DESIGN AND EVALUATION OF A METHOD TO DERIVE DOMAIN METRIC THRESHOLDS

ALLAN VICTOR MORI

DESIGN AND EVALUATION OF A METHOD TO DERIVE DOMAIN METRIC THRESHOLDS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Eduardo Figueiredo
Coorientador: Elder Cirilo

Belo Horizonte
Agosto de 2018

ALLAN VICTOR MORI

DESIGN AND EVALUATION OF A METHOD TO DERIVE DOMAIN METRIC THRESHOLDS

Dissertation presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Eduardo Figueiredo
Co-Advisor: Elder Cirilo

Belo Horizonte
August 2018

© 2018, Allan Victor Mori
Todos os direitos reservados

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Mori, Allan Victor

M854d Design and evaluation of a method to derive domain metric thresholds / Allan Victor Mori. — Belo Horizonte, 2018.
xxii, 59 f.: il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Eduardo Magno Lages Figueiredo.
Coorientador: Elder José ReioliCirilo

1. Computação – Teses. 2. Engenharia de software. 3. Domínios de Software. Orientador. II. Coorientador. III. Título.

CDU 519.6*22(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Design and Evaluation of a Method to Derive Domain Metric Thresholds

ALLAN VICTOR MORI

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Eduardo Figueiredo
PROF. EDUARDO MAGNO LAGES FIGUEIREDO - Orientador
Departamento de Ciência da Computação - UFMG

Elder José Reioli Cirilo
PROF. ELDER JOSÉ REIOLI CIRILO - Coorientador
Departamento de Computação - UFSJ

PROFA. KÉCIA ALINE MARQUES FERREIRA
Departamento de Computação - CEFET-MG

PROF. MARCO TÚLIO DE OLIVEIRA VALENTE
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 24 de agosto de 2018.

Dedico este trabalho à minha família, que sempre me incentivou a sonhar.

Acknowledgments

O trabalho presente nas páginas a seguir não seria possível sem o apoio de muitas pessoas. Eu agradeço a Deus por todas as pessoas que contribuíram com este trabalho e as lições aprendidas ao longo do caminho para a obtenção do título de Mestre.

Agradeço aos meus pais Dorival e Mariluci, meu irmão Alaor e cunhada Gabriela, pelo amor, suporte e compreensão diante dos desafios da vida.

Agradeço a minha namorada Fabiana por compartilhar a jornada com amor e carinho, pelas palavras gentis nos momentos certos e pela felicidade em cada conversa.

Agradeço aos meus orientadores, Prof. Dr. Eduardo Figueiredo e Prof. Dr. Elder Cirilo, pela oportunidade, parceria e colaboração para que este trabalho pudesse ser realizado, pelos ensinamentos e apoio nesses anos de mestrado.

Agradeço aos amigos do LabSoft, pelas conversas, cafés, colaborações e também pela amizade e acolhimento durante esse período de aprendizado.

Agradeço aos amigos das disciplinas e corredores do DCC. Em especial, Aline e Levy por estarem presentes nos momentos felizes e mais difíceis do mestrado.

Agradeço ao professor Dr. Marco Túlio e a professora Dr. Kécia Ferreira pela disponibilidade e valiosa contribuição no trabalho.

Agradeço ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Minas Gerais pela oportunidade do mestrado e a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela bolsa de mestrado concedida.

"O impossível não é um fato: é uma opinião."
(Mario Sergio Cortella)

Abstract

Software metrics provide means to quantify several attributes of software systems. The effective measurement is dependent on appropriate metric thresholds as they allow characterizing the quality of software systems. Indeed, thresholds have been used for detecting a variety software anomalies. Previous methods to derive metric thresholds do not take characteristics of software domains into account, such as the difference between size and complexity of systems from different domains. Instead, they rely on generic thresholds that are derived from heterogeneous systems. Although derivation of reliable thresholds has long been a concern, we also lack empirical evidence about threshold variation across distinct software domains. This work proposes a method to derive domain-sensitive thresholds that respects metric statistics and is based on benchmarks of systems from the same domain. The proposed method is supported by a software tool. This tool helps the developer to write better code since the beginning, by providing a view with class metrics and warnings considering the system domain. To evaluate our method, we performed an evaluation with desktop and mobile systems. The first evaluation, we manually mined one hundred mobile applications from GitHub. We measured all these systems using a set of metrics, derived thresholds, and validated them through qualitative and quantitative analyses. For the second evaluation, we investigated whether and how thresholds vary across domains by presenting a large-scale study on 3,107 software systems from 15 desktop domains. As a result, we observed that our method gathered more reliable thresholds considering software domain as a factor when building benchmarks for threshold derivation. Moreover, for the desktop evaluation, we also observed that domain-specific metric thresholds are more appropriated than generic ones for code smell detection.

Keywords: Metric Thresholds, Software Domains, Software Engineering.

List of Figures

3.1 Domain Metric Threshold Method 15
3.2 Architecture of TWarning. 16
3.3 TWarning View and Project Properties. 16

4.1 Experimental steps for the analysis of mobile domain thresholds. 22

5.1 Experimental steps for thresholds analysis. 32
5.2 God Class Detection Strategy 45

List of Tables

2.1 Software Metrics . . . 9

3.1 Qualitative Evaluation of the Proposed Methods to Derive Thresholds . . . 18

4.1 Description of Mobile Software Domains . . . 23
4.2 Mobile Threshold from the Proposed Method . . . 24
4.3 Difference between Highest and Lowest Thresholds . . . 26
4.4 Coefficient of Variation for Domains Thresholds . . . 26
4.5 Metrics Levels for 90% and 95% thresholds . . . 28

5.1 Description of Enterprise Software Domains . . . 33
5.2 Lines of Code per System Domain . . . 34
5.3 Number of Classes per System Domain . . . 35
5.4 Metric Threshold per Domain Group . . . 36
5.5 Difference between Highest and Lowest Thresholds . . . 39
5.6 High and Very-High Thresholds Levels for each Metric . . . 41
5.7 Thresholds for Different Benchmarks . . . 42
5.8 Precision and Recall using Domain-Specific and Generic Thresholds . . . 46

Contents

Acknowledgments **xi**

Abstract **xv**

List of Figures **xvii**

List of Tables **xix**

1 Introduction **3**
1.1 Proposed Method and Evaluation 4
1.2 Publications 5
1.3 Dissertation Outline 5

2 Background and Related Work **7**
2.1 Software Domains 7
2.2 Software Metrics 8
2.3 Methods to Derive Thresholds 9
2.4 Concluding Remarks 12

3 Domain Threshold Derivation Method **13**
3.1 Method Steps 13
3.2 TWarning Tool 15
3.3 Qualitative Evaluation 17
3.4 Concluding Remarks 18

4 Evaluation of Mobile Domain Thresholds **21**
4.1 Research Questions and Experimental Steps 21
4.2 Benchmark of Mobile Systems 22
4.3 Derived Thresholds 23
4.4 RQ1: Variation of Thresholds for Different Domains 25

4.5 RQ2: Coefficient of Variation for Thresholds 25
4.6 RQ3: Similarity among Domains 27
4.7 Concluding Remarks 28

5 Evaluation of Domains Thresholds with Enterprise Systems 31
5.1 Research Questions and Experimental Steps 31
5.2 Selected Systems and Domain Classification 32
5.3 Measurement and Threshold Derivation 35
5.4 Desktop Domain Thresholds 36
5.5 RQ1: Threshold Variation across Domains 39
5.6 RQ2: Similar Thresholds across Domains 40
5.7 RQ3: Impact of System Size on Thresholds 42
5.8 Code Smell Detection Evaluation 43
5.8.1 Dataset for Code Smell Analysis 43
5.8.2 Metric-based Detection Strategy and Measurement of Effectiveness 44
5.8.3 RQ4: Effectiveness of Thresholds for Code Smell Detection . . . 45
5.9 Concluding Remarks 47

6 Threats to Validity 49
6.1 Internal Validity 49
6.2 External Validity 50
6.3 Concluding Remarks 50

7 Final Considerations 51
7.1 Conclusion 51
7.2 Contribution 52
7.3 Future Work 52

Bibliography 55

Nomenclature

CBO Coupling between Objects

DIT Depth of Inheritance Tree

LCOM Lack of Cohesion in Methods

LOC Lines of Code

NOA Number of Attributes

NOC Number of Children

NOM Number of Methods

WMC Weighted Method per Class

Chapter 1

Introduction

Measurements support project managers to improve their products and processes [DeMarco, 1986]. In software engineering, researchers and practitioners are always looking for better ways to predict the number of faults, errors, and the effort to complete a task [Kitchenham, 2010]. In this sense, many metrics have been proposed and pragmatically used to measure, control, and assess the quality of software systems [Alves et al., 2010; DeMarco, 1986; Fernandes et al., 2016, 2017]. For example, certain metrics can help to indicate specific software components (or modules) that suffer from bad design or poor-quality code [Buschmann et al., 1996; Lanza and Marinescu, 2007; Marinescu, 2004; Oizumi et al., 2016; Vidal et al., 2015]. As a result, developers may check if there is something wrong with these components to avoid future maintenance issues. Nevertheless, the effective use of metrics is directly dependent on the definition of reliable and higher quality metric thresholds [Alves et al., 2010; Padilha et al., 2014; Vale et al., 2015]. Thresholds allow practitioners to objectively characterize or classify software components according to one or more software metrics. Several methods to derive metric thresholds have been proposed over the years [Alves et al., 2010; Chidamber and Kemerer, 1994; Concas et al., 2007; Erni and Lewerentz, 1996; Ferreira et al., 2012; Filó et al., 2015; French, 1999; Spinellis, 2008; Vale et al., 2018; Vasa et al., 2009]. The common practice for deriving thresholds in these methods is to use a set of similar systems (so-called benchmarks) [Alves et al., 2010; Vale et al., 2018]. The idea behind the use of benchmarks is to get information from similar systems (e.g., same programming language) and derive thresholds for a specific context.

Despite the use of benchmarks, methods for deriving thresholds in recent literature do not consider specific characteristics of the systems when building the benchmarks [Alves et al., 2010; Vale et al., 2018]. For instance, the use of localization on navigation systems, transaction management on e-commerce systems, mechanics and

level design of games. The reason for using systems of the same domain to build benchmarks is to group systems that share intrinsic characteristics and derive thresholds for a specific context. Recent studies has shown that grouping systems in a domain support to better understand some analysis. For instance, the relation of language defect proneness with software domain and the presence of smells and quality-related metrics [Ray et al., 2014; Linares-Vásquez et al., 2014]. Such evidences calls for a method that takes the intrinsic characteristics of software systems into account when deriving metric thresholds.

1.1 Proposed Method and Evaluation

This work proposes a tool supported method to derive metric thresholds that considers the intrinsic characteristics of software systems into the same domain and the evaluation of mobile and enterprise systems. The domain threshold derivation method proposes the classification of systems into domains before further threshold derivation steps. Then, it derives thresholds in a step-wise format where each metric has its own thresholds for each software domain. The proposed method shares common activities presented in other methods. However, its design enhances three of major issues in existing threshold derivation methods: (i) the input is a benchmark composed of systems that belong to the same domain; (ii) the core process computes metrics and performs statistical analyses for each separated domain; and (iii) the output is domain-specific thresholds to each analyzed metric.

The key points of this method is: It has systematic steps, it considers the system domain in order to compose the benchmark, it has strong dependence with the number of entities (i.e, classes) and a weak dependency with the number of systems, it calculates upper and lower thresholds and it respects metric statistics of the metrics. A software tool, named TWarning, supports the proposed method. This tool helps the developer to write better code since the beginning, by providing a view with class metrics and warnings considering the system domain.

To evaluate our method, we performed an evaluation with mobile and enterprise systems (i.e., desktop and Web systems). The first evaluation, we manually mined one hundred mobile applications from GitHub. We measured all these systems using a set of metrics, derived thresholds, and validated them through quantitative analyses. For the second evaluation, we investigate whether and how thresholds vary across domains by presenting a large-scale study on 3,107 software systems from 15 desktop domains. As a result, we observed that our method gathered more reliable thresholds consider-

ing software domain as a factor when building benchmarks for threshold derivation. Moreover, for the desktop and Web evaluation, we also observed that domain-specific metric thresholds are more appropriated than generic ones for code smell detection.

1.2 Publications

This dissertation generated the following publications and, therefore, it contains resources from them.

- Allan Mori, Gustavo Vale, Markos Viggiato, Johnatan Oliveira, Eduardo Figueiredo, Elder Cirilo, Pooyan Jamshidi, and Christian Kastner. Evaluating Domain-Specific Metric Thresholds: An Empirical Study. In *Proceedings of the 1st International Conference on Technical Debt (TechDebt)*, Gothenburg, Sweden, 2018.

- Allan Mori, Elder Cirilo, Eduardo Figueiredo. TWarning: A Warning Tool for Domain-Sensitive Thresholds. In *Proceedings of the 9th CBSoft - Tool Session*, São Carlos, Brasil, 2018.

- Allan Mori, Eduardo Figueiredo, and Elder Cirilo. Towards the Definition of Domain-Specific Thresholds. In *Proceedings of the XIII Brazilian Symposium on Information Systems (SBSI)*. Lavras, MG, 2017.

- Allan Mori, Elder Cirilo, and Eduardo Figueiredo. Measuring and Comparing Quality Attributes of Software Development Communities. In *Proceedings of the Master and PhD Workshop on Software Engineering (WTDSoft)*, co-allocated with CBSoft. Fortaleza, CE, 2017.

1.3 Dissertation Outline

This chapter present our motivation and introduce this dissertation. The remainder of this document is organized as follow.

Capter 2: This chapter presents concepts to understand this study and related works that explore software domains, software metrics and methods to derive metric thresholds. We explore how previous studies addressed software domains. We list metrics used in this work and summarize methods to derive thresholds presenting our perspective on all these concepts.

Capter 3: This chapter proposes our method to derive domain-sensitive metric thresholds. This chapter also presents the method process and its composing steps. It shows a tool, called TWarning, which automates the proposed domain-sensitive threshold derivation method and presents the qualitative evaluation for the method proposed.

Capter 4: This chapter illustrates an example of use of the proposed method in a benchmark composed of mobile applications. We present our experimental steps and research questions to evaluate this method. We also present the thresholds derived by our method using the mobile benchmarks. Furthermore, we present the variation of thresholds for different mobile domains, compare our method results using the coefficient of variation and discuss the similarities and differences between domains.

Capter 5: This chapter evaluates the method using desktop and Web software systems from many domains. First, we describe our research questions and experimental steps and, then, we explain how we built our benchmarks. Moreover, we describe how we perform the measurement and domain metric threshold derivation and present the metric thresholds. Finally, we discuss the derived domain thresholds and evaluates whether domain thresholds are better to find code smells.

Capter 6: This chapter presents some potential threats to validity that we discuss and presents the main actions we have taken to mitigate their impact on the research results.

Capter 7: This chapter presents our final considerations and main conclusions about the proposed method and its results. In addition, we present our contributions, and directions for future works.

Chapter 2

Background and Related Work

This chapter presents concepts to understand this study and related works that explore software domains, software metrics and method to derive metric thresholds. Section 2.1 discusses how previous studies address software domains. Section 2.2 lists the metrics used in this work. Section 2.3 summarizes methods to derive thresholds. Section 2.4 brings our perspective on all these concepts.

2.1 Software Domains

Previous studies [Linares-Vásquez et al., 2014; Murphy-Hill et al., 2014; Ray et al., 2014; Silva et al., 2013] have grouped systems based on similar characteristics in domains. The results indicated domains as a factor to better understand some analysis. For instance, Linares-Vásquez et al. [2014] grouped systems into 13 domains to investigate the relationships between the presence of smells and quality-related metrics. Murphy-Hill et al. [2014] compare game development with development of systems from other domains. Ray et al. [2014] grouped systems in 7 different domains to check whether language defect proneness is related with the software domain. These studies investigated software quality in specific domains.

Another study conducted an industry multi-project study [Silva et al., 2013] to evaluate the reusability of detection strategies in systems of a specific domain. They evaluated the degree of reuse for anomaly detection strategies based on the judgment of domain specialists. This study revealed that even though the reuse of strategies in a specific domain should be encouraged, their accuracy is still limited when holistically applied to all the modules of a program. However, the accuracy and reuse were both significantly improved when the metrics, thresholds, and logical operators were tailored to each recurring concern of the domain. Like these studies, we share the idea that

systems from the same software domain have distinct characteristics leading to specific measures and different thresholds to characterizing their entities.

2.2 Software Metrics

Object-oriented software metrics are used to capture different attributes of systems, such as size, complexity, cohesion, and inheritance relationships. In this study, we investigate domain-specific thresholds using eight well-known metrics. To measure size: Lines of Code (LOC), Number of Attributes (NOA), and Number of Methods (NOM). Complexity was measured with: Weighted Method per Class (WMC), Coupling between Objects (CBO) and Lack of Cohesion over Methods (LCOM). For inheritance: Depth of Inheritance Tree (DIT) and Number of Children (NOC). By applying these metrics, it was possible to access different quality attributes of software systems, such as maintainability and changeability [Chidamber and Kemerer, 1994; Lanza and Marinescu, 2007; Lorenz and Kidd, 1994]. To automate the measurements we used CK Metrics tool [CK Metrics, 2018] to extract software metrics, this tool is open-source and was written in Java to measure one system at time. We executed a script to measure all systems.

Table 2.1 presents the metrics, showing their names and a brief description. We choose these metrics because: (i) they capture different attributes of software systems, such as size and complexity; (ii) they are well-known object-oriented software metrics [Kitchenham, 2010]; and (iii) they have been often used by researchers and practitioners to measure several anomalies in components and quality attributes [Chidamber and Kemerer, 1994; Kitchenham, 2010].

Size metrics helps to access software dimensions, which can be size of a file or amount of elements, among others. Lines of Code (LOC) indicates the size of a class, measuring the number of lines of code per class without comments or blank lines [Lorenz and Kidd, 1994]. Weighted Method per Class (WMC) counts the number of methods in a class weighting each method by its cyclomatic complexity [Chidamber and Kemerer, 1994]. Number of Methods (NOM) quantifies the number of methods or constructors and Number of Attributes (NOA) counts the number of fields or class variables. These metrics are mainly used to estimate the size of a class.

Table 2.1. Software Metrics

Metric	Description
Size	
Lines of Code (LOC)	Measures the number of lines of code per class, it counts neither comment lines nor blank lines.
Number of Attributes (NOA)	Quantifies the number of fields and constants in a class.
Number of Methods (NOM)	Quantifies the number of methods and constructors in a class.
Complexity	
Weighted Method per Class (WMC)	Counts the number of methods in a class weighting each method by its cyclomatic complexity.
Lack of Cohesion in Methods (LCOM)	Divides (i) the pairs of methods in a class that do not access any attribute by (ii) the pairs of methods in a class that do access attributes in common.
Coupling Between Objects (CBO)	Counts the number of classes that are coupled to a class, by calling methods or accessing attributes of the other classes.
Inheritance	
Depth of Inheritance Tree (DIT)	Counts the number of levels that a subclass inherits methods and attributes from a superclass in the inheritance tree.
Number of Children (NOC)	Counts the number of direct subclasses of a given class. This metric indicates software reuse by means of inheritance.

Coupling, cohesion and complexity metrics are related to complexity and number of possible changes of flow in classes. Coupling between Objects (CBO) counts the number of classes called by a given class, measuring the degree of coupling among classes based on method calls and attribute accesses [Chidamber and Kemerer, 1994]. Lack of Cohesion over Methods (LCOM) measures the cohesion of methods of a class in terms of the frequency that they share attributes. To calculate this metric, we first get the pairs of methods in a class that do not access any attribute in common and then we subtract by the pairs of methods in a class that do access attributes in common [Chidamber and Kemerer, 1994].

With respect to inheritance metrics, Depth of Inheritance Tree (DIT) counts the number of levels that a subclass inherits methods and attributes from superclasses in the inheritance tree of the system. This metric estimates the class complexity with respect to its inheritance relationships. Number of Children (NOC) counts the number of direct subclasses of a given class. This metric indicates software reuse by means of inheritance [Chidamber and Kemerer, 1994].

2.3 Methods to Derive Thresholds

Thresholds allow us to objectively characterize or classify each component according to one of the quality metrics. The effective use of software metrics depends on the definition of the corresponding thresholds. Thresholds have been calculated on the basis of experience of software engineers or with a single system as a reference [Chidamber and Kemerer, 1994]. In recent years, methods have been proposed using a set of systems to derivate thresholds.

Programming Experience. Programming experience were used to obtain thresholds by Coleman et al. [1995]; McCabe [1976]; Nejmeh [1988]. For instance, Coleman et al. defined the values 65 and 85 as thresholds for Maintainability Index (MI). When MI values are smaller than 65, they are considered as highly maintainable, between 65 and 85 as moderately maintainable, higher than 85 as difficult to maintain. McCabe defined the values 10 for McCabe Complexity and Nejmeh defined 200 for NPATH metric as thresholds. These values can be used to indicate whether there are code smells. Thresholds based on programming experience are appropriately applied in specific cases. These threshold values are arbitrary and the lack of scientific support often leads to divergences. However, unlike these papers [Coleman et al., 1995; McCabe, 1976; Nejmeh, 1988], our research does not aim to propose a method to derive thresholds based on programming experience.

Metric Analysis. Erni and Lewerentz [1996] proposed thresholds (T) that use mean (μ) and standard deviation (σ) from software data. A threshold is calculated using equations, $T = \mu + \sigma$ and $T = \mu - \sigma$ when high and low values of a metric indicate potential design problems, respectively. Lanza and Marinescu [2007] use a similar method in their research for 45 Java projects and 37 C++ projects. However, they use four labels: low, mean, high, and very high. Labels low, mean, and high are calculated in the same way as Erni and Lewerentz [1996]. Label very high is calculated as $T = (\mu + \sigma) \times 1.5$. Abílio et al. [2015] use the same method than Lanza and Marinescu [2007], but they derive thresholds for eight Software Product Lines. These methods rely on a common statistical technique. However, Erni and Lewerentz [1996]; Abílio et al. [2015]; Lanza and Marinescu [2007] do not analyze the underlying distribution of metrics. These methods assume that metrics are normally distributed, limiting the use of these methods. In contrast, our research focuses on a method that does not make assumptions about data normality [Vale and Figueiredo, 2015; Vale et al., 2018]. In addition, we take into consideration the functional domain of the target software systems. French [1999] also proposes a method based on the mean and standard deviation. However, French used the Chebyshevś inequality theorem (whose validity is not restricted to normal distributions). A metric threshold T can be calculated by $T = \mu + kx\sigma$, where k is the number of standard deviations. For metrics with high range or high variation, this method identifies a smaller percentage of observations than its theoretical maximum. Herbold et al. [2011] propose a method for threshold derivation that does not depend on the context of the collected metrics, e.g., the target programming language or abstraction level. For this purpose, the authors rely on machine learning and data mining techniques. Finally, Perkusich et al. [2015] propose a method to support the interpretation of values for software metrics. Instead of deriving

thresholds that indicate acceptable values for a metric, their proposed method relies on Bayesian networks that consider subjective factors of software development. Thus, their method aims to support managers in minimizing wrong decisions based on software measurement and assessment. In contrast to French [1999]; Herbold et al. [2011]; Perkusich et al. [2015], our method was designed to derive thresholds from benchmark data and, as such, it is resilient to high variation of outliers. In addition, we did not use Chebyshevś inequality theorem, machine learning, or Bayesian networks. We focus on the domain of the software systems that compose the benchmark.

Metric Distributions. The method proposed by Chidamber and Kemerer [1994] uses histograms to characterize data. They plotted histograms for each of their 6 software metrics per programming language. Analyzing the metrics distribution they spotted outliers in C++ and Smalltalk systems. Spinellis [2008] compares metrics of four operating system kernels (i.e., Windows, Linux, FreeBSD, and OpenSolaris). For each metric, boxplots of the four kernels are put side-by-side showing the smallest observation, lower quartile, median, mean, higher quartile, the highest observation, and identified outliers. The boxplots are then analyzed by the author and used to give ranks, + or - to each kernel. However, as the author states, the ranks are given subjectively. Vasa et al. [2009] propose the use of Gini coefficients to summarize a metric distribution across a system. The analysis of the Gini coefficient for 10 metrics using 50 Java and C# systems revealed that most of the systems have common values. Moreover, higher Gini coefficient values indicate problems and, when analyzing subsequent releases of source code, a difference higher than 0.04 indicates significant changes in the code. In contrast to Chidamber and Kemerer [1994]; Spinellis [2008]; Vasa et al. [2009], we did not use histograms, mean, median, or Gini coefficient to calculate thresholds and we derive thresholds based on data from a benchmark of systems from the same domain.

Benchmark-based. This section describes methods closer to ours because they are transparent, the thresholds are extracted from benchmark data, and the methods consider the skewed distribution of metrics. However, as far as we are concerned, none of these methods considers the domain of the analyzed software systems. For instance, Alves et al. [2010] proposed a method that weights software metrics by lines of code and aim at labelling each entity of a system based on thresholds. Each label is defined based on a pre-determined percentage of entities. This method proposes 70, 80, and 90% to represent the following labels: low (between 0 and 70%), moderate (70 - 80%), high (80 - 90%), and very high (>90%). Similarly, Ferreira et al. [2012] presented a method for calculating thresholds. This method consists in grouping the extracted metrics in a file and gets three groups, with high, medium, and low frequency. The

groups are called good, regular, and bad measurements, respectively. The authors do not make clear how to extract the three groups since they argue that the groups rely on visual analysis. Vale and Figueiredo [2015] proposed a method based on lessons learned from a comparison of other methods. Their method provides upper and lower thresholds in four different labels: very low (0 - 3%, low (3 - 15%), high (90 - 95%), and very high (>95%). Values between 15 and 90% are considered moderate or common. In contrast to Alves et al. [2010]; Ferreira et al. [2012]; Vale and Figueiredo [2015] methods, our method considers the intrinsic characteristics of software domains to derive metric thresholds.

2.4 Concluding Remarks

This chapter provides an overview about domains, metrics and methods to derivate thresholds. Methods derive thresholds from metric analysis, programmer experience and using benchmarks that consider the skewed distribution of software metrics. Recently, thresholds were derived from benchmarks and calculated using clearly defined derivation methods. For example, Alves et al. [2010] suggested a method that weighted software metrics through code lines. Similarly, Ferreira et al. [2012] group the extracted metrics into one file and get three groups of high, medium, and low frequency (good, regular and bad measurements).

Vale and Figueiredo [2015] method is benchmark-based, with a weak dependency on the number of systems. The method computes the upper and lower thresholds in a step-wise format and maintains the statistical properties of metrics. Unfortunately, existing methods and tools to derive thresholds do not either consider the intrinsic characteristics of software systems in each domain or provide a superficial analysis on thresholds for software domains. That is, they ignore the fact that systems from different domains may have different degrees of complexity and size, for instance. As a result, even when a robust and pragmatic method is used, the derived thresholds can be inappropriate. The next chapter proposes a method to derive thresholds that considers benchmarks of systems from the same domain.

Chapter 3

Domain Threshold Derivation Method

This chapter proposes a method to derive domain-sensitive metric thresholds; i.e., thresholds for systems of the same domain. The proposed method was designed to follow guidelines from previous works (Chapter 2) [Alves et al., 2010; Ferreira et al., 2012; Vale et al., 2015; Vale and Figueiredo, 2015]. These guidelines make the threshold derivation based on data from a representative set of systems (namely benchmark). As suggested by previous work [Vale et al., 2015; Vale and Figueiredo, 2015], the proposed method has a strong dependence with the number of entities and a weak dependency with the number of systems. Other relevant aspects of the proposed method are: It calculates upper and lower thresholds, it is systematic and it guarantees the statistical properties of the metrics. In addition to these points, the proposed method includes a novel classification step to consider the system domain in order to compose the benchmark. We argue that this step is important to achieve more reliable and higher quality thresholds. Section 3.1 presents the method process and its composing steps. Section 3.2 shows a tool, called TWarning, which automate the proposed domain-sensitive threshold derivation method. Section 3.3 presents the qualitative evaluation for the proposed method. Section 3.4 summarizes this chapter.

3.1 Method Steps

Figure 3.1 presents the main steps to the proposed method. The overall derivation process starts with a set of systems and they are classified into benchmarks by separate domains (Step 1). Steps 2 to 5 are executed for each domain-benchmark, composed of the classified systems. As a result, each domain has a derived set of thresholds. We

further describe each step below.

1. Classification: The systems are separated in domain and grouped to compose a benchmark labeled by the domain name. This process can be performed manually or automatically, for instance, using system artifact (e.g., entity names, description files, and others).

2. Compute Metrics: Software metrics are computed for each entity from the benchmark of systems from the same domain. By entity, we mean this method can be applied to architectural components, classes, or methods. The results are compiled into a file in which the first column represents the domain, the second has the entity name followed by other columns with the metric names. The rows are instances of the computed metrics.

3. Entity Weight Ratio: Each benchmark has different number of entities, making necessary to compute the entity weigh ratio to find how much an entity can represent in a specific domain. We compute entity weight percentage in the domain dividing the entity weight by the sum of all entities weights of the same domain and multiply by one hundred. All entities have the same weight and the sum of all entities must be 100% for a given domain. For instance, if one benchmark has 1,000 entities, each entity represents 0.1% of the overall domain ($0.1\% \times 1,000 = 100\%$).

4. Entity Sorting: The method ascending sorts entities by their weight. That is, we take the maximal metric value to represent a specific percentage. For instance, all entities with LOC < 100 must come first than LOC > 100. This is equivalent to computing a density function, in which the x-axis represents the weight ratio (0 - 100%), and the y-axis the metric scale.

5. Entity Grouping: After sorting, we want to represent the percentage of the overall metric values by grouping thresholds with labels defined as very low (0 - 3%), low (3 - 15%), moderate (15 - 90%), high (90 - 95%), and very high (>95%). These labels were defined similar and based on the same argumentation of a previous work [Vale et al., 2018], i.e., as software quality metrics normally have a skewed or (small) common distribution, a large amount of entities will have small values and a small number of entities will have large values. Therefore, to compensate it, the variation in the percentages between the very low and low labels is larger than the variation between the high and very high labels. In the next section where we present our method in practice, it will be clearer how to use these labels. To advance, grouping using the label high, 95% of the overall code for the LOC metric, the derived threshold for the Connectivity domain is 309 (see details in Section 4.3). This threshold is meaningful for systems of this domain, since it means that 95% of the Connectivity classes have less than 309 lines of code. Note that this threshold might be different for systems of

a different domain.

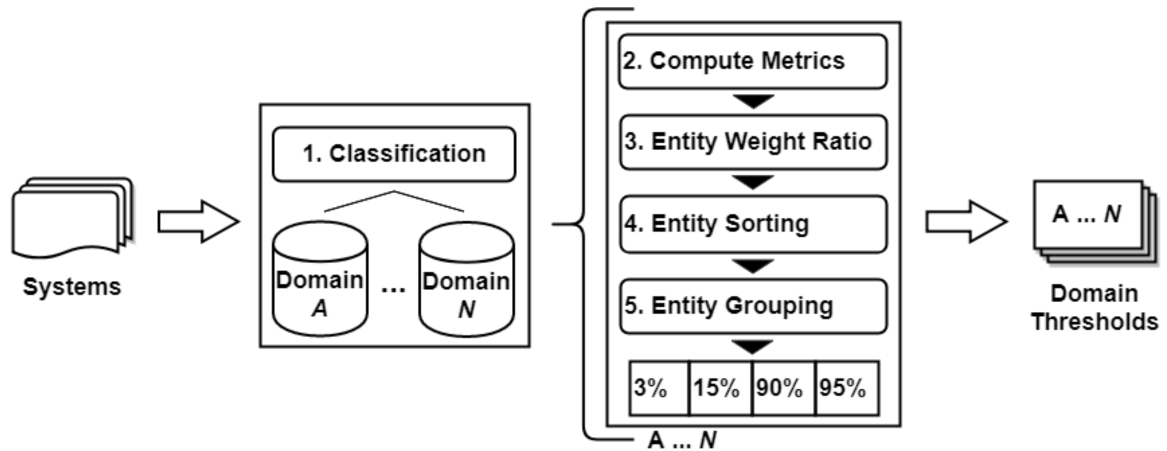

Figure 3.1. Domain Metric Threshold Method

3.2 TWarning Tool

TWarning is a tool that provides warnings based on domain-sensitive thresholds. It Considers that classes belonging to a domain and exceeding the metric thresholds can indicate a future anomaly. That is, this tool alerts the developer about domain-specific classes that can become a future issue. The following sections presents the architecture and design of this tool.

Architecture. TWarning is an Eclipse plug-in and its architecture is illustrated in Figure 3.2. The Project module detects which project was selected in the default Eclipse Project View, and provides to TWarning a reference to the selected project. The Warning module receives the project reference and requests the metrics measurement to the Metric module. CK Metrics tool calculates code metrics in Java projects by means of static analysis and provide the metrics measurement [CK Metrics, 2018]. After receiving the measured metrics, the Warning module requests to the Thresholds module to the domain-specific thresholds. The TWarning tool uses the domain-specific thresholds derived in our work (see Chapter 5). Based on a set of properties chosen by the developer (e.g., selecting E-Commerce domain and High label), an output is generated and exhibited in the View module (see Figure 3.3). This View also offers additional features to re-execute the metrics measurement and to configure domain preferences.

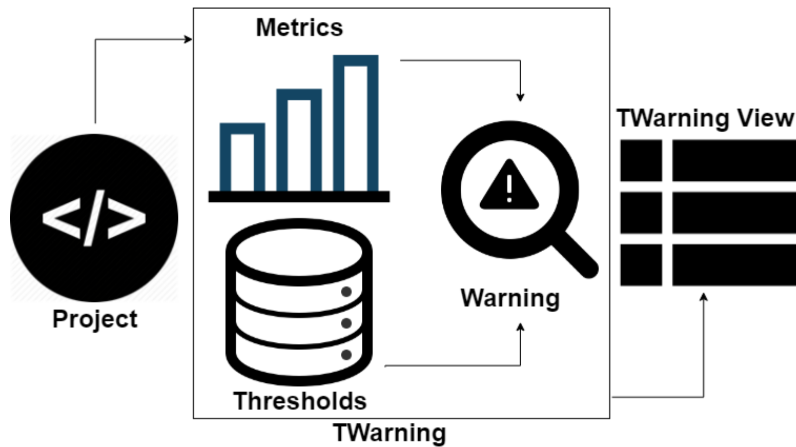

Figure 3.2. Architecture of TWarning.

Design and Implementation Decision. To issue warnings, developer must run TWarning and search for the colored classes in the View, and continue to check how to improve the class code. It is possible to see metrics, but the developer should pay attention to which domain is selected to analyze your project, this will influence the warnings. Figure 3.3 presents project domain selection and check-boxes to mark the labels to be colored in the view. The default properties are Generic Thresholds and Very High Label.

Threshold Warning View

Class	CBO	WMC	DIT	NOC	LCOM	NOM	NOF	LOC
simulation.Simulator	3	6	1	0	1	2	0	55
exceptions.PMFailureException	0	1	3	0	0	1	1	11
interfaces.Remote	1	3	1	0	3	3	0	7
infrastructure.PM	3	29	1	0	152	24	8	141
monitoring.TimeSeriesMonitor	11	4	2				6	93
infrastructure.Address	0	8	1				2	31
constant.HandType	0	0	1				0	4
simulation.FailureSimulator2	5	106	1				12	326
infrastructure.Edge	5	32	1				7	166
simulation.SimulationRun	9	39	1				11	228
interfaces.LocationElement	0	3	1				0	6
simulation.MoveSimulator	2	14	1				3	61
monitoring.EnergyMonitor	3	9	1				5	61
comparables.ComparableEdge	2	8	1				2	24
infrastructure.RemoteClient	3	7	1	0	0	7	2	29
constant.SimuType	0	0	1	0	0	0	0	4

TWarning
Select your project domain
Games
Select threshold labels to be colored
☑ Very High
☐ High
☐ Low
☐ Very Low

Figure 3.3. TWarning View and Project Properties.

The Threshold Warning View presents all project classes and the respective measurements. Each measurement receives a color when the associated measured value does not match with the configured thresholds. We adopted the following color scheme to correlate the measured values with each category of thresholds (Very High, High,

Low or Very Low). The dark blue color denotes the values that do not exceed Very Low threshold. The light blue color, in turn, represents the values that exceed the Very Low threshold. For instance, the class *constant.HandType* has a value for LOC (4 LOC) that is lower than the Low threshold (considering the Game domain) but do not exceed the Very Low threshold. Observe in Figure 3.3 that, in this case, the respective value 4 LOC was colored in dark blue. On the other hand, the light red color characterizes the measured values, which exceed the Very High thresholds, while the dark red color represents the values that exceed High threshold but do not exceed the Very High threshold. As an example, the class *simulation.FailureSimulator2* has a value for LOC metric (326 LOC) higher than the Very High threshold considering the Game domain. Therefore, in the Threshold Warning View, this value was colored as light red. The tool can be find in the supplementary website[1].

3.3 Qualitative Evaluation

A previous study described eight desirable points for methods to derive metric thresholds [Vale and Figueiredo, 2015]. These points motivated our method proposal (see Section 2.3). Another previous study compared five related methods present in the literature [Vale et al., 2018]. This section complements these previous studies by comparing our method to the two most similar threshold derivation methods, namely Alves' and Vale's methods. We extended the previous comparison including the proposed method and the points highlighted in Section 2.3 related to the intrinsic characteristics of software domains.

Table 3.1 highlights the main differences of Alves' and Vale's methods and the proposed method. All three derivation methods - Alves', Vale's, and the proposed method - recommend percentages to derive thresholds. However, the users must decide if they use the recommended percentages or not and, because of that, they are considered partially deterministic.

The methods in Table 3.1 also consider the metric distribution and they identify step-wise outliers. Another similarity is that they all agree on the impact of entities and systems, and they have tool support. On the other hand, only Alves' method correlates metrics for the threshold derivation. That is, it weights every metric by lines of code (LOC), using the labels: low (between 0 and 70%), moderate (70 - 80%), high (80 - 90%), and very high (>90%).

[1] Available at: https://github.com/Allan045/TWarning

Table 3.1. Qualitative Evaluation of the Proposed Methods to Derive Thresholds

Question	Methods		
	Alves	**Vale**	**Proposed**
Is it deterministic?	Partially	Partially	Partially
Are step-wise outliers identified?	Yes	Yes	Yes
Is the metric distribution considered?	Yes	Yes	Yes
Does the number of entities impact?	Strong	Strong	Strong
Does the number of systems impact?	Weak	Weak	Weak
Does it correlate with other metrics?	Yes	No	No
Lower bound thresholds?	No	Yes	Yes
Does it provide tool support?	Yes	Yes	Yes
Does it explicitly consider software domain?	No	No	Yes
Is it iterative over benchmarks?	No	No	Yes

Related to the impact of the number of systems and entities, unlike Alves' method, Vale's and the proposed methods consider the number of entities more important than the number of systems. The proposed method calculates lower bound thresholds, as Vale's method does. Furthermore, all methods have tool support since a recently developed tool, named TDTool [Veado et al., 2016], supports both Alves' and Vale's methods. Our method is also supported by an open source tool (Section 3.2), named TWarning, which extends the Eclipse IDE to show thresholds warnings.

One of the main difference between the proposed method and others is the classification of systems into domains. It leads to thresholds that consider specific characteristics of the systems, such as the intrinsic complexity of geo-localization systems [Mori et al., 2018]. On the other hand, the other evaluated methods let it to the user when they are building their benchmark. In addition, neither Alves' nor Vale's methods mentioned software domains when describing their method. Hence, we consider that Alves' and Vale's methods do not explicitly consider them. Another difference of the proposed method and the two evaluated methods is that the proposed method is iterative in the benchmarks. That is, since more than one benchmark is expected to be available (e.g., one for each domain), the proposed method iterates over the benchmarks in order to calculate domain-specific thresholds for each metric.

3.4 Concluding Remarks

This chapter proposed a method to derive domain-sensitive metric thresholds. The method derives thresholds with a classification step to consider the system domain in order to compose the benchmark. We argue that this step is important to more appropriate to derive thresholds in a domain-specific context. Inspired by a previous

method [Vale et al., 2018], the proposed method calculates upper and lower thresholds, it is systematic and it guarantees the statistical properties of the metrics.

We also perform an qualitative evaluation comparing characteristics of the proposed method with 2 other methods. In this analysis we highlight the differences of our method to two recently proposed methods, including explicitly differentiation of domains from different domains when building benchmarks. In addition, a tool to support the proposed method. TWarning, an Eclipse plug-in to detect warnings based on domain-sensitive metric thresholds. We described the architecture of the tool and its main features. Our tool provides a straightforward approach to help the developer write better code from start by providing a view with class metrics and warnings considering the software domain. In the next chapter we quantitatively evaluate this method for mobile domain benchmarks.

Chapter 4

Evaluation of Mobile Domain Thresholds

The method proposed can be applied in different ways, such as using SIG quality model [Heitlager et al., 2007], using metrics individually, or using a metric-based detection strategy [Marinescu, 2004]. This chapter illustrates an example of use of the proposed method in a benchmark composed of mobile systems. Next chapter presents a similar evaluation with enterprise systems. Before the method being applied, it is necessary to measure the systems. CK Metrics tool [CK Metrics, 2018] is used to extract the software metrics and then we applied the method to obtain domain-sensitive thresholds used in this evaluation. Section 4.1 presents our research questions and experimental steps. Section 4.2 explains how we built the benchmarks. Section 4.3 presents the thresholds derived by our method for the subject metrics (Section 2.2) using the benchmarks (Section 4.2). Section 4.4 presents the variation of thresholds for different mobile domains. Section 4.5 compares our method results using the coefficient of variation. Section 4.6 discusses the domain similarity and differences by clustering the thresholds in levels. Finally, Section 4.7 concludes this chapter and revisits some of the results.

4.1 Research Questions and Experimental Steps

This evaluation aims to investigate whether metric thresholds vary across mobile systems of different software domains and how they vary across domains. We assume that mobile systems from distinct domains have some diverse characteristics, affecting the systems and measures, so it may impair the derived thresholds. We defined four research questions (RQs) as follows.

RQ1. Do thresholds for the same metric vary among different mobile domains?
RQ2. How much thresholds vary between domains?
RQ3. Are there similarity between thresholds levels regardless of the mobile domain?

Regarding the empirical steps, Fig. 4.1 presents an overview of this study described below. First, we built our dataset by mining open-source systems from Fossdroid [Fossdroid, 2018] (Step 1). We then measured the source code of each system using CK Tool [CK Metrics, 2018] (Step 2). Once measured all systems, we derived metrics thresholds with domain threshold method(Step 3). Finally, we analyzed the results and evaluated domain-specific thresholds (Step 4).

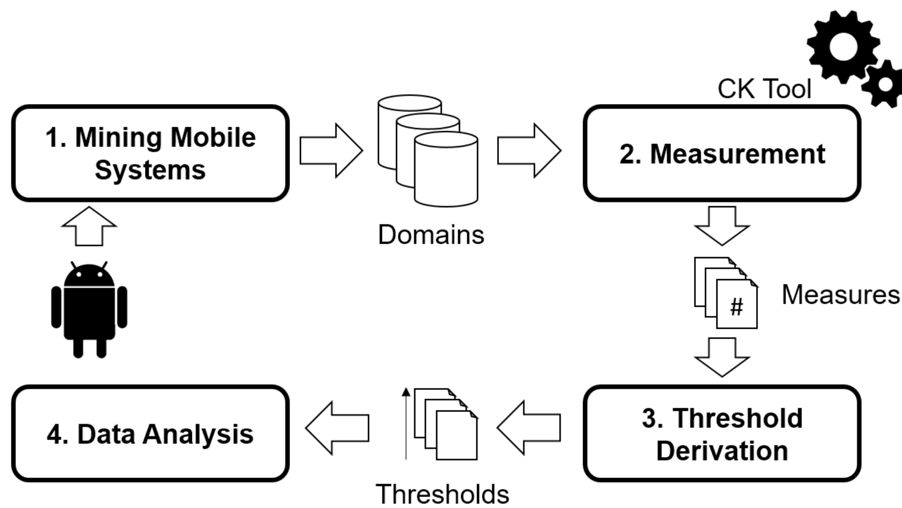

Figure 4.1. Experimental steps for the analysis of mobile domain thresholds.

4.2 Benchmark of Mobile Systems

We mined ten mobile systems for each of the ten different software domains to build the domain benchmarks. We choose mobile systems to compose our benchmarks because: (i) mobile systems have been increasingly used; and (ii) these type of system (i.e., mobile applications) are often not explored by other studies to derive metric thresholds [Alves et al., 2010]. To build the domain-benchmarks, we relied on a repository of open-source Android mobile systems, namely Fossdroid [Fossdroid, 2018]. We performed three steps to select the mobile systems. First, we decided which categories consider as domains, based on close names of domains presented in other studies [Hubbard, 2014; Mori et al., 2018; Murphy-Hill et al., 2014]. We chose the following categories: Connectivity, Development, Games, Internet, Money, Navigation, Phone & SMS and Reading. Table 4.1 presents and describes the 10 mobile software domains explored in

this study. Then, for each category, we selected the ten most popular mobile systems. Popularity on Fossdroid is measure with amount of access and downloads. Third, we manually verified whether each system source code is available on GitHub [Ray et al., 2014]. We also investigated the existence of the selected system on Google Play Store. We argue that systems on Play Stores in general have their domain verified. It is important to note that the category of a mobile systems was defined by its developers. The list of one hundred systems that compose our benchmarks can be find in the supplementary website[2].

Table 4.1. Description of Mobile Software Domains

Domain	Description
Connectivity	Mobile systems that synchronize information between server and client.
Development	Systems that implement simple development tools (i.g. code editor).
Games	Mobile systems that provide entertainment.
Internet	Browsers and Web related systems.
Money	Systems that provide vision and account management
Navigation	Mobile systems that access geolocation sensors and show map information.
Phone & SMS	Systems that manage phone calls and exchange SMS messages.
Reading	Reader applications, dictionaries and text related systems.
Science & Education	Mobile systems to share science knowledge.
Sports & Health	Systems to track the user health data and timers.

4.3 Derived Thresholds

This section presents the derived thresholds we obtained using the proposed domain-sensitive method (Chapter 3). Table 4.2 illustrate the method in practice based on the seven metrics presented in Section 2.2 and on the benchmarks presented in Section 4.2. Only the key percentage values of the proposed method are presented. For example, the proposed method presents five labels, but these labels are established in four percentages, like related work did [Alves et al., 2010; Vale et al., 2018]. Hence, Table 4.2 shows just the values that represent the percentages. This table should be read as follows: the first row of each partition represents the domain, the first column shows the different domain labels and the other columns indicates the thresholds of LOC, NOA, NOM, WMC, LCOM, CBO, and DIT, respectively. For example, the labels for CBO to the Connectivity domain are defined for very low (0 - 3%), low (3 - 15%), moderate (15 - 90%), high (90 - 95%), and very high (>95%) as the intervals 0 - 1; 1 - 3; 3 - 23; 23 - 31, and >31, respectively.

[2] Available at: https://github.com/Allan045/MSc-Data

Table 4.2. Mobile Threshold from the Proposed Method

	LOC	**NOA**	**NOM**	**WMC**	**LCOM**	**CBO**	**DIT**
Connectivity (Conn.)							
0-3%	11	0	1	1	0	1	1
3-15%	31	0	2	2	0	3	1
90-95%	227	9	17	43	103	23	3
>95%	309	14	25	67	204	31	4
Development (Dev.)							
0-3%	17	0	0	0	0	0	1
3-15%	34	0	2	3	0	3	1
90-95%	259	14	16	52	86	27	4
>95%	376	20	22	75	197	35	6
Games (Gam.)							
0-3%	6	0	1	1	0	0	1
3-15%	21	0	2	3	0	2	1
90-95%	369	19	27	82	231	27	4
>95%	591	29	41	144	496	35	5
Internet (Int.)							
0-3%	7	0	1	1	0	1	1
3-15%	23	0	2	3	0	3	1
90-95%	399	16	26	87	231	32	3
>95%	606	27	41	140	636	45	4
Money (Mon.)							
0-3%	7	0	0	0	0	0	1
3-15%	16	0	1	2	0	3	1
90-95%	252	11	17	46	88	27	3
>95%	391	17	24	74	190	36	4
Navigation (Nav.)							
0-3%	12	0	1	1	0	0	1
3-15%	36	0	3	4	0	3	1
90-95%	498	21	32	125	406	38	4
>95%	782	33	48	195	1017	52	5
Phone & SMS (Phone.)							
0-3%	12	0	1	1	0	0	1
3-15%	27	0	1	2	0	2	1
90-95%	287	13	21	63	152	24	3
>95%	439	22	29	104	282	31	4
Reading (Read.)							
0-3%	7	0	1	1	0	0	1
3-15%	18	0	1	2	0	2	1
90-95%	380	15	24	91	246	31	4
>95%	585	22	32	142	605	45	5
Science & Education (Edu.)							
0-3%	12	0	0	0	0	0	1
3-15%	28	0	1	2	0	1	1
90-95%	467	17	26	85	219	29	3
>95%	733	26	36	155	516	40	4
Sports & Health (Sport.)							
0-3%	11	0	1	1	0	0	1
3-15%	32	0	2	3	0	3	1
90-95%	255	13	18	46	90	25	3
>95%	376	18	24	74	217	33	4

4.4 RQ1: Variation of Thresholds for Different Domains

This section quantitatively evaluates the proposed method aiming to investigate if thresholds for the same metrics vary on different software domains (RQ1). For this evaluation, we compare the difference between the higher and the lower thresholds. Table 4.3 presents differences between thresholds for each metric, the lowest to the highest thresholds among the 10 domains of Table 4.2. It also shows, between parentheses, how many times the highest threshold is greater than the lowest threshold. For instance, the lowest 95% threshold for the LOC metric is 227 (see Connectivity in Table 4.2), while the highest 95% threshold for the LOC metric is 498 (see Navigation in Table 4.2). The difference is that the Connectivity threshold for LOC is 2.2 times higher than the Navigation threshold in this case. Results in Table 4.3 show that the investigated thresholds largely vary for all metrics; except Depth of Inheritance Tree (DIT), which have smaller range. Apart from DIT, all metrics have a significant variation in the 90% and 95% thresholds; i.e., higher than 1.5 times. The metrics with higher threshold variations are LCOM and WMC. The rate between the highest and the lowest thresholds in a domain is about 3 to 5 times higher for these two metrics. This large difference between the highest and lowest thresholds suggests that LCOM and WMC are highly sensitive to the software domain. Therefore, developers should be aware of this variation when using thresholds for these metrics in software quality evaluation. The rate between the highest and the lowest thresholds for LOC, NOA and NOM is more than 2 times. This difference suggests that these metrics are not as sensitive to the software domain as LCOM and WMC. Yet, a difference of twice can be considered large, depending on the evaluation goal. The CBO metric is an interesting case. The threshold of CBO presented a 1.7 times variation among domains for 90% and 95%. For instance, the 95% thresholds of CBO varied between 31 and 52 (1.7 times) and most of times, 6 out of 10 domains, varied between 31 to 36. However, CBO thresholds vary sharply (3 times) in the low values (15%).

4.5 RQ2: Coefficient of Variation for Thresholds

In this section, we answer the second research question. We compare our method results using the Coefficient of Variation (CV) to determine the volatility of a threshold value as a percentage. Coefficient of variation is the ratio of the standard deviation to the mean. The higher the coefficient of variation, the greater the level of dispersion

Table 4.3. Difference between Highest and Lowest Thresholds

%	LOC	NOA	NOM	WMC	LCOM	CBO	DIT
3	17 - 6 (2.8)	0 - 0 (n/a)	1 - 0 (n/a)	1 - 0 (n/a)	0 - 0 (n/a)	1 - 0 (n/a)	1 - 1 (1.0)
15	36 - 16 (2.3)	0 - 0 (n/a)	3 - 1 (3.0)	4 - 2 (2.0)	0 - 0 (n/a)	3 - 1 (3.0)	1 - 1 (1.0)
90	498 - 227 (2.2)	21 - 9 (2.3)	32 - 16 (2.0)	125 - 43 (2.9)	406 - 86 (4.7)	38 - 23 (1.7)	4 - 3 (1.3)
95	782 - 309 (2.5)	33 - 14 (2.4)	48 - 22 (2.2)	195 - 67 (2.9)	1017 - 190 (5.4)	52 - 31 (1.7)	6- 4 (1.5)

around the mean. That is, more distinct are the metric thresholds. While the data standard deviation must always be understood in the context of the data means, the value of CV is independent of the unit in which the measurement was made. As a result, it is a dimensionless number and allows comparison between distributions of values whose scales of measurement are not comparable or have very different means (i. e., different domains). For instance, this coefficient relates the standard deviation of thresholds from different domains to the value of this estimate. The higher the value of the coefficient of variation for thresholds, more distributed are the data, making necessary a domain analysis as our method proposes. Table 4.4 presents the Coefficient of Variation for thresholds using seven metrics and ten domains.

Calculating the mean of 95% LOC threshold for all domains is 518, which is near to the Reading threshold (591). This label (95%) has a coefficient of variation of 31.5%. Using the relative deviation and summing it with the mean, we obtain the value 681 for LOC threshold (518 + 31.5% = 681). This value is similar to Internet threshold (606). Decreasing the coefficient of the mean, we find the threshold value 355, which is close to Sports & Health threshold (376). Similar cases occur with other coefficients in Table 4.4 and, so, it leads to consider domain in threshold analyses more appropriate.

Table 4.4. Coefficient of Variation for Domains Thresholds

CV	LOC	NOA	NOM	WMC	LCOM	CBO	DIT
3%	33,6%	(n/a)	69,0%	69,0%	(n/a)	210,8%	(n/a)
15%	25,8%	(n/a)	39,7%	26,9%	(n/a)	28,3%	(n/a)
90%	26,6%	20,5%	23,3%	34,6%	53,5%	14,8%	15,3%
95%	31,5%	25,9%	27,8%	37,7%	62,1%	18,3%	15,7%

4.6 RQ3: Similarity among Domains

This section discusses the third research question, which investigates about domains similarity thresholds across domains. To answer this question, we cluster the thresholds in three levels (low, medium, and high) to find similarity between domain thresholds. Table 4.5 presents the similarity analysis of thresholds across different domains with respect to each metric. The main reason for this analysis is to identify whether and which domains can be grouped to promote larger benchmarks. In addition, if different domains have similar thresholds for the same metric, it means that some metric thresholds can be reused across domains [Fowler et al., 1999].

For this analysis, we cluster thresholds for each metric into three levels according to how far from the mean these values were. Table 4.5 uses gray scale to indicate domains (rows) with low, medium, and high thresholds for each metric (columns). In Table 4.5, light gray boxes mean lower thresholds, gray boxes mean medium thresholds, and dark gray boxes mean high thresholds. Based on Table 4.3, we jointly analyzed both the 90% and 95% thresholds to determine if the metric threshold is low, medium, or high for a domain. For instance, Navigation domain has high thresholds for 6 metrics (LOC, NOA, NOM, WMC, LCOM, and CBO), and a medium threshold for DIT.

It is interesting to observe that Connectivity (Conn.) and Sports & Health (Sport.) domains have the same levels of thresholds for all seven metrics (i.e., same colors in corresponding boxes). This result means that these two domains are very similar in terms of measurement. Therefore, if someone derives thresholds for a set of Connectivity systems, these thresholds are expected to be reliable to be used for code analysis of Sports & Health mobile systems, for instance. This similarity was unexpected, but looking closer the systems in these domains we found functionalities they have in common. One reason for this similarity is because both domains tend to have systems that synchronize data with server. For instance, Connectivity systems synchronize time, notifications and localization, and Sport & Health systems track the user information and localization to synchronize this data with server. In addition, they are usually simple and smaller systems as indicated by low threshold for all metrics.

Other pairs of similar domains in terms of metric thresholds are: (i) Navigation and Internet, (ii) Science & Education and Games, and (iii) Reading, Phone & SMS and Money. The explanation for similar thresholds vary in a case by case basis. For instance, Navigation (Nav.) and Internet (Int.) in our dataset have some of the largest and more complex mobile systems. Therefore, these domains share high thresholds for most metrics. On the other hand, Science & Education (Edu.) and Games (Gam.) tend to be heterogeneous, but both encompass complex algorithms as indicated by

high WMC. For instance, Games often use massive computation for entertainment and leisure purposes, while Science & Education must support a variety of knowledge and make then accessible for a large range of interests. An interesting similarity is between Reading (Read.) and Phone & SMS (Phone) systems. Although these domains deal with different requirements, we observed that they share some similar levels for most of the size metrics. For instance, Reading and Phone & SMS systems have LOC, NOA, NOM and WMC with medium level. On the other hand, LCOM, CBO and DIT are medium level for Reading and low level for Phone & SMS. This characteristic makes them have similar levels in terms of size metrics and thresholds. This explanation can also be given to the similarity between Money (Mon.) and Development (Dev.), but they have low level for almost all metrics.

Table 4.5. Metrics Levels for 90% and 95% thresholds

	LOC	**NOA**	**NOM**	**WMC**	**LCOM**	**CBO**	**DIT**
Nav.							
Int.							
Edu.							
Gam.							
Read.							
Phone							
Dev.							
Mon.							
Conn.							
Sport.							

4.7 Concluding Remarks

In this chapter, we evaluated the method to derive domain metric thresholds. We validated the proposed method through quantitative analyses with mobile systems. We manually mined one hundred mobile systems from GitHub and from ten distinguished domains and analyzed the results. Furthermore, our quantitative analyses indicate that thresholds vary for each domain and that the majority of the analyzed metrics are domain-sensitive. By analyzing the coefficient of variation for domain thresholds, the lower variation was about 15% on 90% for DIT, which was expected due smaller range for this metric, and the higher variation was for LCOM with 95% (62%). In addition, clustering domains with levels using the 95% thresholds range our results indicate some domains share similarities. In chapter 6 we present some potential threats to validity and the main actions we have taken to mitigate their impact on the research results.

In the next chapter, we evaluate this method with benchmarks of desktop and Web systems.

Chapter 5

Evaluation of Domains Thresholds with Enterprise Systems

This chapter evaluates desktop software systems from many domains. Section 5.1 describes our research questions and experimental steps to evaluate desktop software domains. Section 5.2 explains how we built our benchmarks. Section 5.3 describes how we perform the measurement and domain metric threshold derivation. Section 5.4 presents the metric thresholds. Sections 5.5 to 5.7 discuss the derived domain thresholds. Section 5.8 evaluates whether domain thresholds are better to find code smell than generic thresholds. Finally, Section 5.9 concludes this chapter with our main results.

5.1 Research Questions and Experimental Steps

This section presents the research questions to investigate: (i) whether metric thresholds vary across systems of different software domains, (ii) if metric thresholds vary across systems of different sizes, and (ii) if domain-specific thresholds are better than generic thresholds for detecting code smells. We assume that some characteristics of each domain affect the systems and measures, so they may impair the derived thresholds. Given this assumption, we defined four research questions (RQs) as follows.
RQ1. Do thresholds for the same metric vary among different software domains?
RQ2. Are there metrics with the same thresholds regardless of the software domain?
RQ3. Does the system size impact on the derived thresholds?
RQ4. Are domain-specific thresholds better than general thresholds for detecting the God Class code smell?

We expect our findings to provide evidence that software domains and the system size should be considered when building benchmarks, for instance, to identify code anomalies. For example, inaccurate thresholds may influence negatively the derived metric thresholds by providing meaningless values about singular software domain. Regarding the empirical steps, Figure 5.1 presents an overview of this study described below.

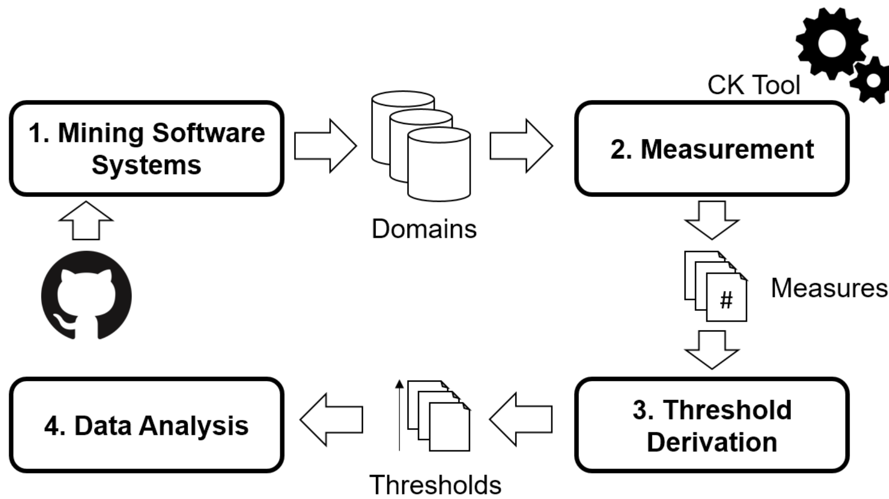

Figure 5.1. Experimental steps for thresholds analysis.

First, we built our dataset by mining open-source systems from GitHub (Step 1). We then measured the source code of each system using CK Tool [CK Metrics, 2018] (Step 2). Once measured all systems, we derived metrics thresholds (Step 3). Finally, we analyzed the results and evaluated the effectiveness of domain-specific thresholds to detect code smells compared to generic thresholds (Step 4). The first three steps are described in the following two sections and the data analysis is described in Sections 5.4 to 5.8.

5.2 Selected Systems and Domain Classification

Table 5.1 presents and describes the 15 domains explored in this study with the number of systems per domain (last column). We choose these domains for the following reasons. First, they are well-defined in terms of requirements and, most of these domains have been used in previous studies [Ferreira et al., 2012; Linares-Vásquez et al., 2014; Murphy-Hill et al., 2014], therewith, we believe that they are representative. Second,

they encompass several types of systems (e.g., frameworks and tools). Third, there is a significant number of systems in these domains publicly available in GitHub.

Table 5.1. Description of Enterprise Software Domains

Domain	Description	Systems
Accounting (Acc)	Systems that record and process accounting transactions, such as accounts payable and receivable, payroll, and trial balance.	216
Business (Bus)	Systems that implement validation, calculation, and law regulations of business requirements, such as pricing, and inventory management.	368
Communication (Com)	Systems that manage connections between server and clients, using protocols to share information.	266
Development (Dev)	Software tools that support developers to implement projects in general.	528
Dictionaries (Dic)	Software tools used to translate a variety of languages.	130
E-commerce (EC)	Systems in charge of supporting the transactions of products buying and selling, as well as providing services to consumers.	34
Education (Edu)	Systems used by students to manage their study life and by school managers to administrate their schools.	165
Free Time (FT)	Entertain systems or applications which provide information for joy, such as travel information.	28
Games (Gam)	Entertainment games that can be played alone or in collaboration.	452
Health (Hea)	Systems that offer health-related services to people in general.	279
Home (Hom)	Systems that control basic many home devices and services.	160
Localization (Loc)	Show local information normally based on GPS. Some of them display maps and location sensitive data for users.	70
Messaging (Mes)	Systems that allow the users to send messages from one client to another.	66
Restaurant (Res)	Systems that provide different services for both managers and users of food houses.	326
Science & Engineering (ScE)	Systems designed to aid users in several fields of science and engineering, such as 3D visualization and data analysis.	19

Mining and Classification. We mined systems from GitHub to compose our dataset in October 2017. For each domain, we collected up to 1,000 systems by automatically searching the project name and description for keywords that match the domain name. For instance, we collected 34 e-commerce systems by searching for "e-commerce" or "ecommerce". Whenever more systems have been returned by the GitHub search, we selected the first 1,000 systems in the descending order of stars. In GitHub, stars are a meaningful measure for repository popularity. We developed and iteratively tested a script that automatically searches relevant systems and applies a set of criteria. For instance, we excluded systems with less than 1,000 lines of code because we considered them toy examples or incipient software projects. In addition, we focus in Web and Desktop-based Java systems and, therefore, we removed other projects (e.g., mobile applications) because they tend to have a different architectural design. Finally, we manually validated the collected systems by checking the $projectname$ and $readmefile$ and excluded the wrongly classified systems. As a result, the dataset used in this study includes 3,107 software systems in the 15 domains, with at least 19 systems per domain (Science & Engineering). The list of systems that compose our benchmarks can be find in the supplementary website[3].

[3]https://github.com/Allan045/MSc-Data

Table 5.2 summarizes the descriptive statistics of our dataset focusing on the source lines of code per domain (excluding code of test cases). The software systems diverge largely in lines of code, while the smallest software system in every domain has about 1,000 LOC (due to the criteria described above), the largest system by domain vary from about 26,633 LOC in the E-Commerce domain to over 7 MLOC in the Business domain. We can also observe a considerable variance (i.e., standard deviation) and that data do not follow a normal distribution. In fact, all domains follow a right skewed distribution for lines of code. This considerable difference among domains corroborates with our assumption that systems in one domain might be more complex than systems in other domains, for instance. Therefore, metric thresholds should be tailored to each specific domain.

Table 5.2. Lines of Code per System Domain

Domain	Min	Max	Mean	Median	SD
Acc	1,003	1,938,805	83,358.40	4,984.00	308,705
Bus	1,011	7,822,498	130,576.84	6,402.50	775,924
Com	1,006	1,351,323	37,604.28	5,042.00	125,571
Dev	1,153	1,498,869	69,427.47	16,931.50	161,548
Dic	1,016	472,642	14,307.52	3,556.00	47,759
EC	1,104	26,633	6,298.76	3,133.50	7,340
Edu	1,025	1,170,720	26,440.87	5,204.00	102,034
FT	1,047	28,422	45,509.96	4,511.00	84,510
Gam	1,009	546,748	17,989.02	4,877.50	49,593
Hea	1,025	1,446,807	27,161.00	4,790.00	109,779
Hom	1,022	299,898	29,538.95	9,119.50	53,988
Loc	1,054	859,393	40,062.38	12,015.50	110,307
Mes	1,000	160,297	14,648.90	3,891.00	29,912
Res	1,063	151,814	7,639.50	2,951.50	18,216
ScE	1,028	439,747	36,939.84	8,649.00	100,668

Table 5.3. Number of Classes per System Domain

Domain	Min	Max	Mean	Median	SD
Acc	3	10,125	349.07	50.00	1061.97
Bus	1	12,470	471.92	65.50	1624.18
Com	3	8,379	249.99	49.00	753.12
Dev	3	8,735	472.20	167.50	962.62
Dic	4	2,296	87.76	30.00	234.74
EC	7	301	80.52	46.00	83.86
Edu	2	5,641	225.33	66.00	586.22
FT	10	2,128	296.67	42.00	564.69
Gam	1	2,439	116.40	45.00	221.61
Hea	3	5,322	186.50	51.00	592.33
Hom	4	2,798	238.14	83.50	426.75
Loc	4	4,041	222.97	57.50	540.42
Mes	1	980	100.00	44.50	192.06
Rest	4	1,263	62.31	35.00	114.14
ScE	12	1,528	184.42	78.00	358.14

Similar to Table 5.2, Table 5.3 shows descriptive statistics for the number of classes in systems of each domain. In general, we can observe a considerable difference among systems inside a domain. For instance, while the median is 167.50 classes in the Development domain, the largest system in this domain has more than 8K classes. Considering the median per domain, Development has the largest systems in terms of classes and Dictionary has the smallest ones. On the other hand, the largest system, in terms of number of classes, belongs to the Business domain, containing more than 12K classes. Based on these observations, it is also expected that the variation among domains reflects in software metrics and in their thresholds.

5.3 Measurement and Threshold Derivation

We investigate domain-specific thresholds using eight metrics, we added Number of Children metric to compare with Depth of Inheritance Tree. We used a measurement tool to compute the source code metrics, [CK Metrics, 2018]. CK tool measures Java programs by means of static analysis and it supports all metrics used. To derive domain thresholds we follow the method presented in Chapter 3. In this chapter, we focus only on top-threshold values (i.e., 90% and 95%) because the differences are higher for these labels making the analysis more applicable as we noted in Chapter 4.

5.4 Desktop Domain Thresholds

This section presents the derived thresholds for each domain on the dataset presented in Section 5.2. Table 5.4 shows the threshold values for 90% and 95% for each software metric and domain analyzed. We classify domains with similar characteristics into four categories to make our discussions more direct and to support us answering RQ1, RQ2 and RQ3 in the next sections. The following topics describe the four categories.

Table 5.4. Metric Threshold per Domain Group

		All High			High Size				High Complexity				All Low			
Metric	**%**	**Acc**	**Bus**	**ScE**	**Gam**	**Hea**	**Mes**	**Rest**	**Dev**	**FT**	**Dic**	**Loc**	**Com**	**EC**	**Edu**	Hom
LOC	90	541	527	456	310	289	296	255	287	314	337	345	303	160	238	248
	95	981	928	718	491	442	447	382	450	464	575	589	467	232	378	403
NOA	90	14	12	11	11	10	11	12	8	9	11	10	10	8	9	8
	95	21	21	19	18	16	18	19	13	14	17	17	16	11	14	13
NOM	90	34	30	23	20	19	18	18	19	19	21	21	18	17	19	17
	95	46	53	36	31	29	27	25	29	29	33	31	26	24	29	27
WMC	90	123	98	88	59	43	52	41	51	49	63	75	52	30	43	44
	95	236	232	148	100	73	81	62	86	80	111	132	92	49	73	76
LCOM	90	317	253	148	120	136	126	134	117	104	136	159	85	91	120	91
	95	741	742	392	330	351	344	256	302	276	405	378	219	218	311	248
CBO	90	15	16	16	13	15	12	11	15	16	15	15	14	13	14	13
	95	23	23	23	18	20	18	15	22	21	22	21	20	17	20	19
DIT	90	3	4	4	4	3	5	6	4	4	5	4	4	2	3	3
	95	5	5	5	5	5	6	6	5	5	6	5	4	2	4	4
NOC	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	95	0	0	1	1	0	0	0	1	1	1	1	0	0	0	0

All High thresholds. As the name suggests, this category present high thresholds for all analyzed metrics when comparing to derived thresholds of all domains studied. Columns 3 to 5 of Table 5.4 present the thresholds for the three domains (Accounting, Business and Science & Engineering) that composes this category. These data show that the 5% largest classes (i.e., 95% threshold), in Accounting systems have 981 or more lines of code. Interestingly, the 95% threshold for Business is very similar to Accounting, that is, 928 lines of code. We speculate that the similarity of Accounting and Business systems might be because these domains involve heterogeneous systems in broader trade fields. In addition, systems in this category tend to be large and complex. For instance, 10% classes in Accounting systems with the highest lack of cohesion (i.e., 90% threshold) present LCOM equals or higher than 317. This value is the highest 90% threshold for LCOM among all domains. The reason for this lack of cohesion in Account systems might be due to the several unrelated functionalities controlled by this kind of systems. For instance, Account systems deal with several changes of tax and rules. We believe that these types of functionalities are loosely coupled, which might result in low cohesive systems. In the Science & Engineering domain, thresholds

are not the highest ones for any metric, compared to the other domains of this category. However, these values are still high for LOC, NOM and WMC compared to domains in other categories (see Table 5.4). In fact, Science & Engineering systems usually involves complex or extensive computations increasing lines of code and complexity. Therefore, these systems naturally belong to this category.

High size thresholds. This topic discusses thresholds derived for domains with large classes (i.e., many Lines of Code and Number of Methods), but low complexity (i.e., few Weighted Method per Class and Coupling Between Objects) when compared to the systems of our dataset. This category includes four domains: Games, Health, Messaging, and Restaurant. Table 5.4 also shows the threshold values for these four domains in columns labeled High Size Thresholds. Comparing thresholds of the previous category with the ones in this category, we observe that thresholds do not vary much for the size metrics (LOC, NOA, and NOM). For instance, all eight domains in these categories have 18 attributes or more for the 95% NOA metric threshold; the exception is Health systems with a 16-attribute cut for 95% threshold. In general, systems in these categories also have more lines of code than systems in the other two categories (see next topics). For instance, apart from Restaurant, the systems from this category show more than 382 LOC for the top-5% largest classes. Classes in Restaurant systems are not much large in terms of LOC, but they have high threshold values for the other size metrics (i.e., NOA and NOM). Hence, we decide to classify this domain in this category instead of the last one (low thresholds).

Systems in this category are not highly complex in terms of coupling (CBO), cohesion (LCOM), and weighted methods by cyclomatic complexity (WMC) when compared to the first category. For instance, while the 95% thresholds of LCOM are above 600 for three out of four domains in the previous category, they are below 400 for four domains in this category. These results can be explained by the fact that systems in this category usually involve simple, yet large, functionalities. In addition, some of these domains, such as Messaging and Restaurant, usually involve a clear set of simple requirements; i.e., message and media exchange for messaging systems or order registration and inventory management for restaurant systems.

Health and Games systems have some particularities because they present high coupling and high cyclomatic complexity, respectively, compared to the other domains in this category. We speculate that Health systems present higher CBO thresholds because they might involve several cases (e.g., many symptoms) to reach a conclusion (e.g., a disease). Therefore, these systems seem highly coupled to other classes, such as domain-specific API classes, using an initial code to provide the basic structure and requirements. In the Games domain case, the result is expected since games usually are

computationally extensive involving long if-then-else statements (or worst, long switch-case statements) [Murphy-Hill et al., 2014]. In fact, we verified that methods in games are commonly large and complex, although each class has few methods. Therefore, this common practice for gaming development contributes to higher WMC values, but lower values for LCOM and NOM.

High Complexity. The current category includes domains with small classes, yet high complexity. Four domains fall into this category: Development, Free Time, Dictionary, and Localization. Columns labeled High Complexity in Table 5.4 present the thresholds derived for systems of this category. Software systems in this category usually have classes with high thresholds for complexity metrics, although thresholds are not very high for size metrics. For instance, the 5% largest classes in systems of this and the previous category have about 450 or more lines of code. Hence, they have similar size in terms of LOC. On the other hand, systems in this category are more complex than systems of the previous category at least in CBO metrics. If we observe the 95% thresholds for CBO, for instance, we see that these values are always higher than 21 in this category and lower than 21 in the previous category.

It is interesting to observe, however, that Dictionary and Localization domains have some commonalities since they present similar variation of thresholds for all metrics in general. For example, the 10% and 5% classes with the highest complexity in Dictionary domain present WMC very similar to the ones in Localization domain; That is, around 63 and 111, respectively. In contrast, despite being in the same category, Development systems often have lower thresholds than Dictionary and Localization systems for all metrics. We decided to classify Development in the current category, instead of the last one, because systems in this domain seem more complex than systems in the last category for at least two out of three complexity metrics (WMC and CBO).

All Low Thresholds. This category consists of domains with low thresholds for most of the eight metrics compared to the other domains and categories. It is composed by four domains (Communication, E-Commerce, Education, and Home) presented in the last four columns of Table 5.4. In particular, size metrics have lower thresholds with respect to the first and second categories, while complexity metrics have lower thresholds compared to the first and third categories. Communication and Home domains have larger systems in terms of LOC than E-Commerce and Education, the other two domains in this category. For instance, the first two domains have more than 403 LOC defined for the 95% threshold, while the last two have 232 LOC and 378 LOC, respectively. This observation suggests that E-Commerce and Education share the common characteristics of holding the smallest classes in terms of LOC considering

all 15 analyzed domains.

Although not as small and simple as E-Commerce and Education, the Communication and Home domains also have small and simple classes. Classes in these domains have a particularly small number of methods and attributes, when compared to other domains. For instance, focusing on the 90% thresholds in Table 5.4, we observed that Communication and Home domains have classes with up to 18 and 17 methods (NOM), respectively. Their 90% threshold values are 10 and 8 attributes (NOA), respectively. These two domains also have simple classes in terms of WMC and LCOM. With respect to coupling, these domains are in the middle; i.e., neither the lowest nor the highest thresholds for CBO. This result is somehow expected for Communication because systems in this domain manage relationship about different sub-systems, such as in a Client-Server architecture [Buschmann et al., 1996].

5.5 RQ1: Threshold Variation across Domains

First, we investigate if thresholds for the same metrics vary on different software domains. If so, it supports our assumption that software domains should be considered when building benchmarks for metrics-based quality assessment. To support our discussion, we compare the highest to the lowest thresholds of each metric. Table 5.5 presents for each metric the lowest to the highest thresholds among the 15 domains. It also shows, between parentheses, how many times the highest threshold is greater than the lowest threshold. For instance, the lowest 95% threshold for the LOC metric is 232 (see E-Commerce in Table 5.4), while the highest 95% threshold for the LOC metric is 981 (see Accounting in Table 5.4). The difference is that the Accounting threshold is 4.2 times higher than the E-Commerce threshold in this case.

Table 5.5. Difference between Highest and Lowest Thresholds

%	LOC	NOA	NOM	WMC	LCOM	CBO	DIT	NOC
90	160 - 541 (3.4)	8 - 14 (1.7)	17 - 34 (2.0)	30 - 123 (4.1)	85 - 317 (3.7)	12 - 16 (1.3)	2 - 6 (3.0)	0 - 0 (n/a)
95	232 - 981 (4.2)	11 - 21 (1.9)	24 - 53 (2.2)	49 - 236 (4.8)	218 - 742 (3.4)	15 - 23 (1.5)	2 - 6 (3.0)	0 - 1 (n/a)

The results in Table 5.5 show that the investigated thresholds largely vary for all metrics; except Number of Children (NOC), which we discuss in Section 5.6. Apart from NOC, all metrics have a significant variation in the 90% and 95% thresholds. The metrics with higher threshold variations are LCOM, WMC, and LOC. The rate between the highest and the lowest thresholds in a domain is about 3.4 times or higher

for these 3 metrics. This large difference between the highest and lowest thresholds suggests that LCOM, WMC, and LOC are highly sensitive to the software domain. Therefore, developers should be aware of this variation when using thresholds for these metrics in software quality evaluation.

The rate between the highest and the lowest thresholds for NOA, NOM, and CBO is about 2 times. This difference suggests that these metrics are not as sensitive to the software domain as LCOM, WMC, and LOC. Yet, a difference of twice can be considered large, depending on the evaluation goal. The DIT metric is an interesting case. The threshold of DIT did not present large variation among domains. For instance, the 95% thresholds of DIT varied between 3 and 6 (2 times) for 14 out of 15 domains. The only exception was E-commerce with both 90% and 95% thresholds of 2 for DIT (see Table 5.4). Therefore, we consider the variation of DIT moderate.

5.6 RQ2: Similar Thresholds across Domains

We analyze similar thresholds across different domains for each metric. The main reason for this analysis is to identify whether and which domains can be grouped to promote more reliable benchmarks. In addition, if different domains have similar thresholds for the same metric, it means that some metric thresholds can be reused across domains.

For this analysis, we cluster thresholds for each metric into three levels according to how high these values are. Table 5.6 uses grey scale to indicate domains (rows) with low, medium, and high thresholds for each metric (columns). In Table 5.6, light grey boxes mean lower thresholds, grey boxes mean medium thresholds, and dark grey boxes mean high thresholds. Based on Table 5.4, we jointly analyzed both the 90% and 95% thresholds to determine if the metric threshold is low, medium, or high for a domain. For instance, Accounting domain has high thresholds for 6 metrics (LOC, NOA, NOM, WMC, LCOM and CBO), medium thresholds for DIT, and low thresholds only for NOC.

It is interesting to observe that Accounting and Business domains have the same levels of thresholds for all eight metrics (i.e., same colors in corresponding boxes). This result means that these two domains are very similar in terms of measurement. Therefore, if someone derives thresholds for a set of Accounting systems, these thresholds are expected to be reliable to be used for detecting anomalies in Business systems, for instance. As discussed before, one reason for this similarity is because both domains involve broader trade fields. In addition, they are usually large and complex systems

Table 5.6. High and Very-High Thresholds Levels for each Metric

	LOC	NOA	NOM	WMC	LCOM	CBO	DIT	NOC
Acc								
Bus								
ScE								
Gam								
Hea								
Mes								
Rest								
Dev								
FT								
Dic								
Loc								
Com								
EC								
Edu								
Hom								

as indicated by high threshold for most metrics.

Other pairs of similar domains in terms of metric thresholds are: Education and Home, Free Time and Development, Dictionary and Science & Engineering, and Dictionary and Localization. The explanation for similar thresholds vary in a case by case basis. For instance, Education and Home in our dataset have some of the smallest and simplest systems. Therefore, these domains share low thresholds for most metrics. On the other hand, Free Time and Development tend to be heterogeneous, but both are used in many situations. For instance, Free Time is often used for entertainment and leisure purposes, while development must support a variety of scenarios for completing development projects. An interesting similarity is between Dictionary and Localization systems, although these domains deal with completely different requirements, we observed that they share some similar coding styles. For instance, Dictionary and Localization systems have large methods coupled to several API classes. This characteristic makes them similar in terms of metrics and thresholds. This explanation can also be given to the similarity between Dictionary and Science & Engineering.

Number of Children (NOC) has similar - and low - thresholds for all domains. In fact, the 90% thresholds are always 0 for all domains while the 95% threshold is either 0 or 1. We observed in the distribution of NOC that its values only vary largely if we select the 3% to 1% classes with more children (i.e., the 97% to 99% thresholds). This result means that only 1% to 3% of classes in a system usually have more than one subclass. Therefore, when defining threshold for NOC, someone needs to consider this particularity of this specific metric.

5.7 RQ3: Impact of System Size on Thresholds

This section discusses if the size of the systems which compose a benchmark affects the metric thresholds. It so, in addition to domains, the system size should be considered when building benchmarks for quality assessment. For instance, metrics-based code smell detection tools should consider the size of the systems in this case. Otherwise, they might use overly high thresholds for small systems, and vice versa. To investigate RQ3, we build four balanced benchmarks with different system sizes and compare the derived thresholds. The first one is *QualitasCorpus* [Tempero et al., 2010] which is a benchmark composed of 111 large Java systems, such as Eclipse, Netbeans, and Ant. The second one is the *AllSystems* benchmark which includes all 3,107 heterogeneous systems of this study that we mined from GitHub. The third benchmark, named *MediumSystems*, is a subset of *AllSystems* comprising 20 systems per domain and, making 300 systems in total. Our aim was to exclude outliers (too large or too small) systems in this benchmark of medium-sized systems. Therefore, we only selected 10 systems immediately above and 10 systems immediately below the median of each domain in terms of LOC. The fourth benchmark, *SmallSystems*, is also a subset of *AllSystems* with 20 systems per domain (i.e., 300 systems in total). However, in this case we randomly selected systems below the median LOC for each domain. For instance, all 20 Health systems in *SmallSystems* have between 1,000 LOC (selection quality criterion) and 4,786 LOC. Since three domains (E-Commerce, Free Time, and Science & Engineering) have less than 40 systems and we have the criterion of 20 systems per domain, *SmallSystems* has exactly the 20 smallest systems for these three domains.

Table 5.7 presents the results of the 95% thresholds for the four benchmarks described in the previous paragraph. This table focuses on 7 metrics because we have not observed difference for NOC; i.e., its 95% thresholds are either 0 (*SmallSystems* and *MediumSystems*) or 1 (*QualitasCorpus* and *AllSystems*). The results in Table 5.7 shows that larger systems have higher thresholds for all metrics. For instance, in the case of LOC, the 95% thresholds are 602, 599, 315, and 286 for *QualitasCorpus*, *AllSystems*, *MediumSystems*, and *SmallSystems*, respectively.

Table 5.7. Thresholds for Different Benchmarks

	LOC	NOA	NOM	WMC	LCOM	CBO	DIT
Qualitas	602	16	34	111	406	20	6
AllSystems	599	18	37	125	475	22	5
MediumSyst.	315	13	23	59	168	17	4
SmallSystems	286	12	20	52	129	15	4

In fact, we already expected that larger systems have higher thresholds for size metrics, such as LOC, NOA, and NOM. However, it is interesting to observe that the system size has also affected the thresholds of complexity and inheritance metrics. This result contradicts previous studies [Ferreira et al., 2012; Vale et al., 2015] that claim metrics like CBO and DIT do not have a high correlation with LOC. In fact, we observe that larger systems have both more complex classes and denser use of inheritance relationships. In addition, we observed in Table 5.7 that *QualitasCorpus* and *AllSystems* have similar thresholds, although they do not have any system in common. This observation suggests that if the benchmark is composed of a high enough number of heterogeneous systems (i.e., from different domains and sizes), the metrics thresholds tend to be comparable.

5.8 Code Smell Detection Evaluation

Code smells describe a anomaly where there are hints that suggest a flaw in the source code [Fernandes et al., 2016]. For instance, one of the most well-known code smell, God Class, is defined as a class that knows or does too much in the software system [Fowler et al., 1999; Fernandes et al., 2016]. God Class is a strong indicator of design flaw because this component is aggregating functionality that should be distributed among several components. In fact, previous work has found that this code smell is related to maintenance problems, such as bugs [Fontana et al., 2013], design flaws [Oizumi et al., 2016; Padilha et al., 2014], and instability [Fernandes et al., 2017].

This section evaluates the God Class detection by comparing the thresholds derived from each domain with the thresholds derived from *AllSystems* benchmark. The first set we call domain-specific thresholds and the second set we call generic thresholds. The evaluation consists of comparing precision and recall of both sets in God Class detection. To perform this evaluation, we followed four steps. First, we randomly selected 43 systems from all 15 domains explored in this study and built an oracle of God Class instances for these systems (Section 5.8.1). Second, we used the metric-based strategy to identify God Classes and, then, to compute precision and recall (Section 5.8.2). Finally, we analyzed the effectiveness results for detecting code smells to answer RQ4 (Section 5.8.3).

5.8.1 Dataset for Code Smell Analysis

To build the Smell Dataset, we randomly selected 43 systems using two criteria: (i) at least two systems per domain; (ii) systems that compiles in Eclipse IDE. The first

criterion is to have a sample that covers all domains and the second one is to address tool support limitations. This dataset is then composed by 3 systems per domain, except two domains (E-Commerce and Free Time) which we only found 2 systems that match the second criterion.

Oracle Creation. For each system of our Smell Dataset, we built an oracle of true positive instances. The oracle can be understood as the reference list of the actual smells found in a system. This oracle is the basis for determining whether the derived thresholds are effective in the identification of God Classes. In order to provide a reliable oracle, we run three well-known code smell detection tools (JDeodorant [Mazinanian et al., 2016], JSpirit [Vidal et al., 2015], and PMD [Fontana et al., 2013]) for all target systems and build a list with possible anomalies pointed by these tools. Then, at least a pair of authors analyzed each class pointed as God Class to validate our oracle. This manual validation consisted of the answer of four questions with a confidence rate varying from 1 to 5:

- Does the class have more than one responsibility?
- Does the class have functionality that would fit better into other classes?
- Do you have problems summarizing the class responsibility in one sentence?
- Would splitting up the class improve the overall design?

These four questions were based on questions of a previous study [Schumacher et al., 2010]. In the cases we had a disagreement between the two evaluators or an average confidence score small than 3, a third evaluator checked the class and the three evaluators discussed to reach a consensus.

5.8.2 Metric-based Detection Strategy and Measurement of Effectiveness

Metrics are often too fine-grained to comprehensively quantify technical flaws [Lanza and Marinescu, 2007]. To overcome this limitation, metric-based detection strategies have been proposed [Marinescu, 2004]. This work selected and adapted a detection strategy from the literature to identify God Class [Oizumi et al., 2016], presented in Figure 5.2. There are two main reasons to use such strategy. First, it has been evaluated in other studies and presented good results for the detection of God Class [Oizumi et al., 2016]. Second, this detection strategy defines a straightforward way for identifying instances of God Class by combining four different metrics. In fact,

we adapted its original definition based on the metrics we investigate in this paper, but the adapted detection strategy captures the same quality characteristics from the original work [Lanza and Marinescu, 2007; Marinescu, 2004].

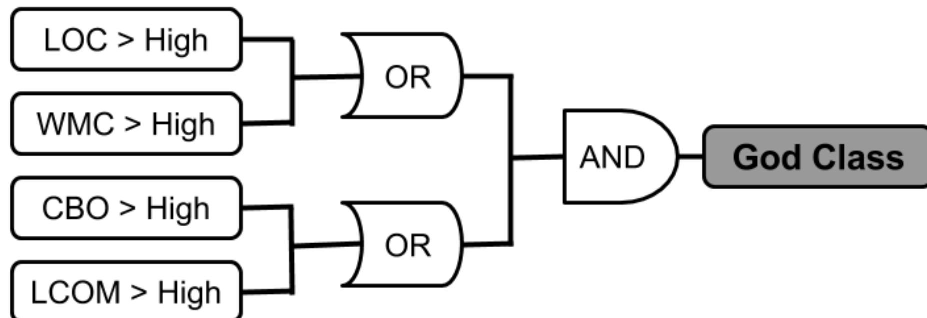

Figure 5.2. God Class Detection Strategy

We use precision and recall as a proxy for effectiveness in code smell detection. Recall measures the fraction of relevant classes listed by the detection strategy using a set of thresholds. Relevant classes are classes that appear in the oracle. Precision measures the ratio of correctly detected code smells by the total classes listed. To compute precision and recall we need to know the values of true positives (TP), false positives (FP), and false negatives (FN). TP and FP quantify the number of correctly and wrongly identified code smells by the detection strategy compared to the oracle. FN, on the other hand, quantifies the number of code smells the detection strategy missed out from the oracle. The computation of recall and precision is: $Recall = TP/(TP + FN)$ and $Precision = TP/(TP + FP)$. Recall and precision vary from 0 to 1 and higher values are related to better effectiveness.

5.8.3 RQ4: Effectiveness of Thresholds for Code Smell Detection

In this section, we answer and discuss RQ4 (Are domain-specific thresholds better than general thresholds for code smell detection?). To do that, we computed precision and recall as just described. Analyzing the results presented in Table 5.8, domain-specific thresholds did not always get the best results in such evaluation. In terms of precision, domain-specific thresholds were better in 3 cases, and technically equal in other 5 cases. We considered technically equal, cases that the difference is smaller than 5%. In terms of recall, the results are more exciting since domain-specific thresholds are better in 8

cases and technically equal in other 5 cases. We use bold to show which is better in the pairwise comparison.

High precision means that the detection strategy indicated more relevant than irrelevant code smells. High recall, on the other hand, means that the detection strategy was able to identify most code smells in the system. Hence, a large number of false positives (high recall) are preferred over a large number of false negatives (high precision) by software engineers, because manual inspection, which is inevitable, tends to uncover false positives. Therefore, considering that recall is higher for domain-specific thresholds in 8 domains, we conclude that domain-specific thresholds fared better in detecting code smells than generic thresholds.

Still talking about Table 5.8 and considering the four categories presented in Section 5.4, higher thresholds got higher precision and lower recall than generic thresholds. Exactly, the opposite result for low thresholds category. For the intermediate categories is harder to conclude something, but it tends to follow low thresholds category where generic thresholds achieved higher precision and domain-specific higher recall. Therefore, the classification of domains presented in Section 5.4 should be considered for selecting the most appropriate thresholds for software quality evaluation.

Table 5.8. Precision and Recall using Domain-Specific and Generic Thresholds

		Precision		**Recall**	
Category	**Dom.**	**DS**	**G**	**DS**	**G**
All High	Acc	**1.00**	0.75	0.20	**0.60**
	Bus	0.00	0.00	0.00	0.00
	ScEng	**0.88**	0.82	0.58	**0.75**
High Size	Gam	0.05	0.09	0.67	0.67
	Hea	0.50	**0.75**	**0.36**	0.27
	Mes	0.40	**1.00**	**0.40**	0.20
	Res	**0.25**	0.00	**0.25**	0.00
High Complextiy	Dev	0.67	**0.76**	0.94	0.94
	FT	0.43	**0.75**	0.50	0.50
	Dic	0.00	0.00	0.00	0.00
	Loc	0.64	0.67	**0.69**	0.46
All Low	Com	0.58	**0.68**	**0.83**	0.72
	EC	0.03	0.00	**0.50**	0.00
	Edu	0.54	**0.80**	**0.54**	0.31
	Hom	0.79	**0.89**	**0.85**	0.62

5.9 Concluding Remarks

In this chapter, we evaluated the method using domain metric thresholds with enterprise systems. We validated the proposed method through quantitative analyses by answering four questions. To perform these analyses, we mined 3,107 desktop systems from GitHub and devised into 15 distinguished domains. As a result for RQ1 (Do thresholds for the same metric vary among different software domains?), we verified that thresholds for the same metric may vary from 1.3× (CBO) to 4.1× (WMC) for the 90% cut, and from 1.5× (CBO) to 4.8× (WMC) for the 95% cut. We have not observed high variation of thresholds only for Number of Children (NOC). Therefore, we concluded that the metric thresholds are typically sensitive to the software domain. For RQ2 (Are there metrics with the same thresholds regardless of the software domain?), some domains have similar thresholds for the same metrics. This result implies that these domains can be grouped to promote more reliable benchmark-based threshold derivation. Accounting and Business are examples of domains with similar thresholds for all metrics. Analyzing RQ3: (Does the system size impact on the derived thresholds?), the size of the systems that compose the benchmark influences the metric thresholds. Benchmarks with larger systems yield higher thresholds for all metrics. Furthermore, benchmarks composed of many heterogeneous systems in terms of size and domains tend to have similar thresholds. Finally RQ4 (Are domain-specific thresholds better than general thresholds for detecting the God Class code smell?), in terms of recall, domain-specific thresholds are usually better than generic thresholds for most domains. In the next chapter, we present some potential threats to validity and the main actions we have taken to mitigate their impact on the research results.

Chapter 6

Threats to Validity

Our empirical studies have some potential threats to validity that we discuss in this chapter by presenting the main actions we have taken to mitigate their impact on the research results. Section 6.1 presents our actions to mitigate system selection and measurement issues. Section 6.2 explains our issues on generalization and extension of the results of this work.

6.1 Internal Validity

Internal validity refers how systematic and well design an experiment is done. This kind of threat can affect the independent variable with respect to causality [Wohlin et al., 2012]. There are two main threats to internal validity: selected domains and measurements. Regarding the first threat, one might consider that we have not selected representative domains. However, as the selected domains are consolidated ones, we expect to have analyzed high quality and frequently used systems. In addition, we have also included in our study only systems with more than 1,000 lines of code.

For the measurements threat, we may get false or wrong measures for the target systems. We quantify metrics for all systems that compose this study and we also derive thresholds for these metrics. To have rigorous measurement processes, we used a specific tool presented in another study [CK Metrics, 2018]. In addition, we also made some tests to check if the results of this tool were the expected ones.

6.2 External Validity

External Validity limits the ability to generalize the results beyond the experiment setting. These treats are reduced by making the experimental environment more realistic [Wohlin et al., 2012]. We consider four external threats to validity. First, it is not possible to ensure that the select systems reflect the recurrent practices of software development. To reduce this risk, for Chapter 4, we selected the most popular mobile systems in each domain, and with a link to a real mobile system on Google Play Store.

For Chapter 5, we filter the total amount of systems by number of stars and lines of code. Second, all systems in our dataset are developed in Java. However, our proposed method is generic and other studies might achieve similar results in the context of other programming languages or technologies.

Third, our results are restricted for the set of metrics we selected, but we believe that similar results can be also found for metrics that quantify similar quality attributes or characteristics, as we found in our results. Finally, even though we have presented a large-size study, additional replications are necessary to determine if our findings can be generalized to other domains and dataset of systems.

6.3 Concluding Remarks

In this chapter, we presented our actions to mitigate potential threats to validity on our empirical studies. For internal validity we focus on system selection and measurement issues. For external validity, we explains the limitation of this work results. In the next chapter we concludes this work by presenting our contributions and final considerations.

Chapter 7

Final Considerations

In this dissertation, we proposed a tool supported method to derivate domain-sensitive metric thresholds and evaluated mobile and enterprise systems. This method considers domain as a relevant factor to derive reliable metric thresholds. We also presented a tool that brings this concept to practice. This chapter presents our conclusions about the method and it empirical evaluation, our contributions, and directions for future work.

7.1 Conclusion

This work proposed and evaluated a domain-sensitive method to derive reliable quality metric thresholds. The method was designed following recommended guidelines: it is based on a data analysis from a representative set of systems (benchmark); It has a strong dependence with the number of entity; and a weak dependency with the number of systems. Therefore, while it shares common activities presented in other methods, it also introduces innovations: (i) the input is a benchmark composed of systems that belong to distinguished domains; (ii) the core process computes metrics and performs threshold derivation for each separated domain; and (iii) the output are domain thresholds to each analyzed metric.

We evaluated the proposed method through qualitative and quantitative analyses. For qualitative analysis, we highlight the differences of our method to two recently proposed methods, including explicitly differentiation of domains when building benchmarks. For the quantitative analyses, we performed two studies. First, we manually mined one hundred mobile systems from GitHub divided into ten distinguished domains. This quantitative analysis indicates that thresholds vary for each domain and that the majority of the analyzed metrics are domain-sensitive. By clustering domains

with levels, our results indicate that some domains share similarities. In addition, analyzing the coefficient of variation for domain thresholds, the lower variation was 15.3% on 90% for DIT, which was expected due smaller range for this metric, and the higher variation was for LCOM with 95%. In the second quantitative evaluation, we presented an empirical study on domain-specific thresholds. We conducted the study by selecting and measuring 3,107 software systems from 15 software domains. Once we have the measurements, we derived 90% and 95% thresholds for each metric per domain and analyzed them in different ways. For instance, we compared the thresholds among domains and investigated the effectiveness of code smell detection between domain-specific and generic thresholds. The results indicate that metric thresholds are sensitive to software domain. For instance, some metrics may vary across domains from 1.5× to 4.8× for the 95%. Moreover, we observed that not only the domains, but also the size of the systems that compose the benchmark is a factor that affect the metric thresholds. That is, the results corroborate with the claim that benchmarks composed of heterogeneous systems tend to have similar thresholds. Finally, in terms of recall, we collect evidence indicating that domain-specific thresholds are better than generic thresholds on average for code smell detection.

7.2 Contribution

We consider four main contributions in this dissertation, as follows.

- A method to derive domain-sensitive metric thresholds.
- A tool to show domain threshold warnings.
- A benchmark of 100 mobile systems classified into 10 domains which we used in the method evaluation.
- A large-size empirical study with more than 3 thousand systems to explore threshold derivation with the corresponding measurements for eight well-known software metrics.

7.3 Future Work

The results presented in this work highlight domains as a factor to bring into account for threshold derivation. Therewith, for further research there are some directions to improve the proposed method and tool. For the method, further investigation with

additional anomalies and additional replications of this studies to determine whether our findings can be generalized to other domains and systems. In addition, commercial software systems should be investigate to confirm whether and how thresholds vary across domains and their impact on anomalies.

For the method application, this work leads to other directions. One suggestions is to study detection of anomalies on mobile applications. A second suggestion, research how to build optimal benchmarks, from which we can derive reliable metric thresholds with the lowest number of systems. Another suggestion is to investigate whether there are domain-specific anomalies.

For the application of the tool, TWarning can be improved by implementing new features, such as: (i) methods to derive thresholds and, (ii) adding more usability functions (i.e. views, domains, thresholds, and other custom data operations). Some additional tool evaluation and survey can capture what the developer most need with the tool use.

Bibliography

Ramon Abílio, Juliana Padilha, Eduardo Figueiredo, and Heitor Costa. Detecting code smells in software product lines–an exploratory study. In *12th International Conference on Information Technology-New Generations (ITNG)*, pages 433–438, 2015.

Tiago L. Alves, Christiaan Ypma, and Joost Visser. Deriving metric thresholds from benchmark data. In *29th International Conference on Software Maintenance (ICSM)*, pages 1–10, 2010.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. Pattern-oriented software architecture, volume 1: A system of patterns. 1996.

Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

CK Metrics, 2018. URL `https://github.com/mauricioaniche/ck`.

Don Coleman, Bruce Lowther, and Paul Oman. The application of software maintainability models in industrial software systems. *Journal of Systems and Software*, 29 (1):3–16, 1995.

Giulio Concas, Michele Marchesi, Sandro Pinna, and Nicola Serra. Power-laws in a large object-oriented software system. *IEEE Transactions on Software Engineering*, 33(10):687–708, 2007.

Tom DeMarco. *Controlling software projects: Management, measurement, and estimates*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1986.

Karin Erni and Claus Lewerentz. Applying design-metrics to object-oriented frameworks. In *3rd International Software Metrics Symposium*, pages 64–74, 1996.

Eduardo Fernandes, Johnatan Oliveira, Gustavo Vale, Thanis Paiva, and Eduardo Figueiredo. A review-based comparative study of bad smell detection tools. In *20th*

International Conference on Evaluation and Assessment in Software Engineering (EASE2016), page 18, 2016.

Eduardo Fernandes, Gustavo Vale, Leonardo Sousa, Eduardo Figueiredo, Alessandro Garcia, and Jaejoon Lee. No code anomaly is an island anomaly agglomeration as sign of product line instabilities. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 48–64, 2017.

Kecia A.M. Ferreira, Mariza A.S. Bigonha, Roberto S. Bigonha, Luiz F.O. Mendes, and Heitor C. Almeida. Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, 85(2):244–257, 2012.

Tarcısio G.S. Filó, Mariza Bigonha, and Kecia Ferreira. A catalogue of thresholds for object-oriented software metrics. *Proc. of the 1st SOFTENG*, pages 48–55, 2015.

Francesca Arcelli Fontana, Marco Zanoni, Alessandro Marino, and Mika V. Mantyla. Code smell detection: Towards a machine learning-based approach. In *29th International Conference on Software Maintenance (ICSM)*, pages 396–399, 2013.

Fossdroid, 2018. URL `https://fossdroid.com/`.

Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: improving the design of existing code.* Addison-Wesley Professional, 1999.

Vern A. French. Establishing software metric thresholds. In *9th International Workshop on Software Measurement (IWSM)*, 1999.

Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability, 2007.

Steffen Herbold, Jens Grabowski, and Stephan Waack. Calculation and optimization of thresholds for sets of software metrics. *Empirical Software Engineering*, 16(6): 812–841, 2011.

Douglas W. Hubbard. *How to measure anything: Finding the value of intangibles in business.* John Wiley & Sons, 2014.

Barbara Kitchenham. What's up with software metrics?–a preliminary mapping study. *Journal of systems and software*, 83(1):37–51, 2010.

Michele Lanza and Radu Marinescu. *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media, 2007.

Mario Linares-Vásquez, Sam Klock, Collin McMillan, Aminata Sabané, Denys Poshyvanyk, and Yann-Gaël Guéhéneuc. Domain matters: bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in Java mobile apps. In *22nd International Conference on Program Comprehension (ICPC)*, pages 232–243, 2014.

Mark Lorenz and Jeff Kidd. *Object-oriented Software Metrics: A Practical Guide*. Prentice-Hall, Inc., 1994.

Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *20th International Conference on Software Maintenance*, pages 350–359, 2004.

Davood Mazinanian, Nikolaos Tsantalis, Raphael Stein, and Zackary Valenta. JDeodorant: Clone Refactoring. In *38th International Conference on Software Engineering Companion (ICSE)*, pages 613–616, 2016.

Thomas J. McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976.

Allan Mori, Gustavo Vale, Markos Viggiato, Johnatan Oliveira, Eduardo Figueiredo, Elder Cirilo, Pooyan Jamshidi, and Christian Kästner. Evaluating domain-specific metric thresholds: an empirical study. In *International Conference on Technical Debt (TechDebt)*, 2018.

Emerson Murphy-Hill, Thomas Zimmermann, and Nachiappan Nagappan. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development? In *36th International Conference on Software Engineering (ICSE)*, pages 1–11, 2014.

Brian A. Nejmeh. Npath: a measure of execution path complexity and its applications. *Communications of the ACM*, 31(2):188–200, 1988.

Willian Oizumi, Alessandro Garcia, Leonardo da Silva Sousa, Bruno Cafeo, and Yixue Zhao. Code anomalies flock together: Exploring code anomaly agglomerations for locating design problems. In *38th International Conference on Software Engineering (ICSE)*, pages 440–451, 2016.

Juliana Padilha, Juliana Pereira, Eduardo Figueiredo, Jussara Almeida, Alessandro Garcia, and Cláudio Sant'Anna. On the effectiveness of concern metrics to detect code smells: an empirical study. In *26th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 656–671, 2014.

Mirko Perkusich, Amaury Medeiros, Kyller Costa Gorgônio, Hyggo Oliveira de Almeida, Angelo Perkusich, et al. A bayesian network approach to assist on the interpretation of software metrics. In *30th Annual ACM Symposium on Applied Computing (SAC)*, pages 1498–1503, 2015.

Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in GitHub. In *22nd International Symposium on Foundations of Software Engineering (SIGSOFT)*, pages 155–165, 2014.

Jan Schumacher, Nico Zazworka, Forrest Shull, Carolyn Seaman, and Michele Shaw. Building empirical support for automated code smell detection. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, page 8, 2010.

Alexandre Leite Silva, Alessandro Garcia, Elder Jose Reioli, and Carlos Jose Pereira De Lucena. Are domain-specific detection strategies for code anomalies reusable? an industry multi-project study. In *27th Brazilian Symposium on Software Engineering (SBES)*, pages 79–88, 2013.

Diomidis Spinellis. A tale of four kernels. In *30th international conference on Software engineering (ICSE)*, pages 381–390, 2008.

Ewan Tempero, Craig Anslow, Jens Dietrich, Ted Han, Jing Li, Markus Lumpe, Hayden Melton, and James Noble. The qualitas corpus: A curated collection of Java code for empirical studies. In *17th Asia Pacific Software Engineering Conference (APSEC)*, pages 336–345, 2010.

Gustavo Vale and Eduardo Figueiredo. A method to derive metric thresholds for software product lines. In *29th Brazilian Symposium on Software Engineering (SBES)*, pages 110–119, 2015.

Gustavo Vale, Danyllo Albuquerque, Eduardo Figueiredo, and Alessandro Garcia. Defining metric thresholds for software product lines. *Proceedings of the 19th International Conference on Software Product Line - SPLC '15*, pages 176–185, 2015.

Gustavo Vale, Eduardo Fernandes, and Eduardo Figueiredo. On the proposal and evaluation of a benchmark-based threshold derivation method. *Software Quality Journal*, pages 1–32, 2018.

Rajesh Vasa, Markus Lumpe, Philip Branch, and Oscar Nierstrasz. Comparative analysis of evolving software systems using the gini coefficient. pages 179–188, 2009.

Lucas Veado, Gustavo Vale, Eduardo Fernandes, and Eduardo Figueiredo. Tdtool: threshold derivation tool. In *20th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, page 24, 2016.

Santiago Vidal, Hernan Vazquez, J Andres Diaz-Pace, Claudia Marcos, Alessandro Garcia, and Willian Oizumi. Jspirit: a flexible tool for the analysis of code smells. In *34th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–6, 2015.

Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.