2012 19th Working Conference on Reverse Engineering

SMURF: A SVM-based Incremental Anti-pattern Detection Approach

Abdou Maiga[1,3], Nasir Ali[1,2], Neelesh Bhattacharya[1,2], Aminata Sabané[1,2]
Yann-Gaël Guéhéneuc[1], and Esma Aimeur[3]
[1] Ptidej Team & [2] Soccer Lab., DGIGL, École Polytechnique de Montréal, Canada
[3] Heron Lab., DIRO, Université de Montréal, Canada
E-mails: maigaabd@iro.umontreal.ca, {nasir.ali,neelesh.bhattacharya,aminata.sabane}@polymtl.ca,
yann-gael.gueheneuc@polymtl.ca, aimeur@iro.umontreal.ca

***Abstract*—In current, typical software development projects, hundreds of developers work asynchronously in space and time and may introduce anti-patterns in their software systems because of time pressure, lack of understanding, communication, and–or skills. Anti-patterns impede development and maintenance activities by making the source code more difficult to understand. Detecting anti-patterns incrementally and on subsets of a system could reduce costs, effort, and resources by allowing practitioners to identify and take into account occurrences of anti-patterns as they find them during their development and maintenance activities. Researchers have proposed approaches to detect occurrences of anti-patterns but these approaches have currently four limitations: (1) they require extensive knowledge of anti-patterns, (2) they have limited precision and recall, (3) they are not incremental, and (4) they cannot be applied on subsets of systems. To overcome these limitations, we introduce SMURF, a novel approach to detect anti-patterns, based on a machine learning technique—support vector machines—and taking into account practitioners' feedback. Indeed, through an empirical study involving three systems and four anti-patterns, we showed that the accuracy of SMURF is greater than that of DETEX and BDTEX when detecting anti-patterns occurrences. We also showed that SMURF can be applied in both intra-system and inter-system configurations. Finally, we reported that SMURF accuracy improves when using practitioners' feedback.**

***Keywords*-Anti-pattern, program comprehension, program maintenance, empirical software engineering.**

I. Introduction

Developers continuously evolve software systems to implement and adapt to new customers' needs, as well as to fix bugs. Due to the time-to-market, lack of understanding, and the developers' experience, developers cannot always follow standard designing and coding techniques, *i.e.*, design patterns [1]. Design patterns are "good" solutions to recurring design problems, conceived to increase reuse, code quality, code readability, resilience to changes and above all, maintainability [2]. Consequently, anti-patterns creep up in software systems. Anti-patterns are "poor" solutions to recurring design and implementation problems [3]. They are generally the result of misuse of the object-oriented paradigm and–or design patterns [4].

Motivation: Researchers have performed empirical studies to show that anti-patterns like Spaghetti Code and Blob create hurdles during program comprehension [5], software evolution and maintenance activities [6]. The Spaghetti code anti-pattern is characteristic of procedural thinking in object-oriented programming [4]. Spaghetti code is related to classes without object-oriented structure. Therefore, such classes do not exploit object-oriented mechanisms, such as polymorphism and inheritance, and also prevent them from being used by developers [7]. Another example of anti-pattern is the Blob. A Blob is a large class that mostly controls the behaviour of a system or part thereof [4]. These two anti-patterns and many more yield to design and implementation of systems that are, at best, cumbersome and hinder evolution, in particular by impeding program comprehension [5]. It is important to detect anti-patterns in order to refactor/or remove them. This will improve software quality and reduce maintenance costs.

Limitations: Current anti-pattern detection approaches as proposed by Marinescu [8], Moha et al. [7] and Alikacem et al. [9], have four limitations: (1) they require extensive knowledge of anti-patterns, (2) they have limited precision and recall, (3) they are not incremental, and (4) they cannot be applied on subsets of systems. We claim that the first and second limitations, *i.e.*, high complexity and low accuracy, would be reduced by taking into account the practitioner's feedback. Indeed, practitioners using the textual description of anti-patterns can easily recognise them (and then provide feedback) but they will require more knowledge to define and set-up rules and thresholds, based on design or program metrics and properties to characterise anti-patterns. Also, the current approaches are not flexible and cannot take advantage of practitioners feedback, so their accuracy can be improved only by changing the approach itself. Using practitioners feedback is an easy and valuable way to improve the accuracy of anti-patterns detection.

Further, the third and fourth limitations are even more important, because they prevent a practitioner to guide the detection process and that today's software systems often weigh hundreds of millions of lines of code. With such large systems, a detection approach cannot be applied frequently

1095-1350/91 $25.00 © 4891 IEEE
DOI 10.1109/WCRE.2012.56

because of parsing and analysis times. Thus, it is important to detect anti-patterns at every stage of software development to reduce the maintenance costs and encourage practitioners in the anti-patterns detection. Indeed, if developers detect anti-patterns independently among few classes of a software (subset of the system) as it is developed, the detection will take less time. This will also permit the easy removal/or refactoring of the anti-patterns compared to, when they perform the detection on the whole system at the end of the development. Moreover, the improvement at this stage will facilitate the next steps in the development. Also, the number of detected occurrences when dealing with the whole system can be huge and discourage the practitioners to analyse and correct them. We argue that these limitations could be taken care of by using support vector machines (SVM).

Answer: SVM have been applied in various areas, *e.g.*, bioinformatics [10] and object recognition [11]. SVM is a recent alternative solution to the classification problem and relies on the existence of a linear classifier in an appropriate space by using a set of training data to train the parameters of the model. It is based on the use of functions called kernel, which allows an optimal separation of data in different categories. When apply to anti-patterns detection, we believe that SVM can yield better precision and recall values when compared to that of previous approaches, and can take into account practitioners' feedback. Also, SVM is by definition incremental because we can increase its training set incrementally. Finally, it can be applied on subsets of systems because it considers system classes one at a time, not collectively as previous rule-based approaches do. To the best of our knowledge, researchers have not yet studied the potential benefits of using SVM to detect anti-patterns.

In our previous study [12], we explored the potential usage of SVM in the context of anti-patterns detection. The preliminary results of our study were promising. However, the data was only trained and tested on the same system and we did not consider expert's feedback. Our conjecture, in this paper, is that SVM combined with expert's feedback can improve the accuracy (precision and recall) of anti-pattern detection as well as SVM and we can generalise the applicability of SVM-based approaches to multiple systems.

Contribution: The contribution of this paper is two-fold. First, we propose our approach SMURF to detect anti-patterns using SVM and practitioners' feedback. We exploit the benefits of SVM to detect the occurrences of anti-patterns while taking into account practitioners' feedback. We use the most studied anti-patterns, *i.e.*, Blob, Spaghetti Code, Functional Decomposition, and Swiss Army Knife, and perform more than 300 experiments to compare the results of DETEX [7] and BDTEX [13], the best two state of the art approaches, respectively, in exact and probabilistic anti-patterns detections, with the results of SMURF. We use both the measures of precision and recall to compare the approaches on a set of three programs namely ArgoUML v0.19.8, Azureus v2.3.0.6, and, Xerces v2.7.0. We showed that the accuracy of SMURF is greater than that of DETEX and BDTEX when detecting anti-pattern occurrences on a set of classes or on the whole system. We also showed that SMURF can be applied in both intra-system and inter-system configurations. Finally, we reported that SMURF accuracy improves when using practitioners' feedback. We thus conclude that our conjecture is correct: a SVM-based approach can overcome the four limitations of previous approaches.

Organisation: The paper is organised as follows. Section II provides a brief description of the state-of-the-art of anti-patterns detection approaches, machine learning approaches, and SVM. Section III describes our approach along with describing the SVM algorithm in some details. Section IV introduces our empirical study while Section V reports and discusses its results. Finally, Section VIII concludes the paper and outlines future work.

II. Related Work

We now recall major related work.

Smell/Anti-pattern Detection: Many researchers studied anti-patterns detection. Rahma et al. [14] used quality metrics to identify and predict anti-patterns in UML designs using structural and behavioral data. Ballis et al. [15], [16] worked on both the detection of design patterns and anti-patterns using a rule-based matching approach for extracting all occurrences of a design pattern/anti-pattern in a graph representation of the source code. Langelier et al. [17] proposed a visual approach to detect anti-patterns. Marinescu [8] presented detection strategies to detect and localise anti-patterns in systems. Marinescu defined 10 detection strategies to capture 10 important anti-patterns, but suffered from two drawbacks: (1) if a user may not successfully detect an anti-pattern without having extensive knowledge of metric-based rules and (2) different results would be obtained using different thresholds (whose definition itself is difficult). Dimitrios et al. [18] explored the ways in which the anti-pattern ontology can be enhanced using Bayesian networks in order to reinforce the existing ontology-based detection process. Their approach allows software developers to quantify the existence of an anti-patterns using Bayesian networks, based on probabilistic knowledge contained in the anti-pattern ontology regarding relationships of anti-patterns through their causes, symptoms and consequences. Sahraoui et al. [19] used search-based techniques to detect anti-patterns conjecturing that the more the code deviates from good practices, the more it is likely to be vulnerable to anti-patterns, without mentioning the values of recall obtained. Moha et al. [7] proposed an approach based on a set of rules (metrics, relations between classes) that describes the characters of each anti-pattern to identify them.

Khomh et al. [13] present BDTEX (Bayesian Detection Expert), a Goal Question Metric (GQM) based approach to build Bayesian Belief Networks (BBNs) from the definitions of anti-patterns. The output of the BBN is a probability that a class is an anti-pattern or not. For example, the probability of a class being a Blob depends directly on the two symptoms: MainClass and DataClass.

The approaches of Moha et al. [7], Marinescu [8], Rahma et al. [14] are mostly based on the use of code/design quality metrics and thresholds to identify anti-patterns. The use of these approaches to characterize new anti-patterns or to customise existing rules of detection for improving their performance, will require practitioners to have an extensive knowledge, at least about code/design quality metrics and software quality. The knowledge is necessary to be able to derive the rules from the textual description of anti-patterns. Another issue to those approaches is the choice of thresholds whose definition is difficult. In the same way, using the approach proposed by Ballis et al. [15], [16] requires also to learn the proposed language and to be able to describe structurally and semantically anti-patterns using that language. Thus all the above approaches are concerned with the first limitation. Khomh et al. [13] approach based on bayesian networks needs extensive knowledge to characterize the anti-patterns and built the bayesian network. However based on probability, it cannot tell to the user that a class is an anti-pattern for sure. Then we will face again the problem of threshold to decide if we consider a class as an occurrence of anti-pattern or not based on the probability. For this the user need an extensive knowledge. The approach of Moha et al. [7] cannot be used successfully on a set of classes because it needs the whole system to be able to automatically set up the thresholds via boxplots. This approach failed to answer to the third limitation. Moreover, any of the above approaches cannot benefit from practitioners' feedback and the only way to improve the performance of one of them is to modify the rules and/or the thresholds or of the pattern formal description. Thus, they are all concerned with the fourth limitation.

SVM: SVM has been used in several domains in the past for various applications, *e.g.*, bioinformatics [10] and object recognition [11]. It is a recent alternative to the classification problem. For examples, Takashi et al. [20] presented a document retrieval method using non-relevant documents. The approach used an active learning technique based on SVM for evaluating the relevant feedback. Guihong et al. [21] used C-SVM, a variant of SVM, for term classification and proved that expansion terms determined in traditional ways for pseudo-relevance feedback were not all useful. SVM has also been used in image retrieval systems when Sethia et al. [22] used invariant feature histograms to compare the efficiency of different SVMs. They claimed that a significant performance gain was obtained only after several feedback rounds. Kim et al. [23] proposed the change classification approach for predicting latent software bugs based on a SVM. Lucia et. al. [24] used incremental user feedback to enhance the rate of true positives found while analyzing the anomaly reports. The approach was validated using the anomaly reports of three real programs and the results showed clear increase in the number of true positives found in the anomaly reports.

No previous approaches used either SVM or users' feedback to detect anti-patterns.

User Feedback: As with SVM, users' feedback is a mechanism widely used in various domains with good results. In the field of image retrieval, Yihua et al. [25] used relevance feedback along with SVM to allow users to unblur images. Sethia et al. [22] also used both SVM and relevance feedback to improve image retrieval, exploiting the users' feedback on the initial results to achieve better results based on the feedback. Takashi et al. [20] proposed an approach for an interactive document retrieval using only non-relevant documents information. Cohn et al. [26] used a clustering algorithm to iteratively integrate feedback.

Similar to SVM, there has been no work on taking into account users feedback in anti-patterns detection.

III. SMURF: SVM USING FEEDBACK

SMURF is based on Support Vector Machines (SVM) to detect occurrences of anti-patterns. We provide some backgrounds on SVM before describing our approach in details.

A. Background

SVM is a set of techniques based on statistical theory of supervised learning introduced by Vapnik [27]. It relies on the existence of a linear classifier in an appropriate space and uses a set of training data to train the parameters of the classifier. SVM is based on the use of functions called kernel, which allows an optimal separation of data into two categories by a hyperplane.

Assuming some training data $x = x_1 \ldots x_n$ and their labels $y = y_1 \ldots y_n$ that are vectors in the space $\mathbb{R}^n$, which has a dot product, where $y_i \in \{-1, 1\}$, SVM builds a function f to assign for a given x_i, the label y_i. The function f uses a hyperplane to assign the labels by separating the hyperspace in two. There are many valid hyperplanes but SVM can compute the optimal hyperplane, which intuitively passes in the "middle"' of the two sets of data. Formally, the SVM finds the hyperplane for which the minimum distance to the training examples is maximal. The margin is the distance between the hyperplane and the closest example. These margins are called support vectors The optimal separating hyperplane is the one that maximises the margin. A hyperplane is defined as a set of points x satisfying $h(x) = w \cdot x + b = 0$ where $\cdot$ is the dot product; w

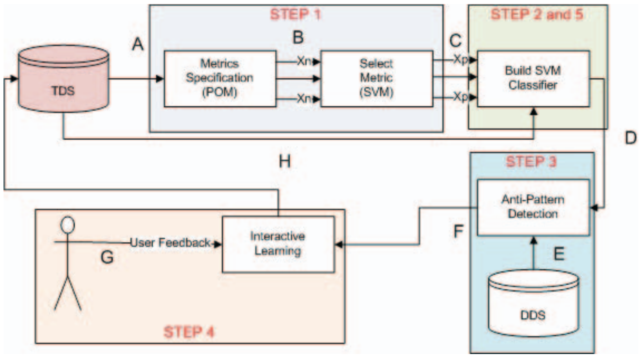

Figure 1. SMURF process overview

is the normal vector to the hyperplane; and the parameter $\frac{b}{||w||}$ is the perpendicular distance from the hyperplane to the origin. Therefore, there are two support vectors: x^{+}_{marge} for the support vector (positive border) near to positive class C_{+} and x^{-}_{marge} for the support vector (negative border) near to negative class C_{-}.

The computation of the maximum margin and the search procedure of separating hyperplane can often only be resolved linearly through separable discrimination problems. SVM suggests reconsidering the problem of the lack of linear separator in a higher dimensional space (possibly infinite dimensional).

The kernel function allows to perform the computations in the original space, which is much less expensive than high-dimensional dot product. Further, the transformation φ does not need to be known explicitly, only the kernel function is involved in the computations.

The kernel function may lead to complex transformations and even re-description of spaces of infinite dimension [28].

B. *SMURF Process*

Our approach to detect anti-patterns, SMURF, is based on a SVM using a polynomial kernel, and can take into account practitioners' feedback. We use SMURF to detect the well-known anti-patterns: Blob, Functional Decomposition, Spaghetti code, and Swiss Army Knife. For each anti-pattern detection, the detection process is identical. Figure 1 shows the overview of SMURF, which we illustrate with the Blob anti-pattern for the sake of clarity. We define:

- $TDS = \{C_i, i = 1, \dots, p\}$, a set of classes C_i derived from an object-oriented system that constitute the training dataset;
- $\forall i, C_i$ is labelled as Blob (B) or not (N);
- DDS is the set of the classes of a system in which we want to detect Blob occurrences.

To detect the Blob classes in the set DDS, we apply SMURF through the following steps:

Step 1 (Object Oriented Metric Specification): SMURF takes as input the training dataset TDS. For each class from TDS, we calculate object-oriented metrics that will be used as the attributes x_i for each class in TDS. We use POM[1] to compute metrics for all the studied systems. POM is an extensible framework, based on the PADL meta-model [29], which provides more than 60 metrics [30], including the well-known metrics by Chidamber and Kemerer.

Step 2 (Train the SVM Classifier): We train the SVM classifier using the dataset TDS and the set of metrics computed in Step 1. We define the training dataset as: $TDS = \{(x_i, y_i) | x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}, \forall i \in (1, \dots, n)\}$ where y_i is either 1 or -1, indicating respectively if a class x_i is a Blob occurrence or not. Each x_i is a p-dimensional real vector with p the number of metrics.

The objective of the training step is to find the maximum-margin hyperplane that divides the classes into the two different groups, Blob or Not-Blob.

Step 3 (Construction of the dataset *DDS* and detection of the occurrences of an anti-pattern): We build the dataset of the system on which we want to detect an anti-pattern as follows: for each class of the system, we compute the same set of metrics as in Step 1. We use the SVM classifier trained in Step 2 to detect the new occurrences of the anti-pattern in the dataset DDS.

Step 4 (Interactive learning and practitioners' feedback): After detecting occurrences of an anti-pattern using SMURF, the practitioner may label some of the classes reported as being anti-pattern according to whether they are indeed instances of this anti-pattern or not. We then give the set of labelled classes to the SVM as supplement to its training dataset to build a new optimal hyperplane that provides more accurate results in the subsequent detection. Practitioners can repeat Steps 2 to 4 as many times as desired.

IV. Empirical Study

The *goal* of our empirical study is, by comparing our approach, SMURF with DETEX [7] and BDTEX [13], to validate that SMURF can overcome the four limitations (mentioned before) of the previous approaches. The *quality focus* of our study is the accuracy of SMURF, in terms of precision and recall. The *perspective* is that of researchers and practitioners interested in verifying if SMURF can be effective in detecting various kinds of anti-patterns, in taking into account feedback, and in overcoming the previous limitations.

A. *Research Questions*

The goal of our empirical study is to evaluate the accuracy of SMURF and the impact of practitioners' feedback on the anti-patterns detection results. We also seek to compare SMURF with DETEX [7] and BDTEX[13], the best two state of the art approaches, respectively, in exact and

[1] http://wiki.ptidej.net/doku.php?id=pom

probabilistic anti-patterns detections. DETEX and SMURF are both exact anti-patterns detection approaches. Thus, we could perform a full comparison between the two approaches using the same set of programs used by Moha et al. [7] in their experiment. In the case of BDTEX, which is a probabilistic anti-patterns detection approach, we use the data provided by the authors to perform a comparison of BDTEX and SMURF.

Our experiment is divided in four steps.

First, we train SMURF on a set of known occurrences of one anti-pattern and non anti-pattern classes. Then, we apply SMURF on a set of classes of anti-patterns and non anti-patterns occurrences. The selection of anti-patterns and non anti-patterns classes is random. We also apply DETEX on the same set of classes (we reproduce this experiment four times: as many times as the number of anti-patterns). The first set of experiments allows us to show whether SMURF overcomes the first, second, and fourth limitations.

Second, we train SMURF on a set of known occurrences of one anti-pattern and then apply it on the classes of an entire subject system. We also apply DETEX on the same system. The second experiment allows us to compare the number of occurrences of Blob which SMURF detected, with that of DETEX in a realistic setting to verify whether SMURF overcomes the first, second, and fourth limitations.

Third, we train SMURF on a set of known occurrences of one anti-pattern from one system and then apply it on the classes of another system. We then add feedback to SMURF in the form of additional known occurrences of the anti-pattern. The last set of experiments allows us to compare the accuracy of SMURF with/without feedback and the impact of practitioners' feedback on its accuracy to verify whether SMURF overcomes the first, second, and third limitations.

Finally, we train and apply SMURF on the same data that was used by BDTEX for training and testing [2] . We then compare the results of the two approaches.

- RQ1: How does the accuracy of SMURF compare with that of DETEX, in terms of precision and recall? We decompose RQ1 as follows:
  - RQ1_1: How does the accuracy of SVMDetect compare with that of DETEX, in terms of precision and recall, when applied on a same subset of a system?
  - RQ1_2: How many occurrences of Blob SVMDetect can detect when comparing with that of DETEX on a same entire system?
- RQ2: How does the accuracy of SMURF compare with that of BDTEX, in terms of precision and recall when applied on a same entire system?
- RQ3: How does the accuracy of SMURF change when trained/applied on the same system and trained/applied on different systems, in terms of precision and recall?
- RQ4: How does the accuracy of SMURF, with relevance feedback, compare with that of SMURF without feedback, in terms of precision and recall?

B. Objects

The objects[3] of our study are ArgoUML v0.19.8 (1,230 classes), Azureus v2.3.0.6 (1,449 classes), and Xerces v2.7.0 (513 classes), three open-source Java systems. We chose these systems due to several factors. First, we selected open-source systems that are freely available so that other researchers can replicate our study. Second, we selected systems that have been used by other researchers to allow comparisons [7]. Moha et al. [7] used these three systems, thus we use these three systems for the comparison between DETEX and SMURF. Khomh et al [13] used only one the three systems (Xerces v2.7.0) and thus, we compare BDTEX and SMURF using that system.

C. Subjects

The subjects of our study are the following four anti-patterns: Blob, Functional Decomposition (FD), Spaghetti Code (SC), and Swiss Army Knife (SAK). We chose these four anti-patterns because they are well known and commonly studied anti-patterns and also for the comparison purpose with Moha et al. [7] and Khomh et al. [13]. Indeed, Moha et al. [7] used these four anti-patterns and Khomh et al. [13] three of them.

D. Data Collection and Oracles

Moha et al. [7] built four oracles of four anti-patterns, for three systems based on the results of DETEX. For each system and each anti-pattern, the detected classes have been checked by independent engineers to assess whether they are true or false positive. The engineers manually validated the detected classes depending on the anti-pattern definition and their context.

We reuse these oracles to assess both the precision and recall of DETEX and SMURF.

For the comparison of BDTEX and SMURF, we used the oracles provided by Khomh et al. [13] [4]. They manually checked each class of the programs to assess whether it is an occurrence of anti-pattern or no.

We use Weka[5] to implement SMURF, using its SVM classifier. In all the experiments, we train SMURF on training datasets TDS and apply it on detection datasets DDS.

E. Analysis Methods

For the purpose of the comparison of DETEX with SMURF, we build for each system and each anti-pattern,

[2] http://www.ptidej.net/downloads/experiments/jss10/

[3] argouml.tigris.org/, azureus.sourceforge.net/, and xerces.apache.org/xerces-c/

[4]

[5] http://www.cs.waikato.ac.nz/ml/weka/

three datasets that are composed of two parts: anti-patterns and non-anti-pattern classes in equal numbers. For example, if we consider Blob and ArgoUML, we build three datasets DDS_1, DDS_2, and DDS_3.

To build each dataset we use 30 Blob classes in ArgoUML (identified with the oracles) and add 30 non-Blob classes by choosing them randomly in the remaining classes of ArgoUML. We then divide the 60 classes randomly into the three datasets DDS_1, DDS_2, and DDS_3, making sure that each dataset contains 10 Blob classes and 10 non-Blob classes. To answer our research questions, we perform a 10-fold cross validation.

For the comparison of SMURF with BDTEX, we use the same trained and test data used by Khomh et al. [13].

1) RQ1: To answer $RQ1_1$ (respectively $RQ1_2$), we train SMURF on a dataset DDS_1 and detect occurrences of an anti-pattern on DDS_2 (respectively on the rest of the whole system). We compute the precision and recall of SMURF on DDS_2 (respectively we compute the number recovered occurrences of BLOB on the rest of the whole system). We then run DETEX on the same dataset, DDS_2 (respectively on the rest of the whole system), and compute its precision and recall.

2) RQ2: To answer RQ2, we train SMURF and detect occurrences of an anti-pattern on respectively the same trained dataset and test dataset used by Khomh et al. [13] for the system Xerces for the three anti-patterns they studied. We compute the precision and recall of SMURF on that test dataset. We then compute the precision and recall of BDTEX at different threshold levels according to its results on that test dataset.

3) RQ3: To answer RQ3, we train SMURF on a dataset DDS_1 from one system and detect occurrences of an anti-pattern on DDS_2, from either the same system or another system. We compute the precision and recall of SMURF applied to the DDS_2 of the same system and also that of SMURF applied to the DDS_2 of another system. We then run DETEX on the same dataset DDS_2 of the same/other system, and compute its precision and recall.

4) RQ4: To answer RQ4, we first train SMURF on DDS_1 and apply it on DDS_2 and compute, as for the other RQs, its precision and recall. We then simulate the practitioners providing their feedback by adding to the trained dataset DDS_1 of SMURF, different percentage (25%, 75% and, 100%) of the content of DDS_3. Indeed, DDS_3 contains a set of labelled anti-patterns and non anti-patterns and then can constitute the set that practitioners would incrementally constitute when validating the detected occurrences as true or false occurrences. For each level of added feedback, we train SMURF on the new trained dataset and re-apply it on DDS_2 and again compute its precision and recall.

Table I
PRECISION OF SMURF VS. DETEX IN SUBSETS WITHOUT FEEDBACK

		ArgoUML (%)	Azureus (%)	Xerces (%)
Blob	DETEX	0.00	0.00	0.00
	SMURF	97.09	97.32	95.51
FD	DETEX	0.00	0.00	0.00
	SMURF	70.68	72.01	66.93
SC	DETEX	0.00	0.00	0.00
	SMURF	85.00	88.00	86.00
SAK	DETEX	10.00	10.00	0.00
	SMURF	75.46	84.54	80.76

Table II
RECALL OF SMURF VS. DETEX IN SUBSETS WITHOUT FEEDBACK

		ArgoUML (%)	Azureus (%)	Xerces (%)
Blob	DETEX	0.00	0.00	0.00
	SMURF	84.09	91.33	95.29
FD	DETEX	0.00	0.00	0.00
	SMURF	57.50	84.28	70.00
SC	DETEX	0.00	0.00	0.00
	SMURF	71.00	89.00	86.00
SAK	DETEX	0.00	0.00	0.00
	SMURF	77.14	85.71	75.50

V. RESULTS

This section reports the results of our empirical study. These results are discussed in Section VI. The data for replication purpose and the other results are available online[6].

A. RQ1

$RQ1_1$: Tables I and II report the precision and recall values when applying DETEX and SMURF on subsets (DDS_2 dataset). The results for DETEX are mostly zeroes because DETEX cannot detect occurrences of anti-patterns without the entire system being present, as it uses boxplots and thresholds to identify anti-patterns. In the contrary, SMURF has acceptable precision and recall values.

$RQ1_2$: Table V shows the total number of anti-patterns' occurrences of Blob detected by DETEX and SMURF. It can be clearly seen that SMURF detects more number of occurences than DETEX for all the three entire systems.

Thus, we answer RQ1 as follows: on subsets of systems, SMURF dramatically outperforms DETEX, while on entire systems, SMURF detects more occurrences of Blob than DETEX .

B. RQ2

Figures 2 and 3 show the accuracy of SMURF and BDTEX when detecting respectively occurrences of blob and spaguetti code on Xerces system. It shows that SMURF perform better than BDTEX and is more stable. Due to the lack of space, we present only the results for the anti-patterns Blob and Spaguetti Code.

Thus, we answer RQ2 as follows: SMURF has a better precision and recall than BDTEX.

[6] http://www.ptidej.net/download/experiments/wcre12a/

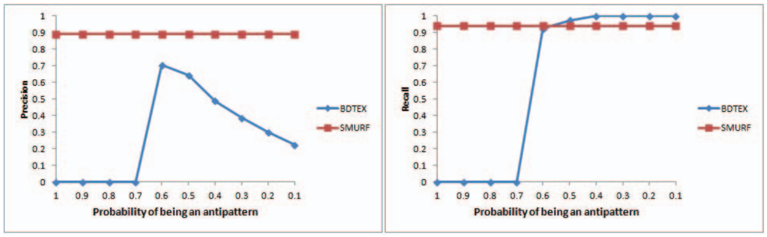

Figure 2. Trends in the increase of precision and recall when decreasing the probabilty of being an antipattern for Blob and Xerces

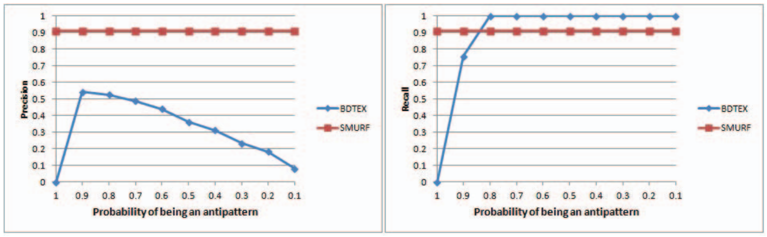

Figure 3. Trends in the increase of precision and recall when decreasing the probabilty of being an antipattern for Spaguetti Code and Xerces

Table III
PRECISION OF SMURF IN INTER-SYSTEMS WITHOUT FEEDBACK

	ArgoUML (%)	Azureus (%)	Xerces (%)
Blob	92.00	96.00	89.00
FD	57.00	62.00	36.00
SC	77.00	74.00	91.00
SAK	56.00	73.00	90.00

Table IV
RECALL OF SMURF IN INTER-SYSTEMS WITHOUT FEEDBACK

	ArgoUML (%)	Azureus (%)	Xerces (%)
Blob	62.00	48.00	94.00
FD	40.00	100.00	20.00
SC	96.00	88.00	91.00
SAK	68.00	84.00	56.00

C. RQ3

Tables III and IV report the values of precision and recall in the inter-system configuration in which SMURF is trained using the classes of one system (chosen randomly) and applied on the subsets of classes of another system. In most cases, SMURF has quite acceptable values for precision and recall, for inter-system configurations.

Thus, we answer RQ3 as follows: SMURF has a better precision and recall than DETEX. Even in the inter-system configuration, its precision and recall are acceptable in the most of cases excepted for the functional decomposition in the programs ArgoUML (the recall is 40%) and Xerces (the precision is 36% and the recall 20%).

D. RQ4

Table V
TOTAL RECOVERED OCCURRENCES OF BLOB BY DETEX AND SMURF

	DETEX	SMURF
ArgoUML	25	40
Azureus	38	48
Xerces	39	55
Total	**102**	**143**

Figure 4 shows the changes in precision and recall values. We observe that the more feedback, the better the precision, up to 100%. For recall, the more feedback, the better the recall but with a slight decrease when we use 100% feedback.

Thus, we answer RQ4 as follows: both precision and recall values increase when taking into account practitioners' feedback.

VI. DISCUSSION

We now provide detailed discussion on SMURF, results, feedback, and some of our observations.

A. SMURF vs. DETEX

Subsets of Systems: When applied on subsets of systems, we observe that DETEX could not detect occurrences of some anti-patterns and, when it did, the precision and recall values were quite low, mostly 0. We explain this observation by the use of boxplots and thresholds by DETEX . When DETEX analyses a few classes, its use of boxplots and thresholds yield most of the classes to fall under (respectively above) the thresholds and hence, not to be reported. The problem does not arise when analysing an entire system because then the boxplots quartiles are different and more classes fall within the threshold values set in the rules. SMURF can work on the whole system and as well as on

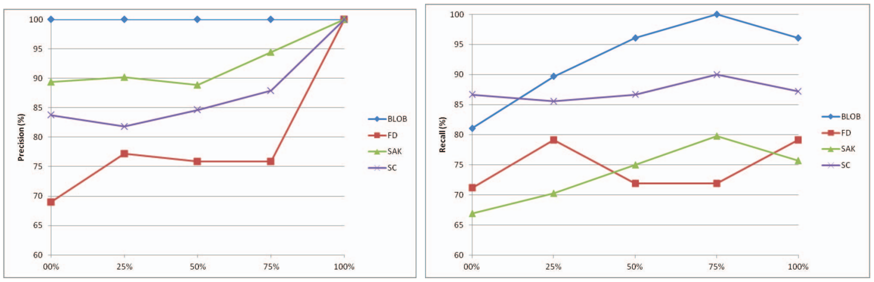

Figure 4. Trends in the increase of precision and recall when integrating incremental feedback

part of a system. In most of cases, for all configurations (intra or inter-system), SMURF provides acceptable recall and precision values. However we observe that for functional decomposition in the inter-system configuration, the recall is low for ArgoUML(40%) and Xerces (20%) and the precision is low for Xerces (36%). We can explain these results by the fact that, we did not have enough detected occurrences of this anti-patterns.

Complete System: When applied on the whole system for detecting a particular anti-pattern Blob, Table V shows that SMURF, on an average, performed better than DETEX. DETEX could detect 102 Blob occurrences whereas SMURF could detect 143 Blob occurrences, in spite of DETEX being able to perform at its best when detecting Blob. Further, we applied SMURF using a trained dataset of one system A for detecting an anti-pattern of another system B. For example, trained dataset of ArgoUML was used to detect anti-patterns of the whole system Xerces. The results obtained were significantly better than DETEX and the results can be generalised for any system. We could not perform experiment on other anti-patterns detection due to the lack of manually validity oracle. However, looking at the nature of Blob detection, we believe that SMURF would even perform better when detecting other anti-patterns.

B. *SMURF vs. BDTEX*

BDTEX is a probabilistic approach which provides probabilities that a class is an occurrence of anti-pattern. However our SMURF approach is an exact detection approach which is a boolean detection approach telling that a class is an occurrence of an anti-pattern or not. Because the user is not necessarily an expert in the field of anti-patterns, she will refer to a threshold to make her decision to consider a class as an anti-pattern or not. And to compare the precisions and recalls of BDTEX versus SMURF, we consider several decisions thresholds based on probability. We consider that the decision is made to consider a class as an instance of anti-pattern when the probability is 1, then we compute the precision and recall. Then we consider a probability threshold of 0.9 and we compute again the precision and recall. And so on by decreasing probability. The results of this comparison are shown with the figures 2 and 3 for Blob and Spaguetti Code on Xerces system. The results show that BDTEX contains a high level of uncertainty. For Example, for spaghetti code, while the precision of SMURF is 90%, from a probability threshold of 0.8, the accuracy begins to drop below 50% quickly and significantly. For the Blob while SMURF is 90% precision BDTEX remains at 0% precision even if the probability constraint relaxes up to 0.7. Its from 0.6 probability that the accuracy of BDTEX goes up without reaching the level of that of SMURF and decreases immediately after. The recall is almost zero when at the beginning the requirement is high, but when we decided to release the constraint, it rises to the level of SMURF or even exceed, which is quite normal since in this case BDTEX accepts most classes. Thus, at this level, BDTEX has high recall but bad precision.

C. *Other Anti-patterns/Systems*

SMURF requires labelled data to train its SVM. Having this labelled data is a limitation of our approach. However, we claim that it is easier for practitioners to label classes as being occurrences of some anti-patterns than to write rules and choose right threshold values. Indeed, it is easier for practitioners to recognise an anti-pattern when seeing one rather than to define the rules to detect it [4]. This observation relates to the concept of “quality without a name” by Alexander [31], who suggested that it is easy to assess the quality of something when seeing it but difficult to define this very same quality ex nihilo. Thus, we claim that practitioners can easily and incrementally build their own oracles to train SMURF. SMURF would fit in their context and, thanks to the feedback loop in SMURF, practitioners could keep on improving its results as time goes, by adding/removing more labelled data as they find new occurrences of some anti-patterns and decide that an existing occurrence should not be reported any longer or not.

D. *Practitioners’ Feedback*

SMURF allows practitioners to provide their feedback. The feedback is an easy and valuable way to improve the approach accuracy as confirmed by the results in figure 4. In most of the cases, when we increase the feedback provided to SMURF, the recall and the precision increase. However, we observe that when using 100% of the feedback, the recall of SMURF decreases slightly when compared to 75% of the feedback (but is still higher than when not using any feedback). We explain this observation by the fact that our oracles, obtained from a previous work [7], may not be completely accurate and, thus, integrating their data could lead to misclassification of some true positive occurrences. Future work includes revising entirely the oracles to identify misclassified occurrences, if any.

E. SMURF and Practitioners

Previous approaches depended heavily on rules and thresholds. To define rules, a practitioner must have a detailed technical knowledge of anti-patterns and the underlying framework. There is no general rule to define what an anti-pattern is. Thus, it is difficult for a practitioner to write a rule. For example, it is quite possible for a practitioner to state that a Blob class must contain 300 LOC and for another practitioner this could be $30,000$ LOC. Defining a wrong threshold and–or rule would negatively impact the results of the approach and would make it useless for the practitioners. SMURF overcomes both of these limitations, *i.e.*, use of anti-pattern rules and thresholds. SMURF can detect occurrences of anti-patterns in object-oriented systems without having to manually set rules for detection. Also, SMURF has a good precision and recall, even in the inter-system configuration. Finally, SMURF provide better results to practitioners as they integrate their feedback.

VII. Threats to Validity

We now discuss threats to the validity of our results [32].

Internal Validity:

Concerning the potential dependence of the obtained results on the chosen anti-patterns and systems, our study is not affected. Indeed, we used four well-known and representative anti-patterns. These occurrences of these anti-patterns have been manually validated by independent engineers. These anti-patterns also have been used in previous works. Further, we applied our approach to three open-source systems with different size and these systems have also been used by previous researchers.

Reliability Validity: Reliability validity threats concern the possibility of replicating the study concerned. To mitigate this threat, we used open-source systems that can be freely downloaded from the Internet. We attempted to provide all the necessary details to replicate our study. Moreover, the results of the validation and the datasets are available online.

External Validity: Threats to external validity concern the possibility to generalise our results. We studied three systems with different sizes and different domains. In spit of the fact that three systems might not be a very large number that can be generalized, as the systems were of varying sizes, and domains, we can claim that the results can be generalized with certain degree of confidence. Further, we also used a representative subset of anti-patterns. However, we will apply SMURF on other systems and anti-patterns in future work to negate the threat completely.

VIII. Conclusion and Future Work

Anti-patterns are a fact of developers' life when developing software systems under the conditions prevailing nowadays: distribution in time and space, time pressure, complexity. In particular, anti-patterns impede program comprehension [5] and thus have negative impact on both development and maintenance activities. We observed that current anti-pattern detection approaches had four limitations: (1) they require extensive knowledge of anti-patterns, (2) they have limited precision and recall, (3) they are not incremental, and (4) they cannot be applied on subsets of systems.

To overcome these limitations, we introduced a novel approach to detect anti-patterns, SMURF, based on support vector machines (SVM). SVM allows (1) learning the detection rules from a set of known occurrences of anti-patterns and non anti-patterns, and (2) improving accuracy. Thus, SMURF, overcomes the previous limitations. Practitioners can easily produce the needed set according to their needs and context.

We designed an empirical study that allowed us to compare the results of DETEX [7] and BDTEX [13], the best two state of the art approaches, respectively, in exact and probabilistic anti-patterns detections, with the results of SMURF. We performed more than 300 experiments to show how SMURF performs on a set of three programs (ArgoUML v0.19.8, Azureus v2.3.0.6, and Xerces v2.7.0) using four anti-patterns (Blob, Functional Decomposition, Spaghetti Code, and Swiss Army Knife).

We showed that the accuracy of SMURF is greater than that of DETEX and BDTEX when detecting anti-patterns on a set of classes or on the whole system. We also showed that SMURF can be applied in both intra-system and inter-system configurations. Finally, we reported that SMURF accuracy improves when using practitioners' feedback.

> We thus conclude that our conjecture is correct: SVM-based approach can overcome the four limitations of previous approaches and could be more readily adopted by practitioners.

Future work includes performing an empirical study about the use of SMURF in real-world environments. We will ask our industrial partners help in realising such a study. Further, we would also reproduce the study with other systems and other anti-patterns to increase our confidence in the generalisability of our conclusions. Another study could be the evaluation of the impact of the quality of feedback on SMURF results.

References

[1] M. Fowler, *Refactoring – Improving the Design of Existing Code*, 1^{st} ed. Addison-Wesley, June 1999.

[2] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[3] B. F. Webster, *Pitfalls of Object Oriented Development*, 1^{st} ed. M & T Books, February 1995.

[4] W. J. Brown, R. C. Malveau, W. H. Brown, H. W. McCormick III, and T. J. Mowbray, *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*, 1^{st} ed. John Wiley and Sons, March 1998.

[5] M. Abbes, F. Khomh, Y.-G. Guéhéneuc, and G. Antoniol, "An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension," in *CSMR, 15th European Conference on Software Maintenance and Reengineering*, T. Mens, Y. Kanellopoulos, and A. Winter, Eds. IEEE Computer Society, 2011, pp. 181–190.

[6] F. Khomh, M. D. Penta, and Y.-G. Guéhéneuc, "An exploratory study of the impact of antipatterns on class change- and fault-proneness," *Journal of Empirical Software Engineering (EMSE)*, 2011.

[7] Naouel Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. L. Meur, "DECOR: A method for the specification and detection of code and design smells," *Transactions on Software Engineering (TSE)*, 2009.

[8] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in *In Proceedings of the IEEE 20th International Conference on Software Maintenance*. IEEE Computer Society Press, 2004, pp. 350–359.

[9] E. H. Alikacem and H. A. Sahraoui, "Détection d'anomalies utilisant un langage de règle de qualité," in *LMO*. Hermes Science Publications, 2006, pp. 185–200.

[10] J. Bedo, C. Sanderson, and A. Kowalczyk, "An efficient alternative to svm based recursive feature elimination with applications in natural language processing and bioinformatics," in *AI 2006: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, A. Sattar and B.-h. Kang, Eds. Springer Berlin Heidelberg, 2006, vol. 4304, pp. 170–180.

[11] M. J. Choi, A. Torralba, and A. S. Willsky, "A tree-based context model for object recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, pp. 240–252, Feb. 2012.

[12] M. Abdou, A. Nasir, B. Neelesh, S. Aminata, G. Yann-Gaël, A. Giuliano, and A. Esma, "Support vector machines for anti-pattern detection," in *ASE*, 2012.

[13] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui, "Bdtex: A gqm-based bayesian approach for the detection of antipatterns," *J. Syst. Softw.*, vol. 84, no. 4, pp. 559–572, Apr. 2011.

[14] N. B. Rahma Fourati and H. B. Abdallah, "A metric-based approach for anti-pattern detection in uml designs," in *COMPUTER AND INFORMATION SCIENCE 2011*, ser. CISW '07. SpringerLink, 2011, pp. 287–290.

[15] D. Ballis, A. Baruzzo, and M. Comini, "A rule-based method to match software patterns against uml models," *Electron. Notes Theor. Comput. Sci.*, vol. 219, pp. 51–66, November 2008.

[16] ——, "A minimalist visual notation for design patterns and antipatterns," in *Proceedings of the Fifth International Conference on Information Technology: New Generations*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 51–56.

[17] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ser. ASE '05. New York, NY, USA: ACM, 2005, pp. 214–223.

[18] D. Settas, A. Cerone, and S. Fenz, "Enhancing ontology-based antipattern detection using bayesian networks," *Expert Syst. Appl.*, vol. 39, no. 10, pp. 9041–9053, Aug. 2012.

[19] M. Kessentini, S. Vaucher, and H. Sahraoui, "Deviance from perfection is a better criterion than closeness to evil when identifying risky code," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 113–122.

[20] T. Onoda, H. Murata, and S. Yamada, "Non-relevance feedback document retrieval based on one class svm and svdd," in *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2006, part of the IEEE World Congress on Computational Intelligence, WCCI 2006, Vancouver, BC, Canada, 16-21 July 2006*, 2006, pp. 1212–1219.

[21] G. Cao, J.-Y. Nie, J. Gao, and S. Robertson, "Selecting good expansion terms for pseudo-relevance feedback," in *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*. ACM, 2008, pp. 243–250.

[22] L. Setia, J. Ick, and H. Burkhardt, "Svm-based relevance feedback in image retrieval using invariant feature histograms," in *In Proc. of the IAPR Workshop on Machine Vision Applications*, 2005, pp. 542–545.

[23] S. Kim, E. J. W. Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Trans. Software Eng.*, vol. 34, no. 2, pp. 181–196, 2008.

[24] Lucia, D. Lo, L. Jiang, and A. Budi, "Active refinement of clone anomaly reports," in *Proceedings of the 2012 International Conference on Software Engineering*, ser. ICSE 2012. Piscataway, NJ, USA: IEEE Press, 2012, pp. 397–407.

[25] Y. Zhou, W. Shi, L. Duan, and C. Niu, "A relevance feedback algorithm based on svm model's dynamic adjusting for image retrieval," in *Proceedings of the 2007 International Conference on Computational Intelligence and Security Workshops*, ser. CISW '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 287–290.

[26] S. Basu, I. Davidson, and K. Wagstaff, *Constrained Clustering: Advances in Algorithms, Theory, and Applications*, 1st ed. Chapman & Hall/CRC, 2008.

[27] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[28] N. C. John Shawe-Taylor, "Support vector machines and other kernel-based learning methods," *Cambridge University Press*, 2000.

[29] Y.-G. Guéhéneuc and G. Antoniol, "DeMIMA: A multi-layered framework for design pattern identification," *Transactions on Software Engineering (TSE)*, vol. 34, no. 5, pp. 667–684, September 2008, 18 pages.

[30] Y.-G. Guéhéneuc, H. Sahraoui, and Farouk Zaidi, "Fingerprinting design patterns," in *Proceedings of the* 11^{th} *Working Conference on Reverse Engineering (WCRE)*, E. Stroulia and A. de Lucia, Eds. IEEE Computer Society Press, November 2004, pp. 172–181, 10 pages.

[31] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, *A Pattern Language*, 1^{st} ed. Oxford University Press, August 1978.

[32] R. K. Yin, *Case Study Research: Design and Methods (Applied Social Research Methods)*. Sage Publications, Inc, 2008.