Textual Analysis for Code Smell Detection

Fabio Palomba
Department of Management & Information Technology
University of Salerno, Fisciano (SA), Italy
Advisors: Andrea De Lucia, Rocco Oliveto

***Abstract*—The negative impact of smells on the quality of a software systems has been empirical investigated in several studies. This has recalled the need to have approaches for the identification and the removal of smells. While approaches to remove 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. In this paper, we bridge the gap analyzing to what extent conceptual information, extracted using textual analysis techniques, can be used to identify smells in source code. The proposed textual-based approach for detecting smells in source code, coined as *TACO* (Textual Analysis for Code smell detectiOn), has been instantiated for detecting the Long Method smell and has been evaluated on three Java open source projects. The results indicate that TACO is able to detect between 50% and 77% of the smell instances with a precision ranging between 63% and 67%. In addition, the results show that TACO identifies smells that are not identified by approaches based on solely structural information.**

I. Research Problem and Motivation

Technical debt is a metaphor used to describe the consequences of poor software design and bad coding. Specifically, the debt represents a piece of code that needs to be rewritten or completed before a particular task can be considered complete [9]. The metaphor explains well the trade-offs between delivering the most appropriate but still immature product, in the shortest time possible [7], [9], [13], [14], [24]. Code smells (shortly *smells*), i.e., symptoms of poor design and implementation choices [11], are one of the most important factors contributing to technical debt. In the past and, most notably, in recent years, several studies investigated the relevance that code smells have for developers [21], [32], the extent to which code smells tend 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 fault-proneness [12] or decrease of software understandability [1] and maintainability [25], [31], [30].

The results achieved in these studies have suggested the need to properly manage smells aiming at improving the quality of a software systems. Thus, several approaches and tools have been proposed for detecting smells [17], [18], [19], [20], [22], [23], [26], [27], [28], and, whenever possible, triggering refactoring operations [5], [4], [27]. While approaches to remove 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 historical information to identify smell. In the context of their study, the authors obtained that using historical information is possible to identify smell instances that are missed using structural information only. In this paper, we conjecture that also by using conceptual information is possible to identify smell instances that are missed by using other sources of information. In other words, we believe that, as obtained in other software engineer 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* (**T**extual **A**nalysis for **C**ode smell detecti**O**n), a textual-based smell detection approach. TACO has been instantiated for the detection of a specific smell, i.e., *Long Method*. However, the approach can be easily extended to other smells. The choice of *Long Method* is not random, but guided by the idea that such a smell is a perfect candidate to evaluate the benefits of conceptual information. Indeed, a method with a high number of lines of code likely implements different responsibilities and thus textual analysis could be particularly suitable to identify such responsibilities.

II. Approach and Uniqueness

Fowler [11] described the *Long Method* as a method in which there is the implementation of a main functionality together with auxiliary functions that should be managed in different methods. Thus, the key idea behind TACO is that a *Long Method* contains a set of code blocks conceptually unrelated each that should be managed separately.

Figure 1 overviews the main steps of the proposed approach. First, TACO extracts from a method M_i the blocks composing it, applying the technique proposed by Wang et al. [29]. Then, from each block TACO extracts the identifiers and comments cleaning 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 generic entry $c_{i,j}$ represent the similarity between the method blocks b_i and b_j. If in the block similarity matrix there is an entry (i.e., similarity between two code blocks) lower than α, then a *Long Method* istance is identified. The parameter α has been empirically evaluated and set to 0.4.

III. Preliminary Evaluation

We evaluate the accuracy of TACO in detecting *Long Method* smell instances in three software systems, namely

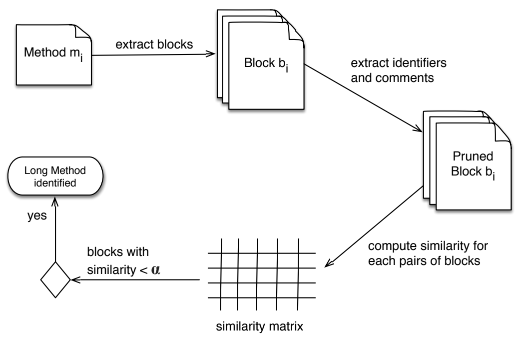

Fig. 1. TACO: Identification of Long Method smell.

TABLE I
RESULTS OBTAINED ON THE THREE OBJECT SYSTEMS

	DECOR			TACO		
Project	Prec.	Recall	F-measure	Prec.	Recall	F-measure
Apache Cassandra	0.84	0.5	0.63	0.63	0.5	0.56
Apache Xerces	0.63	0.71	0.67	0.67	0.57	0.62
Eclipse Core	0.10	1	0.19	0.67	0.77	0.71
Overall	**0.52**	**0.74**	**0.51**	**0.65**	**0.61**	**0.63**

Apache Cassandra[1], Apache Xerces[2] and Eclipse Core[3]. Besides the analysis of the accuracy of TACO we also compare the proposed approach with a structural-based technique, namely DECOR [18].

In order to evaluate the accuracy of the experimented techniques, we compare the set of *Long Method* instances identified by a specific technique with the set of instances manually identified in the object system. Details on how these smells have been manually identified can be found in the paper by Palomba et al. [21]. Then, we measure 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 measure the overlap between TACO and DECOR by measuring the smell instances identified by both the technique (TACO $\cap$ DECOR), the instances identified by TACO only (TACO $\setminus$ DECOR) and the instances identified by DECOR only (DECOR $\setminus$ TACO).

Table I shows the results achieved. As we can see, TACO is able to detect *Long Method* instances with good accuracy in all the object systems. Indeed, TACO is able to achieve, overall, a precision of 65% and a recall of 61% (F-measure=63%), while DECOR is able to achieve a precision of 52% and a recall of 74% (F-measure=51%). An interesting case regards *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 Method*, achieving a good compromise between precision and recall (F-measure=71%). Analyzing more in details the reasons behind this result, we observed that *Eclipse Core* has several number of methods having more than 100 lines of code, and this is why they are detected as Long Methods by the code

[1]http://cassandra.apache.org
[2]http://xerces.apache.org
[3]http://www.eclipse.org/eclipse/platform-core/

TABLE II
OVERLAP BETWEEN TACO AND DECOR

System	TACO $\cap$ DECOR	TACO $\setminus$ DECOR	DECOR $\setminus$ TACO
Apache Cassandra	12%	44%	44%
Apache Xerces	0%	43%	57%
Eclipse Core	77%	23%	0%

analysis technique. However, the most part of these methods manage a single responsibility, but in a long piece of code. For example, the method `findTypesFromImports` of the class `CompletionEngine` in identified by DECOR as Long Method since it has 125 lines of code, but it only contains the implementation of an algorithm that finds the references of a class looking at its imports. On the other hand, our approach is able to identify different types of Long Method. As an example, the method `findTypesAndPackages` of the class `CompletionEngine`, 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 avoid the identification of many false positive candidates, but also to detect instances of *Long Method* that the structural technique is not able to detect. This claim is supported by the results achieved when analyzing the overlap between TACO and DECOR (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 conceptual information are complementary when used to identify smells and thus better accuracy might be obtained by combining the two approaches. Future work will be devoted to investigate such an aspect.

IV. CONTRIBUTIONS AND FUTURE DIRECTIONS

We presented TACO (Textual Analysis for Code smell detectiOn), an approach to detect *Long Method* 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 premier of this paper, since all the detection approaches proposed in the literature so far use structural or historical information. As future work, we plan to instantiate TACO for detecting other kinds of smells. For example, *Blob* and *Promiscuous Package* smells can be detected applying the same technique presented in this paper at a higher level of granularity, i.e., instead of computing similarity between code blocks it is necessary to compute the similarity between methods (in case of *Blob*) or classes (in case of *Promiscuous Package*). 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 evaluation of TACO indicated a quite low overlap between the set of smells identified by TACO and a structural-based detection technique. The possibility of combine the two approaches to define a hybrid and more accurate smell detector is also part of the agenda of our future work.

REFERENCES

[1] 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 *15th European Conference on Software Maintenance and Reengineering, CSMR 2011, 1-4 March 2011, Oldenburg, 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.

[4] 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.

[5] 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.

[6] 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

[7] 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 Symposium on Foundations of Software Engineering*. Santa Fe, NM, USA: ACM, 2010, pp. 47–52.

[8] 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.

[9] 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

[10] 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.

[12] F. Khomh, M. Di Penta, Y.-G. Guéhéneuc, 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in *20th International Conference on Software Maintenance (ICSM 2004), 11-17 September 2004, Chicago, IL, USA*. IEEE Computer Society, 2004, pp. 350–359.

[18] N. Moha, Y.-G. Guéhéneuc, 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.

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

[20] R. Oliveto, F. Khomh, G. Antoniol, and Y.-G. Guéhéneuc, "Numerical signatures of antipatterns: An approach based on b-splines," in *Proceedings of the* 14^{th} *Conference on Software Maintenance and Reengineering*, R. Capilla, R. Ferenc, and J. C. Dueas, Eds. IEEE Computer Society Press, March 2010.

[21] 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.

[22] 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.

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

[24] 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.

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

[26] 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* 14^{th} *Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM Press, 1999, pp. 47–56.

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

[28] 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

[29] 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.

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

[31] 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.

[32] ——, "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.