INTERNATIONAL JOURNAL OF ADVANCED STUDIES IN COMPUTER SCIENCE AND ENGINEERING
IJASCSE VOLUME 5 ISSUE 11, 2016

11/30/2016

Influence of Nominal Project Knowledge in the Detection of Design Smells: An Exploratory Study with God Class

Khalid Alkharabsheh, Shahed Almobydeen, Jose A. Taboada
Universidad de Santiago de Compostela, CITIUS
Santiago de Compostela, Spain
{khalid.alkharabsheh, shahed.almobydeen, joseangel.taboada}@usc.es

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

***Abstract*—Several Design Smell detection tools have been developed for identifying Design Smells in source code or design models. The early prediction of a useful set of Design Smells has a positive impact on software quality. In this paper, we present an exploratory study to check whether some project information can be relevant or not to be supplied to a classifier in order to detect God Class Design Smell. This paper explores if clarifying the domain, the status and the size category of the project to which a class belongs to can lead to variations in the classification accuracy and usefulness for this Design Smell detection. The dataset is formed by the 12,588 classes of 24 projects with different size categories, domains and maturity status. We conduct the experiments with eight different machine learning approaches which are the most recently used in literature. These eight involve all families of classifiers. The results of classifiers are compared based on the accuracy, sensitivity and specificity performance significance tests. It was found that the set of nominal project knowledge studied in this paper have not any impact on the detection of God Class Design Smell based on the set of detection tools were used to identify the God Class Design Smell.**

Keywords-design smell detection; god class; machine learning

I. INTRODUCTION

Software quality is an important concern for software industries, academic and researchers. To improve quality we should perform maintaining activities as early as possible through the software development life cycle. The majority of software development cost was devoted to maintenance process [1]. Maintenance process is influenced by the amount and frequency of maintenance tasks related to adaptive, corrective and predictive maintenance, the wide range of different tools required for controlling, documenting and making changes effectively, and by the degree of code complexity. Design Smells presence is one of the critical problems that impact also on the process.

The concept of Design Smell cover all problems related to the software structure (source code and design) that does not make compile or execution errors [2]. But as consequence, Design Smells' presence negatively affect on software understandability, testability, extensibility, reusability and maintainability factors. These problems can appear in several software artifacts from fine-grained to coarse-grained including (variables, instructions, operations, methods, classes, packages, subsystems, layers and their dependencies). Design Smell detection is an attractive research field for researches community due to the critical role of Design Smell detection over improving software quality.

Several studies, approaches and techniques have been proposed to detect Design Smells. However, they have a set of limitations represented in understanding the precise definition of Design Smells and the process of map definition into effective detection algorithms. According to Fowler who introduce the concept for 22 Bad Smells, Large Class is a class that try to do much (i.e., it has many responsibilities and instance variables) [3]. Brown mentioned that to decide a class as a Blob Antipattern, we should analyze the class and if it contains more than 60 methods and attributes it indicates the presence of the Blob [4]. According to Rațiu suggest to detect God Class considering they are classes that use a lot of data from the closer classes and have high complexity or low cohesion between methods [5]. Santos design a controlled experiment that focus on identifying how different experts understand the concept of God Class, the answers arise that if the class or methods in the class have a high complexity is considered as God Class [6]. Large class Bad Smell, The Blob Antipattern and God Class are closely related Smells in Code defined by different authors with different names. Design Smell is a unifying term we use and God Class is the name we use for bringing both together.

Many commercial or open source tools have been developed to aid developers in detecting Design Smells automatically, semi-automatically or manually through software development phases to support maintenance activities that will improve the software quality. Most of detection techniques handle a set of combined metrics, standard object-oriented metrics, or metrics defined ad hoc for the smell detection purpose [7]. Even in case a set of different detection tools used the same set of metrics, the threshold value will vary for each tool. Consequently, we obtain different results as the tool used is different. Some of tools can generate metrics by their own such as iPlasma [8] [9] and Borland Together [10] and other tools use an input metrics generated by third party tools. A few tools have been adopted machine learning techniques for deriving detection rules.

The rest of paper is organized as follows. Section 2 describes the related work. Section 3 presents the proposed work methodology. Section 4 describes our experiments on nominal project information influence when detecting Design Smells. Section 5 discusses the results of experiments. Section 6 explains our conclusions.

II. RELATED WORK

In our the state of the art of Design Smell detection, we found a few studies regarding Design Smell detection that apply machine learning techniques. Fontana outlined the common problems in the previous Design Smells detection and suggests a new approach based on machine learning[11]. The dataset consists of 76 systems from different sizes, domains, and a large set of relevant metrics. A set of six Design Smells (God Class, Data Class, Feature Envy, God Method, Brain Method, Long Method) were predicted in the experiment. The dataset was used as input to six machine learning algorithms J48, RandomForest, Naive Bayes, JRip, SMO, LibSVM. The accuracy of classifiers was different in prediction from one smell to another and the majority of classifiers obtain performance more than 90% .

Maneerat and Muenchaisri present a methodology for predicting Design Smells [12]. They collect 7 dataset from the previous literatures consist from seven Design Smells (Lazy Class, Feature Envy, Middle Man, Message Chains, Long Method, Long Parameter list, Switch Statement) detected in design models and a set of twenty-seven software metrics. They trained and tested the dataset to predict Design Smells using seven machine learning algorithms RandomForest, Naive Bayes, Logistic Regression, lBl, lBk, VFI, J48. Statistical significant tests were used to evaluate the accuracy, sensitivity, and specificity. They found some Design Smells could be predicted earlier from software design model. Furthermore, the difficulty to identify the best machine learning algorithm that accurately predict the seven Design Smells arose because when they compare the results of accuracy they found them close to each other and some algorithms achieve more than 95% of prediction accuracy for some Design Smells.

BDTEX (Bayesian Detection Expert), a Goal Question Metric (GQM) based approach to building Bayesian Belief Networks (BBNs) from the definitions of Antipatterns stood on rule based representation presented in [13]. The experiment applied on two projects from different domain and size to validate three Design Smells (Antipatterns in this case) Blob, Functional Decomposition and Spaghetti Code. The result with this approach obtained high values in terms of precision, recall and utility comparing with another well know Design smell detection tools such as Décor [18].

Kreimer suggested combining popular methods to detect Design Smells based on metrics and machine learning techniques [14]. The aim of the study was to find errors in design models. The dataset is formed by 688 classes of 2 systems and include five smells Big Class also known as (Blob), Feature Envy, Long Method, Lazy Class and Delegator and use J48 Classifier technique. Good accuracy was obtained with the proposed approach

As we can see, the related works principally focused on numerical knowledge (metrics) on classes to detect sets of Design Smells (Bad Smells or Antipatterns). They did the experimentations without any nominal information on the projects which are the context of these classes. Hence, we do not know if that can influence in the way the classifier predicts. On the other hand, in these works authors usually experiment with a set of machine learning techniques that did not include all families of supervised learning algorithms.

In our work, our attention focused on certain nominal project knowledge and try to explore its importance regarding the prediction of God Class Design Smell. God Class is a class level Design Smell. We want to check whether some of the above mentioned nominal knowledge on the project to which the analyzed classes belong can influence the detection and if it is important to take this information into account in order to obtain better predictions that can be more useful for developers. In this paper we present an exploratory study to check whether some nominal information on the project of the classes is relevant to be supplied to the classifier. We are analyzing if clarifying the domain, the project status and the size category of the project to which the class belongs to can lead to variations in the classification accuracy and usefulness. The dataset we used is form by the 12,588 classes of 24 systems. We conduct the experiments with eight different machine learning approaches which are the most recently used and they jointly involve all families of classifiers.

Wherever Times is specified, Times Roman or Times New Roman may be used. If neither is available on your word processor, please use the font closest in appearance to Times. Avoid using bit-mapped fonts if possible. True-Type 1 or Open Type fonts are preferred. Please embed symbol fonts, as well, for math, etc.

III. PROPOSED WORK

In a previous study on the state of the art we found God Class (class level) Design Smell is one of the most detected smell in software and has a great interest for researchers community. We conducted a comparison between five tools (PMD [15][16], Borland Together [10], JDeodorant [17], iPlasma [8][9] and Decor [18][19]) to assess the degree of agreement on detecting God Class. We found different results for detecting the same smell in the same set of classes. Therefore, in this paper God Class Design smell was the selected as a case study to design our experiments because we are looking for reasons that justifies these lack of agreement.

In this study we investigate and analyze the impact of three nominal project information in the God Class Design Smell detection. These information involve different project domains, project status, and size categories for the whole project. To do this, we introduce a research question addressing the influence of different nominal knowledge in the detection of God Class Design Smell. The research question is:

Q1: Does the differences between project domains, project status, and the size category of the whole project influence in the detection of God Class Design Smell?

In order to answer the research question, we design a set of separated experiments to investigate the impact of domain, status, and size categories in the behavior of classifiers for God Class detection. We determine the dataset we will use to cover these nominal information. Our experiments required dividing the dataset into two groups for training and testing, the size of training set was approximately 75% from all dataset and the rest 25% was used for testing, as we will see in the experimentation section in details. To conduct the proposed approach, the following subsections summarize how we can formalize our dataset to be supplied to machine learning algorithms for obtaining the classifiers.

A. Dataset

Our dataset include a set of 24 open source software systems written in Java obtained from SourceForge source code repository which involved different domains, size categories and status. We follow the same approach as Fontana to classify the projects based on size category and domains [20]. The nominal information of size categories refers to the size of Total Line of Code in the whole project (TLOCP). The size category of the whole project is divided into five categories based on the TLOCP include (Small 0-4999, Small-medium 5000-14999, Medium 15000-39999, Medium-Large 40000-99999, Large 100000-499999). Also, the domain of projects is classified into four different categories where a set of different particular domains belongs to each domain category as following:

- Application software: (Word Processor, Web Browser, Accounting, Graphic and Player.
- Software development: Ide, Parsers/Generators/Make, SDK, Testing.
- Diagram generator/data visualization: GUI Design Tool.
- Client server software: Database, Application Server, Middleware, CMS.

The projects status is obtained as it is declared in the SourceForge source code repository were involve seven categories(planning, pre-alpha, alpha, beta, production/stable, mature, inactive).

The selected projects are analyzed using RefactorIT 2.7 tool to compute a set of important metrics relevant with class level artifacts as we see in Table I [21]. The selected metrics cover different aspects of the source code such as complexity, inheritance, size, cohesion and coupling.

TABLE I. LIST OF METRICS AT CLASS LEVEL

Metric	Definition
LOC	Total Lines of Code
NCLOC	Non-Comment Lines of Code
CLOC	Comment Lines of Code
EXEC	Executable Statements
DC	Density of Comments
NOT	Number of Types
NOTa	Number of Abstract Types
NOTc	Number of Concrete Types
NOTe	Number of Exported Types
RFC	Response for Class
WMC	Weighted Methods per Class
DIT	Depth in Tree
NOC	Number of Children in Tree
DIP	Dependency Inversion Principle
LCOM	Lack of Cohesion of Methods
NOA	Number of Attributes

Table II presents the main characteristics of our dataset were the following abbreviation in the table refer to the different nominal information and stand for [large (L), Medium-Large (M-L), Medium (M), Medium-Small (M-S), Small (S), Application software (App), Software development (Dev), Diagram generator/data visualization (Vis), Client server software (Cli), Production/Stable (P/S), Beta (B), Mature (Mat), Multiple-status (Mul)].

Fig. 1 represent the format of our dataset as we shown every row in the dataset represent sixteen numerical attributes, the three nominal information, and the result of classification if a class smelly or not.

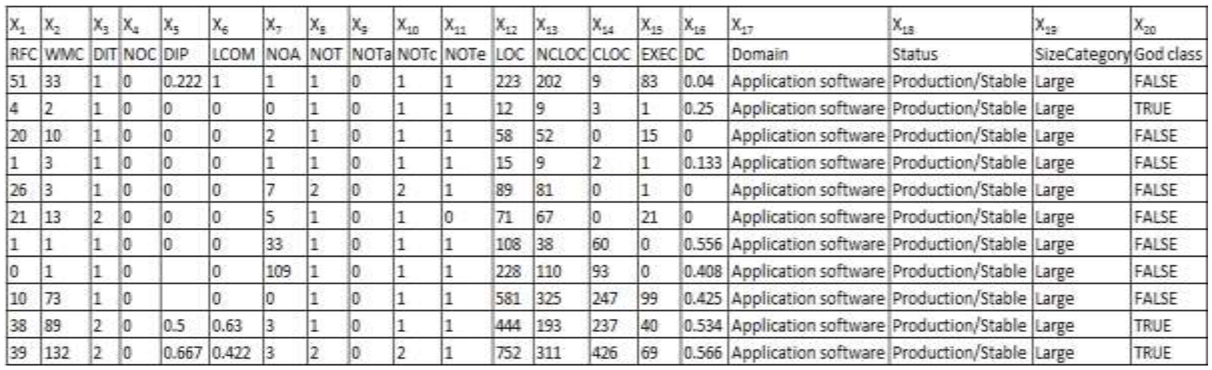
B. Machine Learning Techniques

We choose a set of supervised machine learning techniques that are available in WEKA version 3.7 [22]. In the state of art for Design Smell detection and machine learning techniques in our previous study, the selected techniques was the most recently used, they jointly involve all classifiers' families, and work based on different approaches [NaiveBayes (NB), J48, RandomForest (RF), JRip, LibSVM, IBK, RandomCommittee (RC) and InputMappedClassifier (IMC)].

The methodology we propose consists in obtaining a classifier trained with projects with the same value for the nominal variable (X_{17}, X_{18}, X_{19}). This value is selected as the most frequented in the dataset. The selected values are Application Software for X_{17}, medium-large for X_{18}, and Production/Stable for X_{19}. The 80% of the projects with this values will be the training set. The attributes we use as input to the training process are $X_1,\ldots,X_{16}$ and the information obtained Smell Detection tools, considered as experts, indicating if the class is a God Class or no. We will test if the classifier obtained for projects with the same value for X_{17} or X_{18} or X_{19} behave the same when used for detecting God Class in projects with different values in this nominal scale.

TABLE II. DATASET CHARACTERISTICS

ID	Name	Version	Domain	Status	Size category	TLOCP
1	jAudio	1.0.4	App	P/S	L	117615
2	Freemind	1.0.1	Vis	P/S	L	106396
3	JasperReports	4.7.1	Dev	Mat	L	350690
4	SQuirreL SQL Client	1.2	Cli	P/S	M-L	71626
5	KeyStore Explorer	5.1	Vis	P/S	M-L	83144
6	DigiExtractor	2.5.2	App	P/S	M	15668
7	Angry IP Scanner	3.0	Vis	P/S, B	M	19965
8	Plugfy	0.6	Dev	B	S	2337
9	Matte	1.7	App	P/S	M-L	52067
10	sMeta	1.0.3	App	P/S	M	30843
11	xena	6.1.0	Dev	P/S	M-L	61526
12	pmd	4.3.x	Dev	P/S	M-L	82885
13	checkstyle	6.2.0	Dev	P/S, Mat	M-L	41104
14	JDistlib	0.3.8	App	P/S	M	32081
15	JCLEC	4-base	Dev	P/S	M	37575
16	Java graphplan	1.0	Dev	B	S-M	1049
17	Mpxj	4.7	App	P/S	L	261971
18	Apeiron	2.92	App	P/S	S-M	8908
19	FullSync	0.10.2	App	B	M	24323
20	OmegaT	3.1.8	App	P/S	L	121909
21	Lucene	3.0.0	App	P/S	M-L	81611
22	Ganttproject	2.0.10	App	P/S	M-L	66540
23	JFreechart	1.0.X	App	P/S	L	206559
24	JHotDraw	5.2	App	B	M	17807



X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}	X_{16}	X_{17}	X_{18}	X_{19}	X_{20}
RFC	WMC	DIT	NOC	DIP	LCOM	NOA	NOT	NOTa	NOTc	NOTe	LOC	NCLOC	CLOC	EXEC	DC	Domain	Status	SizeCategory	God class
51	33	1	0	0.222	1	1	1	0	1	1	223	202	9	83	0.04	Application software	Production/Stable	Large	FALSE
4	2	1	0	0	0	0	1	0	1	1	12	9	3	1	0.25	Application software	Production/Stable	Large	TRUE
20	10	1	0	0	0	2	1	0	1	1	58	52	0	15	0	Application software	Production/Stable	Large	FALSE
1	3	1	0	0	0	1	1	0	1	1	15	9	2	1	0.133	Application software	Production/Stable	Large	FALSE
26	3	1	0	0	0	7	2	0	2	1	89	81	0	1	0	Application software	Production/Stable	Large	FALSE
21	13	2	0	0	0	5	1	0	1	0	71	67	0	21	0	Application software	Production/Stable	Large	FALSE
1	1	1	0	0	0	33	1	0	1	1	108	38	60	0	0.556	Application software	Production/Stable	Large	FALSE
0	1	1	0		0	109	1	0	1	1	228	110	93	0	0.408	Application software	Production/Stable	Large	FALSE
10	73	1	0		0	0	1	0	1	1	581	325	247	99	0.425	Application software	Production/Stable	Large	FALSE
38	89	2	0	0.5	0.63	3	1	0	1	1	444	193	237	40	0.534	Application software	Production/Stable	Large	TRUE
39	132	2	0	0.667	0.422	3	2	0	2	1	752	311	426	69	0.566	Application software	Production/Stable	Large	TRUE

Figure 1. Dataset input format

Fig. 2 summarizes the proposed work to obtain the classifiers that we will use in the testing process.

IV. Experiments

We perform a set of experiments to answer the research question. The same strategy is used in selecting training and testing data in all experiments for analyzing influence of project information. In every experiment on the nominal projects information, we select 80% from the category that have the highest number of projects as a training data. The testing data is prepared manually were contain the rest of categories in the same type of nominal projects information plus the rest 20% from the category that have the highest number of projects. According to this criteria, Table II shows that 80% of the projects in the application software domain include the projects (10, 14, 17, 18, 19, 20, 21, 22, 23, 24), 80% of the projects that have a size category medium-large of whole the project and involve projects (9, 11, 12, 13, 21, 22), and 80% of the production/stable project status include the projects (1, 2, 5, 6, 10, 11, 14, 17, 18, 20, 21, 22, 23) are the selected training dataset and the rest of the projects in the same category, once the training dataset is extracted (the remaining 20%) plus the projects in the rest of the categories become the testing dataset. Table III describes the main characteristic for every experiment regarding the particular nominal information.

TABLE III. Training & Testing Dataset

Nominal	Training	Proj	Testing	Proj
Domain	App	10	Dev, Vis, Cli, 20% App	14
Size	M-L	6	S, S-M,M,L, 20% M-L	18
Status	P/S	13	B, Mat, Mul, 20% P/S	11

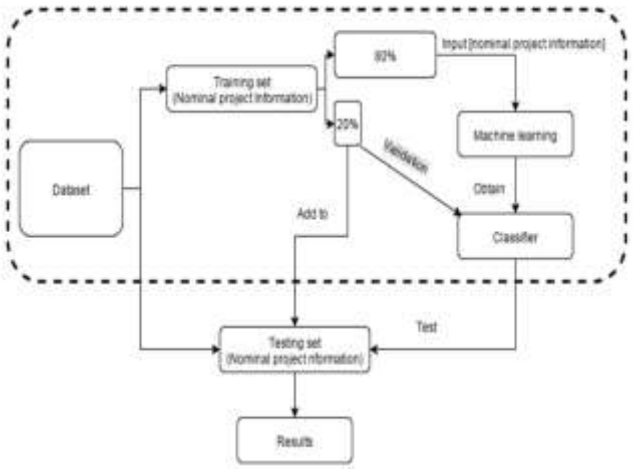

Dataset
Training set (Nominal project information)
80%
20%
Input [nominal project information]
Machine learning
Validation
Obtain
Classifier
Add to
Test
Testing set (Nominal project information)
Results

Figure 2. Proposed approach

Regarding the nominal projects knowledge about the project domain, the training data include 10 projects with 3678 classes while the testing data 14 projects with 8910 classes distributed as the following 7 projects of software development domain with 5238 classes, 3 projects of diagram generator/data visualization with 1436 classes, 1 project of client-server domain with 1138 classes, and 3 projects of Application Software that represent 20% of this category with 1098 classes.

In the size category nominal information the training data involves 6 projects of the M-L category with 5076 classes while the testing data consist of 18 projects formed by 7512 classes partitioned as the following groups: 1 (S) project with 28 classes, 2 projects (S-M) with 112 classes, 1281 classes of 7 (M) projects, 2 projects (M-L) with 1355 classes, and 4763 classes of 6 (L) projects. Finally regarding the project status nominal information the training data include 13 projects of the Production/Stable category with 6964 classes while the testing data formed by 5624 classes divided into 4 projects (P/S) with 2882 classes, 2 (Mul) projects (Checkstyle, Angry IP scanner) with 547 classes, 1 (Mat) project with 1797 classes, and 4 projects (B) with 398 classes.

The experiments were executed on a personal computer with the following properties: Processor Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz, RAM 8.00 GB, Windows 7 Enterprise 64-bit operating system.

A. *Experiments Result*

We used the output of 5 detection tools (PMD, Décor, Together, iPlasma, JDeodorant) as experts in order to introduce this knowledge into the data mining algorithms according to the following criteria. If one tool or more detect God Class in a particular class we assign a true value in the God Class attribute otherwise, this attribute is set false. According to this strategy, the presence of the God Class is distributed along the different categories of the nominal

TABLE IV. Machine learning accuracy performance

Cat/A	IBk	IMC	J48	JRip	libSVM	NB	RF	RC
App	0.77	0.84	0.84	0.86	0.84	0.88	0.87	0.88
Dev	0.78	0.87	0.85	0.86	0.87	0.87	0.85	0.88
Cli	0.80	0.92	0.87	0.88	0.92	0.93	0.86	0.90
Vis	0.81	0.93	0.86	0.87	0.93	0.93	0.88	0.89
S	0.85	0.96	0.96	0.96	0.96	0.96	0.92	0.92
S-M	0.71	0.69	0.75	0.81	0.69	0.71	0.79	0.80
M	0.80	0.84	0.86	0.86	0.84	0.85	0.86	0.87
M-L	0.86	0.91	0.86	0.88	0.91	0.92	0.90	0.92

L	0.78	0.79	0.84	0.85	0.79	0.84	0.84	0.85
B	0.76	0.85	0.85	0.90	0.85	0.86	0.86	0.88
Mat	0.81	0.82	0.85	0.88	0.82	0.86	0.86	0.88
P/S	0.83	0.91	0.90	0.92	0.91	0.92	0.90	0.91
Mul	0.86	0.95	0.92	0.93	0.95	0.94	0.92	0.92

information we are dealing with as follow: 55% of the God Classes are in Application Software projects, 35% are in Software Development projects, 5% are in Client/Server and Diagram generator/data visualization projects, 49% are in Large projects, 39% are in Medium-Large projects, 10% are in Medium projects, 1.7% are in Small-Medium projects, 0.3% are in Small projects, 80% are in Production/Stable projects, 17% are in Mature projects, and 3% in Beta projects.

We present the results of our experiments regarding the influence of certain nominal projects knowledge in the prediction of God Class Design Smell. Tables IV, V, and VI show the results of accuracy, sensitivity, and specificity statistical significance test for the performance of the eight machine learning techniques IBK, IMC, J48, JRip, LibSVM, NB, RC, and RF. The remaining 20% from the category that have the highest number of projects is App, M-L, and P/S. The main observation from Table IV explains that the accuracy performance for all classifiers is closer to each other in the same nominal project information or among different types of nominal projects information.

Table V describes the result of sensitivity performance that present the ratios of true positive in all classifiers, we can show that the performance is low in (Dev), (Cli), and (S-M) nominal projects information. While in the project status nominal information is close to each others.

The specificity performance shown in Table VI presents the true negative percentage in the selected classifiers. (Dev) and (Cli) nominal domains have the highest percentage of performance comparing with all categories in the same nominal projects information or other categories. IMC and LIBSVM classifiers have the lowest percentage of specificity performance.

TABLE V. MACHINE LEARNING SENSITIVITY PERFORMANCE

Cat/A	IBk	IMC	J48	JRip	libSVM	NB	RF	RC
App	0.89	0.84	0.92	0.92	0.84	0.90	0.92	0.92
Dev	0.29	0	0.45	0.47	0	0.53	0.46	0.54
Cli	0.23	0	0.36	0.39	0	0.56	0.34	0.43
Vis	0.98	0.93	0.99	0.99	0.93	0.98	0.99	0.99
S	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96
S-M	0.74	0.69	0.77	0.80	0.69	0.72	0.77	0.78
M	0.86	0.84	0.87	0.87	0.84	0.86	0.87	0.87
M-L	0.94	0.91	0.94	0.96	0.91	0.95	0.94	0.95
L	0.85	0.79	0.85	0.86	0.79	0.85	0.84	0.85
B	0.87	0.85	0.89	0.91	0.85	0.86	0.90	0.91
Mat	0.88	0.82	0.89	0.89	0.82	0.86	0.88	0.89
P/S	0.94	0.91	0.95	0.95	0.91	0.93	0.95	0.95
Mul	0.96	0.95	0.96	0.96	0.95	0.95	0.96	0.96

TABLE VI. MACHINE LEARNING SPECIFICITY PERFORMANCE

Cat/A	IBk	IMC	J48	JRip	libSVM	NB	RF	RC
App	0.34	0	0.50	0.56	0	0.74	0.60	0.65
Dev	0.91	0.87	0.93	0.93	0.87	0.91	0.94	0.93
Cli	0.96	0.92	0.98	0.98	0.92	0.96	0.98	0.98
Vis	0.22	0	0.30	0.33	0	0.49	0.35	0.36
S	0	0	0	0	0	0	0	0
S-M	0.55	0	0.65	0.84	0	0.60	1	0.92
M	0.31	0	0.71	0.67	0	0.70	0.70	0.80
M-L	0.31	0	0.32	0.40	0	0.61	0.49	0.57
L	0.47	0	0.73	0.74	0	0.78	0.78	0.85
B	0.21	0	0.50	0.78	0	1	0.52	0.61
Mat	0.45	0	0.63	0.75	0	0.91	0.70	0.76
P/S	0.23	0	0.41	0.52	0	0.69	0.44	0.48
Mul	0.09	0	0.17	0.24	0	0	0.20	0.23

V. DISCUSSION

In this study our attention focused on investigating and analyzing certain nominal information on projects that can influence on the prediction of God Class Design Smell in a source code. We introduce a research question addressing the impact and importance of different projects status, domains, and the size category of the whole project to which the class belongs to on the detection of God Class Design Smell.

Our results in this study show that all machine learning techniques that participated in the experiments obtain more than 80 % of accuracy performance as we show in Table VI in all categories of nominal projects information and the accuracy values are closer to each other. So, in this case we cannot identify the best classifiers that predict the God Class Design Smell. This mean, all classifiers have the same behavior in the prediction of God Class Design Smell among different nominal projects information. We can say that the selected nominal information were include domain, size category of whole of the project, and the project status have not any influence on the detection of God Class Design Smell based on the knowledge supplied for automatic detection tools PMD, Decor, JDeodorant, Borland Together, and iPlasma that used to identify the smell in our dataset. The detection tool did not take into account from the beginning the importance of this kind of nominal information in order to obtain better prediction.

All classifiers achieved more than 90% of sensitivity performance (see Table V) when the projects domain is Diagram generator/data visualization (Vis), the size category of system is Small (S) or Medium-Large (M-L), and the project status Production/Stable (P/S) or the project have more than one status (Mul). In contrast, all classifiers obtain less than 55% of sensitivity performance when the domain of projects is Software Development or Client-Server. The specificity performance is very low for all classifiers in all nominal projects information except two categories in the domain Software Development and Client-Server.

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper.

VI. THREATS TO VALIDITY

The main factors that negatively affect the internal validity of our experiments are the disproportionate number of projects in size categories, domains, and projects status. Also, the number of classes in each project especially in size categories nominal information. The external validity affected by the nature of analyzed projects were all projects written Java, have not a very large size category, and does not include several versions from the same projects.

VII. CONCLUSION

This paper described the approach we are following based on machine learning techniques to identify the influence of relevant nominal project knowledge in the detection of God Class Design Smells. Our attention focused on the importance of size categories, domains, and projects status nominal information regarding the prediction of God Class Design Smell.

In this paper, we present an exploratory study to check whether some nominal information is relevant to be supplied to the classifier and can lead to variations in the classification accuracy and usefulness. The dataset is formed by the 12,588 classes of 24 systems with different size categories, domains, and project status. We conduct the experiments with eight different machine learning techniques include RandomForest, NaiveBayes, RandomCommittee, JRip, IBK, InputMappedClassifier, LibSVM, and J48 which are the most recently used and involved all families of classifiers. Then, we compare the classifiers performance based on the accuracy, sensitivity, and specificity significance test to identify if the behavior of classifiers will be the same or different in the prediction of God Class Design Smell in the selected nominal projects information.

We find that all classifiers have the same behavior, and this mean no difference among them. We can say that selected nominal knowledge in this paper have not any influence on the God Class detection because the detection tools (PMD, DECOR, Borland Together, iPlasma, JDeodorant) that used in our previous study to detect God Class in the dataset did not include this kind of nominal information in the detection strategy. However, we cannot discard the importance of this nominal information to identify the true God Class in other detection tools.

Our future work will focus on repeating the same study based on human experts to identify the true God Class instead of Automatic tools. The same approach can extended to study other Design Smells.

ACKNOWLEDGMENT

This research is funded by Erasmus program Lot 2 project, led by the University of Santiago de Compostela (Spain).

REFERENCES

[1] T. Mens and T. Tourwe. "A Survey of Software Refactoring". IEEE Transactions on Software Engineering,vol.30, pp. 126-139, Feb. 2004.

[2] F.P. García."Refactoring planning for Design Smell correction in object oriented software." Ph.D. thesis, Universidad de Valladolid, Spain, 2011.

[3] K. Beck and M. Fowler. "Bad Smells in Code," in Refactoring: Improving the Design of Existing Code, 1st ed., vol. 3. J. Shanklin Ed. USA: Addison Wesley, 1999, pp.63-72.

[4] W.J. Brown, R. Malveau, H.W. McCormick III, and T. J. Mowbray. Antipatterns refactoring software architectures and projects in crisis. USA: John Wiley & Sons Inc, 1998, pp. 42-47.

[5] D. Raţiu, S. Ducasse, T. Gîrba, and R. Marinescu. "Using History Information to Improve Design Flaws Detection,". in Proc. CSMR, 2004, pp. 223-232.

[6] J. Santos, M. Mendonça, and C. Silva. "An exploratory study to investigate the impact of conceptualization in God Class detection," in Proc. EASE, 2013, pp. 48-59.

[7] M. Lanza, and R. Marinescu. Object-Oriented Metrics in Practice: Using Software Metrics to Characterize. Evaluate. and Improve the Design of Object-Oriented Systems. Verlag Berlin Heidelberg: Springer, 2006, pp. 65-70.

[8] C. Marinescu, R. Marinescu, P. Mihancea, D. Raţiu, and R. Wettel. "iPlasma: An integrated platform for quality assessment of object oriented design," in Proc. ICSM, 2005, pp. 77-80.

[9] iPlasma. "iPlasma." Internet: http://loose.upt.ro/iplasma/index.html, [May. 10, 2015].

[10] Borland Together. "Borland Together." Internet: http://www.borland.com/us/products/together, [May. 10, 2015].

[11] F. A. Fontana, M. Zanoni, A. Marino, and M.V. Mäntylä. "Code Smell Detection: Towards a Machine Learning-based Approach," presented at the 29th Int. Conf. on Software Maintenance, Eindhoven, Netherlands, 2013.

[12] N. Maneerat and P. Muenchaisri. "Bad-smell Prediction from Software Design Model Using Machine Learning Techniques," presented at the 8th Int. Joint Conf. on Computer Science and Software Engineering (JCSSE), Nakhon Pathom, Thailand, 2011.

[13] F. Khomh, S. Vaucher, Y. Guéhéneuc, and H. Sahraoui. "BDTEX: A GQM-based Bayesian approach for the detection of antipatterns". The Journal of Systems and Software, vol. 84, pp. 559-572, Apr. 2011.

[14] J. Kreimer. "Adaptive Detection of Design Flaws". Electronic Notes in Theoretical Computer Science, vol. 141, pp. 117-136, Dec. 2005.

[15] A.F. Fontana and S. Spinelli. "Impact of Refactoring on Quality Code Evaluation," in Proc. WRT, 2011, pp. 37–40.

[16] PMD. "PMD." Internet: http://pmd.sourceforge.net/, [May. 10, 2015].

[17] N. Tsantalis,T. Chaikalis, and A. Chatzigeorgiou. "JDeodorant: Identification and removal of type checking bad smells," in Proc. CSMR, 2008, pp. 329–331.

[18] N. Moha and Y. Guéhéneuc. "Décor: A Tool for the Detection of Design Defects," in Proc. ASE, 2007, pp. 527-528.

[19] N. Moha, Y. Guéhéneuc, L. Duchien, and A. Le Meur. "DECOR: A method for the specification and detection of code and Design Smells". IEEE Transactions on Software Engineering. vol. 36. pp. 20-36. 2010.

[20] F.A. Fontana, V. Ferme, and A. Marino. "Investigating the Impact of Code Smells on System's Quality: An Empirical Study on Systems of Different Application Domains," presented at the Int. conf. on Software Maintenance, Eindhoven, Eindhoven, 2013.

[21] Refactorit. "Refactorit." Internet: http://refactorit.sourceforge.net/, [May. 10, 2015].

[22] I.H. Witten, E. Frank, and M.A. Hall. Data Mining Practical Machine Learning Tools and Techniques. Burlington MA, USA: Elsevier, 2011, pp. 445-474.