Pós-Graduação em Ciência da Computação

"CLASSIFYING METRICS FOR ASSESSING OBJECT-ORIENTED SOFTWARE MAINTAINABILITY: A FAMILY OF METRICS' CATALOGS"

Por

JULIANA DE A. GONÇALVES SARAIVA

Tese de Doutorado

RECIFE
2014

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JULIANA DE ALBUQUERQUE GONÇALVES SARAIVA

"CLASSIFYING METRICS FOR ASSESSING OBJECT-ORIENTED SOFTWARE MAINTAINABILITY: A FAMILY OF METRICS' CATALOGS"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR(A): SÉRGIO C. BRANCO SOARES
CO-ORIENTADOR(A): FERNANDO J. C. DE L. FILHO

RECIFE
2014

Catalogação na fonte
Bibliotecária Alice Maria dos Santos Costa, CRB 4-711

Saraiva, Juliana de Albuquerque Gonçalves.
Classifying metrics for assessing object-oriented software maintainability: a family of metrics' catalogs. – Recife: O Autor, 2014.
132 f. : fig., tab.

Orientador: Sérgio Castelo Branco Soares.
Tese (Doutorado) - Universidade Federal de Pernambuco. Cin. Ciência da Computação, 2014.
Inclui referências e apêndices.

1. Engenharia de software. 2. Software - Manutenção . 3. Medição de software. I. Soares, Sérgio Castelo Branco. (orientador). II. Título.

005.1 (22. ed.) MEI 2014-100

Tese de Doutorado apresentada por **Juliana de Albuquerque Gonçalves Saraiva** à Pós Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Classifying Metrics for Assessing Object-Oriented Software Maintainability: A Family of Metrics' Catalogs**" orientada pelo **Prof. Sergio Castelo Branco Soares** e aprovada pela Banca Examinadora formada pelos professores:

_____
Prof. Paulo Henrique Monteiro Borba
Centro de Informática / UFPE

_____
Prof. Fabio Queda Bueno da Silva
Centro de Informática / UFPE

_____
Profa. Renata Maria Cardoso Rodrigues de Souza
Centro de Informática / UFPE

_____
Prof. Manoel Gomes de Mendonça Neto
Departamento de Ciência da Computação / UFBA

_____
Prof. Claudio Nogueira Sant'Anna
Departamento de Ciência da Computação / UFBA

Visto e permitida a impressão.
Recife, 21 de fevereiro de 2014.

_____
Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

To my mom, ***Laura A. Mauricio****, my life's example, for her dedication and effort during her whole life to provide me the best education.*

To my sister, ***Jeanine de A. Gonalves****, for supporting me in all my achievements.*

Acknowledgements

First of all, I would like to thank God, for giving me enough strength to conclude this cycle of my life.

I would like to thank my husband, Tibrio de A. Saraiva, for making me feel special (at least for him) and competent when I have to face on any challenge in my life. You are such a lier, but I believed in you, then I finished it! I love you so much, forever!

I have to thank my son, Joo Victor G. A. Saraiva, and my daughter, Las G. A. Saraiva, for their patience with me during this period of my life. You two are my engine of motivation to keep walking.

I am thankful to Evelyn Saraiva (mother in-law) and Joo Saraiva (father in-law) for believing in my competence to conclude this period of my life.

It is important to remember and thank my advisor and co-advisor for guiding me during my Ph.D. program. I am sure that the majority of my professional skills, I learned from you two. Thank you so much!

I also would like to thank the professors and students of the Informatics Center (CIn-UFPE) that contributed to my education and my project execution. In special, thank the SPG (Software Productivity Group) for all your feedback and collaboration during my Ph.D. research.

Adauto Filho and Emanoel Barreiros, I am very thankful to have you as my friends. Thank you so much for supporting me during my Ph.D. research. You are special people that God send us to make the life better. You are my fellows in my personal and professional life.

I would like to thank the professors Cludio Sant'Anna, Fbio Queda, Manoel Mendona, Paulo Borba, and Renata Souza for accepting the invitation to compose my Ph.D. committee. It is my pleasure to have all of you assessing and contributing with my work.

I would like to thank the FACEPE (Foundation for Supporting Science and Technology of Pernambuco State) and INES (National Institute of Science and Technology for Software Engineering) for their financial support.

Finally, I would like to thank my family and friends that, directly or indirectly, collaborated and helped me in these four years of a hard and persistent work to conclude this Ph.D. program.

When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.

—W.T.L.K (William Thomson Lord Kelvin)

Resumo

Atualmente, Programao Orientada a Objetos (POO) um dos paradigmas mais utilizados. Complementarmente, a manutenibilidade de software considerada um atributo de software que desempenha um papel importante com relao ao nvel de qualidade. Neste contexto, a Manutenibilidade de Software Orientado a Objetos (MSOO) foi estudada atravs de anos e vrios pesquisadores propuseram um elevado nmero de mtricas para a medir. Como consequncia do nmero e da diversidade de mtricas existentes, alm da no padronizao nas definies e nomenclatura, a tomada de deciso sobre quais mtricas podem ser adotadas para realizar estudos em MSOO difcil. Desta forma, um mapeamento sistemtico foi realizado a fim de encontrar quais mtricas so usadas como indicadores de MSOO. Houve uma seleo inicial de 5175 estudos primrios e 138 artigos foram selecionados, resultando em 568 mtricas encontradas. Analisando as 568 mtricas, foram encontradas inconsistncias na nomenclatura destas mtricas, pois havia mtricas com nomes iguais mas significados diferentes (8 casos envolvendo 17 mtricas) e tambm mtricas com nomes diferentes e significados semelhantes (32 casos envolvendo 214 mtricas). Alm disto, uma categorizao destas mtricas foi proposta, sendo identificadas 7 categorias e 17 subcategorias. Estas categorias representam os cenrios de adoo de mtricas de MSOO. Adicionalmente, um portal web de mtricas foi desenvolvido para fornecer informaes sobre as mtricas para outros pesquisadores e tambm gerar catlogos de mtricas de acordo com o contexto da aplicao das mesmas. Este portal tambm pode ser alimentado sistematicamente por outros pesquisadores que lidam com mtricas de MSOO, fazendo com que os resultados deste trabalho possam representar os primeiros passos para padronizao e compreenso destas mtricas. Por ltimo, um quasi-experimento foi realizado para checar o grau de cobertura do catlogo proposto pela abordagem aqui apresentada quando o mesmo comparado com catlogos sugeridos por especialistas. 90% de cobertura foi obtido e este resultado foi confirmado com 99% de grau de confiana usando o Teste de Wilcoxon. De forma complementar, houve uma pesquisa de opinio para checar se os especialistas acharam catlogo gerado usando a nossa abordagem semelhante ou melhor do que o sugerido por eles. Sendo assim, os resultados da anlise da cobertura dos catlogos pode servir como indcios da utilidade da abordagem proposta para a escolha de mtricas na avaliao de MSOO.

Palavras-chave: Manutenibilidade de Software. Metricas. Desenvolvimento de Software Orientado a Objetos. Engenharia de Software Experimental

Abstract

Currently, Object-Oriented Programming (OOP) is one of the most used paradigms. Complementarily, the software maintainability is considered a software attribute that plays an important role in quality level. In this context, the Object-Oriented Software Maintainability (OOSM) has been studied through years, and many researchers have proposed a large number of metrics to measure it. As a consequence of the number and diversity of metrics, beyond the no standardization in metrics definition and naming, the decision-making process about which metrics can be adopted in experiments on OOSM, or even their using in software companies is a difficult task. Therefore, a systematic mapping study was conducted in order to find which metrics are used as indicators in OOSM assessments. There was an initial selection of 5175 primary studies and 138 were selected, resulting in 568 metrics found. Analyzing the 568 metrics, inconsistencies in metrics' naming were found because there were metrics with the same names but different meanings (8 cases involving 17 metrics) and also, there were metrics with different names, however with similar meanings (32 cases involving 214 metrics). Moreover, a metrics' categorization has been proposed to facilitate decision-making process about which ones have to be adopted, and 7 categories and 17 subcategories were identified. These categories represent the evaluation scenarios where OOSM metrics should be used. Additionally, a metrics' web portal was developed to provide information about the metrics collected in this research, and to generate metrics' catalogs according to the context of their adoption. This portal can also be systematically fed by other researchers that work with OOSM metrics, making the results of this work the first steps towards metrics' standardization, and the improvement of the metrics' validation. Finally, a quasi-experiment was conducted to check the coverage index of the catalogs generated using our approach over the catalogs suggested by experts. 90% of coverage was obtained and this result was confirmed with 99% of confidential level using the Wilcoxon Test. Complementarily, a survey was conducted to check the experts' opinion about the catalog generated by the portal when they were compared by the catalogs suggested by the experts. Thus, the coverage evaluation can be the first evidences of the usefulness of the proposed approach for metrics' choice in OOSM evaluation.

Keywords: Software Maintainability. Metrics. Object-oriented Software Development. Experimental Software Engineering

List of Figures

List of Tables

List of Acronyms

Summary

1 Introduction

This chapter presents the work's overview. The research context and motivation are shown in Section 1.1. In sequence, Section 1.2 describes the research problem and the research questions raised in this work. After that, the study goals are depicted in Section 1.3. The solution proposed to solve the research problem and the summary of thesis' contributions are shown in Section 1.4. Finally, the thesis' structure is exposed in Section 1.5.

1.1 Context and Motivation

Nowadays, Object-Oriented Programming (OOP) is one of the most widely used programming paradigms (TIOBE, 2013). Thus, programming languages that adhere to the OOP paradigm, such as C#, C++, Visual Basic, Java, Python, .NET, Objective-C, are among the most popular languages adopted in globally known software development companies such as, Microsoft (MICROSOFT, 2013), Apple (APPLE, 2013), and so on. In addition, they are languages predominantly used in source code repositories, such as, Github (GITHUB, 2013), Tigris (TIGRIS, 2013), JavaForge (JAVAFORGE, 2013), CodeFlex (CODEFLEX, 2013), Source Forge (FORGE, 2013), and Google Code (CODE, 2013). Consequently, considering the scenario aforementioned, our research was dedicated to the OOP languages.

Complementarily, maintainability, which is one software attribute observed to measurement of software quality, plays an important role in quality level (LIN, 2010). The less effort/cost during the software maintenance cycle, the higher the software's quality level (SOMMERVILLE, 2007). Therefore, it is necessary more research on software maintainability. Thus, the work presented here is focused on Software Maintainability, specifically, on Object-Oriented Software Maintainability (OOSM).

Since it is difficult to measure software maintainability directly without measuring the actual maintenance process, researchers and practitioners often use product metrics as indicators (BASILI; BRIAND; MELO, 1996; SANTANNA et al., 2003). In this context, OOSM has been studied throughout the years, and several researchers have proposed a large number of metrics for software maintainability (ABDI; LOUNIS; SAHRAOUI, 2006; BESZEDES et al., 2007;

OLIVEIRA et al., 2008; MINGGUANG et al., 2009; KULKARNI; KALSHETTY; G.ARDE, 2010; ALSHAMMARI; FIDGE; CORNEY, 2010; REVELLE; GETHERS; POSHYVANYK, 2011; SARAIVA; SOARES; CASTOR, 2010). As a matter of fact, several metric suites have been used as indicators in both quantitative and qualitative software engineering research. The literature provides a large number of maintainability metrics to analyze different characteristics in object-oriented software.

In general, distinct areas such as Software Development, Project Management, and Software Research can adopt them as means to summarize information, point out anomalies, and support decision-making processes. Developers can use metrics to evaluate the level of code maintainability. With the results in hand, they can, for instance, make decisions about which technologies they should use to evolve code. Metrics can also help project managers to evaluate project progress by quantifying issues that should be in constant assessment, such as software productivity (MANTORO, 2009). In addition, metrics have been used by researchers to evaluate the impact of new technologies, such as a new design pattern or new programming paradigm (ABDI; LOUNIS; SAHRAOUI, 2006; ALSHAMMARI; FIDGE; CORNEY, 2010; SARAIVA; SOARES; CASTOR, 2010).

Nevertheless, there is no standardization or a catalog to summarize the information about these metrics, helping the researchers/ practitioners in their decision-making about which metrics can be adopted in OOSM evaluations. In spite of the widespread use of metrics, several researchers look at this topic with extra caution and skepticism (SHEPPERD; INCE, 1994; JONES, 2010; MAYER; HALL, 1999; UMARJI; SEAMAN, 1999; KANER; BOND, 2004; ARKSEY; O'MALLEY, 2005; UMARJI; SEAMAN, 2008; KITCHENHAM, 2010). One of the reasons for this negative stance is that considering the number of available metrics, many of them are not empirically evaluated (HUDLI; HOSKINS; HUDLI, 1994; CONCAS et al., 2010; YAZBEK, 2010; HAN et al., 2010).

Additionally, the large number of metrics poses many challenges to researchers/ practitioners. Firstly, because there are so many metrics and their descriptions are scattered throughout a number of different papers, where some of them mention the metrics but do not actually explain them (SARAIVA et al., 2012). Therefore, the metrics' finding and understanding is a difficult process. Secondly, because they have different levels of empirical evaluations. There are metrics that have been employed in tens of studies, whereas others are only mentioned in the paper that first presented them. Hence, assessing metrics reliability is also difficult.

This situation leads researchers to duplicate metrics or use only metrics of studies they already know. As a consequence, they might not conduct precise and reliable assessments because they are unable to find adequate metrics for their purposes. Therefore, the thesis' research was conducted considering the previously context described, trying to deal with this huge amount of information and different types of OOSM metrics spread in the literature. It is important to highlight that OOSM metrics are predictors for software maintainability, however these metrics can measure other software characteristics and attributes.

1.2 Research Problem and Questions

Faced by the research motivation presented before, the thesis' research problem is **the lack of useful information about OOSM metrics that support the decision-making process about which ones can be adopted in OOSM evaluations**. Metrics information means metrics' name, description, whom proposed/adopted them, tools that collect them automatically, the process description of metrics' validation, and so on. This study aims to query the literature and map maintainability-specific metrics and their use to measure OOSM. Thus, the thesis' research questions are:

- **RQ1:** What metrics were adopted to assess software maintainability in Object-Oriented Software Development (OOSD)?
- **RQ2:** What OOSM metrics are most widely adopted?
- **RQ3:** How to organize information about OOSM metrics to support the decision-making process in the scenario of metrics adoption?

1.3 Goals

Considering the research problem and research questions previously shown, the main goal of this work is **to provide metrics' information and support the generation of context-based catalogs that can be initially used towards the definition of which metrics will be adopted by researchers/practitioner interested in OOSM evaluation**. We expect provide useful information about OOSM metrics for developers, project managers, and researchers that do not have experience in metrics adoption. Consequently, they can consult a metrics' catalog similar to that one that would be suggested by experts in OOSM metrics. Then, to achieve the main goal, four specific goals were proposed:

1. To consistently map information about existing OOSM metrics;
2. To disseminate information about OOSM metrics and their applicability;
3. To develop tools that support the decision-making process about the adoption of OOSM metrics;
4. To evaluate the usefulness of the proposed catalog and associated tools.

1.4 Solution and Summary of Contribution

The main contribution of this work is the categorization of the metrics, and consequently, the possibility to generate OOSM metrics' catalogs based on the adoption scenarios. Thus, the

availability of a portal with specific information about OOSM metrics implements the solution proposed to solve the lack of useful information about this type of metrics. It can be accessed in `http://julianasaraiva.info/oosmMetricsPortal`. It contains information about the metrics, such as metrics' name and description, published papers that addressed OOSM metrics, metrics' categorization, and so on. In addition, the portal provides a way for systematically collaboration from other researchers interested in this subject. They can login with an specific profile (project collaborator) and insert/update information about the metrics.

In summary, the thesis' contributions are:

- **The gathering and organization of all possible information about object oriented software maintainability metrics available until June of 2011**. We investigated and summarized all possible information about OOSM through a systematic mapping study performed. With it, we could identify 568 metrics that have been adopted in OOSM evaluation in both industrial and academic scenarios.

- **The providing of a list with the relevant metrics in OOSM assessments**. With all the metrics' information in hands, we checked which metrics were most mentioned by other researchers, different from who proposed them. So, it was possible to observe the relevance of them. In addition, we analyzed the Google Scholar portal to identify the number of citations of published papers that addressed metrics in OOSM assessment. From those papers, we extract the metrics contained in them and listed the most relevant/adopted metrics, considering the number of citation of them.

- **The metrics' naming inconsistencies identification and solution**. Out of 568 metrics identified, 2 scenarios of naming' inconsistencies were found: (i) metrics with same names and different meanings, and (ii) metrics with different names and similar meanings. For the first scenario, 8 cases were identified, involving 17 metrics. On the other hand, for the second scenario, 32 cases were identified, involving 214 metrics. These inconsistencies can be one of the consequences of the lack of the useful information about OOSM metrics. Different researchers have been proposing new metrics, which are already published, and sometimes, validated, because they do not know where they can find this kind of metrics consistently.

- **A metrics categorization for characterization of OOSM evaluation context**. Through the 138 published papers that addressed OOSM metrics, selected in the systematic mapping performed, it was possible to extract information that support a metrics' categorization. This categorization was proposed through the assessment of the metrics' application context.

- **The automatically catalogs' generation, based on the assessment context**. As the portal has all information about OOSM metrics gathered in this research, it can provide a

catalog generation based on the context of OOSM evaluation. The researcher/practitioner can choose the evaluation context through the metrics' categorization, and the portal provides a catalog that can be, at least, initially consulted by researchers/practitioners in their research. The catalogs provide the metrics information, identify the metrics' naming inconsistencies, and rank the metrics based on their relevance. In addition, we performed a quasi-experiment to check the usefulness of the catalogs generated with our approach. Complementarily, a survey was done to check the metrics' expert opinions about the catalogs generated through our approach.

1.5 Thesis Structure

This section presents the thesis' structure. The chapters were organized to make easier for any reader the comprehension of the research context, problem, the methodology applied in the research, the presentation and discussion of the results, and finally the concluding remarks with the conclusions and future works. The thesis is composed by six chapters.

First of all, the context and motivation of our research is shown in the current chapter. Chapter 2 presents the fundamentals and concepts of software maintainability and software metrics. In addition, the strategies of empirical studies in Software Engineering adopted in this study are also shown. In Chapter 3, the methods adopted to perform the research are discussed.

After that, the results obtained by the execution of Ph.D. project's research are discussed in Chapter 4. The threats to validity are also depicted in the same chapter. The state-of-the-art of OOSM metrics related to our research are shown in Chapter 5. And finally, Chapter 6 presents the concluding remarks. This chapter summarizes the conclusions and indicates the future works scheduled for our research.

2

Background

After presenting the overview about what was done during the Ph.D. research (Chapter 1), this chapter shows the state of the art in object-oriented software maintainability metrics and software engineering research methods used for laying the ground for this thesis. Section 2.1 depicts the definitions of software maintainability adopted in this work. The state of the art in software metrics is shown in Section 2.2. Finally, Section 2.3 details the theories and concepts about research methods used in Software Engineering that were adopted to conduct the study presented here.

2.1 Software Maintainability

2.1.1 Maintainability Definition

International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 9126 standard defines a software quality model (SQA, 2012). Its goal is to deal with human biases that can affect the delivery and perception of a software development project. However, the **SQUARE!** (**SQUARE!**) quality model framework is a series of standards replacing the ISO/IEC 9126 (ISO/IEC, 2011). Three models compose SQuaRE: the quality in use model, the product quality model, and the data quality model. They are depicted in Figure 2.1. These models provide a set of quality characteristics relevant to stakeholders, such as software developers, system integrators, acquirers, owners, maintainers, contractors, quality assurance and control professionals, and users.

Considering the ISO definitions previously mentioned, the work presented here is focused just in the product quality model, that categorizes system/software product quality properties into eight characteristics: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability, which are depicted in Figure 2.2 (ISO/IEC, 2011). Specifically, from this model, our research addresses just the software maintainability issue.

Based on these concepts, maintainability requirements must be delineated in five sub-

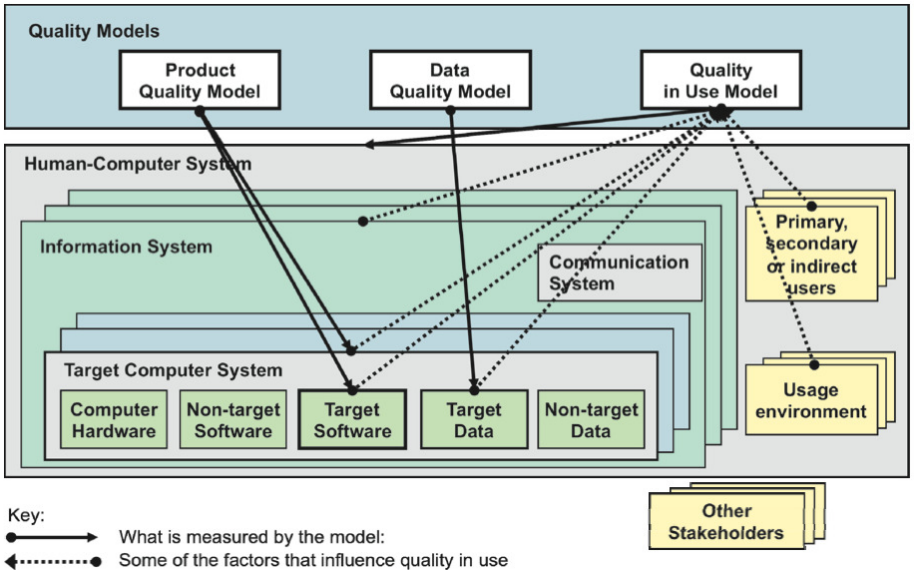

Figure 2.1 Targets of Quality Model (ISO/IEC, 2011).

characteristics: Modularity, Reusability, Analyzability, Modifiability, and Testability (ISO/IEC, 2011). They are shown in the software quality model illustration depicted in Figure 2.2. Firstly, Modularity is the degree to which a system or computer program is composed by a group of components that a change to one component has minimal impact on the other components. By definition, the degree to which an asset can be used in more than one system or in building other assets is known as Reusability.

Analyzability is the degree of effectiveness and efficiency with which it is possible to evaluate the impact on a system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified. It measures the maintainers' effort or resources expended in trying to diagnose deficiencies or causes of failure, or in identifying parts to be modified. It characterizes the ability to identify the root cause of a failure within the software (BOURQUE; DUPUIS, 2004).

Following the definitions, Modifiability is the degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality. And finally, the degree of effectiveness and efficiency with which a test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met is called Testability (ISO/IEC, 2011). Testability measures the maintainers and users effort to test the modified software, characterizing the effort needed to verify (testing) a system change (BOURQUE; DUPUIS, 2004).

Considering the maintainability definitions aforementioned, software maintenance is an activity whose social context is rarely addressed. Thus, Sillito and Wynn conducted an ethnography study at a large technology company observing software engineers (SILLITO; WYNN, 2007). From their observation, they provided a description on how work was divided between groups, the social dependencies that exist between the groups, challenges in managing

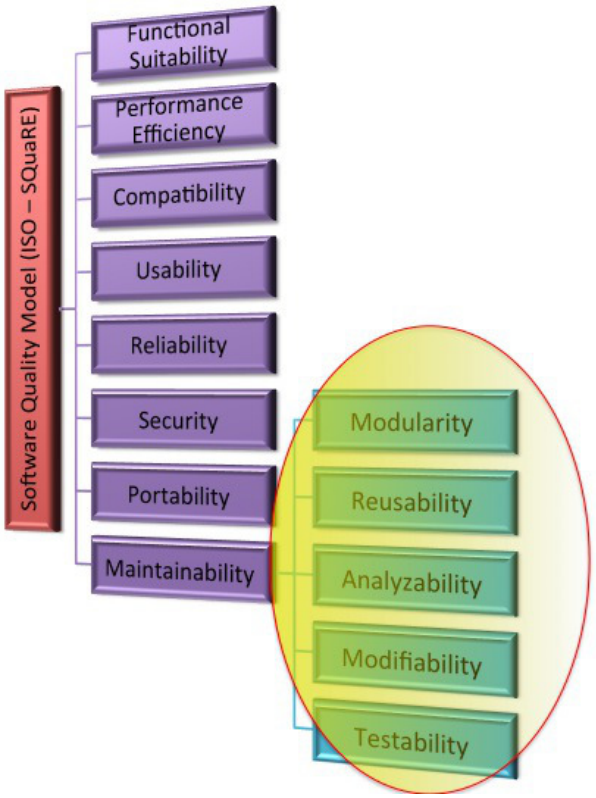

Figure 2.2 Software Quality Model Illustration - ISO/IEC 25010:2011 (ISO/IEC, 2011).

branches, problems of confidence in testing, and so on. One of their interesting findings was that changes were often avoided when testing and managing the impacts of those changes involved dealing with other groups. And more, group structure and geographic locations of group members influenced the decomposition of source code making architecture-level changes difficult (SILLITO; WYNN, 2007). They expected that understanding the social context helped researchers in more effectively addressing challenges faced by practitioners. As their expectations, we hope that research in OOSM (Object-Oriented Software Maintainability) can be improved. Thus, this thesis considered the previously exposed definitions to address software maintainability, regardless of where it is applied.

2.1.2 Maintainability and Software Costs

Maintainability is a characteristic of software commonly related to how easy, accurate, safety, and economic is the performance of software maintenance. A system should be designed such that can be maintained without large investments of time, at least cost, with a minimum impact on environment, and with a minimum expenditure of resources (BLANCHARD; VERMA; PETERSON, 1995). Therefore, some of the objectives for applying maintainability engineering principles are to reduce projected maintenance cost and time through design modifications directed at maintenance simplifications, to use maintainability data for estimating equipment, availability, and to determine the labor hours and other resources required to perform the software

maintenance (DHILLON, 2006).

In this context, maintainability is considered a software attribute that plays a major role in the software quality level. The less effort/cost the software maintenance cycle requires, the higher the software's quality level. Hence, new software development methods, techniques, and tools often aim to minimize future costs in the maintenance process (EADDY et al., 2008). Software Maintenance and Evolution implies a huge cost and slow speed of implementation (BENNETT; RAJLICH, 2000). Bennett and Rajlich performed a study characterizing an overview of maintenance and evolution activities and since that, they mentioned the importance of empirical studies about maintenance and evolution, including process, organization, and human aspects. In that context, they identified reverse engineering as one programming technique which has not been exploited industrially (BENNETT; RAJLICH, 2000). In addition, their work claimed the position of software evolution as the center of software engineering.

In addition, thinking in minimizing software maintenance cost, Tairas proposed some techniques that determined novel factors influencing code clone analysis and refactoring (TAIRAS, 2008). He developed a process to unify phases of code clone detection, analysis, and refactoring. In addition, he also provided an alternative graphical representation of clones maintenance reducing the total time required for maintenance when compared to other techniques where the analysis and refactoring phases are still separated.

The assessment of software maintainability is a concern in both, academic and industrial scenarios. Consequently, there are also researches that address this topic in software companies. Thus, through years researchers had dedicated their studies to minimize the gap between theory and practice, trying to make efficient the new adoptions of technologies in industrial environment. One of this study is the O'Brien et al. work (O'BRIEN; BUCKLEY; EXTON, 2005). They presented a review of the empirical work carried out in the area of program comprehension and illustrates that most of the evidence from these studies derives from lab-based experiments, thus implying a degree of artificial control. After the evaluation of some approaches dedicated to quantitative and qualitative assessments, they suggested that there is a mandatory role for both qualitative and quantitative approaches if we hope to gain a more complete and realistic understanding of industrial programming behavior.

In the same context, Rombach et al. performed a case study providing evidences of both success and failure regarding software application in practice (ROMBACH et al., 2008). Thus their analysis of historic impact chains of research reveals a clear impact of software engineering research on sustained industrial success for inspections, reviews and walkthroughs. More importantly, in impact chains where the empirical results have not been established, they concluded that success has not been achieved or has not been sustained. They found that companies that report repeatable success tend to employ well-defined techniques for the analysis of code and other documents. And more, the benefits observed from sustained review programs have been facilitated by various Software Engineering research results. Finally, they encouraged the community to provide comments, challenges, or support and additions to their findings.

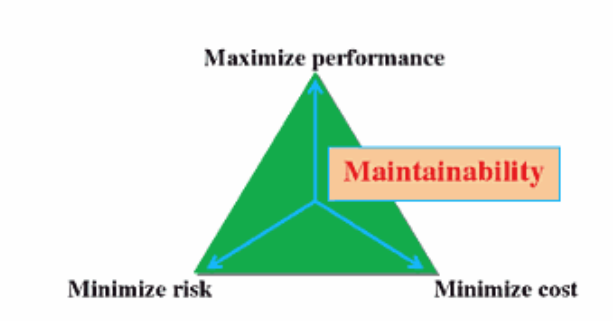

Figure 2.3 Maintainability Definition (LIN, 2010).

Consequently, it is notable the importance and influence of software maintainability in the software system.

2.1.3 Maintainability in Software Life Cycle

In terms of software's life cycle, maintainability is the ability to achieve the optimum performance throughout the lifespan of a software system within the minimum life cycle cost (LIN, 2010). It requires a clear understanding of issues associated with durability, serviceability, and sustainability for the proper selection of materials, components, systems, as well as the effect of and impact to, the environment. So, maintainability has to provide a system maximized performance, minimizing the risks and costs as depicted in Figure 2.3.

It is important to highlight that maintainability requirements must be planned for and included within the overall planning documentation in a software life cycle. So, the maintenance activity should be specified in the top-level specification, designed in the iterative process of functional analysis, and measured in terms of adequacy through system evaluation (DHILLON, 1999).

Therefore, the effective practice of maintainability design, and engineering requires a systematic management approach. So, maintainability issues should be arise throughout the product life cycle (DHILLON, 1999). Specific maintainability functions are associated with each of the software life cycle phases. During the concept development phase, the primary task is to determine the product effectiveness requirements and to determine, from the purpose and intend operation of the product. In this phase, it is necessary to determine the product utilization rates, mission time factors, and product life cycle duration, including product use and out-of-service conditions.

On the other hand, in the validation phase, the maintainability management tasks include developing a maintainability program plan that meets contractual requirements and other plan for maintainability testing and demonstration (DHILLON, 1999). After that, in the production phase, some activities involved reviewing product design with respect to maintainability features, preparing maintainability design criteria and guidelines, participating in design reviews to safe-guard the interests of maintainability. Later, maintainability documentation consists of tasks such

Figure 2.4 Maintainability Life Cycle Phases.

as developing maintainability data and feedback reports, establishing and maintaining documents and information related to maintainability management. Finally, the maintainability coordination is an activity that interfaces with product engineering and other engineering disciplines. The whole product maintainability life cycle phases is depicted in Figure 2.4.

It is widely known that many problem factors in the software development phase affect the maintainability of delivered software systems. Therefore, Chen and Huang performed a study focused on those software development problem factors which may possibly affect software maintainability (CHEN; HUANG, 2009). They classified 25 problem factors into 5 dimensions, and designed a questionnaire and a survey. With that, they suggested that **SPI!** (**SPI!**) can help reducing the severity level of the documentation quality and process management problems, and it is only likely to enhance software maintainability to a medium level. And more, this study had provided empirical evidence that problem factors in the software development phase can negatively affect software maintainability.

In this context, it is possible to understand that maintenance is any actions needed for retailing a system or product in, or restoring it to, a desired operational state, and it can be understood as the following categories (DHILLON, 1999):

- **Corrective Maintenance**: This includes all unscheduled maintenance actions performed, as a result of system/product failure, to restore the system to a specified condition;
- **Preventive Maintenance**: This activity include all scheduled maintenance actions performed to retain a system or product in a specified operation situation. It covers periodic inspections, condition monitoring, critical-item replacement, and periodic calibration;
- **Predictive Maintenance**: This action is related to a condition-monitoring prevention for predicting possible degradation, and for the purposes of highlighting areas where maintenance is required.

Then, considering the organization's needs for maintenance activity, the fulfillment of maintainability objectives is highly dependent on the proper mix of resources and the development of good communications (BLANCHARD; VERMA; PETERSON, 1995). Consequently, the success of the implementation of maintainability program requires a thorough understanding of not only system-level requirements, but also the many organization interfaces that exist. Thus, it is important to identify all people involved in 'Maintainability Engineering' to pinpoint the responsibilities of them in this process.

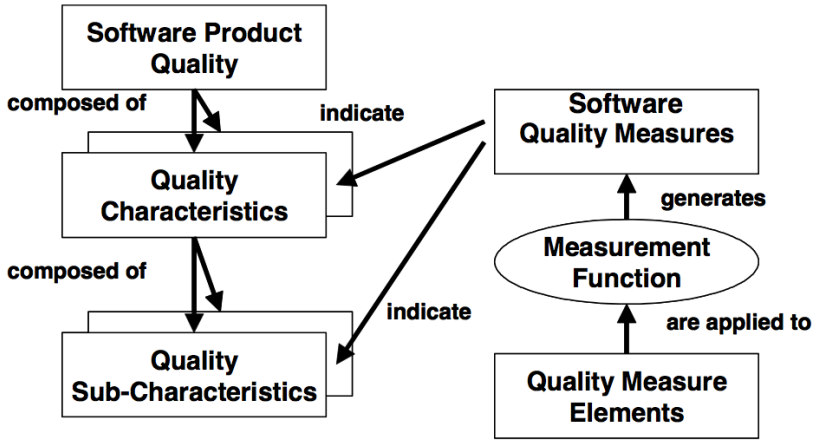

Figure 2.5 Software Product Quality Measurement Reference Model (SPQM-RM) - (JEDLITSCHKA; CIOLKOWSKI, 2006).

Therefore, the customer must recognize the maintainability engineering objectives from the beginning, and the organizational entity needs to be established to ensure that these objectives met. After all exposed, it is clear the importance of researches in software maintainability, which is one of our study's focus. Nevertheless, it is important to highlight that our study is related only to object-oriented software maintainability.

2.2 Software Metrics

2.2.1 Metric Definition

Software metric is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and researchers to bring similar approaches to software development (KAN, 1995). As a matter of fact, software metrics are used in reference to multiple concepts. However in software engineering, they are associated to measurement and quality and estimation models (ABRAN, 2010).

Many of the software measures proposed to the industry have not been seriously analyzed, making them not sufficient mature. In contrast to other fields of science and engineering, these software measures lack the credibility to be used as a basis for decision-making process. The impact of the absence of software measure credibility is when objective and quantitative data are required for decision-making in software engineering. Therefore, software engineering researchers and practitioners must often design and develop their own individual software measures and measurement methods, whereas these already exist in other fields of knowledge (ABRAN, 2010).

Seeking a standardization in software measurement, ISO (International Organization for Standardization) proposed the ISO/IEC 25020:2006 standard (JEDLITSCHKA; CIOLKOWSKI,

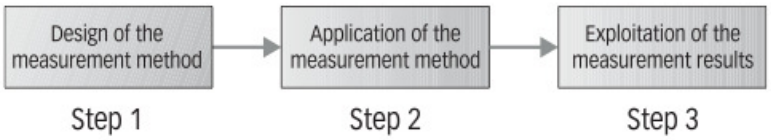

Figure 2.6 Measurement Context Model (The Design View) (ABRAN, 2010).

2006) that deals with the selection and construction of software product quality measures, especially with respect to their use in conjunction with other SQuaRE series documents: Quality Management Division (ISO/IEC 2500n), Quality Model Division (ISO/IEC 2501n), Quality Requirements Division (ISO/IEC 2503n), Quality Evaluation Division (ISO/IEC 2504n), and SQuaRE Extension Division (ISO/IEC 25050 - 25099) (ISO/IEC, 2011; JEDLITSCHKA; CIOLKOWSKI, 2006). It is possible to observe with Figure 2.5 that the model describes the relationship between a quality model and software product attributes with the corresponding software quality measures, measurement functions, quality measure elements, and measurement methods.

According this model, there are three different types of software quality measures defined to correspond to the software product quality life cycle: (i) Internal Software Quality Measures, (ii) External Software Quality Measures, and (iii) Measures of Quality (JEDLITSCHKA; CIOLKOWSKI, 2006). The first group is applied to a part of a software product during its development phases such as, requirements definition, design specification or source code. On the other hand, External Measures are used to measure the quality of the software product based on the behavior of the system of which it is a part. And the last one is used to measure the extent to which a product meets the needs of specific users with respect to their specific personal or business goals.

The purpose of software metrics is to help answer the questions which arise as the life cycle progresses (PERLIS; SAYWARD; SHAW, 1981). They help in making decision about the time to go on the next stage or returning to an earlier stage. There are thousands of metrics available in every scientific and engineering field. However, it is important to highlight that software metrics needs to determine the experimental constructs applied to software. A useful beginning would be the identification of software hypotheses. In this context, experimentation is crucial if we are interested in substantiating or refuting hypotheses.

Software measurement is definitively a young technology, and, as such, it shares many of the characteristics of new technologies and as well as the constraints that must be tackled to facilitate its adoption by industry at large and by individual practitioners (ABRAN, 2010). Then, in measurement context there are three steps to be followed in this process as depicted in Figure 2.6.

Before measuring, it is necessary to either select a measurement method, if one already exists, or to design one (Step 01). Once the measurement method has been designed, its rules

are applied to software or a piece of software to obtain a specific measurement result (Step 02). Finally, the measurement result is exploited in a quantitative or qualitative model, usually in combination with other measurement results of different types (Step 03).

In the same context, a goal-driven method for developing and maintaining a meaningful metrics program that is based on three levels: Goals, Questions and Metrics (BASILI; WEISS, 1984; CSIAC, 2012). The approach uses metrics to improve the software development process (and its resulting software products) while maintaining alignment with organization business and technical goals. GQM is a top-down approach to establish a goal-driven measurement system for software development. The team starts with organizational goals, it defines measurement goals, posing questions to address the goals, and identifying metrics that provide answers to the questions.

The GQM approach defines a measurement model on three levels: (i) Conceptual level (goal), where the goal is defined for an object for a variety of reasons, with respect to various models of quality, from various points of view and relative to a particular environment; (ii) Operational level (question) where a set of questions is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal; and (iii) Quantitative level (metric), where a set of metrics, based on the models, is associated with every question in order to answer it in a measurable way. The use of this approach gives us some benefits such as understanding and defining the baseline for an organization software practices, guiding and monitoring software processes, assessing new software engineering technologies, and evaluating and certifying improvement activities (BASILI; WEISS, 1984; CSIAC, 2012).

2.2.2 Metrics Assessment

As aforementioned, the standard ISO/IEC 25020 presents a software quality measurement (JEDLITSCHKA; CIOLKOWSKI, 2006). It also describes some issues affecting the reliability and/or validity of metrics. According to the standard authors, the procedures and instruments used for collecting quality metrics elements and source of data can affect the measurement reliability. Consequently, trying to minimize the bias during the assessment of measurement validity and reliability, the quality model have suggested some methods that can help metrics' users during those evaluation (JEDLITSCHKA; CIOLKOWSKI, 2006).

First of all, considering the measurement validity, they mentioned five (05) ways to evidence the validity: (i) Correlation, where an measurement user can predict quality characteristics without measuring them directly by using correlated measures; (ii) Tracking, that an measurement user can detect movement of quality characteristics along a time period without measuring directly by using those measures that have tracking ability; (iii) Consistency, in which a measurement user can notice exceptional and error prone components of software by using those measures which are capable of being consistent; (iv) Predictability, where an measurement

user can predict the movement of quality characteristics in the future by using those measures, which are within the allowed prediction error range, and; (v) Discrimination, that indicates that a measure should be able to discriminate between high and low quality for software characteristics and subcharacteristics.

On the other hand, considering the measurement reliability, it should be approached from two perspectives: (i) repeatability: the degree of repeated use of the base measure in the same organizational produces results that can be accepted as being identical, and (ii) reproducibility: the degree of repeated use of the metric in the same organizational unit following the same measurement method under different conditions produces results that can be accepted as being identical (JEDLITSCHKA; CIOLKOWSKI, 2006).

More recent researches in metrics validation have been performed, such as the Shanthi and Duraiswamy study (SHANTI, 2011). They presented an empirical validation of software quality metric suites on open source software for fault-proneness prediction in object-oriented systems. The three metrics used here are Chidamber and Kemerer (CK) Metrics, Robert C. Martin Metric Suite and McCabe Metric Suite. From the results and empirical analysis, it was clear that the different metric suites have different efficiency in faults prediction. With the aid of this empirical analysis, they can suggested software professionals to find out those metric suites can predict faults while developing the quality metric software products using the OO approach.

Furthermore, there is also research assuming the philosophic spectrum of software metrics validation. It is the case of Meneely et al. study (MENEELY; SMITH; WILLIAMS, 2012). They debated over what constitutes a valid metric process on software metrics validation criteria. For that, they conducted a systematic literature review that began with 2,288 papers and ultimately focused on 20 papers. After extracting 47 unique validation criteria from these 20 papers, they performed a comparative analysis to explore the relationships amongst the criteria. With the results they concluded that metrics validation criteria provide answers to questions that researchers have about the merits and limitations of a metric.

In parallel with empirical studies for metrics evaluation, there are some practical assessments of software metrics. Then, Westfall introduced the reader to a practical process for establishing and tailoring a software metrics program that focuses on goals and information needs (WESTFALL; ROAD, 2005). The process provides a practical, systematic, start-to-finish method of selecting, designing, and implementing software metrics. It outlines a cookbook method that the reader can use to simplify the journey from software metrics in concept to delivered information. His contribution indicated that a metrics program based on the goals of an organization will help communicating and measuring progress, and eventually, attaining those goals is a valuable support.

A large number of software metrics have been proposed in the literature, but there is little understanding of how these metrics relate to one another. Then, Cinneide et al. proposed a novel experimental technique, based on search-based refactoring, to assess software metrics and to explore relationships between them (CINNEIDE et al., 2012). They wanted to improve

the program being refactored, but it was necessary to assess the software metrics that guide the automated refactoring through repeated refactoring experiments. Their results demonstrated that cohesion metrics disagree with each other in 55% of cases, and showed how their approach can be used to reveal novel and surprising insights into the software metrics under investigation.

On the other hand, Ciolkowski et al. research was dedicated to develop and evaluate the measurement instrument in industrial scenario (CIOLKOWSKI et al., 2007). The instrument is based on the **TAM!** (**TAM!**) (DAVIS, 1989) and customized to project controlling. They illustrated the application and evaluation of this measurement instrument in the context of industrial case studies and provide lessons learned for further improvement. They concluded that the scale items for the control center administration view will be completely reworked to get an acceptable reliability, and that interactions between different control goals will become an issue in order to detect and resolve goal conflicts. In addition, they supported the idea that easy-to-use visualizations are a key factor for ease of use and, therewith, for the acceptance of a project control center in a productive environment.

2.3 Empirical Studies in Software Engineering SE

Many ideas in Software Engineering (SE) are adopted without empirical data to support them (Juristo; Moreno, 2001). Ideas, whose truthfulness has not been tested against reality, are continually assumed as truth. Although there are some exhaustive experimental studies in the computer science literature, this is not the general rule. Consequently, the main idea of conducting empirical studies in SE is to help empirical investigation of software projects, and evaluate design principles developed in the research literature and elsewhere (Juristo; Moreno, 2001). Section 2.3.1 presents the research methods applied to SE. The research instrumentation, and examples of data collection strategies are described in Section 2.3.2.

2.3.1 Empirical Research Strategies Applied to SE

One way of classifying empirical studies is considering two approaches: quantitative and qualitative (Juristo; Moreno, 2001). The first one aims to get a numerical relationship between several variables or alternatives under examination. On the other hand, the qualitative approach aims to examine objects in their natural setting rather than looking for a quantitative or numerical relationship, attempting to make sense of, or interpret, a phenomenon in terms of explanations that people bring to them (MILES; HUBERMAN, 1994). The data collected from this study are usually composed of text, graphics or even images.

In Empirical Software Engineering (ESE), the studies can be classified based on the type. For some authors, they are classified as (i) laboratory experiment, (ii) quasi-experiment, (iii) case studies, and (iv) surveys (WOHLIN et al., 2000; Juristo; Moreno, 2001). Nevertheless, Easterbrook et al. included two other classifications: (v) ethnography, and (vi) action research

(JEDLITSCHKA; CIOLKOWSKI, 2008). In this work we consider the later classification, with 6 kinds of studies. However, as we performed just Experiment/Quasi-experiment and Survey, they are shown in next sections.

2.3.1.1 Experiment and Quasi-Experiment

Experiment can be understood as a test under controlled conditions that is made to demonstrate a known truth, examine the validity of a hypothesis, or determine the efficacy of something previously untried (SHADISH; COOK; CAMPBELL, 2002). The result of an experiment can define cause, effect, and causal relationships. A controlled experiment is an investigation of a testable hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables (Juristo; Moreno, 2001). Each combination of independent variables values is a treatment. Most SE experiments require human subjects to perform some task. Then, during the experiment it is possible to measure the effect of the treatments on the subjects (Juristo; Moreno, 2001).

A precondition for conducting an experiment is a clear hypothesis. The hypothesis (and the theory from which it is drawn) guides all steps of the experimental design, including the decision about which variables to include in the study and how to measure them. The hypothesis is drawn from a theory that explains the effect. It is important to mention that during the experiment reporting process, the research should keep information such as, problem statement, research objective, research context, goals, experimental units, experimental tasks, hypotheses, parameters and variables, data analysis process, threat to validity, and conclusion (JEDLITSCHKA; CIOLKOWSKI, 2008).

The fact that experiments are theory-driven is both a strength and a weakness. It is a strength because basing analysis on hypotheses derived from theories reduces problems of "fishing for results", and if we look for long enough we will find them. On the other hand, being theory-driven forces researchers to decide in advance which variables to ignore, and they might turn out to be important outside the laboratory setting (SIM; EASTERBROOK; HOLT, 2003). Consequently, variants on experiments are possible and can be used in circumstances where a true experiment is not possible. Quasi-experiments are examples of that.

The experimentation can be classified based on its amplitude (Juristo; Moreno, 2001). They can be ***in vitro*** where the experiments are conducted in laboratories, and ***in vivo*** where the level of experimentation should be carried out on real projects, whose developers are prepared to take risks with the purpose of learning about the latest technological innovations.

Quasi-experiments share with all other experiments a similar purpose to test descriptive causal hypotheses about manipulable cause, but there is no random assignment during the process (SHADISH; COOK; CAMPBELL, 2002). In quasi-experiments, the cause is manipulable and occurs before the effect is measured. However, quasi-experimental design features usually create less compelling support for counter factual inferences. In quasi-experiments, the researcher has to enumerate alternative explanations one by one, deciding which are plausible, and then using logic,

design, and measurement to assess whether each one is operating in a way that might explain any observed effect. The difficulties are that these alternative explanations are never completely enumerable in advance (SHADISH; COOK; CAMPBELL, 2002). In quasi-experiments the subjects are not assigned randomly to the treatments (Juristo; Moreno, 2001; SHADISH; COOK; CAMPBELL, 2002). These variations are less powerful than true experiments, and require more careful interpretation.

2.3.1.2 Survey

In other areas, like the social sciences, surveys are very common practice. In the case of SE, surveys would supply knowledge of what development variables affect certain process or products characteristics. They can be applied to evaluate the mean development productivity of an organization or to analyze the mean surplus cost in software projects run by the above organization. Ideally, the more homogeneous the elements examined in the surveys are, the better the results obtained will be (Juristo; Moreno, 2001).

There are many studies proposing survey building or reporting the application of surveys. Punter et al. arisen the awareness of on-line surveys and discussed methods how to perform them in the context of software engineering (PUNTER et al., 2003). In addition, they reported their experience in performing on-line surveys in the form of lessons learned and guidelines. Their major contribution is showing that surveys has an important role in SE research and they allow researchers to learn about the state of the practice, identifying improvement potentials, or investigating the acceptance of a technology. Also, with the increasing pervasion of the Internet it is possible to perform surveys easily and cost-effectively on-line.

Other report about the survey application in industry is the work of Jedlitschka and colleagues (JEDLITSCHKA et al., 2007). They conducted an online survey among German software industry decision makers trying to get relevant information sources for successful technology transfer. Their main findings were that information regarding the impact of technologies on product quality, cost, and development time, as well as on technology cost-benefit ratio is considered most important among decision makers. The preferred sources of information are colleagues, textbooks, and industry workshops. It is important to highlight that surveys can only serve as an initial insight to determine which information researchers should provide to properly reach industry decision makers.

Later, on 2009, John and Eisenbarth analyzed and compared existing approaches of surveys/scoping and derived open and partially addressed research questions that can be tackled by researchers in product line engineering in the next years (JOHN; EISENBARTH, 2009). They identified 16 scoping approaches that have been developed and used in this decade and characterized them in a framework for scoping approaches that is derived from state of the art characterization frameworks for SE methods and for software product line engineering.

Complementarily, Laukkanen and Mantyla surveyed developers from six industrial software development organizations about the defect report information, from three viewpoints:

quality, usefulness, and automation possibilities of the information (LAUKKANEN; MANTYLA, 2011). 72 out of 142 developers completed their survey. They concluded that the quality of defect reports is a problem in the software industry as well as in the open source community. Thus, they suggested that part of the defect report should be automated since many of the defect reporters lack technical knowledge or interest to produce high-quality defect reports.

Another survey in software industry was done by Rodriguez and colleagues (RODRÍGUEZ et al., 2012). They conducted a study on the current stage of agile and lean adoption and usage in the software industry. Then, they surveyed among Finnish software practitioners in 2011, using the membership registry of The Finnish Information Processing Association (FIPA) as a sampling frame. In Finland, there exists an independent association of Finnish ICT professionals and companies called The Finnish Information Processing Association (FIPA). It has about 16.000 professionals as personal members and the number of company members is more than 500. 408 responses were collected from 200 software intensive organizations in the study. The results of the survey reveal that a majority of respondents organizational units are using agile and/or lean methods (58%). Furthermore, lean appears as a new player, being used by 24% of respondents, mainly in combination with agile (21%).

2.3.2 Research Instrumentation/Data Collection Strategies

Independent of the study's nature addressed in SE, consistent research methods should be applied in empirical studies conducting. Consequently, empirical studies have become an important part of SE research and practice (JEDLITSCHKA; CIOLKOWSKI, 2008). Therefore, this section presents discussions on strategies for collecting data, and which strategies are most appropriate in SE evaluations.

A guideline well know for conduct empirical research in SE was proposed by Shull et al. in 2008 (JEDLITSCHKA; CIOLKOWSKI, 2008). They proposed a data collection taxonomy classifying different techniques to perform empirical studies. This classification is composed by three definitions: (i) direct techniques, (ii) indirect techniques, and (iii) independent techniques. Direct Techniques require the researcher to have direct involvement with the participant population. On the other hand, Indirect Techniques require the researcher to have only indirect access to the participants via direct access to their work environment. Finally, Independent Techniques require researchers to access only work artifacts, such as source code or documentation. They believe that selecting an appropriate technique will be influenced by the questions asked and the amount of resources available to conduct the study (JEDLITSCHKA; CIOLKOWSKI, 2008).

Table 2.1: Empirical Studies Techniques Overview - (JEDLITSCHKA; CIOLKOWSKI, 2008)

Technique Group	**Name**	**Used by researchers to understand:**	**Volume of data**	**Used by software engineers for:**
Direct	Brainstorming and focus groups	Ideas and general background about the process and product, general opinions (also useful to enhance participant rapport)	Small	Requirements gathering, project planning
	Interviews and questionnaires	Mental models of product or process	Small	Requirements
	Conceptual modeling Work diaries	Time spent or frequency of certain tasks (rough approximation, over days or weeks)	Medium	Time sheets
	Think-aloud sessions	Mental models, goals, rationale and patterns of activities	Medium to large	UI evaluation
	Shadowing and observation	Time spent or frequency of tasks (intermittent over relatively short periods), patterns of activities, some goals and rationale	Small	Advanced approaches to use case or task analysis
	Participant observation (joining the team)	Deep understanding, goals and rationale for actions, time spent or frequency over a long period	Medium to large	—
Indirect	Instrumenting systems	Software usage over a long period, for many participants	Large	Software usage analysis
	Fly on the wall	Time spent intermittently in one location, patterns of activities (particularly collaboration)	Medium	—
Independent	Analysis of work databases	Long-term patterns relating to software evolution, faults etc.	Large	Metrics gathering
	Analysis of tool use logs	Details of tool usage	Large	—
	Documentation analysis	Design and documentation practices, general understanding	Medium	Reverse engineering
	Static and dynamic analysis	Design and programming practices, general understanding	Large	Program comprehension, metrics, testing, etc.

Table 2.1 shows an overview of these techniques. Column one indicates the group the represents the techniques. Column two indicates the technique, and column three informs uses of the technique for researchers. The fourth column represents the volume of data required to adopt each technique, and finally, the last column indicates for what software engineers (practitioners) used the technique. Based on the kinds of techniques previously described, two kinds of techniques were adopted to perform the thesis' research: (i) literature database analyses, and (ii) online questionnaires application. The first one was used to map and discovery what metrics have been used in OOSM context. On the other hand, the online questionnaires were applied to assess the OOSM metrics catalogs generation approach proposed in this work. Consequently, systematic mapping study and online questionnaires are presented with more detail.

2.3.2.1 Mapping Studies

Mapping Studies (ARKSEY; O'MALLEY, 2005) try to gather all research related to a specific topic. Among other information, it should contain the research questions that he/she wants to answer with the mapping, the literature sources (digital libraries) that will be used in the study and the inclusion/exclusion primary study (paper) criteria. In general, the mapping is composed by three phases. Initially, only the title, keywords, and abstracts are taken into account for paper inclusion. It is important to stress that only papers that are clearly out of scope were excluded in this phase. Then, all potential primary studies were kept for further analysis.

On the second phase, pairs of researchers are composed and all the potentially relevant papers are evenly distributed among them. Therefore, each paper is reviewed by at least two researchers. Each pair has at least one more experienced researcher. The result of this activity is the selection of the primary studies considering the inclusion/exclusion criteria presented in Chapter 3. Each researcher read the whole paper and makes a list of studies that did not match any criteria. If a paper matches at least one of the exclusion criteria, it is excluded from the mapping. During this phase, each pair builds a report of agreements and disagreements regarding the exclusion of each paper in the mapping study according to the evaluation criteria.

The third and last phase is the conflict resolution and final selection. A conflict resolution meeting has to be organized and the disagreements discussed. In this final phase each researcher screen the full paper. During this phase, each pair had a meeting and presented which inclusion/exclusion criterion(s) was selected for each case of divergence in paper inclusion. This meeting was supervised by one of professors in order to help the students with the selection process discussing. The result of this meeting is the final set of primary studies that answer the research questions raised.

Even if an experiment is well planned, there are still many subjective variables that are very difficult to isolate, such as human interactions and behaviors in software development process. One way to aid organizations is providing evidence about the benefits of new technology applied to some contexts, ideally, real life contexts. However, most companies are not willing to risk a project using a new technology that was not thoroughly assessed or whose benefits were not yet demonstrated or justified. Empirical studies, such as Systematic Mapping Study (SMS) and Systematic Literature Review (SLR), are essential to fill this gap, providing reliable data about a given technology, easing its transfer from academy to industry, and consequently adoption. This issue has been widely discussed by several researchers (WOHLIN et al., 2000; TICHY, 1997; BASILI, 1996; ZANNIER; MELNIK; MAURER, 2006; KITCHENHAM, 2009; RIAZ; MENDES; TEMPERO, 2009; ALMEIDA et al., 2011; BARREIROS et al., 2011; SILVA et al., 2011).

Systematic review is a method to identify, assess and analyze published primary studies to investigate research questions (STAPLES; NIAZI, 2007). Staples and Niazi published a work recommending the using of guidelines to conduct systematic reviews. However, they also

suggested that complementary research questions could help clarifying the main questions and defining selection criteria. This approach is adopted in our work, where we have one main research question, and auxiliaries research questions. In addition, they discussed possibilities for automating and increasing the acceptance of systematic review.

There are also empirical studies that addressed and investigated software metrics with systematic reviews. Riaz et al. (RIAZ; MENDES; TEMPERO, 2009) performed a SLR on Software Maintainability trying to collect evidence on software maintainability and metrics. In this study, 15 studies were selected, and their research suggested that there is little evidence on the effectiveness of software maintainability prediction and models.

Silva et al. (SILVA et al., 2011) performed a tertiary study; a mapping study of Systematic Literature Reviews published in journals and conference proceedings. They analyzed the relevant studies, comparing and integrating their findings with two previous tertiary studies. At the end of their study, their conclusions suggested that the SE research community is starting to adopt SLRs consistently as a research method.

Considering the significant popularity of SLR in SE, Staples and Babar performed an investigation using mixed-methods approach (systematically integrating tertiary literature review, semi-structured interviews and questionnaire-based survey) as it is based on a combination of complementary research methods which are expected to compensate each others limitations (STAPLES; NIAZI, 2007). They argued that there is also an apparent and essential need for evidence-based body of knowledge about different aspects of the adoption of SLRs in SE. With their study, they expected that the findings could provide valuable information to readers about what can be expected from conducting SLRs and the potential impact of such reviews.

On the other hand, in Systematic Mapping Study, questions are broader and more general when compared to the ones present on Systematic Literature Reviews (SLRs) (KITCHENHAM, 2004), for example: *What do we know about a topic T?* Before the Systematic Mapping Study (SMS) execution, the researcher have to develop a study protocol (KITCHENHAM; CHARTERS, 2007). Thus, accordingly to the systematic mapping study definition previously presented, we decided to perform a mapping to get all the information about OOSM metrics in the literature. The detailed mapping study protocol is shown in Chapter 3.

2.3.2.2 Questionnaires

Questionnaires are a group of questions exposed in written format (JEDLITSCHKA; CIOLKOWSKI, 2008). It is a technique where the relation between time and cost is efficient. Researchers do not need to schedule sessions with the software engineers to administer them. In addition, they can be filled out when a software engineer has time between tasks. There are two ways to apply them: (i) Paper form-based questionnaires, where they can be transported to the respondent, and (ii) Web-based questionnaires, that cost less and the data are received in electronic form. The latter can also easily collect data from a large number of respondents in geographically diverse locations.

However, it is important to pay extra attention during the form designing, since there is no interviewer, ambiguous and poorly-worded questions can be a problematic issue (JEDLITSCHKA; CIOLKOWSKI, 2008). Even though it is relatively easy for software engineers to fill out questionnaires, response rates can be relatively low which adversely affects the representativeness of the sample. If the objective of the questionnaire is to gather data for rigorous statistical analysis in order to refute a null hypothesis, then response rates must be high. However, if the objective is to understand trends, then low response rates may be enough (JEDLITSCHKA; CIOLKOWSKI, 2008).

The questions can be open or closed (JEDLITSCHKA; CIOLKOWSKI, 2008). In an open question the respondents are asked to frame their own reply. This kind of question avoids imposing any restrictions on the respondent. There are many different ways respondents may choose to answer a question making answers misinterpreted or confused. On the other hand, a question is closed when the respondents are asked to select an answer from a list of predefined choices. In Chapter 3 the questionnaire applied during this Ph.D. research is explained in depth.

3

Research Methodology

The methodology adopted to conduct this Ph.D. research is described in this chapter. Figure 3.1 illustrates the steps executed in the research. First of all, a systematic mapping study was performed trying to find all possible information about object-oriented software maintainability (OOSM) metrics. The systematic mapping phases are detailed in Section 3.1. Metrics' naming inconsistencies were found during the systematic mapping results' assessment. Consequently, a metrics' naming consolidation was done, and this process is described in Section 3.2. After that, Section 3.3 presents a metrics' categorization proposed to make easier the process of metrics' catalogs generation. In this context, an OOSM metrics portal that contains a context-based catalog generator was developed. The portal building process is depicted in Section 3.4. Finally, the method for assessing the proposed catalogs generation approach is shown in Section 3.5.

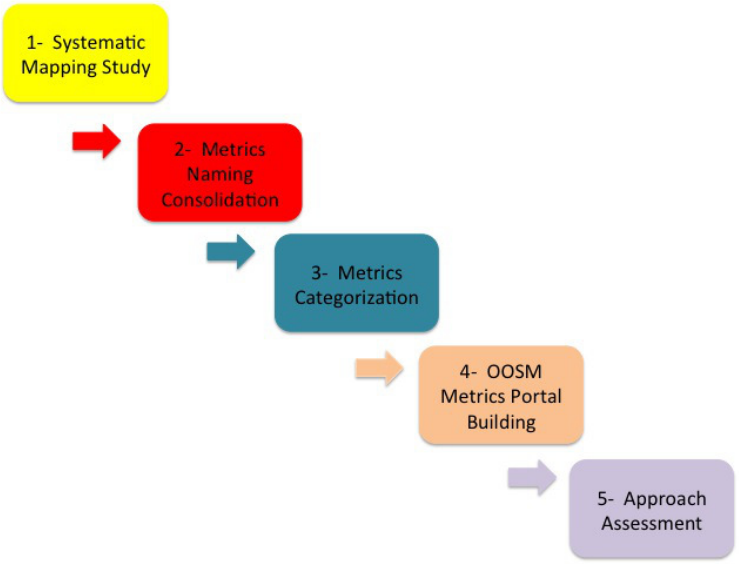

Figure 3.1 Research Methodology Illustration.

3.1 Systematic Mapping Study

Mapping Studies (KITCHENHAM, 2004; ARKSEY; O'MALLEY, 2005; PETERSEN; WOHLIN, 2009) try to gather all research related to a specific topic. As we want to know what metrics have been used as indicators in OOSM assessment in the literature, this method is appropriated to put together information about what we know about this topic. In consequence, this section describes the planning and execution of the systematic mapping study performed. The research questions are presented in Section 3.1.1. After that, Section 3.1.2 shows the digital libraries used, and finally, the search strategy adopted for primary studies selection is detailed in Section 3.1.3.

3.1.1 Mapping Research Questions

This study aims to search the literature and map primary studies that describe maintainability metrics and their use to measure (**!** ((**!**)OO) software development. Thus, the following research question was defined: **What metrics were adopted to assess software maintainability in OOSD?**

3.1.2 Data Source

The search strategy encompasses well-known digital library search engines. They were chosen based on the relevance for the computer science community, and availability of papers for download. The search process was based on automated search using the following digital libraries:

- IEEE Computer Society Digital Library;
- ACM Digital Library;
- EI Compendex;
- Science Direct.

Since the search was performed on different days in different search engines, we decided to limit the final paper publication date to the 1st of June of 2011. Papers published after this date were not considered so as to produce a more homogeneous result and also to allow a future precise update of this study, which can consider publications starting at this date.

3.1.3 Primary Studies Search Strategy

This section describes the search strategy to select the primary studies. The first step was to build a search string. Relevant terms based on key terms used by previous studies and expert insights were defined. The resulting search string from this process was:

Table 3.1 Execution Timetable - Primary Studies Selection

Date	Duration	Phase Description
06/30/2011 to 08/30/2011	2 months	Searching Primary Studies (1st Round).
08/31/2011 to 09/30/2011	1 month	Selection of Papers (2nd Round).
09/05/2011 to 09/25/2011	20 days	Meeting to discuss conflicts.
09/26/2011 to 11/14/2011	1.5 month	Data Extraction and Synthesis.
11/15/2011 to 12/15/2011	1 month	Organize and Reporting Results.

"Software Engineering" **AND** *("Aspect-Oriented Programming" OR "Maintainability" OR "Aspect-Oriented Software Development" OR "Crosscutting Concern" OR "Maintenance" OR "Object-Oriented Programming" OR "Object-Oriented Development" OR "Evolution")* **AND** *("Metrics" OR "Measurement" OR "Measure")*

After formulating the search string, a team of six researchers (3 Ph.D. students and, 3 MSc students) evaluated the search results. In addition, two professors supervised the whole process. Table 3.1 depicts that execution timetable of the papers selection:

In Table 3.1, the first column represents the initial and final dates of each phase execution. The duration of each phase is indicated in second column, and the last column shows the phases. First of all, in the 1st Round potentially relevant primary studies were selected. After that, during the 2nd Round, the result of the first selection was evaluated against the inclusion/exclusion criteria. Lastly, conflicts discussion and final selection were performed.

Initially, during the first round, only the title, keywords and abstracts were taken into account for paper inclusion. It is important to stress that only papers that were clearly out of scope were excluded in this phase. This step was performed by me and another Ph.D. student. Then, all potential primary studies were kept for further analysis. For the second phase of paper selection, the inclusion/exclusion criteria were considered. Following the systematic mapping guidelines, some criteria should be proposed to select a more relevant set of papers, trying to keep only potentially relevant studies after the end of this phase. The inclusion/exclusion criteria were discussed with all researchers involved in this mapping. The aforementioned exclusion criteria are:

- The paper is not a complete research paper (presentation slides or extended abstracts);
- The paper is not related to software engineering;
- The paper does not present maintainability/evolution metrics;
- The paper does not present metrics related to OO or AO programming;
- The paper does not present the metrics description.

On the second phase, three pairs of researchers were composed and all the potentially relevant papers were evenly distributed among them. Each paper was reviewed by at least two researchers. The result of this activity was the selection of the primary studies. For each paper assigned to one of the pairs of researchers, each researcher screened the whole paper to determine whether it matched any exclusion criteria. If a paper matched at least one of the five exclusion criteria, it was excluded from the mapping. It is important to stress that I composed a pair to read the papers. During this part of the process, each pair built a report of agreements and disagreements regarding the permanence of each paper in the mapping study according to the evaluation criteria previously presented. It is important to highlight that the words 'maintainability', 'maintenance', or 'evolution' had to appear in the primary study to be considered a paper that presented a maintainability/evolution metric.

The third and last round was the conflict resolution and final selection. A conflict resolution meeting was organized with me and my advisors, and the disagreements discussed. In this final phase each researcher screened the full paper. The result of this meeting was the final set of primary studies. It is important to highlight that, at the end of the process, a paper was selected if it had at least one maintainability metric, its description, and it was related to OO. All primary studies received a unique identifier so that they could be easily referenced throughout the process. For instance, "SM01" means: Systematic Mapping Study Number 01. For each primary study, I recorded the following information:

- Reviewers (researchers who evaluated the paper);
- Date of data extraction;
- Author(s);
- Journal/Conference where it was published;
- Year of publication;
- Metrics and their acronyms;
- Descriptions of the metrics and information on how to collect them;
- Paradigm (OO or AO).

3.2 Metrics Naming Consolidation

With the list of metrics obtained through the systematic mapping study, some ambiguities and inconsistencies in metrics' naming were an important issue we observed. Consequently, a naming normalization of the metrics found was needed. For the purpose of this study, we consider that the term **consolidation** means to reorganize the information, making it consistent and summarized (useful).

The consolidation process aims to remove the naming ambiguity and inconsistency from the obtained metrics. In this context, two cases of inconsistencies were observed: (i) metrics with different names but essentially representing the same metric, such as **DIT!** (**DIT!**) and **DIH!** (**DIH!**), which are metrics defined as the maximum length from the node to the root of the inheritance tree, and (ii) metrics with the same names and different meanings such as DC, which can be Degree of Cohesion or Descendants Counting. In face of this fact, an assessment of the metrics' definition was demanded to check and identify the naming inconsistencies previously mentioned. For the first case analyzed, the consolidation process was made through four steps:

1. Metrics Grouping;
2. Identification of Ambiguous Description;
3. Matching Correspondence Metrics (same meaning);
4. Selection of the group representative metric.

Firstly, all metrics were grouped based on the software internal attributes that they are related with. Six attributes were identified through the assessment of the primary studies that were selected on the systematic mapping: Size, Inheritance, Coupling, Cohesion, Complexity, and Software Architecture Constraints. Then, the metrics were associated to a software internal attribute based on the metric description contained in the primary study that mentioned it. Figure 3.2 illustrates the process aforementioned.

Afterwards, a metric was chosen to be the representative metric of the group of the correspondent metrics. The choice of the representative metric considered the quantity of primary studies that mentioned it. Then, the representative metric was the one mentioned by the highest number of primary studies. Thus, it was possible to group all the metric. It is important to stress that none of the metrics grouped were disregarded. They were just grouped and identified as similar metrics. In addition, it is important to clarify that similar metrics are not the same metrics. We considered similar metrics, those ones that are associated to the same software internal attribute, and considers the same programming language structures to measure the software attribute.

For the second case of naming inconsistency, all metrics that fit this case were ranked based on the number of times that they appeared on the papers resultant of the systematic mapping study. Consequently, it was possible to know how the metric is widely known by other researchers. Once again, it is important to clarify that none of the metrics were disregarded, but it was possible to identify the naming inconsistency and inform how they are actually understood by the maintainability metric community.

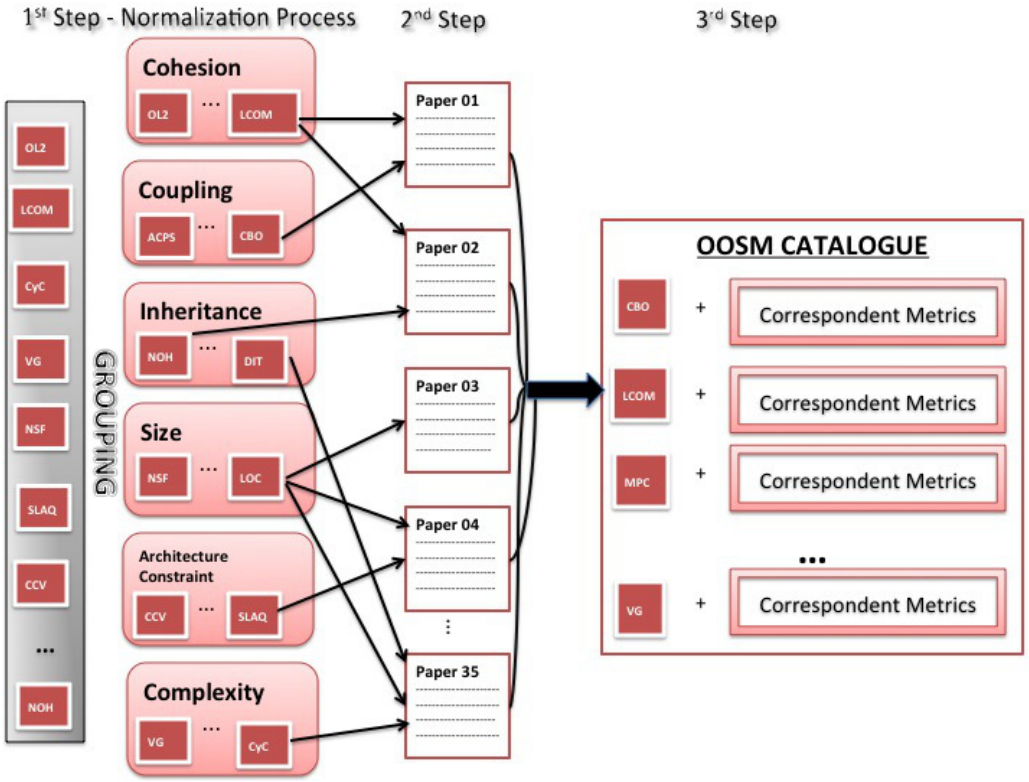

Figure 3.2 Metrics' Naming Consolidation Process.

3.3 Metrics Categorization

As aforementioned, the goal of this research is to provide context-based object-oriented metrics' catalogs to be, at least, initially adopted by researchers and practitioners. Thus, we assumed that the OOSM context can be represented by categories, specifically, metrics categories. The main idea is to take the metrics found by the mapping study as a starting point, and trying to categorize and to summarize these metrics supporting the decision-making process on which metrics to adopt.

We used a Conceptual Grouping Categorization to define the classification groups (VELING; VAN DER WEERD, 1999). For this type of categorization, an element can belong to another group in different levels of pertinence. In this context, a Hierarchy Grouping Assessing method was performed to define the categories (groups) and the level of pertinence for each metric (element) (THE CONCEPTUAL GROUPING EFFECT: CATEGORIES MATTER , AND NAMED CATEGORIES MATTER MORE). The OOSM metrics' categorization were done based on the metrics' definition found in the papers selected in the systematic mapping study previously performed. We believe that with an appropriated categorization considering various domains of metrics' adoptions, a metrics suite will be easily chosen by researchers and software practitioners using a OOSM metrics' catalog generation tool. It is important to mention that a metric can be classified in more than one category because they are not mutually exclusive.

Actually, it is possible that we provide different catalogs for different contexts, based on the metric categorization required by the researcher or practitioner that is using the 'catalog generator'. It is important to clarify that we use the term 'catalog generator' as an infrastructure that provides a list(s) of metrics' catalogs that can be suitable for software maintainability assessment based on the software characteristic to be evaluated. This generator is discussed in more detail in Chapter 4 - Section 4.3.

It is necessary to highlight that all the categories described were proposed based on the assessment of the contexts described by the papers selected in the systematic mapping study. All the contexts were analyzed, and a name was proposed to represent the metric adoption scenario. The categories and subcategories can be seen in Chapter 4 - Section 4.2.3. Once again, it is important to point out that a metric can be fit in more than one category or subcategory.

3.4 OOSM Metrics Portal Building Process

Considering the amount of information about metrics found by this research (see Chapter 4 for more details), a web portal was proposed to propagate all information about OOSM metrics `http://julianasaraiva.info/oosmMetricsPortal`. It contains relevant information about metrics, such as authors that wrote about OOSM metrics, tools that support the automatically metrics collection, published papers related to OOSM metrics, and a tool that generate OOSM metrics catalogs. In addition, the portal provides a questionnaire module that was used to perform the catalog generation approach assessment. Thus, the next sections present the portal building process. Section 3.4.1 presents the portal architecture and project decisions made to build the portal. The modules that compose the portal are explained in Section 3.4.2.

3.4.1 Portal Architecture

As the portal is a web system, all the project decisions were made based on these environment constraints. Consequently, the programming language adopted to develop the portal was Java because it is a multiplatform language, one of the most used object-oriented programming languages, which supports web development. In addition, Java Server Pages (JSP) (GROUP, 2013a), JQuery (GROUP, 2013b), and Java Persistence API (JPA) (ORACLE, 2013) technologies were adopted during the portal building process. The first is a Java technology used to web development. It supports reusing components, and it can be executed in any JSP container, independently of the operation system.

Complementarily, JQuery, that is a JavaScript library, was used to simplify the browser scripts on the client side. JPA was adopted for data persistence. It is a standard Application Programming Interface (API) that facilitates the communication between the applications and databases. The database used was PostgresSQL (POSTGRESQL, 2013). We also used the VRaptor Framework (LAVIERI, 2013), a Model View Controller (MVC) web framework that

aids the programmer in repetitive code through resources, such as validations, dependencies injections, redirection, and so on. The portal adopted layered architecture that can be seen through Figure 3.3. Complementarily, the portal has four users' profile:

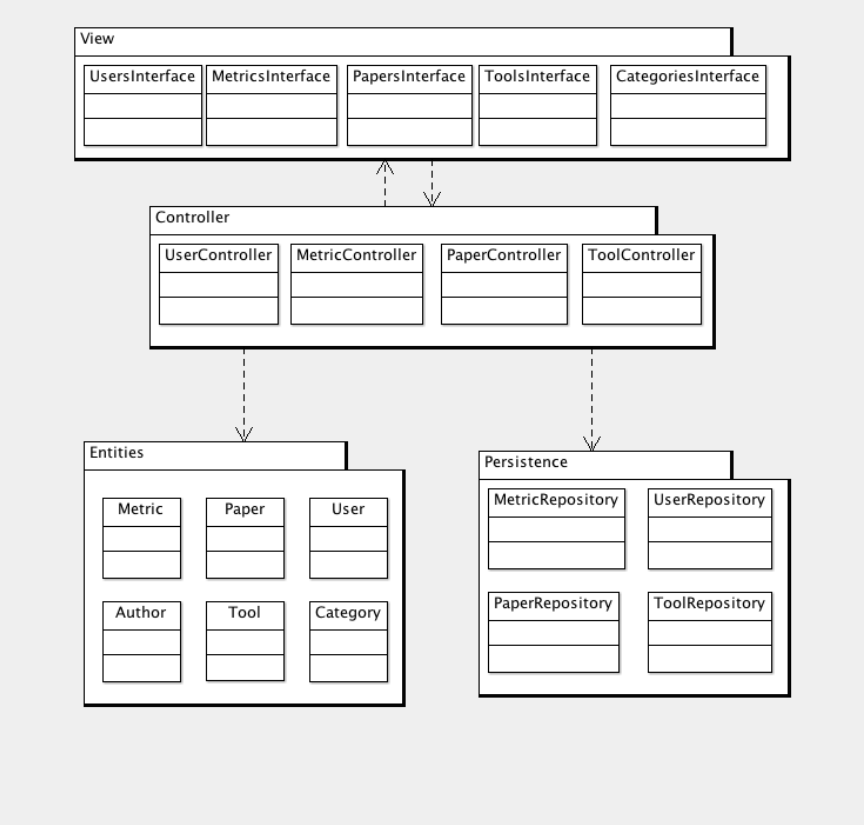

Figure 3.3 Portal Architecture - UML Package Diagram.

- **Visitor**: The users can assess the portal information, however they cannot contribute with the project. They can access Module 01, hereafter explained.
- **Project Collaborator**: They have the same permissions of the Visitors users, but they can also insert information about the metrics, such as new metrics, new descriptions, upload papers in the portal, add new categories/subcategories, and so on. They are directly associated with the Module 02.
- **Quasi-Experiment Participant**: This is a complementary profile, generated for our approach assessment. For these cases, the users have to answer an online questionnaire that will help us in our approach evaluation. Module 03 is allocated for this profile.
- **Portal Administrator**: This is a administrator profile to manage the portal. Features such as permission changing, addition of new users, and others are associated with this profile. Additionally, they can access and manage any module.

Except of the 'Visitor' profile, all other users have to access the portal with a valid login and password. After the login process, the modules options associated with each profile are shown. Even with differences between the profiles, they are not mutually exclusive. It means that a same user can be associated with one or more profiles.

3.4.2 Portal Modules

Three basic modules compose the portal: (i) General Information Module, (ii) Project Information Management Module, and (iii) Catalogs Generation Assessment Module. They were created to support the spreading of OOSM metrics information, and also to generate metrics' catalogs. Additionally, the portal has a module for the catalog generation approach evaluation. It helps us to get some feedback about the proposed approach. All the modules are explained next.

- **Module 01 - General Information**: This is the basic module of the portal. Throughout its website `http://julianasaraiva.info/oosmMetricsPortal`, users around the world can access the portal. By default, all the users have the 'Visitor' profile, and they can access the portal through this module. It contains some metrics information, however the users can only check (read) them. It seems like a metrics' website. If the users would like to collaborate with the project, they have to request a specific profile for the portal's administrators.

  In addition, the user can choose the metrics categories previously indexed, and based on that, a family of catalogs can be generated. We called this process as a 'catalog generation'. After the categories selection, a standard metrics' catalog is provided with information such as, metrics' names, metrics' description, and the categories associated to each metric. In addition, the user can configure the catalog content choosing more information to be shown.

- **Module 02 - Project Information Management**: The systematic feeding of the portal is based on the insertion of the information by the experts and researchers in software maintainability metrics. For this, they need to have the access to this module. Thus, after the login process, they can insert information about metrics, such as news metrics and categories/subcategories, upload papers related to maintainability metrics, tools that collect metrics automatically, and so on.

- **Module 03 - Catalogs Generation Assessment**: This module is dedicated to provide an online questionnaire to be answered by the users that has the system profile as 'quasi-experiment participant'. This module was built because we would like to evaluate the OOSM metrics' catalogs generation of the portal through the users' feedback. Some survey tools were analyzed, however we could not integrate them with the portal. It is important to clarify that the user has to be logged in the system to access this module. All the responses were saved in a database to be statistically assessed.

3.5 Approach Assessment

The research project described here proposes to provide a useful information about OOSM metrics through a web portal that contains, among other things, generation of metrics' catalogs based on the user's context informed (represented by metrics' categories). Therefore, an assessment of the catalog generation was performed to check the feasibility of this approach adoption. Thus, this section describes the quasi-experiment performed. Section 3.5.1 shows the experimentation goals and definitions of the quasi-experiment. The quasi-experiment design and hypotheses are depicted in Section 3.5.2. Finally, Section 3.5.3 presents the method to extract and evaluate the resultant data.

3.5.1 Experimentation Goals

In Software Engineering, an experiment is an investigation of a testable hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables (WOHLIN et al., 2000; Juristo; Moreno, 2001; JEDLITSCHKA; CIOLKOWSKI, 2008). In a true experiment, the environment control is an important issue. If you cannot sample randomly, it is considered a quasi-experiment (Juristo; Moreno, 2001; SHADISH; COOK; CAMPBELL, 2002; SIM; EASTERBROOK; HOLT, 2003; JEDLITSCHKA; CIOLKOWSKI, 2008).

In our case, there are two issues that classify our research method in quasi-experiment: (i) we do not know the actual population of researchers and practitioners that adopt OOSM metrics; (ii) we are not sure about the subjects' knowledge on OOSM metrics. Consequently we cannot categorize them in blocks, based on their expertise. It means that our experimental units were not assigned to conditions randomly. The issues aforementioned make our research method a quasi-experiment.

The quasi-experiment goal is **to check what is the coverage percentage of the OOSM metrics' from the catalog proposed by the catalogs generator over the OOSM metrics' catalogs suggested by researchers** based on their expertise in OOSM evaluations. Some experimental definitions adopted to perform the quasi-experiment can be seen in Appendix E. It is important to highlight that all this experimental definitions were proposed by published papers, guidelines, and books related to experimental software engineering (WOHLIN et al., 2000; Juristo; Moreno, 2001; JEDLITSCHKA; CIOLKOWSKI, 2008).

3.5.2 Quasi-Experiment Design

Two tasks were done during the quasi-experiment design phase: (i) the hypotheses definition, and (ii) the data collection process definition. Therefore, based on the experimentation mentioned at Appendix E, we tried to find any evidence about of the coverage percentage of the

OOSM metrics' catalog generated based on the categories choice over the OOSM metrics' suite proposed by the experts interviewed.

A Coverage Index (CI) was defined for each catalog generated during the experiment (NIE; KAMBHAMPATI, 2004). The CI is the probability of the metrics suggested by the experts belongs to the catalog generated using our approach. Assuming that, we can define Xi as the metrics catalog generated using our approach (categorization choice) by researcher i, and Yi as the metrics catalog generated by the researcher i expertise. Then, for each researcher (i) evaluated, a CI_i was denoted as:

$$CI_i = \frac{\#(X_i \cap Y_i)}{\#Y_i}$$

Where, X_i is the metrics catalog generated using our approach (categorization choice) by researcher i, Y_i is the metrics catalog generated by the researcher i expertise, and # is a function that returns the number of metrics in a catalog. The goal of this evaluation is to infer that our catalogs' coverage is at least 90% over the metrics suggested by experts. Thus, the hypotheses definitions are:

H0: CI < 0.9. In other words, the catalogs generated using our approach has less than 90% of coverage over the catalogs proposed by experts (null hypothesis).

H1: CI >= 0.9. In other words the catalogs generated using our approach has at least 90% of coverage over the catalogs proposed by experts (alternative hypothesis).

We used a direct technique as experiment instrumentation, specifically questionnaires (JEDLITSCHKA; CIOLKOWSKI, 2008). Direct techniques allow the experimenter to obtain a general understanding of the software engineering process. It is composed by brainstorming, focus groups, interviews, questionnaires, and conceptual modeling (JEDLITSCHKA; CIOLKOWSKI, 2008). Interviews and questionnaires are techniques that have been used by researchers when their goal is to understand general information (including opinions) about process, product, or even personal knowledge. It can be adopted for small or large volume of data. As interviews involve at least one researcher talking to at least one respondent, we did not adopt this method, since we would like to interview researchers around the world, and it would be complicated because of the time zone and because of the time to finish this project.

Questionnaires are sets of questions administered in a written format. These are the most common field technique because they can be administered quickly and easily (Juristo; Moreno, 2001; JEDLITSCHKA; CIOLKOWSKI, 2008). As aforementioned, we adopted questionnaires because they are time and cost effective, researchers do not need to schedule sessions with the interviewees to administer them, and they can be filled out when each interviewee has time. Nevertheless, it is important to remember that since there is no interviewer, ambiguous and poorly-worded questions are problematic. Even though it is easy for who will answer the questionnaire to fill out them, they still must do so on their own and may not find the time. This fact can make the response rates relatively low which adversely affects the representativeness of

the sample (WOHLIN et al., 2000; Juristo; Moreno, 2001; JEDLITSCHKA; CIOLKOWSKI, 2008).

The questionnaires were applied online, through the OOSM metrics portal. We generated login and password for each quasi-experiment participant. They could access metrics information contained in the portal only after answering the questionnaire. The subjects group was composed by researchers and practitioners around the world that are involved in OOSM evaluations. It is important to highlight that a dry-run was executed with graduate students and professors of our research group trying to adjust the questionnaire if necessary. The whole questionnaire applied in our research can be observed in Appendix E and it is composed by 5 parts:

1. PROFILE: To inform where the subject came from;
2. EXPERTISE: To indicate the subject's background;
3. METRICS: To collect metrics suggested by the subject for OOSM assessments;
4. CATEGORIES: To characterize the OOSM evaluation's context;
5. COMPARISON: To capture the difference between the catalogs generated by the tool and by the subject.

The respondents were invited based on their experience in software maintainability and software metrics. Their names/emails were collected in the website of known conference proceedings related to the both topics: ICSE (International Conference in Software Engineering), ICSM (International Conference in Software Maintenance), CSMR (European Conference on Software Maintenance and Reengineering), SBES (Simpósio Brasileiro de Engenharia de Software - Brazilian Software Engineering Symposium), SPLASH (ACM Conference on Systems, Programming, Languages and Applications: Software for Humanity), ECOOP (European Conference on Object-Oriented Programming), METRICS Conference, WETSoM (International Workshop on Emerging Trends in Software Metrics). It is an auto-selection since that the invitations were sent based on our individual decision, and we could not sample the subjects randomly.

3.5.3 Data Extraction Method

As the questionnaire was answered in a web system (OOSM metrics portal), it was possible to keep all the responses in a database. From the profile's part, information such as, environment (industrial/academic), position, affiliation and email were kept. The subject's expertise in OO development and software maintainability evaluation was also saved. Lastly, the metrics and categories to be used in OOSM assessment proposed by the subjects were saved.

It is important to highlight that we developed a module specific for applying the questionnaire integrated with the portal because we can compare the catalogs suggested by the subjects and the catalog generated by the tool (Module 02), considering the same categories previously

chosen by the subject. Three tools were tested for building and applying the questionnaire during the quasi-experiment design: eSurveyPro[1], SurveyMonkey[2], and eSurveyCreator[3]. However, we could not integrate any of them with the catalogs generator module. Consequently, a questionnaire application module was also developed in the portal.

Complementarily, we would like to check the experts' opinion about the catalog generated by our approach. Therefore, during the questionnaire application, the subjects have compared the catalog suggested by them with the catalog generated by the tool. After that, the two catalogs were shown, and, the metrics that differed were highlighted. The subjects answered a final question about the equivalence of the two catalogs. This question had two values (evaluation attributes):

1. Yes
2. No

For data assessing, a statistic coverage evaluation was performed checking the CI, i.e., the probability of a metric proposed by the expert belong to the catalog generated through our approach (NIE; KAMBHAMPATI, 2004). We used the Wilcoxon Test for one sample to assess the CI. Additionally, we adopted the Hypotheses Test for a Proportion to assess the experts' opinion about the catalogs generated (WILCOX, 2004). The result of experts' opinion is shown in Appendix F. Therefore, with this statistical measurement, we could check the coverage of catalogs generated by the tool over the catalogs suggested by the experts, and how useful can be the proposed approach for researchers that do not have experience in metrics adoption.

[1] http://www.esurveyspro.com/
[2] https://pt.surveymonkey.com/
[3] https://www.esurveycreator.com/

4

Results Discussion

The results found with the execution of the Ph.D. research project are shown in this chapter. Section 4.1 presents the results of the Systematic Mapping Study (SMS) performed. Information such as digital libraries used in the process, authors who have written about the maintainability metrics, and journal and conferences interested in this kind of metrics are also depicted. Since the initial goal of our research was the assessment of both, aspect-oriented and object-oriented metrics, the Aspect-Oriented Software Maintainability (AOSM) metrics are described in Section 4.1.4. Nevertheless, it is important to remember that after the SMS execution, we decided to focus our research only on OOSM Metrics, the subject of Section 4.2. After that, on Section 4.3, the portal developed to support the research results is presented. Finally, the family of catalogs and its assessment are shown in Section 4.4.

4.1 Systematic Mapping Study (SMS) Results

This section discusses the results found in the systematic mapping study on software maintainability metrics. Information about the digital libraries used by this study is presented in Section 4.1.1. The authors of the selected primary studies are shown in Section 4.1.2. Section 4.1.3 presents information about journals and conferences queried during this study. At the end of this section, on Section 4.1.4, the questions previously raised by the SMS are answered. As the main contribution of this mapping was the compilation of a metrics suite to assess AOSM and OOSM, this section presents some results observed after the execution of the study. These observations help to figure out the actual perspective of the software maintainability metrics adopted in both scenarios.

4.1.1 Digital Libraries

Our systematic mapping selected 138 primary studies, and they are listed on Appendix C. It is relevant to show the digital libraries used in the SMS, and how they helped us in finding the primary studies. Table 4.1 identifies the digital libraries and their websites. There are many

others digital libraries related to Computer Science, however, these were chosen considering their importance in the area, and the availability of papers for download.

Table 4.1 Digital Libraries Used in the SMS

Source	Websites	Search Period
ACM	http://portal.acm.org	06.01.2011 - 06.05.2011
IEEE	http://ieeeexplore.ieee.org	06.05.2011 - 07.10.2011
EI Compendex	http://www.engineeringvillage.com	07.10.2011 - 07.13.2011
Science Direct	www.sciencedirect.com	07.19.2011 - 07.22.2011

During the systematic mapping some papers have been excluded according to the mapping protocol described in Appendix B. For each digital library, Table 4.2 shows the number of papers initially returned (second column), and the result of the first selection (third column), which consisted of evaluating the paper's title, abstract, and keywords. The fourth column exposes the number of papers selected after the second phase of analysis, which consisted of the full paper screening. The fifth column indicates the percentage of the final papers selected, considering the initial number of papers returned. The last column shows the publication year interval of the primary studies returned by the digital libraries engines. The last row represents the number of papers after removing all duplicates. For instance, if the same paper was found on both, ACM and IEEE, digital libraries, it was accounted and considered just once. The other occurrence of primary study was marked as a duplicate and removed from the final study selection on the mapping.

Table 4.2 Evolution Primary Studies Selection

Source	Quantity	#1 Selec.	#2 Selec.	% Included	Interval
ACM	2386	119	41	1.7%	1989 - 2011
IEEE	2180	225	83	3.8%	1982 - 2011
EI Compendex	173	50	25	14.4%	1969 - 2011
Science Direct	881	26	16	1.8%	1985 - 2011
TOTAL	5620	423	165	2.93%	1969 - 2011
TOTAL*	**5175**	**351**	**138**	**2.66%**	**1969 - 2011**

*Number of papers after removing all duplicates.

By analyzing Table 4.2, it is possible to see that EI Compendex had the best performance, with 14.4% of primary studies included. Despite having the lowest number of studies returned, EI Compendex provided a more accurate list of studies when compared to the other search engines, adding less noise, consequently, showing higher precision. The ACM digital library had the worst performance, demanding more work in the studies' selection.

Table 4.2 also shows that, among the 5175 papers returned by the digital libraries, only 138 were selected. We observed that the queries presented a considerable level of noise, since

only 2.7% of the initially studies returned were actually relevant to the mapping. Many factors can contribute to increase the noise, e.g., we might have not used the ideal set of keywords in the search string composition. Kitchenham and Charters have already discussed the problem in using automated search engines like the ones we employed in this mapping (KITCHENHAM; CHARTERS, 2007). Consequently, other options of search string can be used to test this hypothesis.

One more time, EI Compendex appears prominently. It returned the oldest primary study. It is important to notice that after the exclusion process, the oldest study selected dated of 1992, and the most recent one was published on June 2011. This range shows that maintainability metrics have been studied and proposed for at least 19 years. Even though software maintainability metrics have been a research topic through all these 19 years, there is no catalog for them.

4.1.2 Studies' Authors

This section presents an analysis over the primary studies' authors. Table 4.3 shows the authors that published two or more studies about metrics. It is important to clarify that this counting considered just the primary studies selected (138 studies) based on the inclusion/exclusion study's criteria previously shown.

Table 4.3 Primary Studies' Authors

#Papers	Authors
8	Lionel C. Briand.
5	Alessandro Garcia.
4	Avadhesh Kumar, Marcela Genero, Mario Piattini, P.S. Grover, Rajesh Kumar.
3	Claudio Sant'Anna, Denys Poshyvanyk, Doo Hwan Bae, Ewan Tempero, Heung Seok, Chae, Jehad Al Dallal, John W. Daly, Jurgen K. Wust, P. Nesi, R. Nithi, S. Counsell.
2	Alfred Aho, Andrea De Lucia, Bandar Alshammari, Baowen Xu, Colin Fidge, Diane Corney, Doo Hwan Bae, Erik Arisholm, Esperanza Manso, F. Fioravanti, Fernando Castor, Gail C. Murphy, Hany H. Ammar, Jianjun Zhao, Jonas Lundberg, Letha Etzkom, Marc Eaddy, Mikael Lindvall, Nelio Cacho, R. Harrison, Santonu Sarkar, Thais Batista, Victor R. Basili, Welf Lowe, Yuming Zhou.
1	All the others.

As depicted by Table 4.3, Lionel C. Briand is the researcher who published more papers on this mapping study's topic. It is important to highlight that most of the studies found are only using metrics to perform quantitative and qualitative assessments. This means that just a minority of the primary studies actually proposes new metrics.

4.1.3 Journal and Conferences Involved

This section shows the journals and conferences where the primary studies were published. Table 4.4 presents the results. The first column represents how many studies were published in the journal, and its name is indicated in the second column.

The IEEE **TSE!** (**TSE!**) journal is the vehicle that has published more primary studies. This means that, after all exclusion processes and analysis of the remaining studies, this journal provided the highest number of relevant papers. This suggests that, if a researcher is interested in searching for papers that address this topic, TSE can be a good place to start from.

Table 4.4 Number of Selected Primary Studies in Each Journal

# of Selected Primary Studies	Journal
27	TSE
10	INFSOF
4	JSS
3	Information Sciences
3	ACM-SIGSOFT Software Engineering Notes (Newsletter)
2	Sciences of Computer Programming, SMR
1	ESE, IJSEKE, TOSEM, JSA, SQJ

Regarding conferences, METRICS has the highest number of primary studies. This result is expected since this conference is completely related to the subject matter of this study. The result is depicted in Table 4.5.

Table 4.5 Number of Selected Primary Studies in Each Conference Proceedings

# of Selected Primary Studies	Conference
8	METRICS
7	CSMR, ICSM
4	APSEC, ASWEC
2	AOSD, ESEM, STEP, WETSoM
1	ACE, ACoM, ACSC, ASE, ASEW, CASCON, CSSE, CW, CYCSYN, ECSA, EMS, FSE, HIS, ICC, ICCRD, ICECCS, ICETEC, ICIW, ICPC, INFOS, INMIC, ISESE, ISSTA, MOMPES, PerCom, KAMW, QSIC, REV, SAC, SBES, SEAA, SEM, SEW, SIGPLAN, SNPD, SOQUA, TASE, TOOLS, WCIT, WoSQ, WSCS, ACM-SE, ACM-CSUR, ACM-SIGAda.

CSMR and ICSM come close to METRICS in number of published primary studies, which it is also expected, since these two conferences are directly related to software maintainability. Another important insight is that most primary studies were published in conferences. Among the 138 selected papers, 63 (45.7%) were published in journals and 75 (54.3%) were

published on conference proceedings. We believe that this result could be explained by the larger number of conferences, when compared to the number of Software Engineering journals, and the time of evaluation of a paper submission, where this process takes longer for journals than for conference proceedings.

Moreover, it is well-known that the time between the initial submission and the publication of a paper in a journal is longer than in a conference. Furthermore, ideas are often first presented in workshops and conferences, and then they are evolved to a journal publication.

4.1.4 SMS Answer

This section presents the answer to the Research Question (RQ) raised by the systematic mapping study. To make the presentation clearer, the selected primary studies were assigned unique identifiers, for example, SM01 means Systematic Mapping Study #01. All the primary studies selected are listed in *Appendix C*.

RQ: *What metrics were adopted to assess software maintainability in OOSD?*: 568 metrics were adopted to assess software maintainability. Table D.1 lists them. The first column depicts the metrics' names and the second column their descriptions. The studies that mentioned the metrics are presented on the third column.

By observing Table D.1 in *Appendix D*, it is possible to note that the majority of OO metrics, 391 out of the 568 (68.83%) shown, appeared just in one primary study. With this scenario we infer that other researchers have not used these metrics. The lack of a catalog with a list of already proposed metrics could make this situation even worst. Indeed, there are a lot of cases where a certain metric was proposed, however, it was never used/validated by other researchers.

The investigation of this phenomenon is not addressed by the current work, and shall be undertaken by future work. Those cases can indicate some inefficiency of these metrics regarding the assessment of maintainability. This could happen for a number of reasons, e.g., (i) the number of available metrics is too high and it is hard to chose which one to use, (ii) most of the metrics were not validated yet, (iii) there are no tools that can collect them, among others.

4.2 Software Maintainability Metrics

The maintainability metrics identified by this mapping study are candidates to compose a metrics suite for helping researchers in software maintainability assessment. With a metrics suite in mind, researchers can more easily choose the ones that better fit their intent when designing empirical studies (either quantitative or qualitative). In addition, software practitioners can simplify the decision making process about activities related to software maintenance. There are various metrics' features that can be a relevant indicator during this selection process, specifically when looking into the whole set of 568 metrics. The complete list of all metrics can be found in

Table 4.6 Topic Related to Maintainability Metrics

Topic	# Occurrence	Percentage
Cohesion	173	30.4%
Coupling	154	27.1%
Size	149	26.2%
Complexity	30	5.3%
Inheritance	55	9.7%
Software Architecture	7	1.3%
TOTAL	568	100%

Appendix D.

All these metrics were cataloged. Some information about them was saved in a database: name, description, authors who proposed and who employed the metrics, conference proceedings and journals that published papers that describe the metrics, how many times the metrics were used/adopted by other authors, and so on. Those information were shown previously in Section 4.1.

We have also identified a number software maintainability-topics associated with the metrics: Software Architecture Constraints, Inheritance, Cohesion, Coupling, Complexity, and Size. This information about the topics related to the metrics was extracted from the primary studies selected according metrics' descriptions presented by their authors. Thus, if a metric description contained information about cohesion in the primary study selected, we considered that this metric is directly related to the cohesion topic. This occurred for all other topics. For instance, there is a metric called **AIM!** (**AIM!**) that counts the average of local and overridden/inherited methods in a system. So, the topic associated with this metric is inheritance.

Table 4.6 shows the results. The first column presents the topics related to maintainability described by the primary studies. The number of maintainability metrics whose descriptions claimed to investigate the topic or are related to the topics is shown in the second column. Finally, the third column shows the percentage of the metrics in the second column considering the total of 568 maintainability metrics found.

The results show that cohesion, coupling, and size are the most commonly investigated topics addressing software maintainability in the literature. Almost 84% of the papers mentioned these topics. Moreover, complexity, inheritance, and software architecture constraints also appear as important topics directly related to software maintainability among the primary studies.

4.2.1 Metrics' Tools

Based on the primary studies assessment, it was possible to list some tools that collect OOSM metrics automatically. It is important to clarify that we listed only the tools that were mentioned in the 138 primary studies selected in the mapping. Furthermore, some of the tools are not available for downloading or using. However, Table 4.7 shows all the 21 tools found in

this study.

Table 4.7 Tools that Collect Metrics Automatically

Tool's Name	Tool's Website
AdaSTAT	http://www.prnewswire.com/news-releases/adastattm-an-ada-static-analysis-tool-72887862.html
AOP Metrics	aopmetrics.tigris.org/metrics.html
ArchE Meter	https://sites.google.com/site/julianajags/
CoMETA-Lua	Not available
Enterprise Architect	http://www.sparxsystems.de/
FLAT tool	http://www.cs.mcgill.ca/ martin/cm/
Fraunhofer IESE	http://www.fuelairspark.com/Products/FS-'Programmers'-0.aspx
ICR2M	Not available
Metrics Eclipse Plug-in	http://metrics.sourceforge.net/
Ooram case tool	http://pettergraff.blogspot.com.br/2006/02/ooram.html
QMOOD++	http://www.drdobbs.com/web-development/automated-metrics-and-object-oriented-de/184410338
SD Tool e Sara Tool	Not available
Sourcenav	http://sourcenav.sourceforge.net
Together	http://techpubs.borland.com/together/
TOOMS	http://www.researchgate.net/publication/222352538_Metric_framework_for_object-oriented_real-time_systems_specification_languages
Visual Studio Metrication	http://msdn.microsoft.com/en-us/library/bb385910.aspx
M-System	Not available
LOGISCOPE	Not available
TAC++	Not available
PC-Memc for C++	Not available
Corba Components	http://sourceforge.net/projects/ccmtools/files/ccmtools/

4.2.2 Metrics Naming Inconsistencies

In the context of software maintainability metrics, inconsistencies in metrics' naming are another important issue to present and discuss. Thus, this section shows the two types of inconsistencies found in this study: (i) metrics with different names but essentially representing the same metric, such as **DIT!** and **DIH!** , which count the maximum length from a node to the root of the inheritance tree, and (ii) metrics with the same names and different meanings such as DC, which can be Degree of Cohesion or Descendants Counting.

Table 4.8 depicts the cases where different metrics were proposed, but they are the same indicators in the software maintainability assessment. The first column is the metric that can represent the group of the metrics with the same meaning. It is important to clarify that the representative metrics of the group were chosen considering how many times they appeared in the primary studies. Then, the names that appear on the first column are the ones mentioned by the highest number of primary studies selected in the group of metrics names with the same meaning. The second and third columns indicate the metric's name, and its description, respectively. Finally, the last column indicates which primary studies contain each metric. We found 32 cases involving 214 metrics.

Table 4.8: Maintainability Metrics - Different Names and Same Meanings

Representative	Metric	Description	Primary Studies that contain the metric
CBO	CBMU	Coupling between Model Units	SM350
	CMC	Coupling on Method Call	SM47, SM49, SM53, SM161, SM177, SM187, SM325
	CBM	Coupling Between Modules	SM47, SM49, SM92, SM177
	CIM	Coupling on Intercepted Modules	SM53, SM140, SM161, SM177, SM209
	CBO_IUB	CBO Is Used By	SM123
	CBO_NA	CBO No Ancestors	SM123
	CBO_U	CBO Using	SM123
	CBO(d)	Coupling Between Objects	SM66
	CBO(f)	Coupling Between Objects	SM66
	CBOIUB	Coupling Between Objects is Used By	SM73
	CBONA	Coupling Between Objects No Ancestors	SM73
	CBO	Coupling between Objects Classes	SM4, SM8, SM13, SM29, SM38, SM42, SM47, SM48, SM54, SM70, SM73, SM75, SM76, SM87, SM92, SM111, SM119, SM129, SM132, SM146, SM149, SM151, SM164, SM166, SM189, SM194, SM200, SM205, SM209, SM210, SM211, SM217, SM222, SM266, SM301, SM313, SM319, SM329, SM341, SM343, SM346, SM347, SM156
	CBOU	Coupling Between Objects Using	SM73
	CCBC	Conceptual Coupling Between Components	SM343
	CCBO	Conceptual Coupling between Object Classes	SM346
	CCOF	Component Coupling Factor	SM11
	COUPLING	COUPLING	SM212
	C	Coupling	SM37, SM38, SM275, SM323
	CC	Class Coupling	SM7, SM23, SM47, SM54, SM69, SM86, SM138, SM181, SM198, SM199, SM203, SM204, SM208, SM269, SM288, SM335
	CBC	Coupling Between Components	SM112, SM144, SM288
	CBMC	Coupling Between Module Classes	SM92, SM165, SM176
	CBO'	Coupling between Objects Classes	SM151, SM341

Table 4.8: Maintainability Metrics - Different Names and Same Meanings

Representative	Metric	Description	Primary Studies that contain the metric
	CLC	Class Level Coupling	SM38, SM132, SM203
	OLC	Object Level Coupling	SM38, SM132
EC	EC_CD	Export Coupling Dynamic Class	SM129, SM130
	EC_CC	Export Coupling Distinct Class	SM129, SM130
	EC_CM	Export Coupling Distinct Method	SM129, SM130
	EC_OD	Export Coupling Dynamic Message	SM129, SM130
	AEC	Exporting Coupling of a module	SM193
	EC_OM	Export Coupling Distinct Method	SM129, SM130
	EOC	Export Object Coupling	SM132
	EC_OC	Export Coupling Distinct Class	SM130
	IOC	Import Object Coupling	SM132
IC	AIC	Import Coupling of a module	SM193
	IC	Import Coupling	SM193, SM211
	IC_CC	Import Coupling Distinct Class	SM129, SM130
	IC_OC	Import Coupling Distinct Class	SM129, SM130
	IC_OD	Import Coupling Dynamic Message	SM129, SM130
	IC_OM	Import Coupling Distinct Method	SM129, SM130
	IC_CD	Import Coupling Dynamic Message	SM129
	IC_CM	Import Coupling Distinct Method	SM129
HC	H	Horizontal coupling	SM224
	HC	Horizontal coupling	SM23
RFC	RFC	Response For a Class	SM4, SM13, SM38, SM39, SM47, SM48, SM54, SM69, SM70, SM73, SM75, SM76, SM87, SM111, SM123, SM129, SM132, SM146, SM156, SM164, SM189, SM194, SM200, SM205, SM209, SM210, SM217, SM222, SM266, SM313, SM319, SM325, SM329, SM343, SM346, SM347
	RFC1	Response For a Class	SM151
DAC	PDAC	Package Data Abstraction Coupling	SM31

Table 4.8: Maintainability Metrics - Different Names and Same Meanings

Representative	Metric	Description	Primary Studies that contain the metric
	DAC	Data Abstraction Coupling	SM4, SM13, SM31, SM35, SM54, SM76, SM129, SM146, SM151, SM209, SM319, SM341, SM343, SM346
	DAC'	Data Abstraction Coupling	SM151, SM341
	CTA	Coupling Through Abstract Data Types	SM54, SM325
COF	CF	Coupling Factor	SM70, SM76, SM146, SM211
	COF	Coupling Factor	SM11, SM38, SM198, SM211, SM343, SM346
OMMIC	OMMIC	Method-Method interaction between classes	SM73, SM129, SM151, SM153, SM341
	MM	Method-Method Interactions	SM38, SM132
	MMI	Method-Method Interaction	SM181
	AMMIC	Method-Method Interaction between classes	SM73, SM151, SM341, SM129, SM153
DOS	DOS	Degree of scattering	SM169, SM126
	ADOS	Average the Degree of Scattering	SM169
DOF	ADOF	Averaging the Degree of Focus	SM169
	DOF	Degree of focus	SM169
COH	COH	Cohesion	SM3, SM68, SM181, SM219, SM335
	CH	Cohesion	SM193
	CHC	Cohesion of a Component	SM112
	DCD	Degree of Cohesion	SM181
	OL2	Cohesion of Class	SM165
	C3	Cohesion Metric	SM166, SM219, SM345
CACI	CAC	Class Attribut Complexity	SM203, SM226
	CACL	Class Attribute Complexity Local	SM138, SM204
	CACI	Class Attribute Complexity Inherited	SM138, SM204, SM208
CPC	PCT	Path Complexity	SM203
	CCPC	Class Coupling Path Complexity	SM203
	PEC	Path External Complexity	SM203
	PIC	Path Internal Complexity	SM203
	CPC	Class path complexity	SM203, SM112
CCN	VG	Cyclomatic Complexity	SM7, SM132
	MCCABECC	Cyclomatic Complexity	SM290
	PSIC	Provided Service Interface Complexity	SM203
	CyC	Cyclomatic Complexity	SM7
	CCN	Cyclomatic Complexity Number	SM140, SM146, SM217

Table 4.8: Maintainability Metrics - Different Names and Same Meanings

Representative	Metric	Description	Primary Studies that contain the metric
ICC	ICC	Internal Class Complexity	SM203
	CITC	Class internal task complexity	SM156
NOC	DCAE	Descendents from CA-Interactions	SM38
	NOC	Number of Children of a Class	SM4, SM7, SM13, SM31, SM35, SM39, SM42, SM48, SM49, SM53, SM54, SM66, SM69, SM70, SM75, SM76, SM79, SM87, SM132, SM156, SM164, SM187, SM189, SM194, SM200, SM201, SM204, SM205, SM209, SM210, SM217, SM222, SM224, SM266, SM290, SM313, SM319, SM325, SM347
	CO	Classes that Override something	SM55, SM69, SM176, SM277
HAGG	MAXHAGG	Maximum HAgg	SM136
	MOA	Measure of Aggregation	SM15
	HAGG	Height of class within aggregation	SM96, SM337
NOP	ANA	Average Number of Ancestors	SM15
	ANC	Ancestor	SM211
	NAC	Number of Ancestors	SM226, SM325, SM210
	NOA	Number of Ancestors	SM146, SM209, SM290
	CI	Classes Inherited	SM55, SM138, SM204, SM208
	NOP	Number of Parents	SM7, SM15, SM224, SM290
NSA	NOSA	Number of Static Attributes	SM290
	NSA	Number of Static Attributes	SM7
NLM	NAM	Number of Methods Locally	SM138, SM204
	NLM	Number of Local Methods	SM54, SM325
AC	NSUP	Number of Superclasses	SM204
	NUMANC	Number of Ancestors	SM224
	AC	Ancestors Count	SM38, SM198, SM203, SM211, SM288
	PC	Parents Count	SM198
CAA	NAD	Number of Advices	SM82
	CAA	Counting Aspect Advices	SM323, SM343
NP	NOPM	Number of Public Methods	SM187
	CIS	Class Interface Size	SM15, SM301
	MPUB	Method Public	SM234
	NP	Number of Public Methods	SM3, SM337
	NMPUB	Number of Public Methods	SM224
	NUMPUBOP	Number of Public Operations	SM224
	PM	Number of Public Methods	SM70, SM288
	NMO	Number of Methods Overridden	SM55, SM70, SM76

Table 4.8: Maintainability Metrics - Different Names and Same Meanings

Representative	Metric	Description	Primary Studies that contain the metric
	AIM	All Inherited Methods	SM75
	CMI	Coupling Method Inherited	SM84
	MANC	Methods inherited from Ancestor	SM211
	NOOM	Number of Overridden Methods	SM194
	OM	Overridden methods	SM288
	POM	Percentage of Overrided Methods	SM75
	NOI	Number of Inherited Methods	SM75, SM290
	MI	Methods that are Inherited	SM48, SM55, SM68, SM198, SM234
	MIF	Method Inheritance Factor	SM48, SM70, SM76, SM194, SM198
	NMI	Number of Methods Inherited	SM55, SM70, SM129, SM138, SM204
DIT	DIT	Depth of Inheritance Tree	SM123, SM4, SM7, SM13, SM31, SM48, SM49, SM54, SM69, SM70, SM75, SM76, SM87, SM96, SM132, SM144, SM164, SM177, SM187, SM189, SM194, SM200, SM201, SM204, SM205, SM209, SM210, SM217, SM222, SM22, SM266, SM290, SM313, SM319, SM325, SM347, SM39, SM42, SM66
	MAXDIT	Maximum DIT	SM136, SM337
	WIG	Weighted Interaction Graph	SM199
	ADI	Average Depth of Inheritance	SM288
	AID	Average Inheritance Depth of a Class	SM151
	WGT	Weighted Graph Tree	SM199
	DIH	Depth of Inheritance	SM69
	DOIH	Degree of Inheritance	SM7
	DT	Depth of Tree	SM86
	NL	Nesting Level	SM226
	DI	Depth of Inheritance	SM7, SM288
NA	NAI	Number of Attributes	SM204, SM224
	CAS	Class Attribute Size	SM203
	LA	Local Attributes	SM69
	NUMATTR	Number of Attributes	SM224
	TA	Total Attributes	SM69
	NAL	Number of Attributes Locally	SM138, SM204
	A	Number of Attributes	SM38, SM234, SM275, SM290

Table 4.8: Maintainability Metrics - Different Names and Same Meanings

Representative	Metric	Description	Primary Studies that contain the metric
	NA	Number of Attributes	SM10, SM96, SM136, SM144, SM203, SM204, SM205, SM208
NC	NC	Number of Classes	SM96, SM203, SM82, SM226, SM230, SM288, SM136
	BC	Base Classes	SM288
	CS	Class Size	SM203
	CSA	Classes	SM35
	RCS	Real Class Size	SM203
	NCL	Number of Classes	SM203, SM204, SM208
NOC	SNOC	Size Of Number Children	SM35
	DC	Descendants Count	SM48, SM198
	NOOC	Number of Object Children	SM7
	NSUB	Number of Subclasses	SM204
	NUMDESC	Number of Descents	SM224
	NOC*	Number Of Children in sub-tree	SM123
	TC	Total Children	SM48, SM69, SM327
	NOC	Number of Children of a Class	SM4, SM7, SM13, SM31, SM35, SM39, SM42, SM48, SM49, SM53, SM54, SM66, SM69, SM70, SM75, SM76, SM79, SM87, SM132, SM156, SM164, SM187, SM189, SM194, SM200, SM201, SM204, SM205, SM209, SM210, SM217, SM222, SM224, SM266, SM290, SM313, SM319, SM325, SM347
	DCAE	Descendents from CA-Interactions	SM38
NM	NO	Number of Operations	SM35, SM144
	CSO	Class Operations	SM35
	M	Method	SM38
	MA	Methods Available	SM198
	LO	Local Operations	SM69
	MN	Methods (New)	SM198
	N1	Total number of operators	SM269
	NOO	Number of Operations	SM194
	NUMOPS	Number of Operations	SM224
	TO	Total Operations	SM69
	NMC	Number of Methods	SM55
	TNM	Total (System) Number of Methods	SM208, SM138
	NM	Number of Methods	SM70, SM82, SM96, SM136, SM204, SM224, SM230, SM288

Table 4.8: Maintainability Metrics - Different Names and Same Meanings

Representative	Metric	Description	Primary Studies that contain the metric
	NOM	Number of Methods	SM15, SM35, SM54, SM76, SM111, SM146, SM290, SM319
WMC	CAI	Classified Attributes Interaction Weight	SM53, SM84
	CAIW	Classified Attributes Interaction Weight	SM84, SM282
	WOM	Weighted Operations in Module	SM47, SM49, SM177, SM187
	WNCO	Weighted Number of Collections Operation	SM85
	CAMC	Cohesion Among Methods in a Class	SM68, SM181, SM211, SM307, SM335
	CMW	Classified Methods Weight	SM84, SM282
	WMPC1	Weighted Methods Per Class	SM7
	WMC	Weighted Methods Per Class	SM7, SM8, SM31, SM35, SM39, SM48, SM54, SM69, SM70, SM75, SM87, SM111, SM123, SM132, SM138, SM146, SM156, SM164, SM189, SM194, SM200, SM201, SM204, SM210, SM217, SM222, SM266, SM290, SM313, SM319, SM325, SM337, SM347
	WAC	Weighted Attributes per Class	SM313, SM319
	WMA	Weight of Method by Aspect	SM205
	WOC	Weighted Operations per Component	SM209
	CMICL	Class Method Interface Complexity/size Local	SM208
	OAC	Operation Argument Complexity	SM224
	CMICI	Class Method Interface Complexity Inherited	SM138, SM208
LCOM	LCOM	Lack of Cohesion in Methods	SM1, SM3, SM7, SM13, SM35, SM37, SM39, SM48, SM54, SM69, SM70, SM75, SM76, SM87, SM105, SM111, SM123, SM128, SM146, SM164, SM165, SM189, SM194, SM200, SM210, SM217, SM222, SM226, SM66, SM290, SM307, SM313, SM319, SM325, SM330, SM331, SM335, SM345, SM347
	LCOMA	Lack of Cohesion in Methods	SM156
	LCOMB	Lack of Cohesion in Methods	SM156

Table 4.8: Maintainability Metrics - Different Names and Same Meanings

Representative	Metric	Description	Primary Studies that contain the metric
	TLCOM	Transitive LCOM	SM335
	LCO	Lack of Cohesion in Operations	SM49, SM87, SM177, SM187, SM204
	LCOM4	Lack of Cohesion in Methods 4	SM3, SM37, SM176, SM181, SM2667, SM277
	LCOM5	Lack of Cohesion in Methods 5	SM3, SM37, SM176, SM181, SM219, SM267, SM277
	LCOM3	Lack of Cohesion in Methods 3	SM3, SM37, SM68, SM176, SM181, ,SM219, SM267, SM277
	LCOM2	Lack of Cohesion in Methods 2	SM3, SM37, SM68, SM176, SM182, SM219, SM224, SM267, SM277
	LCOM1	Lack of Cohesion in Methods 1	SM3, SM7, SM68, SM176, SM181, SM219, SM224, SM264, SM267, SM277
	LCC	Loose Class Cohesion	SM3, SM37, SM68, SM76, SM181, SM209, SM219, SM236, SM330, SM331
LOC	LOCC	Lines of Class Code	SM126, SM187
	LOCS	Lines of Class Code	SM202
	SLOC	Source Lines of Code	SM169, SM189, SM288
	LOC	Lines of Code	SM7, SM10, SM23, SM47, SM49, SM74, SM111, SM138, SM140, SM144, SM146, SM156, SM176, SM195, SM203, SM208, SM109, SM217, SM219, SM236, SM269, SM275, SM293, SM323, SM327
	NLOC	Number of Lines of Code	SM92
	NOSLOC	Number of Source Lines of Code	SM288
	TLOC	Total Number of Lines of Code	SM7, SM138, SM208, SM217, SM290
	KLOC K	Lines of Code	SM166

As mentioned before, the second case of metrics' naming inconsistency occurred when metrics with the same names have different meanings. For this scenario, eight cases were found involving 17 metrics. Table 4.9 depicts these cases. The first column represents the metrics names. The metrics descriptions are in the second column and the last column indicates in which primary study selected the metric appeared.

Both cases of metrics' naming inconsistencies have happened because there is no convention for naming software metrics. The approach to gather and to list metrics, and after that, to

Table 4.9 Maintainability Metrics - Same Names and Different Meanings

Metric	# Description	Primary Study that contains the metric
DC	Degree of Cohesion Descendants Count	SM335 SM48, SM198
MN	Methods (New) Methods with No implementations replaced	SM158 SM55
NAS	Number of Class Associations Number of Aspects	SM119, SM132 SM82
NC	Number of Clauses in the class Number of Cycle Number of Classes	SM10 SM293 SM96, SM203, SM82, SM226, SM230, SM288, SM136
NLOC	Net LOC Number of Lines of Code	SM296 SM92
NP	Number of Public Methods Number of class Paths	SM3, SM337 SM203
RC	Relative Cost Relational Cohesion	SM291 SM199, SM330
PIM	Permitted Interaction Metric Polymorphically Invoked Methods	SM4 SM38, SM234, SM341, SM343

generate catalogs based on the research context presented here, can help to address this issue. Sometimes, researchers have been proposing distinct metrics around the world because they do not know that the metric already exists.

Therefore, we performed a metrics' naming consolidation (described in Chapter 3) to expose the situation where those inconsistencies happened. We believe this can ease the task of choosing which metrics are more suitable in a given context. Additionally, the inconsistencies tend to decrease or disappear, since information about maintainability metrics can be found in one unique place, together and organized. This can be a first step towards standardizing maintainability metrics naming.

4.2.3 Metrics' Categories

The large number of metrics shown in Section 4.2 poses many challenges to researchers intending to conduct OOSM studies and software practitioners whose goal is to assess maintainability. Additionally, the metrics descriptions were scattered throughout a number of different papers. They had different levels of evaluation, and their names were inconsistent. This issue was addressed in Section 4.2.2.

As mentioned before, we have categorized the metrics in terms of quality attributes that they aim to measure. Complementarily we propose a number of additional dimensions for OOSM metrics categorization, based on the contexts of the studies in which they were employed.

The complete categorization is discussed in this section.

Table 4.10 Examples of Metrics' Descriptions/Adoption Scenario's Descriptions that Support the Categories Definitions

Category	Description to Support the Category Definition
Academic	*The systems used for this study were developed by students participating in an upper division undergraduate/graduate level course offered by the Department of Computer Science at the University of Maryland.*
Industrial	*We performed a measurement and evaluation of various Java standard libraries like J2SE, J2EE...*
Open Source	*...using four open source software systems and 10 cohesion metrics.*
Proprietary	*They are very successful, proprietary, and popular commercial object-oriented systems that are extensively used in real-world software development.*
Coupling	*Efferent Coupling.*
Complexity	*Class Attribute Complexity.*
Size	*Average Method Size.*
Cohesion	*Cohesion Among Methods of Class.*
Inheritance	*Average Inheritance Depth of a Class.*
Architecture Constraint	*Architecture Design Modified.*
Tools Support	*The metric framework is integrated in a CASE tool named TOOMS.*
Changeability	*Strict adherence to the S-S-V-T structure is beneficial for key test design criteria such as changeability.*
Stability	*But other important dependencies are clearly not measured or accounted for, and may not be measurable from code alone, e.g., stability to common requirements.*
Testability	*The use of metrics for the estimation of testability is analyzed.*
Analyzability	*This model relates design properties such as analyzability.*
Evaluated	*This paper first analyzes the limitations of typical cohesion measures for classes in detail.*
Validated	*The proposed approach was validated against an open source software system, namely GanttProject version 1.10.2.*

The large number of OOSM metric that we identified is an obstacle to their use in practice. Thus, taking these metrics as a starting point, we attempt to categorize and summarize these metrics to ease the decision-making process about which metrics to adopt. It is important to clarify that the classification of metrics as OOSM metrics was based on the metrics' definition found in the primary studies selected in the systematic mapping study. Table 4.10 shows some examples of metrics's descriptions or definition of adoption scenarios that support the categorization proposed here. The first column represents the categories, and the second column represents the text extracted from the papers selected from the SMS that support the categories definition. We propose the following categorization for the metrics:

1. **Relevance:** This category is composed by two sub-categories: (i) Most Adopted, and (ii) Most Relevant metrics. For the first one, (i) we counted how many times a metric appeared (proposed or adopted) in the primary studies selected by the systematic mapping study. So, they were ranked as most adopted metrics, considering the number of times they were found in the selected studies. On the other hand, for the Most Relevant metrics sub-category, the primary studies selected in the systematic mapping were ranked based on the number of citations of each study. We considered the Google Scholar engine to count the number of citations. After that, we had the most relevant papers for the area from the 35 studies best ranked (rank's 1st quartile). From the most relevant studies, we extracted the metrics they contained, and classified the metrics according to the number of times they were used in these 35 best-ranked studies.

2. **Environment:** The Environment in which a metric was employed can be (i) Academic, and (ii) Industrial. The Academic metrics includes metrics that were used in academic contexts, such as, software engineering academic research or classes. On the other hand, the Industrial metrics were proposed by or adopted in industrial scenarios by either researchers or practitioners. It is important to clarify that a metric can be considered as both, industrial and academic, if it is adopted in both contexts.

3. **License Type:** This category can classify the metrics in two groups: (i) Open Source Metrics, that were used in open source software maintainability assessments, and (ii) Proprietary Metrics that were used in evaluations of proprietary software.

4. **Internal Attributes:** This classification is composed by the aforementioned five internal quality attributes related to software maintainability: (i) size, (ii) complexity, (iii) coupling, (iv) cohesion, and (v) software architecture constraints.

5. **Tool Support:** This category considers metrics that can be automatically collected by a tool.

6. **External Attributes:** The focus of our research is OOSM, nevertheless, other sub-characteristics and external attributes can be associated or measured by a metric. Consequently, the metrics were also classified considering the following external attributes that have a relationship with maintainability: analyzability, changeability, stability, testability, reliability, extendibility, reusability, readability, flexibility, traceability, scalability, usability, understandability, adaptability, verifiability, variability, instability, modifiability, fault proneness, efficiency, capability, availability, replaceability, predictability, comprehensibility, performance, applicability, accessibility, vulnerability, visibility. It is important to clarify that these external attributes were found in the studies' context described in the selected primary studies.

7. **Assessment:** This category is composed by the metrics that were assessed by any researcher. For this category, the metrics can be classified as (i) Evaluated Metrics, that were not validated, but evaluated in any way, such as compared or discussed by any researcher/practitioner; and (ii) Validated Metrics, that is, composed by the metrics that were validated by any researcher. It is important to highlight that we are considering all the 47 ways of metrics evaluations found by the work of Meneely (MENEELY; SMITH; WILLIAMS, 2012), such as Protocol validity, Notation Validity, Non-uniformity validity, theoretical validity, and so on.

We believe that with an appropriate categorization considering various domains of metrics adoption, a metrics suite can be more easily chosen by researchers and software practitioners. In addition, metrics can also be classified in more than one category because the categories are not mutually exclusive. Figure 4.1 depicts the categories and subcategories proposed. In addition,

we believe that in a future work, the feature model that represents the metrics' categorization can be improved and expanded.

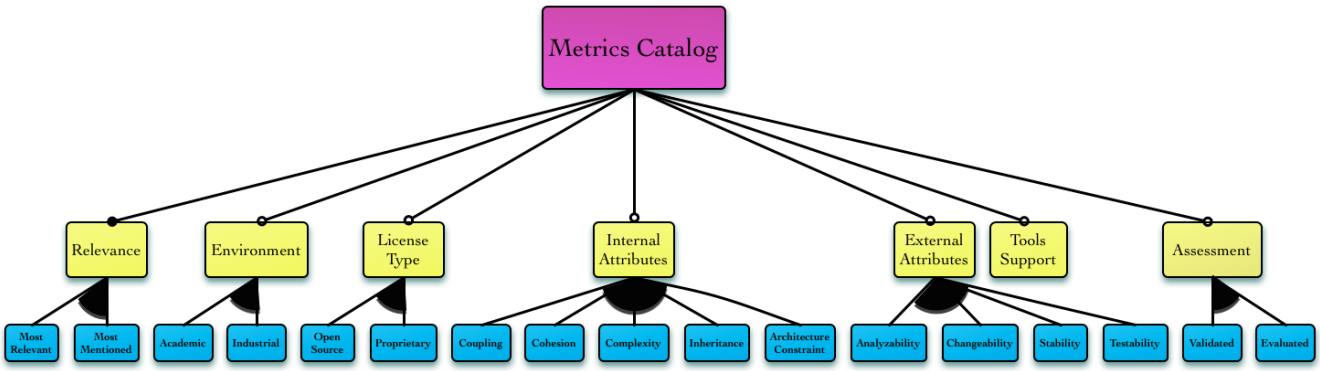

Figure 4.1 Illustration of OOSM Metrics Categorization.

4.3 OOSM Metrics Portal

We have built a metrics portal to support the dissemination of information about OOSM metrics (`http://julianasaraiva.info/oosmMetricsPortal`). The portal is composed by 3 modules, and each one has a specific role as described in Chapter 3 - Section 3.4: (i) General Information, (ii) Project Information Management, and (iii) Catalogs Generation Assessment.

Through the first module, the portal's user can access information such as, the research members contacts, the portal goals and descriptions, published papers that address OOSM metrics, authors that published about this type of metrics, and so on. Any user can access this module. Additionally, metrics' catalogs can be generated according to the categories chosen by the users. They can save the catalogs as PDF files or they can see them in the portal screens.

Nevertheless, due to space constraints, the catalogs generated in this module are not kept in the portal's database. Figures 4.2 and 4.3 depict the screen where users can select the appropriate categories and the catalog generated through the categorization, respectively. We believe that these facilities can support the dissemination of information about OOSM metrics and decrease the time spent by researchers looking for metrics that can be adopted in their studies.

Users associated with the 'Project Collaborator' profile can access the second module. It is important to highlight that adding new categories/subcategories is only possible to users registered as Project Collaborators. Other information, such as new metrics, new descriptions, or even, uploading papers that address OOSM evaluation, can also be added to the portal through this module.

It is possible to observe the portal's functionalities available for this profile. This

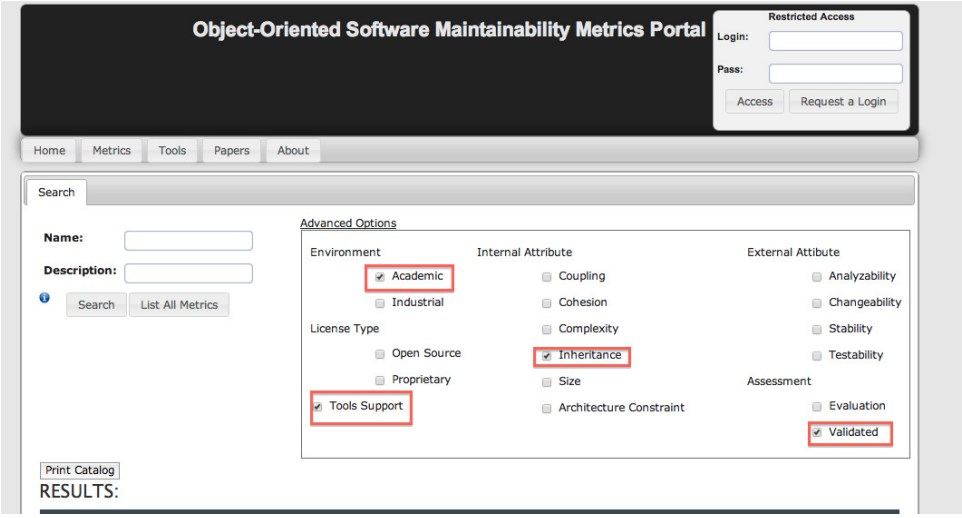

Figure 4.2 OOSM Metrics Portal - Catalogs Generator Module.

Print Catalog

RESULTS:

Metric Name	Description	Calculation Form	Categories Associated
AID	Average Inheritance Depth of a Class	The number of new methods in a class, not inherited, not overriding.	Environment – Academic – Internal Attributes – Inheritance – External Attributes – License Type – Open Source – Evaluation – Validated – Tools Support – Fault Proneness –
CDL	Class To Leaf Depth	CLD of a class is the maximum number of levels in the hierarchy that are below the class.	Environment – Academic – Internal Attributes – Inheritance – External Attributes – License Type – Open Source – Evaluation – Validated – Tools Support – Fault Proneness –
CIC	Class Inheritance-related Coupling	Class Inheritance-related Coupling	Environment – Academic – Internal Attributes – Inheritance – External Attributes – License Type – Proprietary – Evaluation – Validated – Tools Support – Reliability – Extendibility – Reusability –
CO	Classes that Override something	Classes that Override at least one method.	Environment – Academic – Industrial – Internal Attributes – Inheritance – External Attributes – License Type – Open Source – Proprietary – Evaluation – Validated – Tools Support – Reliability – Understandability –
DI	Depth of Inheritance	It indicates the number of class defi- nitions that are above the type in the inheritance tree starting from 0 and excludes interfaces.	Academic – Industrial – Environment – Proprietary – Open Source – License Type – Inheritance – Internal Attributes – Tools Support – Validated – Evaluation –
DIH	Depth of Inheritance	Depth of Inheritance	Academic – Industrial – Environment – Proprietary – Open Source – License Type – Inheritance – Internal Attributes – Tools Support – Validated – Evaluation –
DIT	Depth of Inheritance Tree	This metric is defined as the maximum length from the node to the root of the tree. This means that the DIT measures the number of ancestors of an aspect. The length of the longest path from the class to the root in the inheritance hierarchy.	Academic – Industrial – Environment – Proprietary – Open Source – License Type – Inheritance – Internal Attributes – Tools Support – Validated – Evaluation –
DOIH	Degree of Inheritance	Similar to DIT.	Academic – Industrial – Environment – Proprietary – Open Source – License Type – Inheritance – Internal Attributes – Tools Support – Validated – Evaluation –
MAXHAGG	Maximum HAgg	It is the maximum of the HAgg values obtained for each class of the class diagram. The HAgg value for a class within an aggregation hierarchy is the longest path from the class to the leaves.	Academic – Environment – Proprietary – License Type – Inheritance – Internal Attributes – Tools Support – Validated – Evaluation –
NOI	Number of Inherited Methods	Number of Inherited Methods	Academic – Industrial – Environment – Proprietary – Open Source – License Type – Inheritance – Internal Attributes – Tools Support – Validated – Evaluation –
NOP	Number of Parents	The number of classes that a given class directly inherits from.	Academic – Industrial – Environment – Proprietary – Open Source – License Type – Inheritance – Internal Attributes – Tools Support – Validated – Evaluation –
TOTAL	11 METRICS		

Figure 4.3 Example of Catalog Generated by the Portal.

module allows researchers around the world to feed the portal systematically. Consequently, the information about maintainability metrics can be more accurate and standardized. Therefore, this scenario can represent the first steps for standardization of maintainability metrics naming, searching, and adoption.

The last module is the Catalogs Generation Assessment Module. It is important to highlight that this module was developed just to support our approach's assessment, as discussed in Section 4.4. At the end of the questionnaire application, the resultant catalogs were saved for future comparison and statistical assessments. This module allowed us to collect opinions from experts from different countries about our proposal.

4.4 Results of the Approach Evaluation

This section presents and discusses the results of the quasi-experiment performed to assess the proposed approach for metrics' catalogs generation. Experts in software maintainability metrics answered the questionnaire. These experts were chosen because they have written papers that addressed software maintainability and/or software metrics. Their names/emails were obtained through the website of the major conferences in Software Engineering, Software Maintainability, and Software Metrics mentioned in Section 3.5.2.

An invitation was sent to them explaining the goal of the questionnaire application, and also, a login/password for accessing the portal's restricted area to answer the questionnaire. Seven authors did not answer the questionnaire claiming that although they have published papers about the software maintainability metrics, they did not consider themselves as experts in this topic. In those situations, they sent us an email justifying their position. On the other hand, other researchers, besides answering the questionnaire, gave us a feedback about our research. Researchers such as, Chris Francis Kemerer, who proposed the well-known CK metrics (CHIDAMBER; KEMERER, 1994), sent us emails evaluating positively our research.

Out of 130 invitations sent, just 47 (36.15%) people answered it. The questionnaire was available online during one month (12/06/13 to 01/06/14). We believe that the low number of answers may have been caused by the work holidays that occur during this period. Therefore, researchers and practitioners did not get the invitation email. This number can change in the future if new researchers answer the questionnaire. The following sections show and discuss the results of the evaluation of the proposed approach.

4.4.1 Respondents' Profile Assessment

The first part of the questionnaire was related to the respondent profile. Figures 4.4 and 4.5 show the profile results. The majority of the respondents are from academic environment. This result is expected since authors of conference proceedings papers, which in most of cases are academic researchers, composed the list of potentially respondents. The industrial respondents that answered the questionnaire were researchers/practitioners of companies that has some how employees dedicated to software research.

It is possible to observe through Table 4.11 and Figure 4.6 where the respondents came from. The majority of the respondents are from the United States of America. Another issue

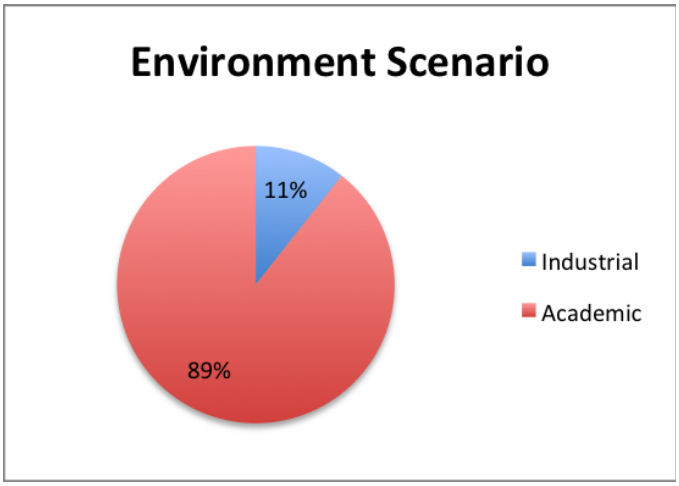

Figure 4.4 Results of the Questionnaire Respondents' Profile - Environment.

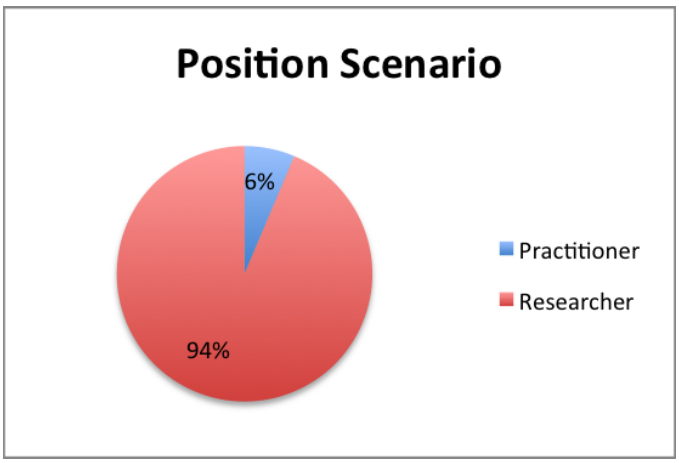

Figure 4.5 Results of the Questionnaire Respondents' Profile - Position.

to point out is the diversity of countries involved in the quasi-experiment. Overall, researchers and practitioners from 15 countries responded our survey. In addition, the majority of them are working in universities. This corroborates with the previous results about the work environment of the respondents. Once again, this result is expected since the source used to compose the respondents' database is directly related to academic research scenario: conference papers.

4.4.2 Respondents' Expertise

The second part of the questionnaire was about the respondents' expertise in software maintainability and software metrics. For measuring the expertise, three levels were adopted: Low Level (0-6 months), Medium Level (> 6 months - 2 years), and High Level (> 2 years). Surprisingly, all questionnaire participants informed that they have the 'Low Level' expertise in software maintainability. Maybe these results stemmed from the respondents not focusing their research on software maintainability as an isolated topic. On the other hand, Figure 4.7 depicts the respondent's expertise in software metrics. It is possible to observe that the majority of the

Table 4.11 Questionnaire's Respondents Affiliation

Affiliation Name	# Occurrences
University of Lille-1	4
Delft University of Technology	2
Eindhoven University of Technology	2
Ecole Polytechnique de Montreal	2
Federal University of Rio Grande do Norte	2
Newcastle University	2
Ozyegin University	2
Southeast University	2
Universidad Politecnica de Madrid	2
University of Pittsburgh	2
Wayne State University	2
Carleton University	1
Federal University of Pernambuco	1
Fraunhofer IESE	1
Gdansk University of Technology	1
Technische Universitat Munchen	1
Keele University	1
Lero Research Centre	1
Microsoft Corporation	1
North Carolina State University	1
Universita' degli Studi dell'Insubria	1
University of Cagliari	1
University of Illinois	1
University of Maryland	1
University of Nebraska	1
University of Passau	1
University of Quebec	1
University of Tartu	1
Federal University of Bahia	1
PUC-Rio	1
CEFET-MG	1
University of Auckland	1
Visa, Inc.	1
Yale University	1

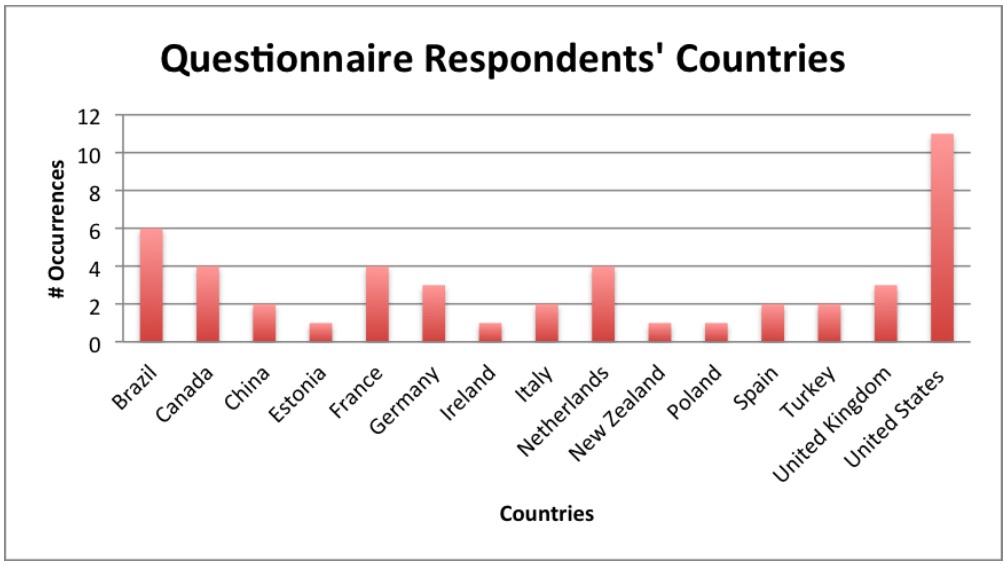

Figure 4.6 Results of the Questionnaire Respondents' Profile - Countries.

respondents consider themselves to be experts in software metrics.

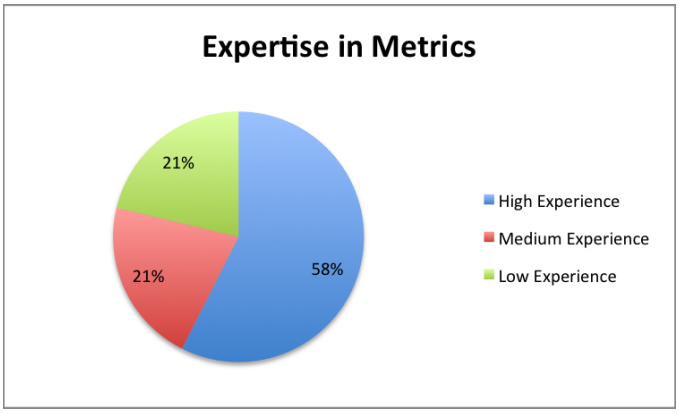

Figure 4.7 Results of the Questionnaire Respondents Expertise in Software Metrics.

4.4.3 Metrics Assessment

Each respondents presented a list of metrics that they consider to be appropriate to assess OOSM. They had to insert the metrics' name, and a brief description. Considering the 47 respondents, 204 metrics were suggested. An important observation is that out of the 204 metrics suggested by the experts, just 25 metrics did not exist in the portal's database. In other words, the set of metrics we identified covered more than 87% of all the metrics suggested by the respondents).

Table 4.12 shows these metrics. It is important to emphasize that the metrics' descriptions presented in Table 4.12 were suggested by the respondents. The first column contains the metrics' name, and the second column represents the metrics' descriptions inserted by the respondents. It is important to highlight that the metrics' descriptions shown here are the descriptions proposed by the respondents. Some of them are not in the portal's database because the collection of metrics information occurred until July 2011. Some of them may have been published after this date. In addition, some of these metrics may have appeared in studies that were excluded from our mapping study due to one of the previously presented exclusion criteria (Section 3.1.3).

Complementarily, there were also metrics that exist in the portal's database that were never suggested by the portal. Table 4.13 shows these cases. It occurred because the respondent fit some metrics in categories that do not match with our database information. For example, the **CBO!** (**CBO!**) metric is classified as a 'Coupling' category in the portal's database. However, if the respondent inserted the 'CBO' metric, and chose any category other than 'Coupling', the portal will never suggest 'CBO'. The occurrences of these cases exposed in Table 4.13 shows once again the non-standardization of metrics' naming and adoption, where some of researchers classify a metric as 'Coupling Metric', and others classify the same metric as 'Size Metric'.

4.4.4 Categories Assessment

The fourth part of the questionnaire was the categories suggested by the respondents. They inserted metrics they considered appropriate in software maintainability assessment, and

Table 4.12 Metrics Suggested by Experts that Do not Exist in Portal's Database

Metric's Name	Description
Atomic Changes	Atomic Changes
MBD	Maximum Block Depth
Branch_coverage	Testing coverage criteria that evaluates the percentage of branches covered by a regression testing suite
Line_coverage	Testing coverage criteria that evaluates the percentage of lines of code covered by a regression testing suite
IND	In Degree of a Class
Moose	Similar to C&K
FL	File length in LOC
Object Coupling	Chidamber
Defects	Number of defects found and fixed historically
Change Effort	Historic average effort to change code
Comment Density	%age of code devoted to comments
Testing Coverage	how much the code was tested
CBY	Commits by year
SOF	Size of Functions
BRL	Bug report / Location
Unit Size risk profile	A quadruple of percentage of code in a certain category (in this case based on method length)
Unit Complexity Risk Profile	A quadruple of percentage of code in a certain category (in this case based on McCabe)
Percentage of Duplicated Lines	To measure duplication
ATFD	Access to Foreign Data
NPF	Number of public fields
#owners	#people making changes
churn	amount of change in past
#changes	#previous changes to code
commentability	quality/correctness of comments in code
#previous bugs	#previous bugs

chose categories/subcategories that match with metrics previously inserted. Then, Table 4.14 shows how many times each category/subcategory was selected. The first column indicates the name of the category and the second column represents the number of occurrences, i.e., how many times that category was chosen.

Table 4.14 allows us to infer what specialists take into account when software maintainability metrics have to be chosen. It is important to highlight that all of the categories we proposed were selected by the questionnaire' participants. This result suggests that the categories/subcategories proposed in this work have the potential to help researchers in the decision-making process about which metrics can be adopted in their evaluations. Additionally, Table 4.15 depicts in detail the categories suggested by the respondents.

4.4.5 Assessment of the Catalogs' Coverage

This section presents a discussion about the catalogs' coverage of the metrics' catalogs generated using our approach over the catalogs suggested by the experts interviewed. As mentioned before, the experts had the opportunity to suggest metrics' catalogs to evaluate software maintainability when responding to the online questionnaire. Therefore, it was possible to check the coverage of catalogs generated using our approach when compared to the ones suggested by the respondents.

Table 4.13 Metrics Suggested by Experts with Inconsistencies in Categorization

Metric's Name	Description
AC	Architecture Compliance
ACHS	Average Cohesion of a System
ACPS	Average Coupling of a System
C	Coupling
C3	Cohesion Metric
CAS	Class Attribute Size
CBO	Coupling between Objects
CC	Clone Coverage
CDC	Concern Diffusion over Component
CHNL	Class Hierarchy nesting level
COH	Cohesion
COU	Coupling
CR	Comment ratio
DIT	Depth of Inheritance Tree
Fan-in	Number of in-calls
Fan-out	Number of out-calls
LCOM	Lack of Cohesion in Methods
LCOO	Lack of Cohesion
LOC	Lines of Code
NAC	Number of Active Contributors
ND	Nesting Dept
NOA	Number of attributes
NOC	Number of children (NOC)
NoD	Number of Dependencies
NOT	Number of types
NP	Number of parameters per method
RFC	Response For a Class
SLOC LOC	without lines contained in comments
TC	Testing Coverage
WMC	Weighted Methods per Class

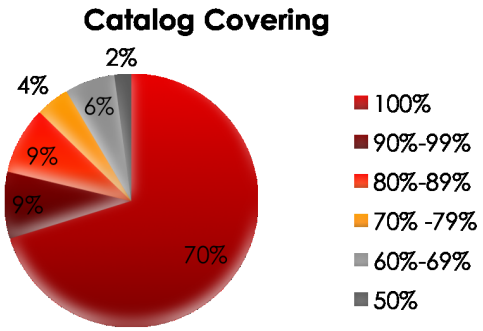

Figure 4.8 Catalogs' Covering.

The main idea is that the decision-making process about the metrics choice that can be adopted in OOSM evaluation can be facilitated using the proposed approach. It is important to highlight that our targets are researchers who do not have experience in software metrics adoption. However, the experts can also use the catalogs generated with our approach to improve their expertise in this scenario. Figure 4.8 shows the results of the catalogs' coverage obtained through the questionnaire answers. It is possible to observe that in the majority of the cases, the catalogs' coverage is at least of 90%. This suggests that when a non-expert researcher uses the catalogs generated by our approach proposed here, the OOSM metrics catalog will closely

Table 4.14 All Categories Suggested

Category	# Occurrences
Coupling	32
Academic	31
Tools Support	27
Complexity	26
Changeability	26
Industrial	26
Open Source	27
Size	27
Evaluated	22
Proprietary	18
Cohesion	18
Validated	15
Inheritance	16
Stability	16
Testability	14
Analyzability	14
Architecture Constraint	9

Table 4.15 Categories Suggested by Experts - In Detail

Category	**Subcategory**	**# Occurrences**
Environment	Academic	31
	Industrial	26
Internal Attribute	Coupling	32
	Complexity	26
	Size	27
	Cohesion	18
	Inheritance	16
	Architecture Constraint	9
External Attribute	Changeability	26
	Stability	16
	Testability	14
	Analyzability	14
License Type	Open Source	27
	Property	18
Assessment	Evaluated	22
	Validates	15
Tool Support	-	27

resemble one suggested by an expert.

A **CI!** (**CI!**) was defined for each catalog generated during the quasi-experiment (NIE; KAMBHAMPATI, 2004). The CI definition is detailed in Section 3.5.2. Table 4.16 shows the CIs of the 47 catalogs generated using our approach. The first columns indicate the number of the catalog, and the subsequent columns represent the CIs. In addition, Table 4.17 depicts the number of metrics suggested by the experts and suggested by the portal. The first column represents the questionnaire participant identification, the second and the third columns represent the number of metrics suggested by the experts and by the portal, respectively. The fourth column indicated the coverage percentage, and the last column informs the percentage of the total metrics (568) suggested by the portal. Based on the CIs of the various catalogs, it was possible to formulate the following hypotheses:

H0: CI < 0.9 of #(Yi). In other words, the catalog generated using our approach has less

Table 4.16 Catalogs' CIs.

#Catalog	CI	#Catalog	CI	#Catalog	CI	#Catalog	CI	#Catalog	CI
1	**1**	11	**1**	21	**0.5**	31	**1**	41	**1**
2	**1**	12	**0.99**	22	**0.95**	32	**1**	42	**0.85**
3	**1**	13	**0.92**	23	**1**	33	**1**	43	**0.90**
4	**1**	14	**1**	24	**1**	34	**1**	44	**0.8**
5	**1**	15	**1**	25	**0.69**	35	**1**	45	**1**
6	**1**	16	**1**	26	**1**	36	**1**	46	**0.75**
7	**1**	17	**1**	27	**0.84**	37	**1**	47	**1**
8	**1**	18	**1**	28	**0.78**	38	**1**		
9	**0.66**	19	**1**	29	**1**	39	**1**		
10	**0.6**	20	**1**	30	**0.83**	40	**1**		

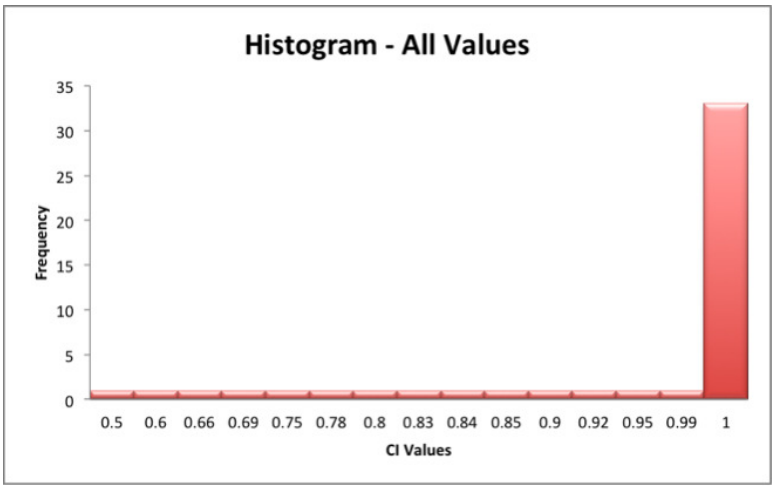

Figure 4.9 CIs Histogram.

than 90% of coverage over catalogs proposed by experts.

H1: CI >= 0.9 of #(Yi). In other words the catalog generated using our approach has at least 90% of coverage over catalogs proposed by experts.

It is important to remember that Yi is the metrics catalog generated by the researcher i expertise. Analyzing the data histogram depicted in Figure 4.9, it is possible to observe a data asymmetry. Consequently, a non-parametric test was applied to assess the data set (WILCOX, 2004). Specifically, the Wilcoxon Test for one-sample was used, and the Z-Test was adopted as the statistic of the test.

For these cases, we should reject the H0 hypothesis if Z0 > Zα (WILCOX, 2004). For this assessment, we used a 99% confidence level (CL). It is the probability that a confidence interval captures the true population parameter given a distribution of samples (WILCOX, 2004).

The values of Zα are obtained through the Z Table. It is a mathematical table used to find the probability of a statistic. Considering 99% of CL, Zα is equal to **2.33** (obtained through the Z Table). Consequently, the Z0 obtained was **5.23**. The statistic of the test says that we should reject the H0 hypothesis if Z0 > Zα (WILCOX, 2004). Consequently, we reject H0. This result indicates that the generated catalogs cover the catalogs suggested by experts in at least 90%.

It is important to highlight that the population was not sampled. Consequently, the

conclusion of the quasi-experiment is based on the observed data. In our case the observational data was the 47 catalogs suggested by the experts and generated using our approach. Nevertheless, even with the unknown population, it is possible to study some population characteristics through the observed data. For this quasi-experiment, characteristics such as expertise in software metrics, and professional profile defined the type of population. Thus, the results assessment is valid for the observational data.

4.5 Answers of the Thesis's RQs

This section is dedicated to answer the Research Questions (RQs) raised by the Ph.D. research. We considered that the lack of useful information about OOSM metrics that support the decision-making process about which ones can be adopted in OOSM evaluations is a research problem. Hereafter, the RQs are exposed and discussed:

- **RQ1: What metrics were adopted to assess software maintainability in OOSD?** The goal of this question was to map all information about software maintainability metrics. The answer of this question is presented in Section 4.2, where a large number of metrics that address this software characteristic is presented. Considering this context, it is clear that the decision-making process about metrics' adoption is a hard task.

- **RQ2: What OOSM metrics are most widely adopted?** This question was proposed because 568 OOSM metrics came up after we performed the systematic mapping study. It is infeasible to adopt this large number of metrics in a software maintainability assessment. Consequently, organizing and listing the most adopted metrics can facilitate the process of choosing the metrics, since it allows presenting a subset of the 568 metrics to researchers. This organization is represented by the categories 'Most Relevant' and 'Most Mentioned' previously described and discussed.

  Table 4.18 depicts a catalog with the top of 12 metrics most often used metrics in the academia environment. The first column indicates the metric's name, and the second column shows a brief metric's description. Finally, the number of times the metric appeared in the papers is presented in the last column. It is important to highlight that we considered the 138 primary studies selected in the systematic mapping to rank the metrics through the number of times that they appeared in these primary studies.

  Nevertheless, we also classified hierarchically these studies according to the number of citations, using Google Scholar. After that, we extracted the metrics contained in those studies. The metrics selected by this process ended up being the same metrics shown in Table 4.18.

- **RQ3: How to organize information about OOSM metrics to support the decision-making process in the scenario of metrics adoption?** Analyzing the scenarios of met-

rics' adoption, some features seem to be good indicators of adoption, e.g., tool support, metrics validation, external attributes related to maintainability, and so on. It is clear that the number of citation of a metric in academic papers can also be an indicator of metrics adoption. However, other ways should be take into account to build a useful OOSM metrics catalog. According to the researcher evaluation scenario, it is possible to pick a metric's catalog up to be adopted.

In this context, the categories and subcategories shown in Section 4.2.3 are the base of this approach. The categories were extracted through the metrics' descriptions found in the primary studies selected in the systematic mapping study. They can be used as references of the metrics' features the researchers want to observe for measuring software maintainability. Other facilities, such as tool support and metric's validation, can also be decisive for the metrics catalog building process. Some metrics were clearly proposed and never adopted again by other researchers because of the lack of information about their use.

However, the representation of the metric' evaluation scenario by the categories can increase the knowledge about this kind of metrics and make it easier to effectively adopt them. Therefore, we believe that the categories and subcategories mentioned in Section 4.2.3 answer this question.

4.6 Limitations and Threats to Validity

It is known that threats to validity can influence or limit the interpretation of the research conclusions (PERRY; PORTER; VOTTA, 2000). Therefore, this section discusses the threats that can interfere in the research results. Next, the internal, external, construct, and conclusion validities are presented.

Internal Validity is related to the bias that can occur when the hypothesis captures the objectives and the generalizability of the findings (PERRY; PORTER; VOTTA, 2000). Generally it is related to the error occurred during the research instrumentation choices. Thus, the choice of July of 2011 as the limit date of published studies that address OOSM metrics can be considered a threat to internal validity. Other metrics may have been proposed after this date and not selected by us. However, the limitation of a date to collect primary studies during a Systematic Mapping Study (SMS) is a necessary task.

Another threat to internal validity was the choice of the exclusion criteria of the primary studies during the SMS. Those criteria may have excluded important studies related to OOSM. Specifically, the exigency of the metric's description in the primary study as exclusion criterion can have excluded important studies. Studies that did not present descriptions for the metrics and instead pointed to other papers were disregarded. An in-depth analysis of the studies selected after the 1st round could help to reduce this threat to validity.

The last threat to internal validity identified was the subjectivity used to assess in the metrics' naming inconsistencies. First of all, we identified some metrics with different names but same meanings. However, the semantics of each one of the metrics was not evaluated. Because of the amount of information about this kind of metrics, we just assessed the metrics' names and their descriptions and it is possible to find other kind of inconsistencies in OOSM metrics adoption.

We have identified two threats to the external validity of our results. First, the number of questionnaire's respondents cannot represent the whole community of researchers and practitioners that deal with OOSM. Just 47 people answered the questionnaire. However, among these respondents there are some researchers with high level of expertise in metrics. Another issue to point out as a threat to external validity is the low level of expertise in software maintainability informed by the quasi-experiment's participants. With this scenario, it is possible that we cannot generalize a positive evaluation of the OOSM metrics' catalogs generation, since researchers who consider themselves to be experts in OOSM might have answered the questionnaire differently.

We proposed a number of categories to represent the context of the application of OOSM metrics. Nevertheless, it is possible that some categories do not express the typical scenario of OOSM metrics adoption. Moreover, other categories could be identified to represent other contexts of the adoption of this kind of metrics. Since construct validity is related to the choice of the right measure to be used in a study, we can conclude that the metrics' categorization proposed is also a threat to validity.

Another threat to construct validity was the use of the number of times that a metric appears in a paper to rank it. We assumed that the more relevant metrics were mentioned by the highest number of papers. It is important to highlight that we considered only the 138 primary studies selected in the SMS. Nevertheless, maybe a more expressive OOSM metric was ranked in a low level of this ranking because is not known and other researchers did not employ it.

Considering the statistics adopted for the research evaluation, the hypotheses tests used to assess the OOSM metrics catalogs' generation could be insufficient to infer that this approach is valid to facilitate the decision-making process about the metrics choice in software evaluations.

Conclusion validity is the ability to reach a correct conclusion about the collected data, the employed statistical test, and the reliability of the measures (PERRY; PORTER; VOTTA, 2000). A threat to conclusion validity identified in this study was the high number of metrics that compose the catalogs generated by the portal. We know that this is a study that tries to gather all information about metrics, and that there are many ambiguities and inconsistencies in OOSM metrics adoption. The actual scenario of OOSM metrics' adoption can cause redundancy and consequent increasing the number of metrics proposed for the measure the same software attribute. Consequently, a decision support system should be developed to make more efficient the decision-making process about which metrics to adopt in OOSM evaluation.

Table 4.17 Metrics Scenario (Expert Suggestion x Portal Suggestion).

Participant_ID	#Metrics (Expert Suggestion)	#Metrics (Portal Suggestion)	Coverage %	% of portal metrics x total (568 metrics)
124	8	343	100.00	60.39
78	2	340	100.00	59.86
75	7	250	100.00	44.01
121	4	237	99.58	41.73
162	7	203	100.00	35.74
175	2	158	100.00	27.82
153	10	133	100.00	23.42
91	9	80	100.00	14.08
79	17	53	100.00	9.33
97	6	34	100.00	5.99
127	13	34	100.00	5.99
81	10	29	100.00	5.11
151	4	26	100.00	4.58
94	5	23	100.00	4.05
131	8	22	100.00	3.87
139	11	22	100.00	3.87
111	5	21	90.48	3.70
71	9	20	100.00	3.52
82	10	20	85.00	3.52
84	9	20	100.00	3.52
88	4	20	95.00	3.52
128	7	20	100.00	3.52
74	6	19	100.00	3.35
145	8	19	84.21	3.35
180	4	19	100.00	3.35
59	6	18	100.00	3.17
95	6	18	100.00	3.17
147	9	18	100.00	3.17
62	1	17	100.00	2.99
135	9	17	100.00	2.99
192	7	16	75.00	2.82
53	8	15	100.00	2.64
73	4	15	100.00	2.64
145	5	15	80.00	2.64
114	5	14	92.86	2.46
179	7	14	78.57	2.46
112	7	13	100.00	2.29
138	5	13	69.23	2.29
72	8	12	100.00	2.11
110	7	12	100.00	2.11
125	3	6	100.00	1.06
191	4	6	83.33	1.06
83	3	5	60.00	0.88
189	2	5	100.00	0.88
87	3	3	66.67	0.53
120	1	3	100.00	0.53
117	5	2	50.00	0.35

Table 4.18 Top of 12 Metrics More Used

Metric	Description	# Occurrences
CBO	Coupling Between Objects Classes	43
DIT	Depth of Inheritance Tree	39
LCOM	Lack of Cohesion in Methods	39
NOC	Number of Children of a Class	39
RFC	Response For a Class	36
WMC	Weighted Methods Per Class	33
LOC	Lines of Code	25
MPC	Message Passing Coupling	17
CC	Class Coupling	16
DAC	Data Abstraction Coupling	14
TCC	Tight Class Cohesion	13
LCC	Loose Class Cohesion	10

5

Related Works

This chapter presents a relationship between the thesis's contributions and the state of-the-art published in the literature about Object-Oriented Software Maintainability (OOSM) metrics. There are a lot of works related to software maintainability, addressing specifically how to measure the maintainability. Riaz et al. conducted a systematic review of software maintainability prediction and metrics (RIAZ; MENDES; TEMPERO, 2009). The evidence was gathered from the selected studies against a set of meaningful and focused questions. Their results suggested that there was little evidence on the effectiveness of software maintainability prediction techniques and models. Additionally, they showed that maintainability, as understood in the context of software systems, was in conformance with the definition provided by IEEE (The Institute of Electrical and Eletronics Engineers, 1990).

In addition, they point out that the commonly used maintainability prediction models were based on algorithmic techniques and there was no distinction of which models should be applied to which maintainability sub-characteristic or maintenance type. The most commonly used predictors were those based on size, complexity and coupling, and gathered at source code level. The use of prediction techniques and models, as well as measures accuracy and cross-validation methods was found scarce for validating maintainability prediction models. And more, they found that the most commonly used maintainability metric employed an ordinal scale and was based on expert judgment (RIAZ; MENDES; TEMPERO, 2009).

Another study that collected information about software maintainability metrics was conducted by Kitchenham (KITCHENHAM, 2009). She observed that there are too many papers describing software metrics. This hinders the assessment of the current status of metrics' research. This study confirms that there is a large body of research related to software metrics, and she suggested that researchers might need to refine their empirical methodology before trying to answer useful empirical questions.

Indeed, our study' results corroborate with their results. A systematic mapping study was performed and 568 OOSM metrics were found. In addition, 138 papers addressing OOSM metrics were selected, since 1992 to 2011, while Kitchenham listed 75 papers from 2000 to 2005. The OOSM metrics were related to size, complexity, and coupling, as Riaz et al. said.

Table 5.1 Metrics from Canfora et al. Study (CANFORA et al., 2005)

Name	Description
NA	Number of activities of the software process model
NWP	Number of work products of the software process model
NDWPIn	Number of input dependences of the work products with the activities in the process
NDW-POut	Number of output dependences of the work products with the activities in the process
NDWP	Number of dependences between work products and activities NDWP(PM) = NDW-PIn(MP) + NDWPOut(MP)
NDA	Number of precedence dependences between activities)

However, other internal attributes such as cohesion, inheritance, and architecture constraint were also found in our study. Although their work had analyzed the software maintainability metrics, it is important to highlight that they did not focus in a specific paradigm as our work, which is focused on object-oriented software maintainability metrics.

Nowadays, software systems are complex and an efficient measurement of software quality is necessary. Consequently, the adoption of metrics to measure their quality is part to the development process. However, the lack of standard formalism for defining software metrics has led to ambiguity in their definitions, hampering their applicability, comparison, and implementation.

Considering the importance of software maintenance in academic and industrial scenarios, a way to measure the level of maintainability is using software metrics (BESZEDES et al., 2007; OLIVEIRA et al., 2008; MINGGUANG et al., 2009; KULKARNI; KALSHETTY; G.ARDE, 2010; REVELLE; GETHERS; POSHYVANYK, 2011; ABDI; LOUNIS; SAHRAOUI, 2006; ALSHAMMARI; FIDGE; CORNEY, 2010; SARAIVA; SOARES; CASTOR, 2010). Then, there are lots of studies that adhere to this issue. One of those is the Haynes et al. work, where they derived a model for estimating adaptive software maintenance effort in person/hours, the Adaptive Maintenance Effort Model (AMEffMo) (HAYES; PATEL; ZHAO, 2004). With this estimation, they proposed a method for estimating the effort to perform adaptive maintenance based on the estimated number of lines of code to be changed and/or the number of operators to be changed.

At the same time, it is known that the management of software processes is largely recognized as a key factor for improving both the productivity of an organization and the quality of the software delivered. Then, Canfora et al. introduced a set of metrics for software process models and discussed how these can be used as maintainability indicators (CANFORA et al., 2005). They also reported the results of a family of experiments that assess relationships between the structural properties, as measured by the defined metrics, of the process models and their maintainability. Considering their proposal of empirically evaluating a set of representative metrics to evaluate the maintainability of descriptive Software Process Model (SPM) they concluded that the following set of metrics (shown in Table 5.1) are good maintainability indicators.

$$171 - 5.2 ln(\text{HV}) - 0.23\text{CC} - 16.2 ln(\text{LOC}) + 50.0 sin\sqrt{2.46 * \text{COM}}$$

Figure 5.1 Maintainability Index Calculation Form (HEITLAGER; KUIPERS; VISSER, 2007).

Then, Mattsson et al. emphasized the lack of commonly definition of maintainability models (KAJKO-MATTSSON et al., 2006"). They listed ideas for a maintainability model providing an initial milestone for ample discussion in software engineering community. They believed that the maintainability domain is too wide to be done by individual researchers or groups, consequently, this topic requires a combined academic and industrial mobilization and word-wide level effort (KAJKO-MATTSSON et al., 2006").

As a software characteristic, the maintainability can be measured in three different ways (SPINELLIS, 2006): (i) We can check out the system maintainability over time to see how well we are battling code entropy, the natural tendency of our system design to disintegrate as it evolves; (ii) We can compare different systems performing the same task to judge which one is most maintainable; and (ii) We can evaluate parts of a system to see which parts appear less maintainable and could therefore be a source of maintenance problems. These parts could also become refactoring targets. So, these parts of a system are the pieces of code that can impact in cost during the maintenance activity. In the case of OOSD, attributes, methods, classes, interfaces, packages, and so on will be also analyzed to check the software maintainability level.

Considering the concern about software maintainability previously mentioned, Heitlager et al. proposed a study discussing the **MI!** (**MI!**), which calculates a single number that express the system maintainability (HEITLAGER; KUIPERS; VISSER, 2007). The higher the MI, the more maintainable a system is deemed to be. The MI is a composite number, based on several different metrics for a software system: **HV!** (**HV!**), Class Complexity (CC), the average number of lines of code per module (LOC), and optionally the percentage of **COM!** (**COM!**). It is important to highlight that Halstead Volume, in turn, is a composite metric based on the number of (distinct) operators and operands in source code. The complete fitting function is depicted in Figure 5.1.

They indicated and discussed several problems with the MI, and identified a number of requirements to be fulfilled by a maintainability model to be used in practice. In addition, they argued that a well-chosen selection measures and guidelines for aggregating and rating them can provide a useful bridge between source code metrics and the ISO 9126, which standardized software product quality. It is important to highlight that, for our research we are using the SQuARE standard, which substituted the ISO 9126, for software maintainability definition.

Focusing on the same scenario, Serban et al. proposed a conceptual framework for defining metrics for component-based systems (SERBAN; VESCAN; POP, 2010). The proposed approach defines a metamodel of the corresponding context where metrics are applied. The use of algebraic sets and relations allows us to formally define metrics, thus providing clear and precise definitions. As result, the main advantages of their framework are simplicity, clarity,

and scalability, where new entities can be added or new properties for the existing entities, for example, new properties for components.

Observing the studies aforementioned, and the list of OOSM metrics found, it is possible to note that lot of researchers proposed new metrics without know the actual world of already proposed/adopted metrics in this context. In consequence of this, lots of metrics' naming inconsistencies were found in our study. For the first scenario, there are metrics with the same names and different meanings, and for the second scenario, there are metrics with the different names and same meanings. We identified all the cases for the two scenarios and we believe that with this inconsistencies' identification, the number of new metrics proposals can decrease and the validation of the already existent metrics can be improved.

The research done during this Ph.D. did not proposed new metrics, however, a metrics' categorization was proposed to make easier the decision-make process about which metrics can be adopted in OOSM evaluations. The categories were proposed based on the description of the scenarios where the metrics were applied. These definitions were obtained through the papers selected on the systematic mapping studies. Internal and external software's attributes, tools support, static and dynamic characteristics, and others issues are contemplated in this categorization. We believe that with the research context definition, the process of metrics selection will be quicker. In addition, with the OOSM Metrics' portal available, other researchers can collaborate with the project and since they can check all the metrics (and their descriptions/categorization) already proposed. Consequently, they will not propose new metrics with naming inconsistencies.

Other researchers have proposed metrics' catalogs. For example, Chidamber et al. (CHIDAMBER; KEMERER, 1994) developed a suite where they propose new metrics to assess OO design. This particular work also focuses on the improvements the adoption of software metrics can bring to existing processes, which by that time was a motivating factor for adopting software metrics in general.

In a similar fashion, Sant'anna et al. (SANTANNA et al., 2003) proposed an assessment framework tailored to measure reusability and maintainability of aspect-oriented software. In addition, the framework defines a quality model, which establishes the relationships between the external attributes, internal attributes, and the metrics. Another work (SULTAN; EN-NOUAARY; HAMOU-LHADJ, 2008) presents a new set of metrics proposed using the goal/question/metric method for building secure software systems. Software engineers might use these metrics in combination with other techniques to detect security risks and prevent these risks from becoming reality. Despite defining a catalog of metrics, these studies did not looked at the literature in a wider and rigorous sense as we did as a step to build the metrics catalogs, considering for example, the context of the OOSM evaluation.

On the other hand, considering the measurement and the development of metrics as a key research area in software engineering, Vaishnavi et al. proposed a generic framework has not emerged for structuring work on object-oriented (OO) metrics (VAISHNAVI; PURAO; LIEGLE, 2007). They believe that the framework allows researchers to examine new metrics

for their compliance with the framework. Since the framework is built on a sound theoretical basis and has been validated with the existing metrics, conformance to the framework should add considerable weight to new metrics.

So, the different inferences for the framework can help researchers and practitioners to automatically generate variations of existing metrics. In other words, unless a seed metric is used such as **LOCm!** (**LOCm!**), the framework cannot generate the logical variations such as avgLOCm (Average of Lines of Code in a Module). While the framework can be used to generate a large number of metrics variations, this does not mean that the metrics are indeed meaningful or useful as predictors. Therefore, the framework did not intend to replace research related to analyses of individual metrics such as characterizing behaviors of metrics in response to ranges assumed by input parameters or complexity analyses of existing metrics. Instead, its use can benefit from assurances of appropriate theoretical foundations for measurement of internal properties.

The work shown here does not propose a unique metrics' catalog, as the works aforementioned. With our research it was possible to observe that the best metrics suite for an OOSM assessment depends on the context of the evaluation. Software Maintainability is related to many subcharacteristics, and can be evaluated by different internal attributes. Consequently, the choice of which metrics can be fit in OOSM assessment depends on what the researcher would like to analyze in software system. Additionally, a metrics' suite can be determined through the existence of a tool support that makes easier and quicker the metrics collection.

Facing this reality, we proposed a dynamic catalog generation based on the OOSM evaluation context. The context is defined by the categories, and the researcher/practitioner can choose which metrics' categories characterize their context. As all metrics found in this study are mapped with the categories, the OOSM Metrics' Portal provides a family of catalogs. We believe that the results of this thesis can be the first steps of OOSM metrics standardization, considering metrics' naming, proposal, description and evaluation/validation.

Besides our research that is analyzing the adoption of OOSM metrics, other researchers performed studies related to software metrics applying survey. Ragab and Ammar's work identified a limited set of metrics that have significant impact on design quality attributes (RAGAB; AMMAR, 2010). They adopted the notion of defining design metrics as independent variables that can be measured to assess their impact on design quality attributes as dependent variables. In addition, they also presented survey of existing object oriented design metrics tools that can be used to automate the measurement process. Different from their work where they analyzed four metrics' tool, in our study we found 26 tools dedicated to automatic metrics collection. The focus of our research is not directly related to metrics tools. Nevertheless, as the availability of tool for metrics collection is one issue to be considered in decision-making process about the choice of the most appropriated metrics suite, we listed some tools that collect OOSM metrics, which is another contribution.

6

Concluding Remarks

This chapter presents the concluding remarks obtained after the results assessment. The conclusions are presented in Section 6.1, and Section 6.2 shows the future works based on the gaps found in the results.

6.1 Conclusions

Hereafter, the conclusions are described considering the results of this thesis. This work presented a systematic mapping on object-oriented software maintainability (OOSM) metrics. The studies selection process used in this systematic mapping was detailed in Chapter 3. Based on that, other researchers can evaluate the adoption scenario of this kind of metrics, and eventually, they can reproduce it to validate or refute the results obtained.

It is important to notice that the participation of eight people in this mapping (three MSc students, three PhD students, and two professors) is an important mechanism to try to reduce the evaluation bias. This diversity of opinions caused conflicts and discussions about the ideal studies to be selected as the final result of this mapping study. We also show all the information about the digital libraries, authors, conferences, and journals that addressed the subject analyzed in this study.

Out of 5175 papers returned by the queries done at four digital libraries (ACM, IEEE, Science Direct, EI Compendex) were analyzed, 138 primary studies were selected identifying a set of 568 OOSM metrics. Observing the publication of the primary studies, EI Compendex was the most effective digital library used in this systematic mapping study. This happened because this digital library indexes many other publication databases. METRICS Conference is the conference that published more papers related to software maintainability metrics. This was expected since the conference is directly related to the study subject. In addition, TSE - IEEE Transactions on Software Engineering Journal published the highest number of primary studies selected (27 occurrences). Interestingly, most of the selected papers just proposed the metrics instead of use other metrics already proposed. This result was concluded because from the 568 metrics found, just 194 (34.15%) appeared in more than 2 different primary studies. Thus, we

inferred that very few studies actually used the metrics' suites previously proposed.

Metrics can be used as indicators in the software assessments, supporting both quantitative and qualitative studies. In our previous research, we were not able to find any other work that lists such an extensive number of metrics like the current mapping. Analyzing the results, it is possible to notice that the majority of metrics, almost 84%, addressing software maintainability were related to software cohesion, coupling, and size.

Analyzing the set of the 568 OO metrics found, some inconsistencies and ambiguities in metrics naming and definition were identified. For instance, there were some metrics that, despite having different names, had actually the similar meanings. On the other hand, there were cases where metrics had the same name but different meanings. Thus, we removed these inconsistencies and ambiguities and called this process as Metrics Naming Consolidation.

For the first case, we grouped the metrics with different names, according the internal attributes related to them. After that, based on their adoption, we chosen a metric to be a group representative. We considered the representative metric the one more cited among the most cited papers in the group. For the second case, we inform the researcher how many times each metric with the same name was adopted. This could help researchers in their decision-making process about their adoption. Indeed, the inconsistencies metrics' naming points out the lack of information about the metrics. Probably, some authors proposed new metrics because they did not know about other existent that could be used in their evaluation. We hope these results can be used towards a standardization of OOSM metrics' naming and adoption.

After solving the metrics' inconsistencies, we analyzed the metrics description and we have identified some categories for the metrics. We believe that these categories can represent the context of the metrics' adoption. Consequently, 17 categories was proposed and they were discussed in Chapter 4. It is important to highlight that during the quasi-experiment performed, the respondents chosen categories to represent the context of metrics adoption, and all of the categories proposed by us were selected. Thus, we believe that the categories/subcategories proposed in this work can help researchers in the decision-making process about which metrics can be adopted in their evaluations because they represent the scenario of metrics' adoption.

After all that was exposed, the lack of information about the OOSM metrics is evident, and this can cause misunderstanding, inconsistencies, and ambiguity in metrics' using and proposal, making difficult the process of choosing the metrics. Thus, a portal containing all the information obtained during this research was developed (`http://julianasaraiva.info/oosmMetricsPortal`). At the portal, the information of this type of metrics is available to be checked by any researcher. In addition, other researchers can contribute inserting new data of OOSM metrics through the portal, such as new metrics, new papers that address this type of metrics, new metrics' descriptions, new tools that collect them automatically, and so on.

The portal also contains a catalog generator. The metrics' categories were used as identifiers of assessment context during the OOSM metrics' catalog generation. The researcher can choose which category is relevant for his/her evaluation, and the portal informs a list of

metrics that belongs to those categories. The idea of the metrics catalogs generation based on the context of the adoption's scenario is to facilitate the metrics choice by researchers with none or low experience in metrics adoption, by only showing a subset of all metrics available.

A quasi-experiment was conducted to identify evidences about the usefulness of the catalogs generated by the portal. 47 participants (metrics' experts) answered the online questionnaire. They suggested a metrics catalog based on their own experience and after that, they used the portal to select categories to represent their context of maintainability metrics. Using these categories, the portal generated OOSM metrics' catalogs using the approach proposed here. Assessing the generated catalogs, it was possible to observe that the catalogs generated by the portal had a coverage of at least 90% over the metrics' catalogs suggested by the experts base don their experience. A non-parametric test, Wilcoxon Test, was used to confirm this result.

In addition, the majority of experts that participated in our study (87.2%) answered that the catalogs generated by the portal are equivalent or better than the metrics' catalogs they proposed.

An issue that is important to point out is that renowned researchers, such as, Chris Francis Kemerer, that proposed very known metrics, the CK metrics (CHIDAMBER; KEMERER, 1994) was one of the questionnaire participants. He gave us a positive feedback about the research and about the catalogs generated by the portal.

In addition, all the RQs raised by the thesis' research were answered, and discussed in Section 4.5. The specific goals of this PhD thesis were also achieved in the following way:

1. To consistently map information about existing OOSM metrics - Achieved with the systematic mapping study performance.

2. To disseminate information about OOSM metrics and their applicability - Achieved with the development of the 'General Information Module' of the OOSM Metrics Portal `http://julianasaraiva.info/oosmMetricsPortal`.

3. To develop tools that support the decision-making process about the adoption of OOSM metrics - Achieved with the development of the 'Project Information Management Module' of the OOSM Metrics Portal `http://julianasaraiva.info/oosmMetricsPortal`.

4. To evaluate the usefulness of the proposed catalog and associated tools - Achieved with the assessment of the catalogs' coverage over the catalogs proposed by the experts.

Analyzing the whole scenario of OOSM metrics, the unification and standardization of the naming and adoption is not an easy task. Nevertheless, we believe that this research can be the first step to better understand the application of them as indicators in software maintainability assessments.

6.2 Future Work

As aforementioned, the systematic mapping study covered studies published until June of 2011. Consequently, a mapping update can be performed starting from July of 2011 to check the availability of any new metrics proposed. In addition, other digital libraries not contemplated in the mapping performed can be used in futures mapping trying to cover the gap of any non-selected studies.

On the other hand, some studies were disregarded during the selection process in the mapping study because they did not have the metrics' description. At the majority of cases of disregarding, they contained known metrics and a reference for its descriptions. Nevertheless, since they did not have an explicit description in the study, they were not considered for final evaluation, according to the mapping protocol. Therefore, a depth assessment of these studies should be done to check how impactful they are. Additionally, an investigation of why there are a lot of cases where a certain metric was proposed, was never used/validated by other researchers, and shall be considered by a future work.

The analysis of other concerns that could have relation with software maintainability is needed. Only software size, cohesion, complexity, coupling, inheritance, and software architecture constraints were identified. Considering the OOSM metrics, the inconsistencies found in metrics naming was solved based on the number of adoption of each metric. However, number of adoption can be not the unique relevant point to taken into account for it. Sometimes, powerful metrics are not adopted because they are not known. Thus, other ways to solve the ambiguity in metrics naming have to be analyzed.

The metrics' categories proposed in this work were based on the metrics' descriptions and scenarios of application of these metrics. However, a depth investigation can be done trying to find other categories or subcategories for metrics' classification. The categorization is the engine of the catalog generation. Consequently, the more diversity is the categorization the better can be the catalog generation.

An alternative way to facilitate the metrics decision-making process can be the mapping of each metric to the adoption goal and to the question to be answered of the **GQM!** (**GQM!**) Model. This can also help the inconsistencies in metrics' naming since the ambiguous metrics (same names different meanings or different names e same meanings) can be chosen based on the GQM Model for their adoption. This is another way to execute the Metrics Consolidation.

Even with the automatically catalog generation, depending on the categories chosen, the number of metrics in the catalogs can still be high. The more categories are chosen the shorter is the catalog. Nevertheless, other ways to optimize and improve the catalog generation is demanded. This gap reflected directly in the catalog generation assessment. Researchers with low experience in software maintainability and/or in software metrics did not see the catalog generated by the portal efficient because of the number of metrics returned to them. Thus, the questionnaire has to be applied and answered by other expert researchers to decrease this gap.

Finally, case studies using the metrics suites generated by the approach proposed here have to be performed. Complementarily to catalogs' coverage evaluation, the using of these catalogs can be another evidence to validate our approach.

References

ABDI, M. K.; LOUNIS, H.; SAHRAOUI, H. A. Analyzing Change Impact in Object-Oriented Systems. In: SEAA'06: PROCEEDINGS OF THE 32ND EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS. **Anais...** [S.l.: s.n.], 2006.

ABRAN, A. **Software Metrics and Software Metrology**. Hoboken: John Wiley & Sons Inc, 2010.

ALMEIDA, A. et al. Mecanismos para Guiar Estudos Empiricos em Engenharia de Software: um mapeamento sistematico. (in portuguese). In: ESELAW'11: PROCEEDINGS OF THE 8TH EXPERIMENTAL SOFTWARE ENGINEERING LATIN AMERICA WORKSHOP. **Anais...** [S.l.: s.n.], 2011.

ALSHAMMARI, B.; FIDGE, C.; CORNEY, D. Assessing The Impact of Refactoring on Security-Critical Object-Oriented Designs. In: APSEC'10: PROCEEDINGS OF THE ASIA PACIFIC SOFTWARE ENGINEERING CONFERENCE. **Anais...** [S.l.: s.n.], 2010.

APPLE. **Apple Company**. 2013.

ARKSEY, H.; O'MALLEY, L. Scoping studies: towards a methodological framework. **International Journal of Social Research Methodology**, [S.l.], v.8, n.1, p.19–32, 2005.

BARREIROS, E. et al. A Systematic Mapping Study on Software Engineering Testbeds. In: ESEM'11: PROCEEDINGS OF THE 5TH INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT. **Anais...** [S.l.: s.n.], 2011.

BASILI, V. R. The role of experimentation in software engineering: past, current, and future. In: ICSE'96: PROCEEDINGS OF THE 18TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, Washington, DC, USA. **Anais...** IEEE Computer Society, 1996. p.442–449.

BASILI, V. R.; BRIAND, L. C.; MELO, W. L. Using structural and textual information to capture feature coupling in object-oriented software. **Journal of Software Engineering**, [S.l.], v.22, n.10, p.751 – 761, 1996.

BASILI, V. R.; WEISS, D. M. A Methodology for Collecting Valid Software Engineering Data. **IEEE Trans. Software Eng.**, [S.l.], v.10, n.6, p.728–738, 1984.

BENNETT, K.; RAJLICH, V. Software Maintenance and Evolution: a roadmap. **The Future of Software Engineering**, [S.l.], p.73–90, 2000.

BESZEDES, A. et al. The Dynamic Function Coupling Metric and Its Use in Software Evolution. In: CSMR'07: PROCEEDINGS OF 11TH EUROPEAN CONFERENCE ON SOFTWARE MAINTENANCE AND REENGINEERING. **Anais...** [S.l.: s.n.], 2007.

BLANCHARD, B.; VERMA, D.; PETERSON, E. **Maintainability**: a key to effective serviceability and maintenance management. [S.l.]: Wiley, 1995. (New Dimensions In Engineering Series).

BOURQUE, P.; DUPUIS, R. (Ed.). **Software Engineering Body of Knowledge (SWEBOK)**. EUA: Angela Burgess, 2004.

CANFORA, G. et al. A family of experiments to validate metrics for software process models. **J. Syst. Softw.**, New York, NY, USA, v.77, n.2, p.113–129, Aug. 2005.

CHEN, J.-C.; HUANG, S.-J. An empirical analysis of the impact of software development problem factors on software maintainability. **J. Syst. Softw.**, New York, NY, USA, v.82, n.6, p.981–992, June 2009.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. **Software Engineering, IEEE Transactions on**, [S.l.], v.20, n.6, p.476 –493, jun 1994.

CINNEIDE, M. et al. Experimental assessment of software metrics using automated refactoring. In: ACM-IEEE INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, New York, NY, USA. **Proceedings...** ACM, 2012. p.49–58. (ESEM '12).

CIOLKOWSKI, M. et al. Evaluating Software Project Control Centers in Industrial Environments. In: FIRST INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2007. p.314–323. (ESEM '07).

CODE, G. **Google Code Repository**. 2013.

CODEFLEX. **Microsoft's free open source project hosting site**. 2013.

CONCAS, G. et al. Assessing Traditional and New Metrics for Object-Oriented Systems. In: WETSOM'10: PROCEEDINGS OF THE 1ST INTERNATIONAL WORKSHOP ON EMERGING TRENDS IN SOFTWARE METRICS. **Anais...** [S.l.: s.n.], 2010.

CSIAC. **SOFTWARE ACQUISITION GOLD PRACTICE**. 2012.

DAVIS, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. **MIS Q.**, Minneapolis, MN, USA, v.13, n.3, p.319–340, Sept. 1989.

DHILLON, B. S. **Engineering Maintainability**: how to design for reliability and easy maintenance. [S.l.]: Elsevier Science, 1999.

DHILLON, B. S. **Maintainability, maintenance, and reliability for engineers**. Boca Raton: CRC - Taylor and Francis, 2006. 1 online resource (217 p.)p. Description based on print format version record.

EADDY, M. et al. Do Crosscutting Concerns Cause Defects? **IEEE Transactions on Software Engineering**, [S.l.], v.34, n.4, p.497–515, july-aug. 2008.

FORGE, S. **Source Forge Repository**. 2013.

GITHUB. **Github Repository**. 2013.

GROUP, J. **JSP Website**. 2013.

GROUP, J. **JQuery Website**. 2013.

HAN, A.-R. et al. Measuring behavioral dependency for improving change-proneness prediction in UML-based design models. **Journal of Systems and Software**, [S.l.], v.83, p.222–234, 2010.

HAYES, J. H.; PATEL, S. C.; ZHAO, L. A Metrics-Based Software Maintenance Effort Model. In: EIGHTH EUROMICRO WORKING CONFERENCE ON SOFTWARE MAINTENANCE AND REENGINEERING (CSMR'04), Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2004. p.254–.

HEITLAGER, I.; KUIPERS, T.; VISSER, J. A Practical Model for Measuring Maintainability. In: INTERNATIONAL CONFERENCE ON QUALITY OF INFORMATION AND COMMUNICATIONS TECHNOLOGY, 6., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2007. p.30–39. (QUATIC '07).

HUDLI, R. V.; HOSKINS, C. L.; HUDLI, A. V. Software Metrics for Object-Oriented Designs. In: ICCD'94: PROCEEDINGS OF IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN: VLSI IN COMPUTERS AND PROCESSORS. **Anais...** [S.l.: s.n.], 1994.

ISO/IEC, B. S. P. **Systems and Software Quality Requirements and Evaluation (SQuaRE) Models**. [S.l.]: ISO/IEC, 2011.

JAVAFORGE. **JavaForge Repository**. 2013.

JEDLITSCHKA, A.; CIOLKOWSKI, M. ISO/IEC 25020: software product quality requirements and evaluation. In: AZUMA, M. et al. (Ed.). **SQuaRE Measurement reference model and guide**. [S.l.]: ISO/IEC JTC1/SC7/WG6, 2006.

JEDLITSCHKA, A.; CIOLKOWSKI, M. Reporting Experiments in Software Engineering. In: SHULL, F.; SINGER, J.; SJoBERG, D. (Ed.). **Guide to Advanced Empirical Software Engineering**. London: Springer-Verlag, 2008.

JEDLITSCHKA, A. et al. Relevant Information Sources for Successful Technology Transfer: a survey using inspections as an example. In: FIRST INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2007. p.31–40. (ESEM '07).

JOHN, I.; EISENBARTH, M. A decade of scoping: a survey. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 13., Pittsburgh, PA, USA. **Proceedings...** Carnegie Mellon University, 2009. p.31–40. (SPLC '09).

JONES, C. Software metrics: good, bad and missing. **Computer Journal**, [S.l.], v.27, n.9, p.98 – 100, 2010.

Juristo; Moreno. **Basics of Software Engineering Experimentation**. Norwell, MA, USA: Kluwer Academic Publishers, 2001.

KAJKO-MATTSSON, M. et al. A Model of Maintainability - Suggestion for Future Research. In: SOFTWARE ENGINEERING RESEARCH AND PRACTICE. **Anais...** CSREA Press, 2006". p.436–441.

KAN, S. H. **Metrics and Models in Software Quality Engineering**. Reading, MA: Addison Wesley, 1995.

KANER, C.; BOND, W. P. Software Engineering Metrics: what do they measure and how do we know? In: METRICS'04: PROCEEDINGS OF THE 10ND INTERNATIONAL SOFTWARE METRICS SYMPOSIUM. **Anais...** [S.l.: s.n.], 2004.

KITCHENHAM, B. A. What's up with software metrics? A preliminary mapping study. **International Journal of System and Software**, [S.l.], p.37–51, 2009.

KITCHENHAM, B. A. Scoping studies: towards a methodological framework. **Journal of Systems and Software**, [S.l.], v.83, n.1, p.37–51, 2010.

KITCHENHAM, B. **Procedures for performing systematic reviews**. [S.l.]: Keele University, 2004.

KITCHENHAM, B.; CHARTERS, S. **Guidelines for Performing Systematic Literature Reviews in Software Engineering**. [S.l.]: Software Engineering Group, School of Computer Science and Mathematics, Keele University, 2007.

KULKARNI, U. L.; KALSHETTY, Y. R.; G.ARDE, V. Validation of CK metrics for Object Oriented Design Measurement. In: ICETET.'10: PROCEEDINGS OF 3RD INTERNATIONAL CONFERENCE ON EMERGING TRENDS IN ENGINEERING AND TECHNOLOGY. **Anais...** [S.l.: s.n.], 2010.

LAUKKANEN, E.; MANTYLA, M. Survey Reproduction of Defect Reporting in Industrial Software Development. In: EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT (ESEM), 2011 INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2011. p.197 –206.

LAVIERI, T. **VRaptor Website**. 2013.

LIN, M. **Maintainability of Facilities**: for building professionals. [S.l.]: World Scientific, 2010.

MANTORO, T. Metrics Evaluation for Context-Aware Computing. In: MOMM'09: PROCEEDINGS OF 7TH INTERNATIONAL CONFERENCE ON ADVANCES IN MOBILE COMPUTING AND MULTIMEDIA. **Anais...** [S.l.: s.n.], 2009.

MAYER, T.; HALL, T. A Critical Analysis of Current OO Design Metrics. **Software Quality Journal**, [S.l.], v.8, n.2, p.97–110, 1999.

MENEELY, A.; SMITH, B.; WILLIAMS, L. Validating Software Metrics: a spectrum of philosophies. **ACM Transactions on Software Engineering and Methodology (TOSEM)**, [S.l.], v.21, n.2, p.24–48, 2012.

MICROSOFT. **Microsoft Company**. 2013.

MILES, M.; HUBERMAN, A. **Qualitative data analysis**: a sourcebook of new methods. [S.l.]: Sage Publications, 1994. (Sage Library of Social Research).

MINGGUANG, Z. et al. The Measurement and Evaluation for Large-scale Object-oriented Software System. In: HIS.'09: PROCEEDINGS OF 9TH INTERNATIONAL CONFERENCE ON HYBRID INTELLIGENT SYSTEMS. **Anais...** [S.l.: s.n.], 2009.

NIE, Z.; KAMBHAMPATI, S. In: IIWEB. **Anais...** [S.l.: s.n.], 2004. p.123–128.

O'BRIEN, M. P.; BUCKLEY, J.; EXTON, C. Empirically Studying Software Practitioners " Bridging the Gap between Theory and Practice. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 21., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2005. p.433–442. (ICSM '05).

OLIVEIRA, M. et al. Software Quality Metrics and their Impact on Embedded Software. In: MOMPES'08: PROCEEDINGS OF 5TH INTERNATIONAL WORKSHOP ON MODEL-BASED METHODOLOGIES FOR PERVASIVE AND EMBEDDED SOFTWARE. **Anais...** [S.l.: s.n.], 2008.

ORACLE. **JPA Oracle Website**. 2013.

PERLIS, A.; SAYWARD, F.; SHAW, M. **Software Metrics**: an analysis and evaluation. [S.l.]: Mit Press, 1981. (The Mit Press Series in Computer Science).

PERRY, D. E.; PORTER, A. A.; VOTTA, L. G. Empirical Studies of Software Engineering: a roadmap. In: CONFERENCE ON THE FUTURE OF SOFTWARE ENGINEERING, New York, NY, USA. **Proceedings...** ACM, 2000. p.345–355. (ICSE '00).

PETERSEN, K.; WOHLIN, C. Context in industrial software engineering research. In: EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, 2009. ESEM 2009. 3RD INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2009. p.401 –404.

POSTGRESQL. **PostgreSQL Website**. 2013.

PUNTER, T. et al. Conducting On-line Surveys in Software Engineering. **Empirical Software Engineering, International Symposium on**, Los Alamitos, CA, USA, v.0, p.80, 2003.

RAGAB, S. R.; AMMAR, H. H. Object oriented design metrics and tools a survey. In: INFORMATICS AND SYSTEMS (INFOS), 2010 THE 7TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.1 –7.

REVELLE, M.; GETHERS, M.; POSHYVANYK, D. Using structural and textual information to capture feature coupling in object-oriented software. **Journal of Empirical Software Engineering**, [S.l.], v.16, n.6, p.773–811, 2011.

RIAZ, M.; MENDES, E.; TEMPERO, E. A Systematic Review of Software Maintainability Prediction and Metrics. In: ESEM'09: PROCEEDINGS OF THE 3RD INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT. **Anais...** IEEE Computer Society, 2009.

RODRÍGUEZ, P. et al. Survey on agile and lean usage in finnish software industry. In: ACM-IEEE INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, New York, NY, USA. **Proceedings...** [S.l.: s.n.], 2012. p.139–148. (ESEM '12).

ROMBACH, D. et al. Impact of research on practice in the field of inspections, reviews and walkthroughs: learning from successful industrial uses. **SIGSOFT Softw. Eng. Notes**, New York, NY, USA, v.33, n.6, p.26–35, Oct. 2008.

SANTANNA, C. et al. On the Reuse and Maintenance of Aspect-Oriented Software: an assessment framework. In: SBES'10: PROCEEDINGS OF THE BRAZILIAN SYMPOSIUM OF SOFTWARE ENGINEERING. **Anais...** [S.l.: s.n.], 2003.

SARAIVA, J. et al. Aspect-Oriented Software Maintenance Metrics: a systematic mapping study. In: EASE'12: PROCEEDINGS OF 16TH INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING. **Anais...** [S.l.: s.n.], 2012.

SARAIVA, J.; SOARES, S.; CASTOR, F. Assessing the Impact of AOSD on Layered Software Architectures. In: ECSA'10: PROCEEDINGS OF THE 4TH EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE. **Anais...** [S.l.: s.n.], 2010.

SERBAN, C.; VESCAN, A.; POP, H. F. A conceptual framework for component-based system metrics definition. In: ROEDUNET INTERNATIONAL CONFERENCE (ROEDUNET), 2010 9TH. **Anais...** [S.l.: s.n.], 2010. p.73 –78.

SHADISH, W.; COOK, T.; CAMPBELL, D. **Experimental and quasi-experimental designs for generalized causal inference**. [S.l.]: Houghton Mifflin, 2002.

SHANTI. An Empirical Validation of Software Quality Metric Suites on Open Source Software for Fault-Proneness Prediction in Object Oriented Systems. **European Journal of Scientific Research**, [S.l.], v.51, n.2, p.168–181, 2011.

SHEPPERD, M.; INCE, D. A critique of three metrics. **Journal of Systems and Software**, [S.l.], v.26, n.3, p.197–210, 1994.

SILLITO, J.; WYNN, E. The social context of software maintenance. In: TWENTY-THIRD IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE (ICSM). **Proceedings...** [S.l.: s.n.], 2007. p.325–334.

SILVA, F. et al. Six years of systematic literature reviews in software engineering: an updated tertiary study. **International Journal Information and Software Technology**, [S.l.], p.889–913, 2011.

SIM, S. E.; EASTERBROOK, S.; HOLT, R. C. Using benchmarking to advance research: a challenge to software engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 25., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2003. p.74–83. (ICSE'03).

SOMMERVILLE, I. **Software Engineering**. [S.l.]: Addison-Wesley, 2007. (International Computer Science Series).

SPINELLIS, D. **Code Quality**: the open source perspective (effective software development series). [S.l.]: Addison-Wesley Professional, 2006.

SQA. **ISO 9126 Software Quality Characteristics**. 2012.

STAPLES, M.; NIAZI, M. Experiences using systematic review guidelines. **J. Syst. Softw.**, New York, NY, USA, v.80, n.9, p.1425–1437, Sept. 2007.

SULTAN, K.; EN-NOUAARY, A.; HAMOU-LHADJ, A. Catalog of Metrics for Assessing Security Risks of Software throughout the Software Development Life Cycle. In: INTERNATIONAL CONFERENCE ON INFORMATION SECURITY AND ASSURANCE. **Anais...** [S.l.: s.n.], 2008. p.461 –465.

TAIRAS, R. Clone maintenance through analysis and refactoring. In: FOUNDATIONS OF SOFTWARE ENGINEERING DOCTORAL SYMPOSIUM, 2008., New York, NY, USA. **Proceedings...** ACM, 2008. p.29–32. (FSEDS '08).

The conceptual grouping effect: categories matter (and named categories matter more). **Cognition**, [S.l.], v.108, n.2, p.566 – 577, 2008.

The Institute of Electrical and Eletronics Engineers. **IEEE Standard Glossary of Software Engineering Terminology**. 1990.

TICHY, W. F. **Should Computer Scientists Experiment More?** Los Alamitos, CA, USA: IEEE Computer Society, 1997. 32-40p. v.31.

TIGRIS. **Tigris Repository**. 2013.

TIOBE. **Tiobe Website**. 2013.

UMARJI, M.; SEAMAN, C. Measuring OO Systems: a critical analysis of the mood metrics. In: TOOLS'99: PROCEEDINGS OF THE CONFERENCE OF TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES AND SYSTEMS. **Anais...** [S.l.: s.n.], 1999.

UMARJI, M.; SEAMAN, C. Why do programmers avoid metrics? In: ESEM'08: PROCEEDINGS OF THE 2ND INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT. **Anais...** [S.l.: s.n.], 2008.

VAISHNAVI, V. K.; PURAO, S.; LIEGLE, J. Object-oriented product metrics: a generic framework. **Inf. Sci.**, [S.l.], v.177, n.2, p.587–606, 2007.

VELING, A.; VAN DER WEERD, P. Conceptual Grouping in Word Co-occurrence Networks. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE - VOLUME 2, 16., San Francisco, CA, USA. **Proceedings...** Morgan Kaufmann Publishers Inc., 1999. p.694–699. (IJCAI'99).

WESTFALL, L.; ROAD, C. 12 Steps to Useful Software Metrics. **Proceedings of the Seventeenth Annual Pacific Northwest Software Quality Conference**, [S.l.], v.57 Suppl 1, n.May 2006, p.S40–3, 2005.

WILCOX, R. R. **Introduction to Robust Estimation and Hypothesis Testing**. 2nd.ed. [S.l.]: Academic Press, 2004. (Statistical Modeling and Decision Science).

WOHLIN, C. et al. **Experimentation in software engineering**: an introduction. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

YAZBEK, H. A concept of quality assurance for metrics in CASE-tools. **ACM SIGSOFT Software Engineering Notes**, [S.l.], v.35, n.5, p.1–8, 2010.

ZANNIER, C.; MELNIK, G.; MAURER, F. On the Success of Empirical Studies in the International Conference on Software Engineering. In: ICSE'06: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE OF SOFTWARE ENGINEERING, Shanghai, China. **Anais...** [S.l.: s.n.], 2006.

Appendix

A

Publications and Awards

A.1 Awards

- SRC-ICSE 2013. 4th place in Student Research Competition on 35th International Conference on Software Engineering. San Francisco/CA - USA.

A.2 Publications Directly Related to the Research

- 2014 - Saraiva, J., Soares, S., and Castor, F.; Assessment of a Roadmap for Software Maintainability Measurement in Object-Oriented Context. Journal of Systems and Software. (to be submitted)

- 2013 - Saraiva, J. A. G.; A Roadmap for Software Maintainability Measurement. In: 35th International Conference on Software Engineering (ICSE'13), 2013, San Francisco - USA (ACM - SRC).

- 2013 - Saraiva, J., Soares, S., and Castor, F.; Towards a Catalog of Object-Oriented Software Maintainability Metrics. In: 4th International Workshop on Emerging Trends in Software Metrics (WeTSOM'13), 2013, San Francisco - USA (Short paper).

- 2012 - Saraiva, J., Barreiros, E., Almeida, A., Lima, F., Alencar, A., Lima, G., Soares, S., and Castor, F.; Aspect-Oriented Software Maintenance Metrics: A Systematic Mapping Study. In: Proceedings of 16th International Conference on Evaluation and Assessment in Software Engineering (EASE'12), 2012, Ciudad Real - Spain (Full paper).

- 2011 - Saraiva, J., Soares, S., and Castor, F.; A metrics suite to evaluate the impact of AOSD on Layered Software Architectures. In: Proceedings of the Empirical Evaluation of Software Composition and Techniques (ESCOT'11), 2011, Lancaster, UK (Full paper).

- 2011 - Saraiva, J., Soares, S., and Castor, F.; Assessing a set of metrics used in a testbed for Aspect-Oriented Software Maintenance. In: Proceedings of the 10th International

Conference on Aspect-Oriented Software Development Companion (AOSD'11), 2011, Porto de Galinhas - Brasil (POSTER).

A.3 Other Publications

- 2012 - Lima, F.; Filho, A. A.; Barreiros, E.; Saraiva, J. A. G.; Alencar, A.; Lima, G.; Soares, S.; Métodos, Técnicas e Ferramentas para o Desenvolvimento de Software Educacional: Um Mapeamento Sistemático (*In portuguese*). In: Brazilian Symposium of Information Technology in Education (SBIE'12), 2012, Rio de Janeiro - Brasil (Full paper).

- 2011 - Barreiros, E.; Filho, A. A.; Saraiva, J. A. G.; Soares, S.; A Systematic Mapping Study on Software Engineering Testbeds. In: 5th International Symposium on Empirical Software Engineering (ESEM'11), 2011, Alberta - Canada (Full paper).

- 2011 - Almeida, A., Barreiros, E., Saraiva, J., and Soares, S.; Mecanismos para Guiar Estudos Empiricos em Engenharia de Software: Um Mapeamento Sistematico. (In portuguese). In: Proceedings of the 8th Experimental Software Engineering Latin America Workshop (ESELAW'11), 2011, Rio de Janeiro - Brasil (Full paper).

- 2011 - Saraiva, J., Soares, S., and Castor, F.; Analyzing Architectural Conformance of Layered Aspect-Oriented Systems with Arche Meter. In: Proceedings of the 10th International Conference on Aspect-Oriented Software Development Companion (AOSD'11), 2011, Porto de Galinhas - Brasil (Full paper).

- 2011 - Saraiva, J., Soares, S., and Castor, F.; Analyzing Architectural Layering Violations in Aspect-Oriented Software with ArchE Meter. In: Brazilian Software Conference (CBSoft'11), 2011, São Paulo - Brasil (Full paper).

- 2010 - Saraiva, J., Soares, S., and Castor, F.; Assessing the Impact of AOP on Layered Software Architectures. In: 4th European Conference on Software Architecture (ECSA'10), 2010, Copenhagen (Full paper).

B

Systematic Mapping Protocol

B.1 Background

The focus of this work is building a testbed for Object-Oriented Software Maintainability (OOSM). This testbed will provide end-to-end systematic comparison of OOSD with mainstream development approaches. In this context, it is necessary a set of metrics that can be suited to support this proposed environment (testbed). A lot of researchers use a large numbers of metrics to measure their target systems adopted in their experiments. Thus, we will conduct a systematic mapping to study and analyze metrics that can be appropriated to help researchers in their both qualitative and quantitative OOSM assessments. Thus, it is necessary to summarize all possible existing information about OOSM metrics in order to draw more general conclusion about this point. Then, this is the actual necessity of the proposed systematic mapping.

B.2 Review Question

- RQ2: What metrics were adopted to assess software maintainability in OOSD?

B.3 Roles and Responsibilities

Juliana Saraiva and Emanoel Barreiros will read and perform the data extraction from all studies during the first selection. After that, three pairs of researchers will be composed by Juliana and Gustavo, Aline and Adauto, Emanoel and Flavio. The pairs will be composed based on their experience in Systematic Literature Review (SLR). Then, we will match a researcher that already performed a SLR with another that never did. The studies will be shared between the pairs, and each member of each pair will read and select the study in conformance with the inclusion criteria. In sequence, the pairs will reach agreement regarding the subjective evaluations and primary study selection. The pairs will confront their results, supervised by the professors Sergio Soares/Fernando Castor. After this phase, all the primary studies that will compose the result of systematic mapping study will be selected.

Table B.1 Documentation

Data Source	Documentation
Digital Library	Name of database, Search strategy for the database, Date of search, Years covered by search
Conferences Proceedings	Title of proceedings, Name of conference (if different), Title translation (if necessary), Journal name (if published as part of a journal)

B.4 Search Process

The strategy will be search for primary studies that have to include: search terms and sources (all digital libraries, specific journals and conference proceedings). The search process for this study will be based on an automated search of the following digital libraries:

- ACM
- IEEE
- EI Compendex
- Science Direct

B.4.1 Search String

All searches will be based on the full text. For all the sources, a set of simple search strings will be used and the outcome from each of the searches for each source will be aggregated. The search string was based on the terms that represent the research context. After its proposal, the string was analyzed and validated with researchers with more expertise (Sergio Soares and Fernando Castor). Consequently, the final search string was:

SEARCH STRING: (Software Engineering AND (Aspect Oriented Programming, OR Maintainability OR Aspect Oriented Software Development OR Crosscutting Concern OR Maintenance OR Object Oriented Programming OR Object Oriented Development OR Evolution) AND (Metrics OR Measurement OR Measure)).

B.4.2 Study Documentation

In our research, the search processes and results will be registered as follow:

B.4.3 Study Selection

Once the search string has been fully proposed and tested, Juliana Saraiva will use the string to search studies in the digital libraries aforementioned. The five steps shown below

compose the whole process:

Stage 1: The title and abstract of each article from each digital library will be reviewed against the inclusion criteria by Juliana Saraiva and Emanoel Barreiros, and any papers that are clearly irrelevant will be excluded.

Stage 2: The lists of studies from each digital library will be collated in a search documentation, and duplicated references will be removed.

Stage 3: Full copies of all the candidate studies remaining after stage 2 will be reviewed against the inclusion criteria. To complete Stage 3, Juliana Saraiva will assign each paper to two members (the pair) of the research team, and each person will decide whether or not to include the paper. Excluded papers will be marked as either an irrelevant paper. Sergio Soares will mediate any disagreements about the inclusion of a paper. The members of each pair must discuss the disagreement until the status of the study be finally solved.

Stage 4: During this stage, each pair will do:

- To associate a unique reference number for each primary study.
- To identify as journal article, a conference paper, book chapter, technical report or other.

B.4.3.1 Inclusion Criteria

After the initial selection, the full versions of each paper will be obtained so that we can perform a more detailed analysis. The following criteria will be applied in order to include or exclude the paper from the review. It is important to remember that they might be refined during the extraction process. The inclusion criteria are:

- It is a real paper, not a power point presentation or extended abstract;
- The paper is not a duplicate;
- The paper is related to software engineering;
- The paper presents a maintainability/evolution metric;
- The paper presents at least one metric related to OO or OA programming.
- The paper contains metrics description or a reference where it is described. We understand that the metric description is the metric definition, the way that it can collected.

B.5 Project Timetable

The Table B.2 indicates the duration of each phase of the mapping process.

Table B.2 Timetable

Data	Duration	Phase Duration
30/06/2011 to 30/08/2011	2 months	Searching Primary Studies. (1st Round).
31/08/2011 to 30/09/2011	1 month	Selection of Papers. (2nd Round).
05/09/2011 to 25/09/2011	20 days	Meeting to discuss conflicts.
26/09/2011 to 14/11/2011	1.5 month	Data Extraction and Synthesis.
15/11/2011 to 15/12/2011	1 month	Organize and Reporting Results.

B.6 Data Extraction and Synthesis

This form should be developed to collect information needed to address the review questions. Data extraction form needs to be piloted on a sample of primary studies. Our form will be as the model:

- Reviewers: the name of reviewer that includes this paper on the systematic mapping;
- Date of Data Extraction: informs when the information was extracted;
- Title
- Author(s)
- Journal/Conferences
- Metric(s) presented in the paper
- Metric's description(s)
- Paradigm(s) Associated

C

Selected Primary Studies

[SM1] Heung Seok Chae, Yong Rae Know. A Cohesion Measure for Classes in Object-Oriented Systems. METRICS'98.

[SM3] Letha H. Etzkorna, Sampson E. Gholstonb, Julie L. Fortuneb, Cara E. Steina, Dawn Utleyb, Phillip A. Farringtonb, Glenn W. Coxa. A Comparison of Cohesion Metrics for Object-Oriented Systems. INFSOF'04.

[SM4] Aaron B. Binkley, Stephen R. Schach. A Comparison of Sixteen Quality Metrics for Object-Oriented Design. IS'96

[SM7] Hashem Yazbek. A Concept of Quality Assurance for Metrics in CASE Tools. SIGSOFT'10.

[SM8] Camelia Serban. A Conceptual Framework for Object-Oriented Design Assessment. EMS'10.

[SM10] Vu Nguyen, Barry Boehm, Phongphan Danphitsanuphan. A Controlled Experiment in Assessing and Estimating Software Maintenance Tasks. INFSOF'10.

[SM11] Hironori Washizaki. A Coupling-based Complexity Metric for Remote Component-based Software Systems Toward Maintainability Estimation. APSEC'06.

[SM13] Youssef Hassoun, Roger Johnson, Steve Counsell. A Dynamic Runtime Coupling Metric for Metalevel Architectures. CSMR'04.

[SM15] Jagdish Bansiya, and Carl G. Davis. A Hierarchical Model for Object-Oriented Design Quality Assessment. TSE'02.

[SM23] Tatsuya Miyake, Yoshiki Higo, Katsuro Inoue. A Metric Based Approach for Reconstructing Methods in Object-Oriented Systems. WoSQ'08.

[SM29] Huan Li. A Novel Coupling Metric for Object-Oriented Software Systems. KAM'08.

[SM31] Thomas Panas, Rudiger Lincke, Jonas Lundberg, Welf Lowe. A Qualitative Evaluation of a Software Development and ReEngineering Project. SEW'05.

[SM35] Mehwish Riaz, Emilia Mendes, Ewan Tempero. A Systematic Review of Software Maintainability Prediction and Metrics. ESEM'09.

[SM37] Lionel C. Briand, John W. Daly, and Jurgen K. Wust. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. TSE'97.

[SM38] Lionel C. Briand, John W. Daly, and Jurgen K. Wust. A Unified Framework for Coupling Measurement in Object-Oriented Systems. TSE'99.

[SM39] Victor R. Basili, Fellow, Lionel C. Briand, and Walcelio L. Melo. A Validation of Object-Oriented Design Metrics as Quality Indicators. TSE'96.

[SM42] Jill Doake, Ishbel Duncan. Amber Metrics for the Testing and Maintenance of Object-Oriented Designs. CSMR'98.

[SM45] Amjed Tahir and Rodina Ahmad. An AOP-Based Approach for Collecting Software Maintainability Dynamic Metrics. ICCRD'10.

[SM47] Jennifer Munnelly, Serena Fritsch, Siobhan Clarke. An Aspect-Oriented Approach to the Modularisation of Context. PerCom'07.

[SM48] Ayaz Farooq, Rene Braungarten, Reiner R. Dumke. An Empirical Analysis of Object-Oriented Metrics for Java Technologies. INMIC'05.

[SM49] Rachel Burrows, Fabiano Cutigi Ferrari, Alessandro Garcia, Francois Taiani. An Empirical Evaluation of Coupling Metrics on Aspect-Oriented Programs. WETSoM'10.

[SM53] Haihao Shen, Sai Zhang, Jianjun Zhao. An Empirical Study of Maintainability in Aspect-Oriented System Evolution Using Coupling Metrics. TASE'08.

[SM54] W. Lia, L. Etzkorna, C. Davisa, J. Talburt. An Empirical Study of Object-Oriented System Evolution. INFSOF'00.

[SM55] Ewan Tempero, Steve Counsell and James Noble. An Empirical Study of Overriding in Open Source Java. ACSC'10.

[SM66] F.G. Wilkiea, B.A. Kitchenham. An Investigation of Coupling, Reuse and Maintenance in a Commercial C++ Application. INFSOF'01.

[SM68] Jehad Al Dallal, Lionel C. Briand. An Object-Oriented High Level Design Based Class Cohesion Metric. INFSOF'10.

[SM69] William W. Pritchett IV. An Object-Oriented Metrics Suite for Ada 95. SIGAda'01.

[SM70] R. Harrison, S. Counsell, R. Nithi. An Overview of Object-Oriented Design Metrics. STEP'97.

[SM73] M.K Abdi, H. Lounis, H. Sahraoui. Analyzing Change Impact in Object-Oriented Systems. SEAA'06.

[SM74] Santonu Sarkar, Girish Maskeri Rama, and Avinash C. Kak. API Based and Information Theoretic Metrics for Measuring the Quality of Software Modularization. TSE'07.

[SM75] Ghassan Allcadi, Doris L. Carver. Application of Metrics to Object-Oriented Designs. TSE'98.

[SM76] Usha Kumari, Sucheta Bhasin. Application of Object-Oriented Metrics to C++ and Java a Comparative Study. SIGSOFT'11.

[SM79] Juha Gustafsson, Jukka Paakki, Lilli Nenonen, and A. Inkeri Verkamo. Architecture Centric Software Evolution by Software Metrics and Design Patterns. CSMR'05.

[SM81] Marcela Genero Bocco1, Daniel L. Moody and Mario Piattini. Assessing the Capability of Internal Metrics as Early Indicators of Maintenance Effort Through Experimentation. SMR'05.

[SM82] Juliana Saraiva, Sergio Soares, Fernando Castor. Assessing the Impact of AOSD on Layered Software Architectures. ECSA'10.

[SM84] Bandar Alshammari, Colin Fidge, Diane Corney. Assessing the Impact of Refactoring on Security Critical Object-Oriented Designs. APSEC'10.

[SM85] Luis Reynoso, Esperanza Manso, Marcela Genero and Mario Piattini. Assessing the Impact of Refactoring on Securitycritical Object-Oriented Designs. IS'10.

[SM86] Ebrahim Bagheri, Dragan Gasevic. Assessing the Influence of Import-Coupling on OCL Expression Maintainability: A Cognitive Theory-Based Perspective. SQJ'10.

[SM87] Giulio Concas, Michele Marchesi, Alessandro Murgia, Sandro Pinna, Roberto Tonelli. Assessing Traditional and New Metrics for Object-Oriented Systems. WETSoM'10.

[SM92] Mikael Lindvall, Roseanne Tesoriero, Patricia Costa. Avoiding Architectural Degeneration an Evaluation Process for Software Architecture. METRICS'02.

[SM95] Henk van der Schuur, Slinger Jansen and Sjaak Brinkkemper. Becoming Responsive to Service Usage and Performance Changes by Applying Service Feedback Metrics to Software Maintenance. ASEW'08.

[SM96] Marcela Genero, Mario Piattini, Esperanza Manso, Giovanni Cantone. Building UML Class Diagram Maintainability Prediction Models Based on Early Metrics. METRICS'03.

[SM105] Sami Makela, Ville Leppanen. Client Based Cohesion Metrics for Java Programs. SCP'09.

[SM109] Mikhail Perepletchikov, Caspar Ryan, and Keith Frampton. Cohesion Metrics for Predicting Maintainability of ServiceOriented Software. QSIC'07.

[SM111] Rudiger Lincke, Jonas Lundberg and Welf Lowe. Comparing Software Metrics Tools. ISSTA'08.

[SM112] Misook Choi, Injoo J. Kim, Jiman Hong and Jungyeop Kim. Component-Based Metrics Applying the Strength of Dependency Between Classes. SAC'09.

[SM119] R. Harrison, S. Counsell, R. Nithi. Coupling Metrics for Object-Oriented Design. METRICS'98.

[SM123] M. Ajmal Chaumun, Hind Kabaili, Rudolf K. Keller, Francois Lustman and Guy SaintDenis. Design Properties and Object-Oriented Software Changeability. CSMR'00.

[SM124] Hong Yul Yang , Ewan Tempero and Rebecca Berrigan. Detecting Indirect Coupling. ASWEC'05.

[SM126] Marc Eaddy, Thomas Zimmermann, Kaitlin D. Sherwood, Vibhav Garg, Gail C. Murphy, Nachiappan Nagappan, Alfred V. Aho. Do Crosscutting Concerns Cause Defects? TSE'08.

[SM127] T.H. Ng, S.C. Cheung, W.K. Chan and Y.T. Yu. Do Maintainers Utilize Deployed Design Patterns Effectively? ICSE'07.

[SM128] Varun Gupta, Jitender Kumar Chhabra. Dynamic Cohesion Measures for Object-Oriented Software. JSA'11.

[SM129] Erik Arisholm, Lionel C. Briand, Audun Fyen. Dynamic Coupling Measurement for Object-Oriented Software. TSE'04.

[SM130] Erik Arisholm. Dynamic Coupling Measures for Object-Oriented Software. METRICS'02.

[SM131] Y. Hassoun, S. Counsell and R. Johnson. Dynamic Coupling Metric: Proof of Concept. TSE'05.

[SM132] Sherif M. Yacoub, Hany H. Ammar, and Tom Robinson. Dynamic Metrics for Object-Oriented Designs. METRICS'99.

[SM136] Marcela Genero, Mario Piattini and Coral Calero. Empirical Validation of Class Diagram Metrics. ISESE'02.

[SM138] Fabrizio Fioravanti, Paolo Nesi. Estimation and Prediction Metrics for Adaptive Maintenance Effort. TSE'01.

[SM140] Guadalupe Ortiz, Behzad Bordbar, Juan Hernandez. Evaluating the Use of AOP and MDA in Web Service Development. ICIW'08.

[SM141] R. Harrison, L.G., Samaraweera, M.R., Dobie and P.H. Lewis. Evaluation of Code Metrics for Object-Oriented Programs. IST'96.

[SM144] Fernando Castor Filho, Nelio Cacho, Eduardo Figueiredo, Raquel Maranhao, Alessandro Garcia, Cecilia Mary F. Rubira. Exceptions and Aspects The Devil is in the Details. FSE'06.

[SM146] Lalji Prasad,Aditi Nagar. Experimental Analysis of Different Metrics (Object-Oriented and Structural of Software). CYCSYN'09.

[SM148] Eugen C. Nistor and Andre Van der Hoek. Explicit Concern-Driven Development with ArchEvol. ASE'09.

[SM149] V. Krishnapriya, K. Ramar. Exploring the Difference Between Object-Oriented Class Inheritance and Interfaces Using Coupling Measures. ACE'10.

[SM151] Lionel C. Briand, Jurgen Wu, John W. Daly, D. Victor Porter. Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems. JSS'00.

[SM153] Michael English, Jim Buckley and Tony Cahill. Fine-Grained Software Metrics in Practice. ESEM'07.

[SM156] Horst Zuse. Foundations Of Object-Oriented Software Measures. METRICS'96.

[SM157] Aram Hovsepyan, Riccardo Scandariato, Stefan Van Baelen, Yolande Berbers, and Wouter Joosen. From Aspect-Oriented Models to Aspect-Oriented Code? The Maintenance Perspective. AOSD'10.

[SM158] Maria Teresa Baldassarre, Alessandro Bianchi, Danilo Caivano, and Corrado Aaron Visaggio. Full Reuse Maintenance Process for Reducing Software Degradation. CSMR'03.

[SM161] Avadhesh Kumar, Rajesh Kumar, and P.S. Grover. Generalized Coupling Measure for Aspect-Oriented Systems. SIGSOFT'09.

[SM164] Bruno Stiglic, Marjan Heri6ko, and Ivan Rozlnan. How to Evaluate Object-Oriented Software Development? SIGPLAN'05.

[SM165] Yuming Zhou, Baowen Xu, Jianjun Zhao, and Hongji Yang. ICBMC: An Improved Cohesion Measure for Classes. ICSM'02.

[SM166] Gabriele Bavota, Andrea De Lucia, and Rocco Oliveto. Identifying Extract Class Refactoring Opportunities using Structural and Semantic Cohesion Measures. JSS'10.

[SM169] Marc Eaddy, Alfred Aho, and Gail C. Murphy. Identifying, Assigning, and Quantifying Crosscutting Concerns. Acom'07.

[SM174] M. M. Lehman, D. E. Perry, and J. F. Ramil. Implications of Evolution Metrics on Software Maintenance. ICSM'08.

[SM176] Heung Seok Chae, Yong Rae Kwon, and Doo Hwan Bae. Improving Cohesion Metrics for Classes by Considering Dependent Instance Variables. TSE'04.

[SM177] Mario Luca Bernardi and Giuseppe Antonio Di Lucca. Improving Design Pattern Quality using Aspect-Orientation. STEP'05.

[SM178] Nelio Cacho, Thais Batista, Alessandro Garcia, Claudio Santanna, and Gordon Blair. Improving Modularity of Reflective Middleware with Aspect-Oriented Programming. SEM'06.

[SM181] Jehad Al Dallal. Improving the Applicability of Object-Oriented Class Cohesion Metrics. INFSOF'11.

[SM182] Charles H. House. Information Worker Tools Selection, Adoption and Evaluation Lessons From Software Development History. ICSS'05

[SM187] Mathupayas Thongmak and Pornsiri Muenchaisri. Maintainability Metrics for Aspect-Oriented Software. IJSEKE'09.

[SM189] Shyam R. Chidamber, David P. Darcy, and Chris F. Kemerer. Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis. TSE'98.

[SM193] Lionel C. Briand, Sandro Morasca, and Victor R. Basili. Measuring and Assessing Maintainability at the End of High Level Design. TSE'93.

[SM194] Ahrim Han, Sanguk Jeon, Doohwan Bae, and Jangeui Hong. Measuring Behavioral Dependency for Improving Change Proneness Prediction in UML Based Design Models. JSS'10.

[SM195] P.S. Grover, Rajesh Kumar, and Avadhesh Kumar. Measuring Changeability for Generic Aspect-Oriented Systems. SIGSOFT'08.

[SM198] Tobias Mayer and Tracy Hall. Measuring OO Systems: A Critical Analysis of the MOOD Metrics. TOOLS'99.

[SM199] Tianlin Zhou, Baowen Xu, Liang Shi, Yuming Zhou, and Lin Chen. Measuring Package Cohesion Based on Context. WSCS'08.

[SM200] G. Manduchi, C. Taliercio. Measuring Software Evolution at a Nuclear Fusion Experiment Site: A Test Case for the Applicability of OO. INFSOF'02.

[SM201] Philippe Lithiaott, Jessie Kennedy, and John Owens. Mechanisms for Interpretation Of OO Systems Design Metrics. TSE'98.

[SM202] P. Nesi and M. Campanai. Mendel: A Model, Metrics, and Rules to Understand Class Hierarchies. ICPC'08.

[SM203] P. Nesi and M. Campanai. Metric Framework for Object-Oriented Real Time Systems Specification Languages. JSS'96.

[SM204] G. Bucci, F. Fioravanti, P. Nesi, and S. Perlini. Metrics and Tool for System Assessment. ICECCS'98.

[SM205] N. Debnath, L. Baigorria, D. Riesco, and G. Montejano. Metrics Applied to Aspect-Oriented Design Using UML Profiles. ISCC'08.

[SM208] F. Fioravanti, P. Nesi, and F. Stortoni. Metrics for Controlling Effort during Adaptive Maintenance of Object-Oriented Systems. ICSM'99.

[SM209] Tassia A. V. Freitas, Thais V. Batista, Flavia C. Delicato, and Paulo F. Pires. Metrics for Evaluation of Aspect-Oriented Middleware. SBES'09.

[SM210] Frederick T. Sheldon, Kshamta Jerath, and Hong Chung. Metrics for Maintainability of Class Inheritance Hierarchies. SMR'02.

[SM211] Santonu Sarkar, Avinash C. Kak, and Girish Maskeri Rama. Metrics for Measuring the Quality of Modularization of Largescale Object-Oriented Software. TSE'08.

[SM212] Christof Ebert and Ivan Morschelb. Metrics for Quality Analysis and Improvement of Object-Oriented Software. IST'97.

[SM217] Shen Zhang, Yongji Wang, and Junchao Xiao. Mining Individual Performance Indicators in Collaborative Development using Software Repositories. APSEC'08.

[SM219] Yixun Liu, Denys Poshyvanyk, Rudolf Ferenc, Tibor Gyimothy, and Nikos Chrisochoides. Modeling Class Cohesion as Mixtures of Latent Topics. ICSM'09.

[SM222] Dapeng Liu and Shaochun Xu. New Quality Metrics for Object-Oriented Programs. SNPD'07.

[SM224] Sahar R. Ragab and Hany H. Ammar. Object-Oriented Design Metrics And Tools: A Survey. INFOS'10.

[SM226] Brian Keith Miller, Dr. Pei Hsia, and Dr. Chenho Kung. Object-Oriented Architecture Measures. TSE'99.

[SM230] David Bellin, Manish Tyagi, and Maurice Tyler. Object-Oriented Metrics: An Overview. CASCON'94.

[SM234] Bart Van Rompaey, Bart Du Bois, Serge Demeyer, and Matthias Rieger. On the Detection of Test Smells: A Metrics Based Approach for General Fixture and Eager Test. TSE'07.

[SM236] Eduardo Figueiredo, Claudio Sant'Anna, Alessandro Garcia, Thiago T. Bartolomei, Walter Cazzola, and Alessandro Marchetto. On the Maintainability of Aspect-Oriented Software: A Concern Oriented Measurement Framework. TSE'08.

[SM256] K. Liu, S. Zhou and H. Yang. Quality Metrics of Object Oriented Design for Software Development and Re-Development. ASWEC'08.

[SM264] Michal Hocko and Tomas Kalibera. Reducing Performance Non-determinism via Cache aware Page Allocation Strategies. WOSP'10.

[SM266] Yoshiki Higo, Yoshihiro Matsumoto, Shinji Kusumoto, and Katsuro Inoue. Refactoring Effect Estimation Based on Complexity Metrics. ASWEC'08.

[SM267] Mohammad Alshayeb. Refactoring Effect on Cohesion Metrics. ICC'09.

[SM269] Chandrashekar Rajaraman and Michael R. Lyu. Reliability And Maintainability related Software Coupling Metrics in C++ Programs. TSE'92.

[SM272] Young Lee and Kai H. Chang. Reusability and Maintainability Metrics for Object-Oriented Software. ACM-SE'00.

[SM275] Landry Chouambe, Benjamin Klatt, and Klaus Krogmann. Reverse Engineering Software Models of Component Based. TSE'08.

[SM277] Gyun Woo, Heung Seok Chae , Jian Feng Cui, and Jeonghoon Ji. Revising Cohesion Measures by Considering the Impact of Write Interactions Between Class Members. INFSOF'08.

[SM282] Bandar Alshammari, Colin Fidge and Diane Corney. Security Metrics for Object-Oriented Class Designs. QSIC'09.

[SM288] Raghu V. Hudli, Curtis L. Hoskins, and Anand V. Hudli. Software Metrics for Object-Oriented Designs. TSE'94.

[SM290] Marcio F. S.Oliveira, Ricardo Miotto Redin, Luigi Carro, Luis Da Cunha Lamb, and Flavio Rech Wagner. Software Quality Metrics and their Impact on Embedded Software. MOMPES'08.

[SM291] William Frakes and Carol Terry. Software Reuse: Metrics and Models. CSUR'96.

[SM293] V. Lakshmi Narasimhan and B. Hendradjaya. Some Theoretical Considerations for a Suite of Metrics for the Integration of Software Components. IS'07.

[SM296] Taku Fujii and Yahiko Kambayashi. Strategies to Suppress Productivity Degradation with Unknown Issues under Iterative Development Process. CW'02.

[SM301] Miro Casanova, Ragnhild Van Der Straeten, and Viviane Jonckers. Supporting Evolution in Component Based Development Using Component Libraries. CSMR'07.

[SM305] Arpad Besze, Tama Gergely, Szabolcs Farago, Tibor Gyimothy, and Ferenc Fischer. The Dynamic Function Coupling Metric and its Use in Software Evolution. CSMR'07.

[SM307] Steve Counsell, Stephen Swift. The Measurement and Evaluation for Large-Scale Object-Oriented Software System. TOSEM'06.

[SM309] Zhu Mingguang, Zhang Haohua, Qi Weiyi, Ma Shijun and Wang Chuanyin. The Interpretation and Utility of Three Cohesion Metrics for Object-Oriented Design. HIS'09.

[SM313] Robert C. Sharblet and Samuel S. Cohen. The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. SIGSOFT'93.

[SM316] Mikael Lindvall and Magnus Runesson. The Visibility of Maintenance in Object Models: An Empirical Study. ICSM'98.

[SM319] Reiner R. Diiinke and Ines Kuhrau. Tool Based Quality Management in Object-Oriented Software Development. TSE'94.

[SM320] Jan Wloka, Robert Hirschfeld, and Joachim Hansel. Tool Supported Refactoring of Aspect-Oriented Programs. AOSD'08.

[SM323] Shmuel Rotenstreich. Toward Measuring Potential Coupling. TSE'94.

[SM325] Letha Etzkom, Harry Delugach. Towards a Semantic Metrics Suite for Object-Oriented Design. TSE'00.

[SM327] Cristina Marinescu, Radu Marinescu, and Tudor Grba. Towards a Simplified Implementation of Object-Oriented Design Metrics. METRICS'05.

[SM329] Thiago T. Bartolomei, Alessandro Garcia, Claudio Santanna, and Eduardo Figueiredo. Towards a Unified Coupling Framework for Measuring Aspect-Oriented Programs. SOQUA'06.

[SM330] Avadhesh Kumar, Rajesh Kumar, and P.S. Grover. Towards a Unified Framework for Cohesion Measurement in Aspect-Oriented Systems. ASWEC'08.

[SM331] Avadhesh Kumar, Rajesh Kumar, and P.S. Grover. Towards a Unified Framework for Complexity Measurement in Aspect-Oriented Systems. CSSE'08.

[SM335] Jehad Al Dallal. Transitive based Object-Oriented Lack of Cohesion Metric. WCIT'10.

[SM337] Moheb R. Girgis, Tarek. M. Mahmoud, and Rehab R. Nour. UML Class Diagram Metrics Tool. TSE'09.

[SM341] Lionel C. Briand, Jurgen Wust, and Hakim Lounis. Using Coupling Measurement for Impact Analysis in Object-Oriented Systems. ICSM'99.

[SM343] Malcom Gethers and Denys Poshyvanyk. Using Relational Topic Models to Capture Coupling among Classes in Object-Oriented Software Systems. ICSM'10.

[SM345] Andrea De Lucia, Rocco Oliveto, and Luigi Vorraro. Using Structural and Semantic Metrics to Improve Class Cohesion. TSE'11.

[SM346] Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Using Structural and Textual Information to Capture Feature Coupling in Object-Oriented Software. ESE'11.

[SM347] Mr. U. L. Kulkarni, Mr. Y. R. Kalshetty, and Ms. Vrushali G. Arde. Validation of CK Metrics for Object-Oriented Design Measurement. ICETEC'10.

[SM350] Gunter Mussbacher, Daniel Amyot, Joao Araujo, Ana Moreira, and Michael Weiss. Visualizing Aspect-Oriented Goal Models with Aogrl. REV'07.

D

Object-Oriented Software Maintainability Metrics

Table D.1: OO Maintainability Metrics

Metric	Description	Primary Studies that contain the metric
CBO	Coupling between Objects Classes	SM4, SM8, SM13, SM29, SM38, SM42, SM47, SM48, SM54, SM70, SM73, SM75, SM76, SM87, SM92, SM111, SM119, SM129, SM132, SM146, SM149, SM151, SM164, SM166, SM189, SM194, SM200, SM205, SM209, SM210, SM211, SM217, SM222, SM266, SM301, SM313, SM319, SM329, SM341, SM343, SM346, SM347, SM156
DIT	Depth of Inheritance Tree	SM123, SM4, SM7, SM13, SM31, SM48, SM49, SM54, SM69, SM70, SM75, SM76, SM87, SM96, SM132, SM144, SM164, SM177, SM187, SM189, SM194, SM200, SM201, SM204, SM205, SM209, SM210, SM217, SM222, SM22, SM266, SM290, SM313, SM319, SM325, SM347, SM39, SM42, SM66
LCOM	Lack of Cohesion in Methods	SM1, SM3, SM7, SM13, SM35, SM37, SM39, SM48, SM54, SM69, SM70, SM75, SM76, SM87, SM105, SM111, SM123, SM128, SM146, SM164, SM165, SM189, SM194, SM200, SM210, SM217, SM222, SM226, SM66, SM290, SM307, SM313, SM319, SM325, SM330, SM331, SM335, SM345, SM347
NOC	Number of Children of a Class	SM4, SM7, SM13, SM31, SM35, SM39, SM42, SM48, SM49, SM53, SM54, SM66, SM69, SM70, SM75, SM76, SM79, SM87, SM132, SM156, SM164, SM187, SM189, SM194, SM200, SM201, SM204, SM205, SM209, SM210, SM217, SM222, SM224, SM266, SM290, SM313, SM319, SM325, SM347
RFC	Response For a Class	SM4, SM13, SM38, SM39, SM47, SM48, SM54, SM69, SM70, SM73, SM75, SM76, SM87, SM111, SM123, SM129, SM132, SM146, SM156, SM164, SM189, SM194, SM200, SM205, SM209, SM210, SM217, SM222, SM266, SM313, SM319, SM325, , SM329, SM343, SM346, SM347
WMC	Weighted Methods Per Class	SM7, SM8, SM31, SM35, SM39, SM48, SM54, SM69, SM70, SM75, SM87, SM111, SM123, SM132, SM138, SM146, SM156, SM164, SM189, SM194, SM200, SM201, SM204, SM210, SM217, SM222, SM266, SM290, SM313, SM319, SM325, SM337, SM347
LOC	Lines of Code	SM7, SM10, SM23, SM47, SM49, SM74, SM111, SM138, SM140, SM144, SM146, SM156, SM176, SM195, SM203, SM208, SM109, SM217, SM219, SM236, SM269, SM275, SM293, SM323, SM327
MPC	Message Passing Coupling	SM13, SM38, SM54, SM66, SM73, SM76, SM132, SM146, SM151, SM166, SM209, SM301, SM319, SM341, SM343, SM346, SM234
CC	Class Coupling	SM7, SM23, SM47, SM54, SM69, SM86, SM138, SM181, SM198, SM199, SM203, SM204, SM208, SM269, SM288, SM335

Table D.1: OO Maintainability Metrics

Metric	Description	Primary Studies that contain the metric
DAC	Data Abstraction Coupling	SM4, SM13, SM31, SM35, SM54, SM76, SM129, SM146, SM151, SM209, SM319, SM341, SM343, SM346
TCC	Tight Class Cohesion	SM3, SM31, SM37, SM68, SM76, SM105, SM111, SM146, SM176, SM181, SM219, SM277, SM335
LCC	Loose Class Cohesion	SM3, SM37, SM68, SM76, SM181, SM209, SM219, SM236, SM330, SM331
LCOM1	Lack of Cohesion in Methods 1	SM3, SM7, SM68, SM176, SM181, SM219, SM224, SM264, SM267, SM277
LCOM2	Lack of Cohesion in Methods 2	SM3, SM37, SM68, SM176, SM182, SM219, SM224, SM267, SM277
CDC	Concern Diffusion over Components	SM126, SM140, SM144, SM169, SM178, SM209, SM236, SM325
LCOM3	Lack of Cohesion in Methods 3	SM3, SM37, SM68, SM176, SM181, ,SM219, SM267, SM277
NA	Number of Attributes	SM10, SM96, SM136, SM144, SM203, SM204, SM205, SM208
NM	Number of Methods	SM70, SM82, SM96, SM136, SM204, SM224, SM230, SM288
NOM	Number of Methods	SM15, SM35, SM54, SM76, SM111, SM146, SM290, SM319
CMC	Coupling on Method Call	SM47, SM49, SM53, SM161, SM177, SM187, SM325
LCOM5	Lack of Cohesion in Methods 5	SM3, SM37, SM176, SM181, SM219, SM267, SM277
NC	Number of Classes	SM96, SM203, SM82, SM226, SM230, SM288, SM136
CA	Afferent Coupling	SM7, SM38, SM47, SM290, SM323, SM343
COF	Coupling Factor	SM11, SM38, SM198, SM211, SM343, SM346
ICP	Information Flow-Based Coupling	SM38, SM132, SM151, SM341, SM343, SM346
LCOM4	Lack of Cohesion in Methods 4	SM3, SM37, SM176, SM181, SM2667, SM277
AC	Ancestors Count	SM38, SM198, SM203, SM211, SM288
ACAIC	Class-Attribute Interaction Between Classes	SM38, SM129, SM151, SM341, SM343
ACMIC	Class-Method Interaction Between Classes	SM38, SM129, SM151, SM341, SM343
AHF	Attribute Hiding Factor	SM48, SM79, SM76, SM198, SM211
AMMIC	Method-Method Interaction Between Classes	SM73, SM151, SM341, SM129, SM153
CAMC	Cohesion Among Methods in a Class	SM68, SM181, SM211, SM307, SM335
CE	Efferent Coupling	SM7, SM38, SM47, SM290, SM346
COH	Cohesion	SM3, SM68, SM181, SM219, SM335
LCO	Lack of Cohesion in Operations	SM49, SM87, SM177, SM187, SM204
MHF	Method Hiding Factor	SM48, SM70, SM76, SM198, SM211
MI	Methods that are Inherited	SM48, SM55, SM68, SM198, SM234
MIF	Method Inheritance Factor	SM48, SM70, SM76, SM194, SM198
NMI	Number of Methods Inherited	SM55, SM70, SM129, SM138, SM204
OMMIC	Method-Method interaction between classes	SM73, SM129, SM151, SM153, SM341
PIM	Permitted Interaction Metric	SM4, SM38, SM234, SM341, SM343
TLOC	Total Number of Lines of Code	SM7, SM138, SM208, SM217, SM290
A	Number of Attributes	SM38, SM234, SM275, SM290
C	Coupling	SM37, SM38, SM275, SM323
CBM	Coupling Between Modules	SM47, SM49, SM92, SM177
CF	Coupling Factor	SM70, SM76, SM146, SM211
CI	Classes Inherited	SM55, SM138, SM204, SM208
CL	Class Locality	SM138, SM194, SM204, SM208
CM	Code Modified	SM10, SM38, SM282, SM327
CO	Classes that Override Something	SM55, SM69, SM176, SM277
EC	Estimated Cost	SM7, SM38, SM174, SM193
NHD	Normalized Hamming Distance	SM68, SM181, SM211, SM307, SM335
NOP	Number of Parents	SM7, SM15, SM224, SM290
OCAIC	Class-Attribute interaction between classes	SM129, SM151, SM341, SM343
OCMIC	Class-Method interaction between classes	SM38, SM129, SM151, SM341
RFM	Response For a Module	SM49, SM53, SM161, SM329
WOM	Weighted Operations in Module	SM47, SM49, SM177, SM187
C3	Cohesion Metric	SM166, SM219, SM345
CACI	Class Attribute Complexity Inherited	SM138, SM204, SM208
CBC	Coupling Between Components	SM112, SM144, SM288
CBMC	Coupling Between Module Classes	SM92, SM165, SM176
CLC	Class Level Coupling	SM38, SM132, SM203
COA	Classified Operation Accessibility	SM84, SM282, SM325
CR	Comment Ration	SM7, SM55, SM203
CSM	Conceptual Similarity between Methods	SM345, SM346, SM166
ECD	External Class Description	SM138, SM203, SM204
FCAEC	Friends from CA-Interactions	SM38, SM151, SM343
IFCAIC	Inverse Friend from CA-Interactions	SM38, SM151, SM343
IH_ICP	Information Flow-Based Inheritance Coupling	SM151, SM341, SM346
NAC	Number of Ancestors	SM226, SM325, SM210

Table D.1: OO Maintainability Metrics

Metric	Description	Primary Studies that contain the metric
NAGG	Number of Aggregation	SM96, SM136, SM337
NAGGH	Number of Aggregation Hierarchies	SM96, SM136, SM337
NCL	Number of Classes	SM203, SM204, SM208
NCO	Number of Comparison Operators	SM85, SM111, SM217
NDC	Number of Direct Connections	SM3, SM210, SM325
NGEN	Number of Generalizations	SM96, SM337, SM136
NML	Number of Methods Locally	SM138, SM204, SM208
NMO	Number of Methods Overridden	SM55, SM70, SM76
NOA	Number of Ancestors	SM146, SM209, SM290
NOT	Number of Tramps	SM211, SM313, SM319
PF	Polymorphism Factor	SM70, SM76, SM194
RCI	Ration of Cohesion Interactions	SM37, SM1, SM165
SLOC	Source Lines of Code	SM169, SM189, SM288
SNHD	Scaled NHD	SM68, SM307, SM335
TC	Total Children	SM48, SM69, SM327
VG	Cyclomatic Complexity	SM7, SM132, SM146
VS	Vocabulary Size	SM144, SM209, SM350
ADT	Number of Abstract Data Type	SM4, SM69
AM	Average Degree of Maintainability	SM187, SM210
CAAI	Classified Accessor Attribute Interactions	SM84, SM282
CAC	Class Attribut Complexity	SM203, SM226
CACL	Class Attribute Complexity Local	SM138, SM204
CAIW	Classified Attributes Interaction Weight	SM84, SM282
CAM	Cohesion Among Methods of Class	SM15, SM53
CBO'	Coupling between Objects Classes	SM151, SM341
CCC	Class Clause Complexity	SM84, SM203
CCDA	Classified Class Data Accessibility	SM84, SM282
CCN	Cyclomatic Complexity Number	SM140, SM217
CDBC	Change Dependency Between Classes	SM7, SM31
CDM	Call-based Dependence between Methods	SM166, SM74
CID	Interaction Density of a Component	SM53, SM293
CIDA	Classified Instance Data Accessibility	SM84, SM282
CIS	Class Interface Size	SM15, SM301
CMAI	Classified Mutator Attribute Interactions	SM84, SM282
CMICI	Class Method Interface Complexity Inherited	SM138, SM208
CMW	Classified Methods Weight	SM84, SM282
COC	Clients Of Class	SM86, SM194
COCC	Conceptual Coupling	SM343, SM346
CPC	Class path complexity	SM203, SM112
CSI	Critical Superclass Inherited	SM84, SM291
CTA	Coupling Through Abstract Data Types	SM54, SM325
CTM	Coupling Through Message Passing	SM54, SM325
DAC'	Data Abstraction Coupling	SM151, SM341
DC	Descendants Count	SM48, SM198
DCC	Direct Class Coupling	SM15, SM301
DCM	Dynamic Coupling Metric	SM13, SM131
DD	Data Declarations	SM193, SM320
DI	Depth of Inheritance	SM7, SM288
DN	Normalized Distance from Main Sequence	SM85, SM290
DOS	Degree of scattering	SM169, SM126
DS	Data Declaration-Subprogram	SM193, SM288
EC_CC	Export Coupling Distinct Class	SM129, SM130
EC_CD	Export Coupling Dynamic Class	SM129, SM130
EC_CM	Export Coupling Distinct Method	SM129, SM130
EC_OD	Export Coupling Dynamic Message	SM129, SM130
EC_OM	Export Coupling Distinct Method	SM129, SM130
ECC	External Class Complexity	SM199, SM203
ECDI	External Class Description Inherited	SM138, SM204
ECDL	External Class Description Local	SM138, SM204
HAGG	Height of class within aggregation	SM96, SM337
IC	Import Coupling	SM193, SM211
IC_CC	Import Coupling Distinct Class	SM129, SM130
IC_OC	Import Coupling Distinct Class	SM129, SM130
IC_OD	Import Coupling Dynamic Message	SM129, SM130
IC_OM	Import Coupling Distinct Method	SM129, SM130
ICH	Information Based Cohesion	SM37, SM76
ICI	Internal Class Implementation	SM138, SM204
IFMMIC	Interaction Friends from Call Method Interactions from Class	SM38, SM151, SM153
IM	Integration and testing	SM10, SM288
LCOO	Lack of Cohesion in Operations	SM330, SM331, SM219
LOCC	Lines of Class Code	SM126, SM187
LORM	Logical Relatedness of Methods	SM219, SM325
MAXDIT	Maximum DIT	SM136, SM337

Table D.1: OO Maintainability Metrics

Metric	Description	Primary Studies that contain the metric
MM	Method-Method Interactions	SM38, SM132
NAI	Number of Attributes	SM204, SM224
NAL	Number of Attributes Locally	SM138, SM204
NAM	Number of Methods Locally	SM138, SM204
NAMI	Number of Attributes and Methods Locally	SM138, SM204
NAS	Number of Class Associations	SM119, SM132
NASSCO	Number of Associations	SM136, SM337
NDEP	Number of Dependencies	SM96, SM337
NDEPIN	Number of Dependencies In	SM149, SM337
NDEPOUT	Number of Dependencies Out	SM149, SM337
NES	Number of Executable Semicolon	SM35, SM85
NF	Number of times a Class is Reused	SM70, SM86
NGENH	Number of Generalizations Hierarchs	SM96, SM136
NIH	Non-Inheritance Coupling	SM151, SM341
NIH_ICP	Information Flow-Based Non-Inheritance Coupling	SM151, SM341
NLM	Number of Local Methods	SM54, SM325
NMA	Number of Methods Added	SM151, SM70
NO	Number of Operations	SM35, SM144
NOI	Number of Inherited Methods	SM75, SM290
NORM	Number of Remote Methods	SM7, SM194
NP	Number of Public Methods	SM3, SM337
NRCI	Neutral Ratio of Cohesive Interactions	SM37, SM193
OLC	Object Level Coupling	SM38, SM132
ORCI	Optimistic Ratio of Cohesive Interactions	SM37, SM193
PM	Number of Public Methods	SM70, SM288
POF	Polymorphism Factor	SM48, SM198
PPRIVM	Percentage of Private Members	SM7, SM194
PPUBM	Percentage of Public Members	SM7, SM194
PRCI	Pessimistic Ratio of Cohesive Interactions	SM37, SM193
RC	Relational Cohesion	SM199, SM330
RTC	Relational Topic based Coupling	SM343, SM346
SCOM	Sensitive Class Cohesion Metric	SM181, SM335
SSM	Structural Similarity between Methods	SM166, SM345
TNM	Total (System) Number of Methods	SM208, SM138
V	Volume	SM48, SM269
VC	Visibility Control	SM193, SM277
VOD	Violations of the Law of Demeter	SM313, SM319
W	Weight of each edge	SM166, SM277
WAC	Weighted Attributes per Class	SM313, SM319
A-EC	Exporting Coupling of a Module	SM193
A-IC	Import Coupling of a module	SM193
AA	Assessment and Assimilation	SM10
AACD	Average Active Component Density	SM293
AAD	Attribute Access Dependencies	SM82
ACC	Attributes and Methods Cohesion	SM68
ACD	Active Component Density	SM293
ACHS	Average Cohesion of a System	SM112
ACPS	Average Coupling of a System	SM112
ACTORS	Actors	SM230
ADI	Average Depth of Inheritance	SM288
AFIU	Average Fixture Usage	SM234
AID	Average Inheritance Depth of a Class	SM151
AIF	Attribute Inheritance Factor	SM48
AIM	All Inherited Methods	SM75
ALV	Available Local Variable	SM23
AMC	Average Method Coupling	SM269
AMC_LOC	Lines Of Code of Average Method Coupling	SM269
AMS	Average Method Size	SM70
ANA	Average Number of Ancestors	SM15
ANAC	Average Number of Active Components	SM293
APIU API	Function Usage Index	SM74
APIUC	API Unit Cohesion	SM211
APPM	Average Parameters Per Method	SM337
AT	Attribute Type	SM68
ATTRINH	Number of inherited attributes	SM224
AVNME	Average Number of Model Elements	SM350
BC	Base Classes	SM288
BCV	Back Call Violation	SM82
BMS	Budding Classes	SM202
BRKZ	Brokerage	SM87
CAED	Clients Access the Encapsulated Data	SM288
CALOC	Cumulative Added LOC	SM296
CAS	Class Attribute Size	SM203
CBMU	Coupling between Model Units	SM350

Table D.1: OO Maintainability Metrics

Metric	Description	Primary Studies that contain the metric
CBO_IUB	CBO Is Used By	SM123
CBO_NA	CBO No Ancestors	SM123
CBO_U	CBO Using	SM123
CBO(d)	Coupling Between Objects	SM66
CBO(f)	Coupling Between Objects	SM66
CBOIUB	Coupling Between Objects is Used By	SM73
CBONA	Coupling Between Objects No Ancestors	SM73
CBOU	Coupling Between Objects Using	SM73
CC_LOC	Lines of Code of Class Coupling	SM269
CCBC	Conceptual Coupling Between Components	SM343
CCBO	Conceptual Coupling between Object Classes	SM346
CCM	Concept Coherency Metric	SM74
CCOF	Component Coupling Factor	SM11
CCPC	Class Coupling Path Complexity	SM203
CCV	Cyclic Call Violations	SM82
CDD	Class Dynamic Description	SM203
CDL	Class To Leaf Depth	SM151
CH	Cohesion	SM193
CHC	Cohesion of a Component	SM112
CHNL	Class Hierarchy nesting level	SM4
CIC	Class Inheritance-related Coupling	SM269
CII	Class Implementation Instability	SM54
CIID	Component Incoming Interaction Density	SM293
CINT	Class Intersection	SM156
CITC	Class Internal Task Complexity	SM156
CMA	Company Dependent Coefficient	SM203
CMD	Coupling Dependecy Metric	SM4
CME	Coupling Methods Existing	SM84
CMI	Coupling Method Inherited	SM84
CMICL	Class Method Interface Complexity/size Local	SM208
CMT	Comments	SM35
CN	Classes with No implementations replaced	SM55
CNIC	ClassNon-Inheritance-related Coupling	SM269
COB	Class Overlap B	SM325
COID	Component Outgoing Interaction Density	SM293
COMP	Average number of internal relationships per class/interface	SM224
CONNECTORS	Connectors	SM224
COUPLING	COUPLING	SM212
CPCC	Critical Composed-Part Classes	SM84
CPD	Component Packing Density	SM293
CS	Class Size	SM203
CSA	Classes	SM35
CSC	Concern Sensitive Coupling	SM236
CSD	Class Static Description	SM203
CSNS	Closeness	SM87
CSO	Class Operations	SM35
CSP	Critical Super Class	SM84
CTC	Cross-Tree Constraints	SM86
CUNI	Class Unification	SM156
CyC	Cyclomatic Complexity	SM7
DAG	Ddirected Acyclic Graph	SM210
DAM	Data Access Metric	SM282
DAT	Direct Attribute Type	SM68
DC	Degree of Cohesion-Direct	SM335
DC_AAX	Dynamic Cohesion due to Reference dependency between attributes	SM128
DC_AMX	Dynamic Cohesion due to Write dependency of Attributes on Methods	SM128
DC_MA	Dynamic Cohesion due to Read dependency of Methods on	SM128
DC_MMX	Dynamic Cohesion due to Call dependency between Methods	SM128
DCAE	Descendants from CA-Interactions	SM38
DCD	Degree of Cohesion	SM181
DD-INTERACTIONS	Data declaration-Data declaration (OD)Interaction.	SM193
DEP_IN	Dependency In	SM224
DEP_OUT	Dependency Out	SM224
DFC	Dynamic Function Coupling	SM305
DIH	Depth of Inheritance	SM69
DIST	Distance Between Features	SM346
DM	Design Modified	SM10
DMC	Dependence Matrix-based Cohesion	SM68

Table D.1: OO Maintainability Metrics

Metric	Description	Primary Studies that contain the metric
DMI	Direct Method Invocation	SM68
DMS	Distance from the Main Sequence	SM275
DOCX	Dynamic Object Cohesion	SM128
DOIH	Degree of Inheritance	SM7
DR	Discarded Rate of Operations	SM296
DSC	Design in Size Class	SM15
DT	Depth of Tree	SM86
DWRH dw	Reach	SM87
EC_ATTR	Externally Class Attribute	SM224
EC_OC	Export Coupling Distinct Class	SM130
EC_PAR	Externally Class Parameter	SM224
ECF	Environment Complexity Factor	SM7
ECS	External Class Size	SM203
EFFSZ	Effective Size	SM87
EHD	Exception Handling Dependencies	SM82
EOC	Export Object Coupling	SM132
EV	Encapsulated Variables	SM288
EWE	Estimated Work Effort	SM7
F-MEASURE	F-MEASURE	SM166
FAN-IN	FAN-IN	SM87
FAN-OUT	FAN-OUT	SM87
FCR	Functionalities Change Rate	SM296
FOC	Flexibility of Configuration	SM86
GLOBAL	Global	SM193
H	Horizontal coupling	SM224
HB	Hierarchy Brittleness	SM226
HC	Horizontal coupling	SM23
HFC	Hybrid Feature Coupling	SM346
HIH	Height of the inheritance Hierarchy	SM4
HLD	High-Level Design	SM335
HM	Helper methods	SM288
IC_ATTR	Interface Attribute	SM224
IC_CD	Import Coupling Dynamic Message	SM129
IC_CM	Import Coupling Distinct Method	SM129
IC_PAR	Interface Parameter	SM224
ICBM	Improved Cohesion Measure for Classes	SM165
ICC	Internal Class Complexity	SM203
ID	Inheritance Dependencies	SM82
IDI	Implicit Dependency Index	SM74
IDM	Inherited Data Members	SM288
IDS	Independence Degree of a System	SM112
IHC	Information Flow Based Cohesion	SM219
IIF	Internal Inheritance Factor	SM76
INAG	Indirect Aggregation	SM341
INCCS	Increment of Class Size	SM203
INFCY	Information centrality	SM87
Inheritance Coupling	Inheritance Coupling	SM29
Inheritance Tree	Inheritance Tree	SM212
Invocation Coupling	Invocation Coupling	SM29
IOC	Import Object Coupling	SM132
IP	Implementation Productivity	SM296
IPTU	Indirect Production-Type Uses	SM234
IV	Interface Violation	SM275
KCI	key Class Identity	SM325
KE	known Errors	SM141
KLOC K	Lines of Code	SM166
LA	Local Attributes	SM69
LBC	Lines of Block Comment	SM217
LCIC	Lack of Coherence In Clients	SM105
LCOKME	Lack of Cohesion in Key Model Elements	SM350
LCOMA	Lack of Cohesion in Methods	SM156
LCOMB	Lack of Cohesion in Methods	SM156
LCSM	Lack of Conceptual Cohesion in Methods	SM219
LLD	Low-Level Design	SM335
LO	Local Operations	SM69
LOCS	Lines of Class Code	SM202
LOD	Lack of Documentation	SM31
LSCC	LLD Similarity-based Class Cohesion	SM181
LSI	Latent Semantic Indexing	SM346
M	Method	SM38

Table D.1: OO Maintainability Metrics

Metric	Description	Primary Studies that contain the metric
MA	Methods Available	SM198
MANC	Methods inherited from Ancestor	SM211
MAXHAGG	Maximum HAgg	SM136
MCC	Mean value of Class Complexity	SM208
MCCABECC	McCabe Cyclomatic Complexity	SM290
MCD	Method Call Dependencies	SM82
MDR	Multiple Descendant Redefinition	SM201
Methods Structure	Methods Structure	SM212
MFA	Measure of Functional Abstraction	SM15
MII	Module Interaction Index	SM74
MISI	Module Interaction Stability Index	SM74
MLOC	Method Lines of Code	SM290
MLOC	New Method Lines Of Code	SM7
MMAC	Method?Method through Attributes Cohesion	SM68
MMI	Method?Method Interaction	SM181
MMIC	Method?Method Invocation Cohesion	SM68
MMRE	Model performance measures	SM10
MN	Methods (New)	SM198
MN	Methods with No implementations replaced	SM55
MNA	Mean Number of Class Attributes	SM208
MNOL	Maximum Number of Kevels	SM194
MO	Method Overridden	SM198
MO	Methods that are Overriding	SM55
MOA	Measure of Aggregation	SM15
MPUB	Method Public	SM234
MR	Modifications Requested	SM141
MR	Methods that have implementations Replaced	SM55
MSGC	Message Complexity	SM203
MSGRECV	Message Received	SM224
MSGSELF	Messages Self	SM224
MSGSENT	Messages Sent	SM224
MSOO	Maximum Number of Levels	SM194
MWE	Maximal Weighted Entropy	SM219
n1	Number of unique operators	SM269
N1	Total number of operators	SM269
n2	Number of unique operands	SM269
N2	Total number of operands	SM269
NAD	Number of Advices	SM82
NAML	Number of Methods Locally and Inherited	SM138
NAN	Number of Attributes referred through Navigations	SM85
NAR	Number of Autonomous Requests in the class	SM203
NAS	Number of Aspects	SM82
NASC	Number of Autonomous Requests	SM203
NASSCOC	Number of Association	SM96
NASSOC	Number of Associations	SM337
NASSOCC	Number of Associations Between Classes	SM149
NAV	Number of Assigned Variables	SM23
NB	NB	SM10
NC	Number of Clauses in the class	SM10
NC	Number of Cycle	SM293
NC	Non-API Function Closedness Index	SM74
NCC	Number of Collaboration Classes	SM4
NCI	Number of Classes that Inherit	SM55
NCIM	Number of Classes Inheriting a Given Method	SM4
NCR	Number of times a Class is Reused	SM70
NCSL	Non Comment Source Line	SM119
NEI	Number of Explicit Iterator	SM8
NEV	Number of Non-encapsulated Variables	SM288
NFOB	Number of Fixture Objects	SM234
NFPT	Number of Fixture Production Type	SM234
NIC	Number of Indirect Connections	SM3
NKW	Number of OCL Key Words	SM85
NL	Nesting Level	SM226
NLEAF	Number of Leaf Features	SM86
NLO	Number of Local Methods	SM75
NLOC	Net LOC	SM296
NLOC	Number of Lines of Code	SM92
NMC	Number of Methods	SM55
NME	Number of Model Elements	SM350
NMES	Number of Messages	SM230
NMIC	Number of Methods Inherited	SM55
NMNPUB	Number of Non-Public Methods	SM151
NMPUB	Number of Public Methods	SM224

Table D.1: OO Maintainability Metrics

Metric	Description	Primary Studies that contain the metric
NNBOC	Number of Non Basic Object Classes	SM203
NNC	Number of Navigated Classes	SM85
NNR	Number of Navigated Relationships	SM85
NOAM	Number of added methods	SM194
NOBU	Number of non test OBject Uses	SM234
NOC*	Number Of Children in sub-tree	SM123
NODP	Number of Direct Parts	SM337
NOH	Number of Hierarchies	SM15
NOIS	Number of Import Statements	SM194
NOO	Number of Operations	SM194
NOOC	Number of Object Children	SM7
NOOM	Number of Overridden Methods	SM194
NOPK	Number of Packages	SM290
NOPM	Number of Public Methods	SM187
NOSA	Number of Static Attributes	SM290
NOSLOC	Number of Source Lines of Code	SM288
NOVM	Number of Overridden Methods	SM290
NP	Number of class Paths	SM203
NPI	Number of Polymorphic Invocations	SM38
NPII	Not-Programming-to-Interfaces Index	SM211
NPS	Number of Provided Services	SM203
NPV	Number of Public Variables per class	SM70
NRV	Number of Referred Variables	SM23
NSF	Number of Static Attributes	SM7
NSM	Number of Static Methods	SM7
NSSR	Number of subsystem-subsystem relationships	SM4
NSUB	Number of Subclasses	SM204
NSUP	Number of Superclasses	SM204
NTOP	Number of Top Features	SM86
NUMANC	Number of Ancestors	SM224
NUMATTR	Number of Attributes	SM224
NUMDESC	Number of Descents	SM224
NUMOPS	Number of Operations	SM224
NUMPARA	Number of Parameters	SM224
NUMPUBOP	Number of Public Operations	SM224
NV	Number of Variables per Class	SM70
NVC	Number of Valid Configurations	SM86
NVI	Novelty Index	SM202
NVS	Novelty Score	SM202
NW	Number of Wholes	SM337
NWCP	Normalized Nr. of Weak Components	SM87
OAC	Operation Argument Complexity	SM224
OAM	Operation Access Metric	SM282
OCPX	Object Cyclomatic Complexity	SM132
OL2	Cohesion of Class	SM165
OM	Overridden methods	SM288
OPFS	Object Response for Service	SM132
OPSIHN	Operations Inherited	SM224
OQFS	Object Request for Service	SM132
PC	Parents Count	SM198
PCRM	Percentage of Completely Redefined Methods	SM201
PCT	Path Complexity	SM203
PDAC	Package Data Abstraction Coupling	SM31
PEC	Path External Complexity	SM203
PEM	Percentage of Extended Methods	SM201
PI	Productivity Index	SM203
PIC	Path Internal Complexity	SM203
PII	Pure Inheritance Index	SM226
PIMAS	Polymorphism Invocations Method	SM341
PINCCS	Productivity Increment of Class Size	SM203
PNAC	Peak number of active components	SM293
POM	Percentage of Overrided Methods	SM75
PPROTM	Percentage of Protected Members	SM194
PRECISION	PRECISION	SM166
PRM	Percentage of Redefined Method	SM201
PSIC	Provided Service Interface Complexity	SM203
PTCPS	Presence of Temporal Constraint for the Provided Service	SM203
PTMI	Production-Type Method Invocations	SM234
QACC	Quality - Accuracy	SM95
QL	Quality - Latency	SM95
QREL	Quality - Reliability	SM95
QREP	Quality - Reputation	SM95
QSLA	Quality - Service Level Agreement	SM95

Table D.1: OO Maintainability Metrics

Metric	Description	Primary Studies that contain the metric
QT	Quality - Throughput of a Method	SM95
QU	Quality - Usability	SM95
RAI	Relationship Abstraction Index	SM226
RC	Relative Cost	SM291
RCC	Real Class Complexity	SM203
RCS	Real Class Size	SM203
RD	Relative Dependency	SM193
RECALL	RECALL	SM166
Reference Coupling	Reference Coupling	SM29
REFF	Reach-Efficiency	SM87
RFC1	Response For a Class	SM151
RMA	Abstractness	SM7
RMD	Normalized Distance from Main Sequence	SM7
RMI	Instability	SM7
ROV	Ratio of variability	SM86
RP	Relative Productive	SM291
RSI	Reuse percent	SM291
RSIC	Required Service Interface Complexity	SM203
SAVI	State Access Violation Index	SM211
SC	Subjective assessment of Complexity	SM141
SCC	Similarity-based Class Cohesion	SM68
SCCS	Source Code Control Systems	SM189
SCOPE	SCOPE	SM193
SCR	Signature Change Rate	SM296
SCV	Skip Call Violations	SM82
SDC	Strength of Dependency between Classes	SM112
SDI	System Design Instability	SM54
SERVERS	Number of Receiving Classes	SM230
SFC	Structural Feature Coupling	SM346
SII	System Implementation Instability	SM54
SIMAS	Static Method Invocations	SM341
SIX	Specialization	Index SM151
SIZE	Size	SM87
SLAQ	Slice Layer Architecture Quality	SM275
SNOC	Size Of Number Children	SM35
SPN	Sum of the avg. variable spans	SM35
STMTS	Statements	SM224
SU	Software Understandability	SM10
TA	Total Attributes	SM69
TCF	Technical Complexity Factors	SM7
TCR	True Comment Ration	SM7
TFC	Textual Feature Coupling	SM346
TIES	Ties	SM87
TKE	Time to fix the Known Errors	SM141
TLCOM	Transitive LCOM	SM335
TM	Total of modifiability	SM210
TMR	Time to Implement Modifications	SM141
TO	Total Operations	SM69
TPC	Tight Package Cohesion	SM31
TRCF	Task-relevant code fragments	SM10
TU	Total of Understandability	SM10
UDT	User Defined Type	SM38
UNFM	Programmer?s Unfamiliarity with the software	SM10
UPCALLBY	Up Call By	SM211
Use Case Complexity	Use Case Complexity	SM7
VI	Verifiability Index	SM203
VOLUME	Volume	SM212
WFD	Weighted sum of all the bug reports at criticality level	SM217
WGT	Weighted Graph Tree	SM199
WIG	Weighted Interaction Graph	SM199
WIH	Width of the Inheritance Hierarchy	SM4
WMA	Weight of Method by Aspect	SM205
WMPC1	Weighted Methods Per Class	SM7
WNCO	Weighted Number of Collections Operation	SM85
WNN	Weighted Number of Navigations	SM85
WOC	Weighted Operations per Component	SM209
PIM	Polymorphically Invoked Methods	SM38, SM234, SM341, SM343

E

Quasi-Experiment Protocol

E.1 Experimentation Goal

To assess the OOSM metrics catalog(s) generation through a context-based categories approach. Specifically, to check what is the coverage percentage of the OOSM metrics' from the catalog proposed by the catalogs' generator over the OOSM metrics' catalogs suggested by researchers.

E.2 Quasi-Experiment Definitions

This section presents some experimental definitions to be considered throughout the quasi-experiment planning/execution.

E.2.1 Quasi-Experiment Subjects

The person who applies the treatment to the experimental units. The subjects are the expert researchers consulted.

E.2.2 Quasi-Experiment Objects/Units

The object on which the quasi-experiment is run. In our case, we have one object per subject. The objects are each individual contexts of maintainability that each subject has (his/her point of view). Therefore, both treatments will be applied (in pairs) to the same object for each subject.

E.2.3 Factor and Treatment

The quasi-experiment has one factor and two treatments. The factor is how maintainability metrics are defined by researchers and the treatments are two different ways of doing so. One

treatment is the researcher expertise and the other is using our catalog generator based on the metrics' categorization choice.

E.2.4 Independents Variables/Parameter

The independents variables, also called parameters, are variable we fix (control) in the experiment since they might change the result if other values are assumed. The independent variables and their values are:

- Metrics definition domain: Object-Oriented Software Maintainability;
- Expert research set: Authors of papers published at ICSE, SPLASH/OOPLSA, ICSM, CSMR, SBES, ECOOP, METRICS Conference, WETSoM (2009 to 2013);
- Quasi-Experiment Application Period: December-January;
- The order that the treatments are applied by subject: First the researcher generates the catalog based on his/her expertise and than using our approach (catalog generator based on the metrics' categorization choice). We choose such order since exposing the researchers to the catalog our approach generates would influence the metrics the researchers would report based on his/her experience.

E.2.5 Dependents Variables

The dependent variable is the one through we measure the effect of the different treatments. In this quasi-experiment, the dependent variables are the metrics' catalogs generated.

E.2.6 Control Group

Another way of looking at this quasi-experiment is considering a control group. In this case the control group for this quasi-experiment is the metrics catalog suggested by the experts considering their expertise (treatment 1).

E.3 Quasi-Experiment Design

E.3.1 Research Goal and Hypotheses

The research goal here is to assess the statement that the metrics' catalogs generation proposed here is useful. We tried to find any evidence about of the coverage percentage of the OOSM metrics' catalog generated based on the categories choice over the OOSM metrics' suite proposed by the experts interviewed.

A Coverage Index (CI) was defined for each catalog generated during the quasi-experiment. The CI is the probability of the metrics suggested by the experts belong to the catalog generated using our approach.

For each researcher (i) evaluated, a CI_i was denoted as:

$$CI_i = \frac{\#(X_i \cap Y_i)}{\#Y_i}$$

Where, X_i is the metrics catalog generated using our approach (categorization choice) by researcher i, Y_i is the metrics catalog generated by the researcher i expertise, and # is a function that returns the number of metrics in a catalog.

The goal of this evaluation is to infer that our catalogs' coverage is at least 90% over the metrics suggested by experts. Thus, the hypotheses definition are:

H_0: CI < 0.9. In other words, the catalogs generated using our approach has less than 90% of coverage over the catalogs proposed by experts (null hypothesis).

H_1: CI >= 0.9. In other words the catalogs generated using our approach has at least 90% of coverage over the catalogs proposed by experts (alternative hypothesis).

E.3.2 Research Method

Our study is a quasi-experiment because we do not have a random assignment of subjects to treatment. This occurs mainly because we do not know the actual population of OOSM researchers and practitioners. In fact, we invited all researchers we could, considering our definition of metric experts (see independent variables definition at Section E.2.4) and all participants where exposed to both treatments.

E.3.3 Data Collection Technique (Instrumentation)

We used the Direct Technique, specifically questionnaires for data collection. Direct techniques allow the experimenter to obtain a general understanding of the software engineering process. Such techniques are probably the only way to gauge how enjoyable or motivating certain tools are to use or certain activities to perform. However, they are often subjective, and additionally do not allow for accurate time measurements. It is composed by brainstorming, focus groups, interviews, questionnaires, conceptual modeling.

Interviews and questionnaires are techniques that have been used by researchers when their goal is to understand general information (including opinions) about process, product, personal knowledge etc. It is can be adopted for small to large volume of data. As Interviews involve at least one researcher talking to at least one respondent, we did not adopt this method. Questionnaires are sets of questions administered in a written format. These are the most common field technique because they can be administered quickly and easily. However, very careful attention needs to be paid to the wording of the questions, the layout of the forms, and the ordering of the questions in order to ensure valid results.

E.4 Data Assessment

For data assessing, a statistic coverage evaluation was performed checking the catalog CI. We calculated the CI based on the description previously shown (Section E.3.1) for each expert interviewed. It was found a data asymmetry analyzing the histogram of the CIs calculated. Consequently, a non-parametric test was applied to assess the data set (WILCOX, 2004). Specifically, the Wilcoxon Test for one-sample was used, and the Z-Test was adopted as the statistic of the test.

For these cases, we should reject the H_0 hypothesis if $Z > Z\alpha$ (WILCOX, 2004). For this assessment, we used a 99% confidence level (CL). It is the probability that a confidence interval captures the true population parameter given a distribution of samples (WILCOX, 2004). The values of $Z\alpha$ are obtained through the normal distribution. It is a mathematical table used to find the probability of a statistic. Considering 99% of CL, $Z\alpha$ is equal to **2.33** (obtained through the normal distribution).

It is important to highlight that the population was not sampled. Consequently, the conclusion of the quasi-experiment is based on the observed data. In our case the observational data was the 47 catalogs suggested by the experts and generated using our approach. Nevertheless, even with the unknown population, it is possible to study some population characteristics through the observed data. For this quasi-experiment, characteristics such as expertise in software metrics, and professional profile defined the type of population. Thus, the results assessment is valid for the observational data.

E.5 Questionnaire Applied

This section shows the online questionnaire used in this quasi-experiment. Figures E.1 E.2 E.3 E.4 E.5 depict the online questionnaire pages. The first part is composed by questions related to interviewee's background. It is depicted in Figure E.1. Questions 4 and 5, shown in Figure E.2 measure the interviewee expertise.

Figure E.3 presents how the questionnaire participant can suggest metrics to compose the his/her catalog. He/she has to inform the metric's name and metric's description to compose his/her catalog. After that, it is necessary to choose which category(s) fit with the metrics previously added. Figure E.4 shows this questionnaire part. Finally, a catalog is generated according the information about metrics and categories that the interviewee informed previously. And, a last question is raised to check the equivalence between the catalog generated by the tool and the catalog suggested by the expert. It is possible to check this situation in Figure E.5.

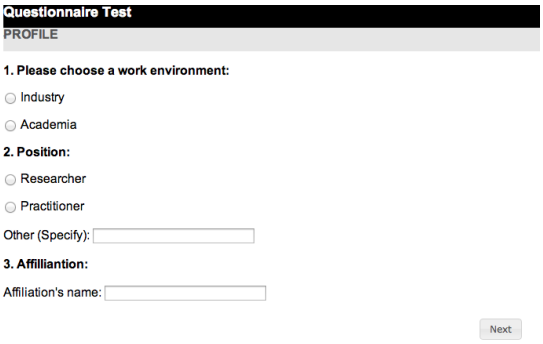

Figure E.1 Questionnaire - First Part (Page 01).

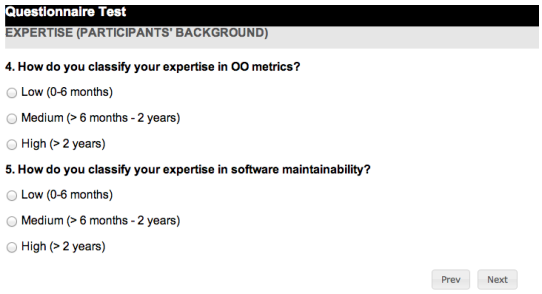

Figure E.2 Questionnaire - Second Part (Page 02).

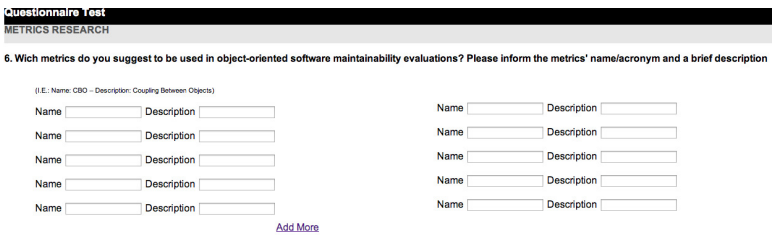

Figure E.3 Questionnaire - Third Part (Page 03).

Figure E.4 Questionnaire - Fourth Part (Page 04).

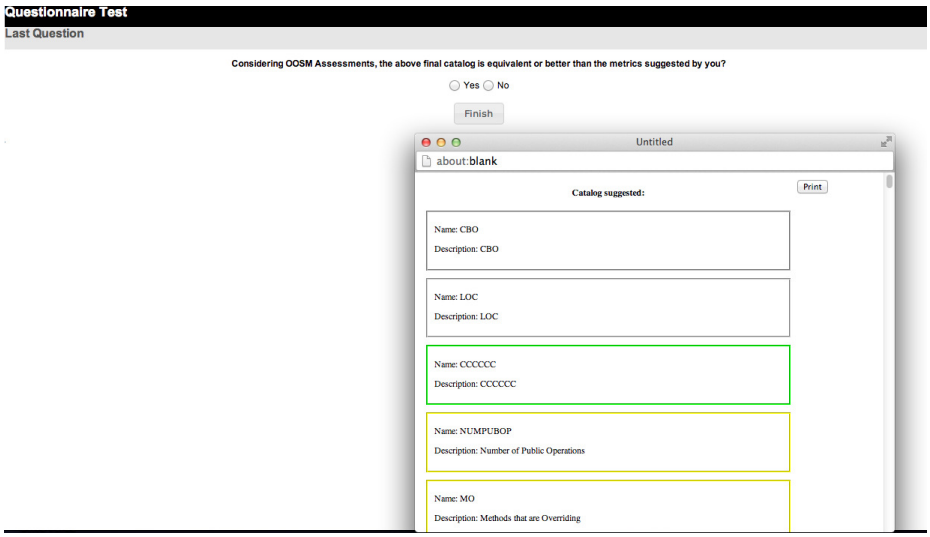

Figure E.5 Questionnaire - Last Question (Page 05).

F

Experts' Opinion About the Proposed Approach

The last question of the questionnaire was related to the experts' assessment of the catalog generated by the portal. The goal of this question is to check the experts opinion about of the catalogs generation. The respondents were questioned if the catalog generated was equivalent or better than the catalog proposed/used by them in OOSM evaluations. It is important to clarify that is a complementary evaluation to the coeverage assessment presented in Section 4.4.5.

The Hypotheses Test of Proportion (WILCOX, 2004) was used to assess the data. This is used to test hypotheses related to sample proportion, which is our case. We wanted to check if the majority of the respondents agree that the catalog generated by the portal is equivalent or better than the catalog previously suggested by them. Consequently, we assumed that the majority of the respondents assessed positively the catalog generated by the portal. Based on that, the hypotheses shown in Section 3.5.2 were raised. This implies that:

1. H_0: $\hat{P} = 0.7$
2. H_1: $P_0 > 0.7$

The statistic for this type of test is to compare the Z value with the value of $Z\alpha$. α is the probability that represents the CL. For this assessment, we used a CL of 99%. The values of $Z\alpha$ are obtained through the normal distribution. Considering 99% of CL, $Z\alpha$ is equal to **2.33** (obtained through the normal distribution).

On the other hand, to calculate Z we use the formula proposed for the statistic of this test, depicted in Figure F.1. Therefore, considering 47 answers, 41 respondents informed that the catalog proposed by our approach was equivalent or better than the catalog suggested by them. Consequently, the Z value obtained was **2.57**.

The statistic of the test says that we should reject the H_0 hypothesis if $Z > Z\alpha$ (WILCOX, 2004). Consequently, we reject the H_0. Based on this results, we claim that respondents assessed the portal positively. Thus, this assessment represents an initial approval of the approach and idea behind the portal of OOSM metrics.

$$z = \frac{\hat{p} - p_o}{\sqrt{\frac{p_o(1 - p_o)}{n}}}$$

Figure F.1 Z0 Calculation Formula (WILCOX, 2004).

Nevertheless, it is important to highlight that this result has some limitations since the experts' opinion is a subjective evaluation and we did not explained what should be considered good or bad, therefore, what it is good for one researcher might not be good enough for others. In addition, since the last question of the questionnaire is related to the equivalence of the catalogs (generated by the portal and suggested by the experts), and it was not clear/explicit what does mean "equivalent catalogs" during the questionnaire's answering, the experts might have misunderstood the question. Also, there was just two options to answer this last question: "YES" or "NO'. Consequently, there was not a chance for experts to inform what they actually think about the catalogs' comparison (generated by the portal and suggested by them). Even with aforementioned limitations, we believe that it is important register the results found as a complementary evaluation.

G

Invitation Letter for the Questionnaire

Dear (RESEARCHER/PRACTITIONER),

I am Juliana Saraiva, a Computer Science Ph.D. student on the Informatics Center at the Federal University of Pernambuco (`http://www.cin.ufpe.br`). My research project is related to Object-Oriented Software Maintainability (OOSM) metrics. One of my work contributions is to provide a portal (website) that has information about this kind of metrics (e.g.: Metrics' Name and Description, the way to collect the metric, papers that address this subject, authors that proposed or adopted the metrics, and so on). Additionally, this environment contains a metrics' catalog generator.

This generator is based on each user's context of adoption/research (academic research or industrial practitioner scenarios). The goal of the portal is to simplify the decision making process about metrics adoption in OOSM context. Indeed, considering the number of existing metrics, many of them are probably known for very few people besides the ones who designed them. Because of this, your expert opinion is so important for this research.

Thus, I am emailing you to ask your participation in an opinion research about the effectiveness of catalog(s) generated by the portal. It is important to highlight that you were selected for this process because we believe that you are an expert and have some experience in metrics adoption for software maintainability assessment.

Consequently, you are invited to answer an online questionnaire about metrics composed by 05 (five) objectives questions and 01(one) open question. The whole process takes between 10 and 15 minutes. In case you agree, you should follow the link below and login (using the information provided below) in the restricted area. The login and password fields are in the upper-right part of the page.

WEBSITE: `http://julianasaraiva.info/oosmMetricsPortal`

USERNAME: XXXX

PASSWORD: XXXXXX

I am very thankful for your attention and I'd really appreciate if you could help us in evaluation of this portal, more specifically, in the assessment of the OOSM metrics catalogs generator.

Best Regards,
Juliana Saraiva