International Journal of Combined Research & Development (IJCRD)
eISSN:2321-225X;pISSN:2321-2241 Volume: 3; Issue: 2; August - 2014

A Machine Learning Approach for Anti-pattern Detection

Meenakshi T.

Assistant Professor
Department of Computer Science
Government First Grade College
Vijayanagar, Bangalore,India

Abstract: Developers may induce anti-patterns in their software frameworks due to time limit, lack of understanding, communication, and so on. Anti-patterns obstruct advancement and maintenance activities by making the source code more difficult to understand. Recognizing anti-patterns in an entire software framework may be infeasible due to the obliged parsing time and of the consequent required manual acceptance. Recognizing anti-patterns on subsets of a framework could diminish costs, effort, and resources. Scientists have proposed methodologies to identify events of anti-patterns in any case these methodologies have at present a few confinements: they require extensive knowledge of anti-patterns, they have constrained precision and recall, and they cannot be employed on subsets of frameworks. To overcome these challenges a machine learning based approach has been proposed to identify anti-patterns. Indeed, through experimental study, it is showed that the accuracy of proposed technique is greater than other existing approaches in detecting anti-patterns.

Keywords: Anti-patterns, precision, recall, machine-learning approach.

I. Introduction

Software testability is a noteworthy property of software quality that encourages testing activities and decreases testing cost and effort. The significance of software quality lies in the intricacy and notoriety of the software frameworks. Anti-patterns contrarily influence the testability and subsequently quality of the object oriented frameworks. Anti-pattern classes oblige much more prominent testability effort than non anti-pattern classes. Their initial recognition and revision is important to comfort the improvement and support process.

Anti-patterns are poor solutions to repeating configuration and execution issues. Researchers have performed empirical studies to demonstrate that anti-patterns make obstacles during project perception, software development and maintenance activities [1]. It is paramount to discover anti-patterns at an early phase of software development, to diminish the support costs.

Current anti-design identification approaches as proposed by Marinescu [2], Moha et al. [2] and Alikacem et al. [3] have a few constraints, for example, they require broad knowledge of anti-patterns, they have restricted accuracy and recall and cannot be utilized on subsets of frameworks. These limitations are overcome by utilizing support vector machine.

Support vector machines (SVM) have been employed in various fields, e.g., bioinformatics [42], data recovery [5]. It is another solution for the classification problems. We can apply SVM on subsets of frameworks in light of the fact that it considers framework classes each one in turn, not collectively as rule based methodologies do. In this paper, an SVM based machine learning approach is used to detect anti-pattern.

The remainder of the paper is as follows: Section II presents related work on detecting anti-patterns. In section III the proposed work is presented. Section IV presents the research question related to the current work. Section V Presents the results of the work and finally section VI concludes the paper.

II. Related Work

Several research works have been carried out to detect anti-patterns in the software design. Moha et al. [6] presented a DSL focused around set of rules (measurements, connection between classes) that portrays the character of every anti-pattern. They characterized a platform for automatic change of rule cards into identification algorithms furthermore proposed a few algorithms [7].

Jug et al. [8] extricate a formal data models from the ORM determination of framework, and create heuristics to find anti-patterns in the data model. Their system can then naturally propose solutions to redress the data models. Maiga et.al [9] additionally proposed SMURF. This methodology additionally utilizes a machine learning method (SVM) utilizing polynomial kernel as a premise for identification; however it considers the specialists' input. SMURF is intended to work on both intra and inter framework designs.

Sahraoui et al. [10] utilized search based methods to recognize anti-patterns deducing that the more the code digresses from great practices, the more it is prone to be susceptible against anti-patterns. Cortellessa et al [11] examine different methodologies of catching design level execution anti-patterns utilizing software modeling and outline change rules.

Tamayo et al [12] develop the project dependency graph utilizing element data, and join the data with the relating database operations to distinguish execution bottlenecks. Their system might additionally be utilized to recognize issues identified with grouping, SQL synchronization, redundant SQL queries, and additional operations.

III. SVM Based Detection Approach

SVM based approach uses a polynomial kernel for detecting the anti-pattern occurrences. It is based on linear classifier and utilizes training data in order to train the classifier. The SVM is utilized to detect familiar anti-patterns like functional decomposition, blob and spaghetti code. The detection process is illustrated as follows:

- Let DS={ C_i } where i=1,2,3,…n is the set of classes of object oriented system that comprises the training data set.
- C_i, is marked as spaghetti (S) or not.
- SC is the set of classes where we have to identify the spaghetti code.

In order to identify the spaghetti code in SC, the following steps are followed:

Specifying Object Oriented Metrics:

The SVM takes the training dataset DS as input. For each and every class in DS, the object-oriented metrics is estimated which will be utilized as an attribute y_i in every class present in the DS. Here, POM is used to calculate the metrics.

Training the SVM:

The SVM is trained using the data set DS and the metrics that are calculated. The training set is defined as follows: $DS = a_i, b_i \quad a_i \in R^n, b_i \in \; -1,1 \;, \forall i \in (1, \ldots m)$ where b_i is either -1 or 1 representing whether the class is spaghetti or not.

The intent of the training phase is to discover a best hyper plane that separates the classes into two groups, spaghetti or not-spaghetti.

Constructing the data set and identifying the anti-pattern occurrence:

The data set is built in such a way that it identifies the anti-pattern as follows: For every class in the system, the metrics are calculated and the trained SVM classifier is utilized to identify the occurrence of the anti-patterns contained in the data set.

IV. Research Questions

1. How much is the accuracy of SVM based approach, in terms of precision and recall?
2. How many occurrences of anti-pattern does the approach detects?

V. Results and Discussion

In this section, the results of empirical study are presented. Table 1 shows the total number of spaghetti code occurrences detected by our approach. The SVM is applied utilizing the trained data set of one system such as X and detects the anti-pattern on other system like Y.

System	**Total Occurrences**
Azureus	47
Xerces	53
Total	100

Table 1 Total number of spaghetti occurrences detected by proposed approach

When applied on the whole system for spaghetti code detection, the proposed approach detected 100 occurrences of spaghetti code. The results obtained shows that the approach is considerably better than other anti-pattern detection techniques. Table 2 and 3 shows the precision and recall for the proposed technique.

	Azureus	Xerces
SC	83	85
FD	70	69
SAK	73	70

Table 3 : Precision Value

	Azureus	Xerces
SC	80	78
FD	71	68
SAK	70	69

Table 4 : Recall Value

Threats to Validity

Internal Validity:

Threats to inside legitimacy concern the reliance of the acquired results that rely on upon the chosen anti-patterns and frameworks. These dangers don't affect our study in light of the fact that we utilized well-known and delegate anti-patterns. These anti-patterns likewise have been utilized within past works. We additionally utilized two open-source frameworks with different sizes, which have been utilized by past scientists**.**

Reliability Validity:

Reliability legitimacy dangers concern the likelihood of duplicating the study concerned. To mitigate this danger, we utilized two open-source frameworks that might be uninhibitedly downloaded from the Internet. We endeavored to give all the essential points of interest to recreate our study**.**

VI. Conclusion

Anti-patterns are a certainty of designers' life when creating software frameworks, under the conditions predominating these days: dissemination in time and space, time pressure and intricacy. Anti-patterns specifically block program cognizance and subsequently have negative effect on both advancement furthermore maintenance exercises. It is observed, that current anti-patterns detection methodologies have a few confinements: they require broad knowledge of anti-patterns, they have constrained accuracy and recall, and they can't be used on subsets of frameworks. To overcome these restrictions, a novel methodology to locate anti-patterns, taking into account support vector machines (SVM), a machine learning based approach is proposed. The proposed approach performs better than other approaches in detecting anti-patterns.

VII. References

1. F. Khomh, M. D. Penta, and Y.-G. Gu_eh_eneuc. An exploratory study of the impact of antipatterns on class change-and fault-proneness. Journal of Empirical Software Engineering (EMSE), 2011.

2. R. Marinescu. Detection strategies: Metrics-based rules for detecting design aws. In In Proceedings of the IEEE 20th International Conference on Software Maintenance, pages 350{359. IEEE Computer Society Press, 2004.

3. Naouel Moha, Y.-G. Gu_eh_eneuc, L. Duchien, and A.-F. L. Meur. DECOR: A method for the speci_cation and detection of code

and design smells. Transactions on Software Engineering (TSE), 2009.

4. E. H. Alikacem and H. A. Sahraoui. Detection d'anomalies utilisant un langage de regle de qualit_e. In LMO, pages 185-200. Hermes Science Publications, 2006.
5. 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 A. Sattar and B.-h. Kang, editors, AI 2006: Advances in Artificial Intelligence, volume 4304 of Lecture Notes in Computer Science, pages 170-180. Springer Berlin Heidelberg,2006.

6. Z. Ye, J. X. Huang, and H. Lin. Incorporating rich features to boost information retrieval performance: A SVM-regression based re-ranking approach. Expert Syst. Appl., 38:7569-7574,June 2011.

7. Moha, N., Guéhéneuc, Y. -G., Duchien, L., Meur, A. -F. L. 2010. DECOR: a method for the specification and detection of code and design smells. IEEE Transactions on Software Engineering(2010a), vol. 36, no.1, pp. 20–36.

8. Moha, N., Guéhéneuc, Y. -G., Meur, A. -F. L., Duchien, L., Tiberghien, A. 2010. From a domain analysis to the specification and detection of code and design smells. Formal Aspects of Computing (FAC), vol. 22, no. 3-4, 2010b, pp. 345-361.

9. Jaideep Nijjar and Tevfik Bultan. Data model property inference and repair. In Proceedings of the 2013 International Symposium on Software Testing and Analysis, ISSTA ’13, pages 202–212, 2013.

10. Maiga, A. et al. 2012. SMURF: a SVM based incremental anti-pattern detection approach.In Proceedings of the 19th Working Conference on Reverse Engineering (WCRE). IEEE Computer Society Press.

11. 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,ASE '10, pages 113{122, New York, NY, USA, 2010. ACM.

12. Vittorio Cortellessa, Antinisca Di Marco, and Catia Trubiani. Software performance antipatterns: Modeling and analysis. In Proceedings of the 12th International Conference on Formal Methods for the Design of Computer, Communication, and Software Systems: Formal Methods for Model-driven Engineering, SFM'12, pages 290-335, 2012.