Assessing the Influence of Size Category of the Project in God Class Detection, an Experimental Approach based on Machine Learning (MLA)

Khalid Alkharabsheh
CiTIUS
Universidad de Santiago de Compostela
Santiago de Compostela, Spain
khalid.alkharabsheh@usc.es

Yania Crespo
Departamento de Informática
Universidad de Valladolid
Valladolid, Spain
yania@infor.uva.es

Manuel Fernández-Delgado
CiTIUS
Universidad de Santiago de Compostela
Santiago de Compostela, Spain
manuel.fernandez.delgado@usc.es

José M. Cotos
CiTIUS
Universidad de Santiago de Compostela
Santiago de Compostela, Spain
manel.cotos@usc.es

José A. Taboada
CiTIUS
Universidad de Santiago de Compostela
Santiago de Compostela, Spain
joseangel.taboada@usc.es

***Abstract*—Design Smell detection has proven to be an effective strategy to improve software quality and consequently decrease maintainability expenses. In this work, we explore the influence of the size category of the software project on the automatic detection of God Class Design Smell by different machine learning techniques. A set of experiments were conducted with eight different learning classifiers on a dataset formed by 12,588 classes of 24 systems. The results were evaluated using ROC area and Kappa tests. The classifiers change their behaviour when they are used in sets that differ in the value of the selected size information of their classes. This study concludes that it is possible to improve results, mainly in agreement, of God Class detection feeding machine learning classifiers with project size information of the classes to analyze.**

***Index Terms*—Design Smell Detection; Machine Learning; God Class.**

I. Introduction

Software systems available in different departments of organizations to provide many services in different domains. Even though the provided services are essential, but the more critical is the software continues operating without problems and mistakes. Software quality is an important concern for software industries, academic, and researchers. Identifying problems in source code or design of software, correcting and modifying them are some of the main activities of the maintenance process in order to increase quality.

All problems related to the software structure that does not make compile or run-time errors are described by "Design Smell" [11]. The presence of Design Smells negatively affects software quality factors, though technical debt increases [6]. Smells can appear in several software artifacts including variables, instructions, operations, methods, classes, packages, subsystems, layers, and their dependencies.

DOI reference number:10.18293/SEKE2019-140

God Class Design Smell is one of the most detected smells in software according to our systematic mapping study [1]. It has attracted the attention of the research community. God Class is defined with different names as the Large Class Bad Smell [10], the Blob Antipattern [5], and the God Class Disharmony [15]. The term God Class Design Smell is the name we use for bringing both together. It is considered as a class level Design Smell.

Several approaches, tools, and techniques have been proposed to improve the detection of God Class Design Smell. However, some studies evidence the lack of agreement among different tools and human experts. Moreover, they have a set of limitations represented in understanding the precise definition of God Class Design Smell and the process of mapping the definition into effective detection algorithms. The current trend in Design Smell detection has adopted machine learning for deriving detection rules [12], [21].

This paper aims through an exploratory study to investigate whether the size category of the project is relevant in God Class detection. A set of experiments were conducted with eight different classifiers. The selected classifiers are the most recently reported in the literature and they jointly involve all families of classifiers. Results are evaluated using ROC area [4] and Kappa tests [2].

The main contributions of this study can be summarized by determining the influence of project size information on God Class Design Smell detection using machine learning techniques. Moreover, a large dataset for God Class Design Smell detection.

The rest of this paper is organized as follows: Section II describes the related work. Section III presents the problem statement ,research question, and hypothesis. Section IV describes the dataset we built. Section V exposes the methodol-

ogy we follow to identify the effectiveness of project domain information in God Class Design Smell detection. Section VI discusses the results of experiments. Section VII presents the main threats to the validity and Section VIII explains our conclusions.

II. Related Work

In this section, we present studies exploiting machine learning techniques for Design Smell detection.

Jochen Kreimer [14] presents a method to detect Design Flaws by combining object-oriented metrics and machine learning. The proposed approach was validated using five Design Flaws: Big Class also known as (The Blob), Feature Envy, Long Method, Lazy Class and Delegator and two software systems: IYC (91 classes) and WEKA (597 classes). Khomh et al. [13] introduce BDTEX (Bayesian Detection Expert) a Goal Question Metric-Based approach to detect Antipatterns using Bayesian Belief Networks (BBNs) stood on rule-based representation. The approach was validated using three Antipatterns: The Blob, Functional Decomposition and Spaghetti Code and two Java programs: Xerces v2.7.0 (589 classes) and GanttProject v1.10.2 (188 classes). Maiga et al. [16] introduce SVMDetect, an approach to detect Antipatterns based on support vector machine. They compute the object-oriented metrics for each class. The empirical study involves four Antipatterns: The Blob, Functional Decomposition, Spaghetti Code and Swiss Army Knife and three open source software: ArgoUML v0.19.8 (1,230 classes), Azureus v2.3.0.6 (1,449 classes) and Xerces v2.7.0 (513 classes). Fontana et al. [9] present their approach based on machine learning techniques for outline the common problems in the previous Design Smells detection. The dataset has formed by the 59,333 classes of 76 systems and a large set of relevant metrics. A set of six code smells: God Class, Data Class, Feature Envy, God Method, Brain Method, and Long Method were detected in the experiment.

As we can see, the related works principally focused on numerical information (mainly object-oriented metrics and other well-known sets of metrics) on classes to detect a set of smells. In our work, we focus the attention on certain nominal project information (size category) and try to explore its influence in God Class detection in order to take this information into account in future work, in the aim of obtaining better results that can be more useful for developers, improving agreement among tools and experts.

The aim of this paper is analyzing if clarifying the size category can lead to variations in the detection. The detection problem is seen as a classification problem; using automatic classifiers that separate classes in having God Class Design Smell or not. Our dataset is formed by the 12,588 classes of 24 systems with different sizes and domains. In order to define our dataset, we analyze these systems using five different tools common in detecting God Class Borland Together [3], PMD [8], iPlasma [18], DÉCOR [20] and JDeodorant [22].

III. Problem Statement

The problem that we examine in this study that most of the research community has not taken into account the impact of nominal project information on Design Smell detection. The suspicion that we have raised involves this type of information can be useful for developers to obtain better detection results. For this reason, in this work, we address the effectiveness and importance of project size nominal information in the detection of God Class Design Smell. To address the study problem, we introduce the following research question:

RQ: Does the differences between the size category of the whole project influence in the detection of God Class Design Smell?

We want to reject the null hypothesis formulated as:

H_0^{size}: Project size does not influence on God Class detection.

We propose to obtain several classifiers trained with and without the project size information under examination, working with the dataset taken as a whole in order to verify whether this kind of information will affect on the behaviour of classifiers or not. If this first exploration succeeds, the next step should be to design the same experiment to investigate the impact of the project size category in detecting God Class as explained in Section V. The behaviour of classifiers is evaluated by ROC area and Kappa performance measurements as we will see in Sections V and VI.

IV. Dataset

Our dataset is formed by the 12,588 classes of 24 open source systems written in Java obtained from SourceForge source code repository which involved different domains and size categories. We selected the SourceForge repository because it is the most widely known and used in the context of open source software, and it supplies useful metadata for projects. The projects are selected according to particular criteria, such as should be written in Java, long life cycle with several version, has a significant change history information (development, maintenance), and the projects should be of different size categories.

The high number of projects and classes is intended to discard the probability of dependencies between classifiers and a particular subset of data. Table I presents the main characteristics of our dataset where includes [Project name, Category of size, Number of Classes, Total Line of Code in the project (TLOCP)].

The dataset is formed as follows: rows x_1 to x_{16} represent numerical attributes (metrics), x_{17}, represent the project size information and x_{18}, the result of classification if a class is God Class or not (resulting from the selected tools as a logical or among them). Table II shows the full list of selected variable x_1 to x_{18} and their definition.

TABLE I
DATASET CHARACTERISTCS

ProjectName	SizeCat.	#Class	TLOCP
jAudio-1.0.4	L	416	117,615
Freemind-1.0.1	L	782	106,396
JasperReports-4.7.1	L	1,797	350,690
SQuirreL-1.2	M-L	1,138	71,626
KeyStoreExplorer-5.1	M-L	384	83,144
DigiExtractor-2.5.2	M	80	15,668
AngryIPScanner-3.0	M	270	19,965
Plugfy-0.6	S	28	2,337
Matte-1.7	M-L	603	52,067
sMeta-1.0.3	M	222	30,843
xena-6.1.0	M-L	1,975	61,526
pmd-4.3.x	M-L	800	82,885
checkstyle-6.2.0	M-L	277	41,104
JDistlib-0.3.8	M	78	32,081
JCLEC-4-base	M	311	37,575
Javagraphplan-1.0	S-M	50	1,049
Mpxj-4.7	L	553	261,971
Apeiro-2.92	S-M	62	8,908
FullSync-0.10.2	M	169	24,323
OmegaT-3.1.8	L	716	121,909
Lucene-3.0.0	M-L	606	81,611
Ganttproject-2.0.10	M-L	621	66,540
JFreechart-1.0.X	L	499	206,559
JHotDraw-5.2	M	151	17,807

TABLE II
VARIABLE DEFINITION

Metrics of class and package level (ratio scale)		
Var	Metric	Definition
x_1	LOC	Total Lines of Code
x_2	NCLOC	Non-Comment Lines of Code
x_3	CLOC	Comment Lines of Code
x_4	EXEC	Executable Statements
x_5	DC	Density of Comments
x_6	NOT	Number of Types
x_7	NOTa	Number of Abstract Types
x_8	NOTc	Number of Concrete Types
x_9	NOTe	Number of Exported Types
x_{10}	RFC	Response for Class
x_{11}	WMC	Weighted Methods per Class
x_{12}	DIT	Depth in Tree
x_{13}	NOC	Number of Children in Tree
x_{14}	DIP	Dependency Inversion Principle
x_{15}	LCOM	Lack of Cohesion of Methods
x_{16}	NOA	Number of Attributes
Project level information (nominal scale)		
Var	Information	Values
x_{17}	Size category	L, M-L, M, M-S, S
Smell detection (binary)		
Var	Design Smell	According to
x_{18}	God Class	PMD, iPlasma, JDeodorant, Décor, Together

The chosen projects are analyzed with RefactorIT v2.7 tool [1] [19] to compute a set of important class and package level metrics. The selected metrics are widely used in state of the art [13], [14], [17] for Design Smell detection, and some of them are particularly related to God Class characteristics such as cohesion (Lack of Cohesion (LCOM)), size (Line of Code (LOC), Number of Attributes (NOA), Number of Children in Tree (NOC), and complexity (Weighted Methods per Class (WMA). The second part of Table II is regarding projects level information, and we can see the size is consists of different categories. We follow the same approach as [7] to classify the projects based on categories.

As can be seen, the nominal data of size categories refers to the size of the Total Line of Code in the whole project (TLOCP). The project size is divided into six categories based on the TLOCP include (Small (S) $\leq 4,999$; $5,000 \leq$ Small-Medium (S-M)$\leq 14,999$; $15000 \leq$ Medium (M) $\leq 39,999$; $40,000 \leq$ Medium-large (M-L) $\leq 99,999$; $100,000 \leq$ Large (L) $\leq 499,999$; Very large (VL)$\geq$500,000). In fact, this is more precisely an ordinal scale. But we are not going to treat it as ordinal just as a nominal category.

The final part of the table focused on the selected God Class detection tools. To obtain the value of variable x_{18}, we used the following five Design Smell detection tool: The five tools are DÉCOR, JDeodorant, iPlasma, PMD, and Together. The tools were selected based on the results of a systematic mapping study [1] we conducted on the state of play in the field of Design Smell detection for the period 2000 to 2017. The selected tools are one of the most cited and used in the literature, and are commonly employed in God Class detection. Moreover, all the tools support projects implemented in the Java language and the input source is source code while the output is text in different formats such as CSV, txt, XML. Also, most of the tools focus only on smell detection, except for JDeodorant which includes a refactoring operation that is performed after the Design Smells have been identified. Nearly all of the tools have a GUI except for PMD, which is the only tool that has been developed to support both a Textual User Interface (TUI) and a GUI.

We use the output of these tools considered as experts for feeding the data mining algorithms according to the following criteria. If one tool or more detect God Class Design Smell in a particular class, we assign a true value in the God Class attribute. Otherwise, this attribute is set false (as a logical or among them). According to this strategy, the presence of the God Class smell is distributed along the different size categories of the data we are dealing with. Dataset is available on the web[2].

Despite the high number of projects, our dataset includes 1,958 God Classes against 12,588 classes. According to our experience in the area, the nature of God Class Design Smell leads to obtaining low ratios of smelly classes detected in each project. As we can observe from Table III, the number of God Classes in all categories are enough as inputs to the classifiers for God Class detection. Our dataset includes only one God Class smell in the small size category (S). In our experience it is normal in a small size project to detect one or at most two God Classes if the project is not a complete disaster. We can either discard this category or add more projects if we find. But we do not, because in this case, the small category do not represent a subset of all dataset or cause a problem if the H_0^{size}. We decide to include this category in our experiments

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

[2]https://citius.usc.es/investigacion/datasets/project-nominal-information

because we choose all classes in each project to send feedback to the project developers.

TABLE III
BUILDING SET, TESTING SET, AND GOD CLASS DISTRIBUTION BY THE SIZE CATEGORIES.(NP: NUMBER OF PROJECT, NOC: NUMBER OF CLASS, GC: GOD CLASSES)

Category	#NP	#NOC	#GC	%GC
L (Testing set)	6	4,763	967	20%
M-L (Building set)	8	5,123	623	12%
M-L (Testing set)		1,281	132	10%
M (Testing set)	7	1,281	198	16%
S-M (Testing set)	2	112	34	30%
S (Testing set)	1	28	1	4%

The high number of projects in a specific category of the size does not mean having more God classes such as (L, M) categories, but when the number of classes increases in a particular category, it implies increasing the number of God Classes. Therefore, it is difficult to balance the dataset categories. The high number of categories implies a lot of subsets to have God classes distributed across them. We would need to increase the number of projects (and classes) significantly to obtain God Class in all of them. Instead of this, we modify our intention and only will use the M-L size category to build classifiers as explained in the next section.

V. METHODOLOGY

In state of the art for Design Smell detection with machine learning algorithms, a different set of classifiers have been used. We choose a set of eight supervised machine learning techniques that are available in Weka version 3.7 [23], which is a comprehensive collection of machine learning algorithm tools used for data analysis and predictive modeling. The high number of selected techniques allows discarding one of them if it is obtained a particular behaviour or it seems to depend on specific data. We work based on different approaches with the default parameters [NaiveBayes (NB), J48, RandomForest (RF), JRip, SMO, IBK, CHIRP and RandomCommittee (RC)]. We decide training with default parameters in all cases because we are not tuning for obtaining the best classifier. What we want to show is that whatever the classifier is, it is a better classifier if the project size category is taken into account.

Firstly, as an exploratory work, we have trained several classifiers with and without size information (x_{17}) to verify whether the use of this type of information increases the quality of classifiers or not. Secondly, the methodology we propose consists of obtaining a classifier trained with projects with the same value for size nominal variable (x_{17}).

As we stated in Section III, the null hypothesis was formulated as size project information does not affect on God Class detection. The intention is to reject each null hypothesis (H_0^{size}). We assume in our experimental design that if size information is not important, a classifier built for the category Medium-Large (**M-L**) of size category info (x_{17}), should behave the same when classifying projects from different size categories. If this does not happen and the classifiers behave worse, we can reject the null hypothesis H_0^{size} and say that the size category information is important.

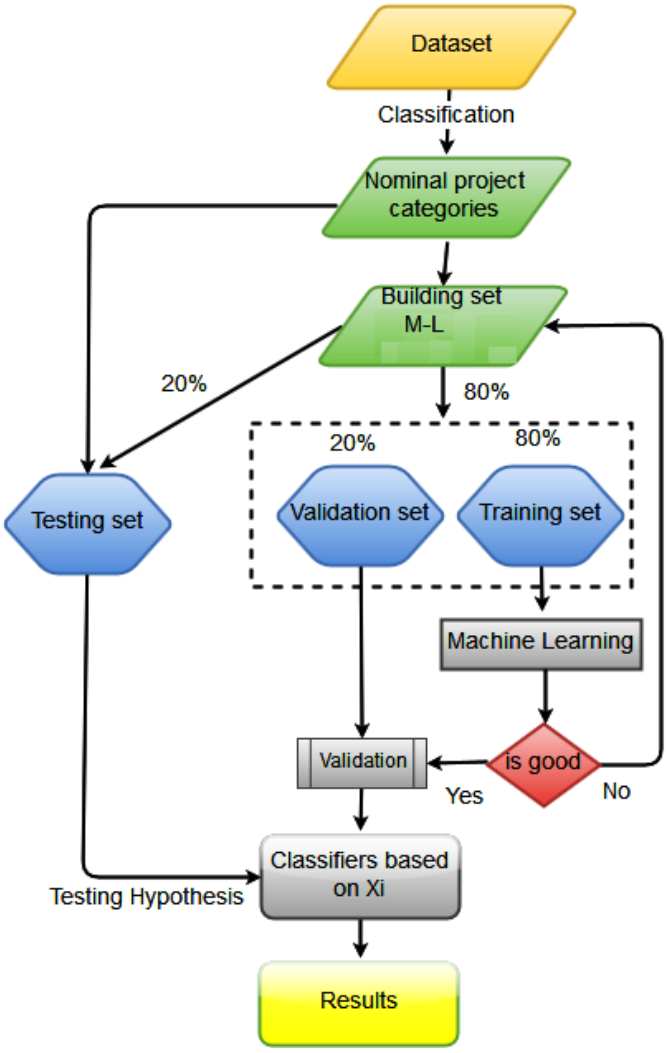

Fig. 1. proposed approach.

Figure 1 summarizes the proposed methodology to obtain the trained classifiers that we will use in the hypothesis testing process. The building set, i.e., the set devoted for building a classifier uses the 80% of the projects in **M-L** category for x_{17}. The testing set contains the projects in the rest of the categories of size (L, M, S-M, S) plus the remaining 20% from the category that has the highest number of classes (M-L) previously selected to be the building set. This is the set used to test the hypothesis. The set of classes in building and testing data are completely different and are distributed on different project categories.

The building set is divided into two parts. The first part includes 80% to be the training set and the remaining 20% to be the validation set. The training set is supplied to the machine learning algorithm to obtain the required classifier. After that, the validation set is used to validate it. If the classifier is good, the testing set is supplied to the classifier to test the hypothesis. Otherwise, we repeat the process of

using the training and validation set until a good classifier is obtained.

The ROC area test obtains a comprehensive effectiveness evaluation of classifiers. This test shows the relation between the sensitivity and the specificity of the classifiers. Table IV shows the traditional classification of this test.

TABLE IV
INTERPRETATION OF THE ROC AREA (R).

ROC value	Interpretation
$0.5 < ROC \leq 0.6$	Fail
$0.6 < ROC \leq 0.7$	Poor
$0.7 < ROC \leq 0.8$	Fair
$0.8 < ROC \leq 0.9$	Good
$0.9 < ROC \leq 1$	Excellent

Kappa measures the degree of agreement (or concordance) of the nominal or ordinal assessments made by appraisers when assessing the same samples. Kappa can range from -1 to $+1$. The higher the value of Kappa, the stronger the agreement. Table V shows the interpretation of this coefficient.

TABLE V
INTERPRETATION OF THE KAPPA VALUES (K).

Kappa value	Degree of Agreement
$k < 0.20$	Poor
$0.21 \leq k < 0.40$	Fair (Weak)
$0.41 \leq k < 0.60$	Moderate
$0.61 \leq k < 0.80$	Substantial (Good)
$0.81 \leq k \leq 1.00$	Almost perfect (Very Good)

VI. RESULTS AND DISCUSSION

In this section, we present the results of our experiments regarding the influence of the size category nominal information on the detection of God Class to reject the null hypothesis.

Table VI shows the classifiers results when we trained them with all dataset as a whole without size category information (cat. info.) and with size category information. In the ROC area, the majority of classifiers obtain a Good to Excellent behaviour with and without size information except for IBk and SMO in the first case (without size) and JRip in the second case(with size). In particular, SMO's behaviour according to ROC is fair and almost fair in both cases. On the other hand, in the Kappa test, which is considered better indicator because the ratio of detected God Class in the most categories is less than 20%, the classifiers obtain a better result with the selected size information than without this information. In this case, SMO and NB obtained the worst results in general (0.467, 0.4967), respectively. According to Kappa analysis, the behaviour of classifiers is improved with the size information. Based on this, we conducted the experiment explained in Figure 1 on the size in order to analyze H_0^{size}.

Figure 2 shows the ROC area results of the classifiers trained regarding a single category (M-L building set) for project size information x_{17}. All classifiers when exercised with the testing set for the same values of the categories (M-L) for size,

TABLE VI
TRAINED CLASSIFIERS

	Without size cat. Info.		*With size cat. Info.*	
Classifier	ROC	Kappa	ROC	Kappa
IBK	0.776	0.3062	0.989	0.934
CHAIRP	0.85	0.5505	0.812	0.7041
J48	0.874	0.4572	0.941	0.7936
JRip	0.864	0.529	0.772	0.6097
SMO	0.703	0.5054	0.683	0.467
NB	0.946	0.5659	0.86	0.4967
RC	0.918	0.4799	1	0.9986
RF	0.944	0.5479	1	0.9982

that were used for building the classifiers obtain the highest values of ROC area compared with the rest of the categories in size nominal information (the top black line of the figure). It deserves to say that we are not training the classifiers to be the best. As we stated before, we are just working with default parameters. What we are showing here is that the same algorithm is better being sensitive to the project size category in which it was trained. So, it is considered good evidence that taking into account the information we are analyzing in this work for God Class detection would improve results.

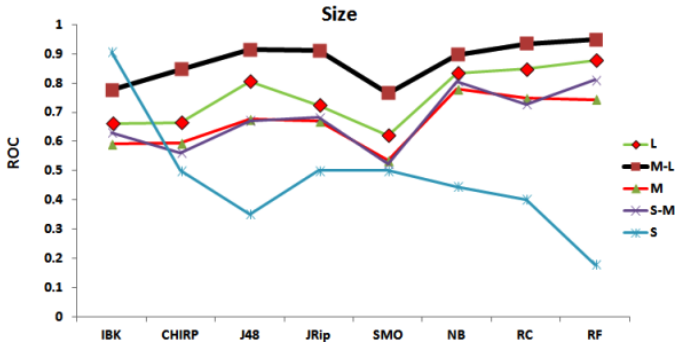

Fig. 2. ROC area performance.

Figure 3 presents the results of the Kappa test for the classifiers trained regarding a single category (building set) for project size information. The testing set with the same value in the size category (M-L) is marked as the black highlighted line in the top of the figure. The better Kappa values are obtained, compared with the rest of the categories except NB in the (M-L) category. Based on the Kappa results, we can say, the size nominal information influence on the detection of God Class. Therefore, we can reject the null hypothesis H_0^{size}.

VII. THREATS TO VALIDITY

In this paper, we highlighted a set of threats to internal validity that affects negatively on our experiment such as the disproportionate number of project in every category of the size nominal information. On the other hand, as threats to external validity, all project is written in Java in order to use the selected set of tools. We did not include the "very large" size projects. Also, the dataset did not include a set of different versions of the same project. This work is focusing on one type of smell (God Class). We need to study other smells that can be detected in different software artifacts. Another threat is

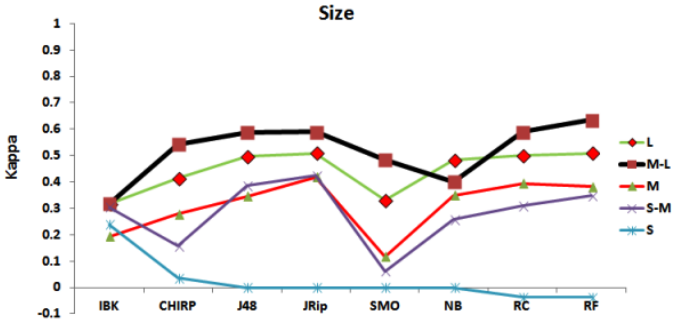

Fig. 3. Kappa values.

the limited number of class level metrics we use and the tools used to analyze the projects (in the metric collection and smell detection).

VIII. Conclusions and Future Work

Our study presented an experimental approach based on machine learning techniques to identify the effective influence of the size category of the project regarding the detection of God Class. In this paper, we present an exploratory study to check whether this information is relevant to be supplied to the classifier and can lead to variations in the classification usefulness, mainly concerning effectiveness (ROC) and agreement (Kappa). All classifiers behave worst according to ROC area and Kappa value tests for categories that were not present when building the classifiers. This study concludes the importance of the size category of the project in the Design Smell detection through machine learning classifiers and should be taken into account in the future works in order to obtain more useful classifiers.

A large dataset formed by the $12,588$ classes of 24 systems with different domains and size categories was defined. The classes in the dataset were analyzed by the five different tools selected as experts for God Class detection.

Our future work will focus on replicating the experiments reported in this work rather than using automatic tools as experts we will involve professional human experts from the industry in identifying true God Classes. The same approach could also be extended to study other software context information, such as domain, programming language. Also including other types of design smell, machine learning techniques, and metrics, to confirm whether the proposed methodology has general applicability.

References

[1] Khalid Alkharabsheh, Yania Crespo, Esperanza Manso, and José A. Taboada. Software design smell detection: a systematic mapping study. *Software Quality Journal*, pages 1–80, Oct 2018.

[2] N.J.M. Blackman and J.J. Koval. Interval estimation for cohen's kappa as a measure of agreement. *Statistics in medicine*, 19(5):723–741, 2000.

[3] Borland. Together. http://www.borland.com/us/products/together. [Accessed: 2014-04-06].

[4] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[5] William J. Brown, Raphael C. Malveau, Hays W. McCormick III, and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, March 1998.

[6] Ward Cunningham. The wycash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2):29–30, 1993.

[7] Francesca Arcelli Fontana, Vincenzo Ferme, Armando Marino, Bohme Walter, and Pawel Martenka. Investigating the impact of code smells on systems quality: An empirical study on systems of different application domains. In 29^{th} *IEEE International Conference on Software Maintenance*, pages 260 – 269, September 2013.

[8] Francesca Arcelli Fontana and Stefano Spinelli. Impact of refactoring on quality code evaluation. In 4^{th} *Workshop on Refactoring Tools*, pages 37–40, New York, NY, USA, 2011.

[9] Francesca Arcelli Fontana, M. Zanoni, Armando Marino, and Mika V. Mantyla. Code smell detection: Towards a machine learning-based approach. In 29^{th} *IEEE International Conference on Software Maintenance*, pages 396 – 399, September 2013.

[10] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Object Technology Series. Addison-Wesley, June 1999.

[11] Francisco Javier Prez García. *Refactoring Planning for Design Smell Correction in Object-Oriented Software*. PhD thesis, Universidad de Valladolid, Valladolid, 2011.

[12] A. Hamid, M. Ilyas, M. Hummayun, and A. Nawaz. A Comparative Study on Code Smell Detection Tools. *International Journal of Advanced Science and Technology*, 60:25–32, 2013.

[13] Foutse Khomh, Stephane Vaucher, Yann-Gaél Guéhéneuc, and Houari Sahraoui. BDTEX: A GQM-based bayesian approach for the detection of antipatterns. *The Journal of Systems and Software*, 84(4):559–572, 2011.

[14] Jochen Kreimer. Adaptive detection of design flaws. *Electronic Notes in Theoretical Computer Science*, 141(4):117–136, December 2005.

[15] 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, 2006.

[16] Abdou Maiga, Nasir Ali, Neelesh Bhattacharya, Aminata Saban andYann Gal Guéhéneuc, Giuliano Antoniol, and Esma Ameur. Support vector machines for anti-pattern detection. In 27^{th} *IEEE/ACM International Conference on Automated Software Engineering.*, pages 278–281, September 2012.

[17] Nakarin Maneerat and Pomsiri Muenchaisri. Bad-smell prediction from software design model using machine learning techniques. In 8^{th} *International Joint Conference on Computer Science and Software Engineering*, pages 331 – 336, May 2011.

[18] Cristina Marinescu, Radu Marinescu, Petru Florin Mihancea, Daniel Ratiu, and Richard Wettel. iPlasma: An integrated platform for quality assessment of object-oriented design. In 21^{st} *International Conference on Software Maintenance - Industrial and Tool Volume*, pages 77–80, Budapest, Hungary, September 2005.

[19] Raúl Marticorena, Carlos López, and Yania Crespo. Extending a taxonomy of bad code smells with metrics. In *7th ECCOP International Workshop on Object-Oriented Reengineering (WOOR)*, page 6. Citeseer, 2006.

[20] Naouel Moha and Yann-Gael Guéhéneuc. DÉCOR: A tool for the detection of design defects. In 22^{nd} *IEEE/ACM International Conference on Automated Software Engineering*, pages 527–528, New York, NY, USA, 2007.

[21] Javier Pérez, Carlos López, Naouel Moha, and Tom Mens. A classification framework and survey for design smell management. Technical Report 2011/01, Grupo GIRO, Departamento de Informática, Universidad de Valladolid, March 2011.

[22] Nikolaos Tsantalis, Tsantalis Chaikalis, and Alexander Chatzigeorgiou. JDeodorant: Identification and removal of type-checking bad smells. In 12^{th} *European Conference on Software Maintenance and Reengineering*, pages 329–331, April 2008.

[23] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining Practical Machine Learning Tools and Techniques*. Elsevier, 2011.