Contemporary Approach for Technical Reckoning Code Smells Detection using Textual Analysis

Dr. P. Sengottuvelan,
Associate Professor, Department of computer science,
Periyar University PG Extension Centre,
Dharmapuri - 636705, INDIA (phone: 04342-230399;
e-mail: sengottuvelan@gmail.com

M. Sangeetha,
Ph.D Research scholar, Department of computer science,
Periyar University PG Extension Centre,
Dharmapuri - 636705, INDIA
e-mail: mslion2010@gmail.com

***Abstract*—**Software Designers should be aware of address design smells that can evident as results of design and decision. In a software project, technical debt needs to be repaid habitually to avoid its accretion. Large technical debt significantly degrades the quality of the software system and affects the productivity of the development team. In tremendous cases, when the accumulated technical reckoning becomes so enormous that it cannot be paid off to any further extent the product has to be abandoned. In this paper, we bridge the gap analyzing to what coverage abstract information, extracted using textual analysis techniques, can be used to identify smells in source code. The proposed textual-based move toward for detecting smells in source code, fabricated as TACO (Textual Analysis for Code smell detection), has been instantiated for detecting the long parameter list smell and has been evaluated on three sampling Java open source projects. The results determined that TACO is able to indentified between 50% and 77% of the smell instances with a exactitude ranging between 63% and 67%. In addition, the results show that TACO identifies smells that are not recognized by approaches based on exclusively structural information.

***Index Terms*—** *Code Smell, Software refactoring, Technical Debt, Code Debt.*

*****

I. INTRODUCTION

In real world environment source code of software is becomes more intricate to read or debug and even harder to widen. Cleaning up bad smells in the source code is used to improve software readability and extensibility. Software refactoring is the process of changing the internal structure of object-oriented software to improve the quality of software code, especially in terms of maintainability, extensibility, and reusability while software outer performance remains unchanged. In order to improve software refactoring, several tools has been employed for code smell detection.

Bad smells in the code

The term bad smell was introduced by Fowler and Beck. According to Martin Fowler, "A code smell is a exterior suggestion that usually corresponds to a deeper problem in the system". Code smells are not usually bugs-that are technically weaknesses in design the term appears to have been coined by Kent Back and Wards Wiki in late 1990's. The term code smell is also used by agile programs. Smells are come from some recurring, poor designs solution also known as anti patterns. The smells need to the carefully detected and monitored by Researchers.

TABLE I

The Primary smells are	
Code Smells	**Descriptions**
Feature Envy	This smell, in which class is involved to use data or function of another class in the source code
Large Class	Too much functionality is collected into one class. Some programmers or developers make a large class for their handiness but it lead to many confusions where the code is analyzed or read by the programmer it is really hard to understand the functionality of large classes
Duplicate Code	The simplest duplicated code problem is when the same expressions in two methods of same classes
Long Method	Long procedure or method used in classes, so it is difficult to understand
Long Parameter list	Parameter list is too long
Divergent Change	One class is commonly changed in different ways from different reasons
Temporary field	Class has a variable which is used in some situations
Dead Code	Code that is never run or does not perform any functionality in the source code

II. TECHNICAL DEBT

Technical debt (also known as design debt or code debt) is "A concept in training that reflects the extra development work that arises when code that is easy to implement in the short run is used as a substitute of applying the best overall solution". Technical Debt is metaphor coined by Ward Cumminghan in a 1992 report. Technical Debt is analogous to financial debt. There is multiple source of technical debt. Dimension of technical debt include

a. Code debt:
b. Design debt
c. Test debt
d. Documentation debt

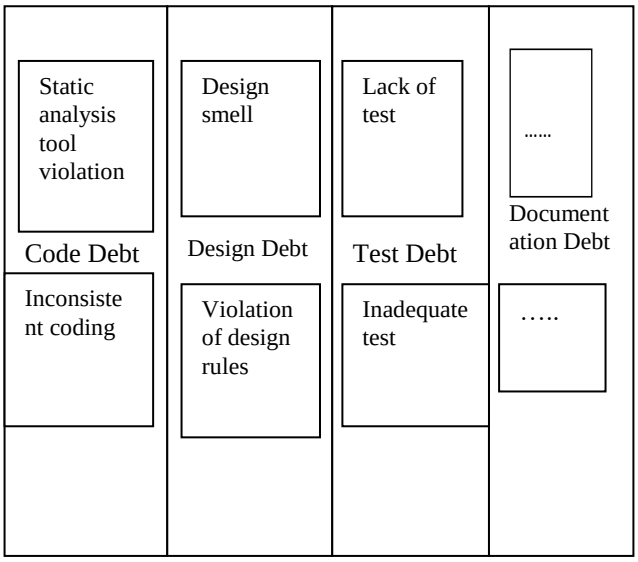

Fig 1. DIMENSION OF TECHNICAL DEBT

III. RESEARCH PROBLEM AND MOTIVATION

Technical debt is an symbol used to describe the consequences of poor software design and bad coding. In particular, the debt represents a measure of code that needs to be re- written or accomplished before a particular task can be considered complete [9]. The image explains well the trade-offs between delivering the most suitable but still immature product, in the shortest time possible [7], [9], [13], [14], [24]. Code smells i.e., symptoms of poor design and implementation choices [11], are one of the most important factors subscribe to technical debt. In the past and, most conspicuously in recent years, several studies investigated the relevance that code smells have for developers [21], [32], the extent to which code smells favor to remain in a software system for long periods of time [2], [8], as well as the side effects of code smells, such as increase in change and responsibility proneness[12] or decrease of software understandability [1] and maintainability [25], [31], [30].

The results achieved in these studies have recommended the need to properly manage smells aiming at improving the quality of software systems. Thus, several tools and methods approaches have been projected for detecting smells [17], [18], [19], [20], [22], [23], [26], [27], [28], and, whenever possible, triggering refactoring operations [5], [4], [27]. While approaches to appropriate smells have investigated the use of both structural and conceptual information extracted from source code, approaches to identify smells are based on structural information only. Recently, Palomba et al. [22] have also used chronological information to identify smell. In the framework of their study, the authors obtained that using chronological information it is possible to identify smell instances that are missed using structural in sequence only. In this paper, we speculate that also by using conceptual information it is possible to categorize smell instances that are missed by using other sources of information. In other words, we suppose that, as obtained in other software engineering tasks (see e.g., [6], [15], [16]), conceptual properties can provide complementary information to structural properties when identifying smells in source code. In order to verify our conjecture, we present TACO (Textual Analysis for Code smell detection), a textual-based smell detection approach. TACO has been instantiated for the detection of a specific smell, i.e., Long parameter list. However, the approach can be easily extended to other smells. The choice of Long parameter list is not random, but guided by the idea that such a smell is a ideal candidate to evaluate the benefits of conceptual information. certainly, a method with a high number of lines of code likely implements different responsibilities and thus textual examination could be mostly suitable to identify such responsibilities.

IV. APPROACH AND UNIQUENESS

Fowler [11] described the Long parameter list as a method Long parameter list is a parameter list that is too long and thus difficult to understand.

Symptoms: A method with too many parameters that is difficult to understand Solution: Introduce Parameter Object, Replace Method with Method Object

Thus, the key idea behind TACO is that a Long parameter list contains a set of code blocks conceptually unrelated each that should be managed separately. Figure 1overviews the main steps of the proposed approach. First, TACO extracts from a method Mi the blocks composing it, applying the technique proposed by Wang et al. [29]. Then, from each block TACO extracts the identifiers and comments concentrated effort the text from non-relevant words, such as language keywords. Each cleaned block of code is viewed as a document, and for each pair of code block is computed a value of similarity using Latent Semantic Indexing (LSI) [10]. The similarity values between all the possible pairs of blocks are stored in a block similarity matrix, where a nonspecific entry ci,j represent the similarity between the method blocks bi and bj . If in the block similarity matrix there is ingress (i.e., similarity between two code blocks) lower than α, then a Long parameter list instance is identified. The constraint 'α' has been

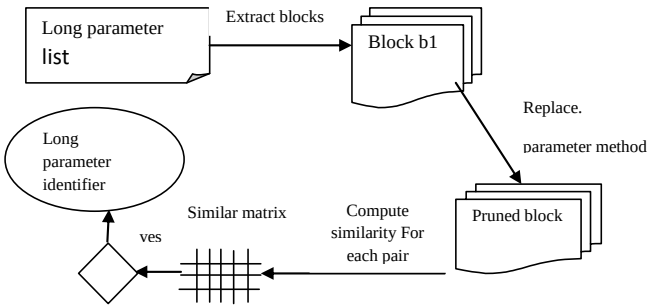

Fig. 2. TACO Identification of Long parameter smell.

empirically evaluated and set to 0.4.

V. Preliminary Evaluation

We estimate the accuracy of TACO in detecting Long Parameter list smell instances in three softwares, Phython , Apache Xerces2 and Eclipse Core3 . Be- sides the investigation of the accuracy of TACO we also prepare the projected approach with a structural-based technique, namely DECOR [18]. In order to appraise the precision of the experimented techniques, we compare the set of Long parameter list instance identified by a specific technique with the set of instances manually identified in the object system. Details on how these smells have been manually recognized can be found in the paper by Palomba et al. [21]. Then, we appraise the accuracy of the experimented techniques by using three widely-adopted Information Retrieval (IR) metrics, namely recall, precision, and F-measure [3].

In addition, we also evaluate the overlap between TACO and DECOR by measuring the smell instances accepted by both the technique (TACO ∩ DECOR), the instances identified by TACO only (TACO \ DECOR) and the instances identified by DECOR only (DECOR \ TACO). Table I shows the results achieved. As we can see, TACO is able to detect Long parameter list instances with good accuracy in all the object systems. certainly, TACO is able to achieve, overall, a precision of 75% and a recall of 62% (F-measure=63%), while DECOR is able to achieve a precision of 54% and a recall of 74% (F-measure=51%).

TABLE II

OVERLAP BETWEEN TACO AND DECOR			
System	TACO \ DECOR	TACO n DECOR	DECOR n TACO
phython	12%	44%	44%
Clion	0%	43%	57%
Eclipse	77%	23%	0%

In Eclipse Core, where DECOR detects a large number of candidate smells (i.e., 122), obtaining a very low value of precision. On this system, TACO detects 6 instances of Long parameter list, achieving a good concession between precision and recall (F- measure=71%). Analyzing more in details the reasons behind this result, we observed that Eclipse has several number of methods having more than 100 lines of code, and this is why they are detected as Long parameter list by the code analysis technique. However, the most part of these methods manage a single responsibility, but in a long piece of code. For example, the method find Types From Imports of the -class Completion Engine is identified by DECOR as Long parameter list since it has 125 lines of code, but it only contains the implementation of an algorithm that ends the reference of a class looking at its imports. On the other hand, our approach is able to identify different types of Long parameter list.

As an example, the method finds Types and Packages of the class Completion Engine allows to discover the classes and the packages of a given project. Clearly, this method manages different tasks, even if its size is not high. This means that the use of textual analysis is actually useful to let alone the identification of many false positive candidates, but also to detect instances of Long parameter list that the structural technique is not able to detect. This claim is supported by the results achieve when analyze the overlap between TACO and DECOR .

TABLE III

Project	Prec.	Recall	F-measure	Prec.	Recall	F-measure
Phython	0.84	0.5	0.63	0.63	0.5	0.56
Clion	0.63	0.72	0.67	0.68	0.57	0.62
Eclipse	0.10	1 0.	19	0.67	0.77	0.71
Overall	0.52	0.74	0.51	0.65	0.61	0.63
	DECOR			TACO		

(see Table II). The two approaches are highly complementary on two out of three systems analyzed in the study. This result suggests that structural and abstract information are complementary when used to identify smells and thus better accuracy might be obtained by combining the two approaches. Future work will be loyal to investigate such an aspect.

VI. Conclusion

We presented TACO (Textual Analysis for Code smell detection), an approach to detect Long parameter list smells in source code by analyzing the textual information extracted by the code blocks in a method. The analysis of textual information for smell detection represent a ruler of this paper, since all the detection approaches proposed in the literature so far use structural or past information. As future work, we plan to instantiate TACO for detecting other kinds of smells. For example, Eclipse and Gene-based Algorithm used detected

smells and applying the same technique presented in this paper at a higher level of granularity, i.e., instead of computing comparison between code blocks it is necessary to compute the relationship between methods (in case of Eclipse) or classes (in case of Gene-based). Also the Feature Envy smell can be detected by using TACO. In this case it is necessary to compute the similarity between a method and all the used classes aiming at identifying the envied class. In addition, the preliminary estimate of TACO indicated a quite low overlap between the set of smells identified by TACO and a structural based detection technique.

In future, a new approach is the possibility of combine the two approaches to concentrated a mixture and more accurate smell detector

REFERENCES

[1] M. Abbes, F. Khomh, Y.-G. Gue´he´neuc, and G. Antoniol, "An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension," in 15th European Conference on Software Maintenance and Reengineering, CSMR 2011, 1-4 March 2011, Old- enburg, Germany. IEEE Computer Society, 2011, pp. 181–190.

[2] R. Arcoverde, A. Garcia, and E. Figueiredo, "Understanding the longevity of code smells: preliminary results of an explanatory survey," in Proceedings of the International Workshop on Refactoring Tools. ACM, 2011, pp. 33–36. [3] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval. Addison-Wesley, 1999.

[3] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "Improving software modularization via automated analysis of latent topics and dependencies," ACM Transactions on Software Engineering and Methodologies, vol. 23, no. 1, pp. 1–33, 2014.

[4] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "Methodbook: Recommending move method refactorings via relational topic models," IEEE Transactions on Software Engineering, 2014.

[5] G. Bavota, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "An empirical study on the developers' perception of software coupling," in Proceedings of the 2013 International Conference on Software Engineering, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 692–701. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486879

[6] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. L. Nord, I. Ozkaya, R. S. Sangwan, C. B. Seaman, K. J. Sullivan, and N. Zazworka, "Managing technical debt in software- reliant systems," in Proceedings of the Workshop on Future of Software Engineering Research, at the 18th ACM SIGSOFT International Sym- posium on Foundations of Software Engineering. Santa Fe, NM, USA: ACM, 2010, pp. 47–52.

[7] A. Chatzigeorgiou and A. Manakos, "Investigating the evolution of bad smells in object-oriented code," in International Conference on the Quality of Information and Communications Technology (QUATIC). IEEE, 2010, pp. 106–115.

[8] W. Cunningham, "The wycash portfolio management system," SIGPLAN OOPS Mess., vol. 4, no. 2, pp. 29–30, Dec. 1992. [Online]. Available: http://doi.acm.org/10.1145/157710.157715

[9] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," Journal of the American Society for Information Science, vol. 41, no. 6, pp. 391–407,1990. [11] M. Fowler, Refactoring: improving the design of existing code. Addison-Wesley, 1999.

[10] F. Khomh, M. Di Penta, Y.-G. Gue´he´neuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on class change- and fault-proneness," Empirical Software Engineering, vol. 17, no. 3, pp. 243–275, 2012.

[11] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," IEEE Software, vol. 29, no. 6, pp. 18–21, 2012.

[12] E. Lim, N. Taksande, and C. B. Seaman, "A balancing act: What software practitioners have to say about technical debt," IEEE Software, vol. 29, no. 6, pp. 22–27, 2012.

[13] A. Marcus and D. Poshyvanyk, "The conceptual cohesion of classes," in Proceedings of 21st IEEE International Conference on Software Maintenance, Budapest, Hungary, 2005, pp. 133–142.

[14] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems," IEEE Transaction on Software Engineering, vol. 34, no. 2, pp. 287–300, 2008.

[15] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in 20th International Conference on Software Mainte-

[16] N. Moha, Y.-G. Gue´he´neuc, L. Duchien, and A.-F. L. Meur, "Decor: A method for the specification and detection of code and design smells," IEEE Transactions on Software Engineering, vol. 36, no. 1, pp. 20–36, 2010.

[17] M. J. Munro, "Product metrics for automatic identification of "bad smell" design problems in java source-code," in Proceedings of the 11th International Software Metrics Symposium. IEEE Computer Society Press, September 2005.

[18] R. Oliveto, F. Khomh, G. Antoniol, and Y.-G. Gue´he´neuc, "Numer- ical signatures of antipatterns: An approach based on b-splines," in Proceedings of the 14th Conference on Software Maintenance and Reengineering, R. Capilla, R. Ferenc, and J. C. Dueas, Eds. IEEE Computer Society Press, March 2010.

[19] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, and A. De Lucia, "Do they really smell bad? a study on developers' perception of bad code smells," in In Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME'14), Victoria, Canada, 2014, to appear

[20] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "Mining version histories for detecting code smells," IEEE Transactions on Software Engineering, 2015.

[21] D. Ratiu, S. Ducasse, T. Gˆırba, and R. Marinescu, "Using history information to improve design flaws detection," in 8th European Confer- ence on Software Maintenance and Reengineering (CSMR 2004), 24-26 March 2004, Tampere, Finland, Proceeding. IEEE Computer Society,2004, pp. 223–232.

[22] F. Shull, D. Falessi, C. Seaman, M. Diep, and L. Layman, Perspectives on the Future of Software Engineering. Springer,

2013, ch. Technical Debt: Showing the Way for Better Transfer of Empirical Results, pp.179–190.

[23] D. I. K. Sjøberg, A. F. Yamashita, B. C. D. Anda, A. Mockus, and T. Dyba˚, "Quantifying the effect of code smells on maintenance effort," IEEE Trans. Software Eng., vol. 39, no. 8, pp. 1144–1156, 2013.

[24] G. Travassos, F. Shull, M. Fredericks, and V. R. Basili, "Detecting defects in object-oriented designs: using reading techniques to increase software quality," in Proceedings of the 14th Conference on Object- Oriented Programming, Systems, Languages, and Applications. ACM Press, 1999, pp. 47–56.

[25] N. Tsantalis and A. Chatzigeorgiou, "Identification of move method refactoring opportunities," IEEE Transactions on Software Engineering, vol. 35, no. 3, pp. 347–367, 2011.

[26] E. van Emden and L. Moonen, "Java quality assurance by detecting code smells," in Proceedings of the 9th Working Conference on Reverse Engineering (WCRE'02). IEEE CS Press, Oct. 2002. [Online]. Available: citeseer.ist.psu.edu/vanemden02java.html

[27] X. Wang, L. Pollock, and K. Vijay-Shanker, "Automatic segmentation of method code into meaningful blocks to improve readability," in Reverse Engineering (WCRE), 2011 18th Working Conference on, Oct 2011, pp.35–44.

[28] A. Yamashita and L. Moonen, "Exploring the impact of inter-smell rela- tions on software maintainability: An empirical study," in International Conference on Software Engineering (ICSE). IEEE, 2013, pp. 682–691.

[29] A. F. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?" in 28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23-28, 2012. IEEE Computer Society, 2012, pp. 306–315.

[30] ——, "Do developers care about code smells? an exploratory survey," in 20th Working Conference on Reverse Engineering, WCRE 2013, Koblenz, Germany, October 14-17, 2013. IEEE, 2013, pp. 242–251...