Abstract—The negative impact of smells on the quality of a software system 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, 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 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 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 \( M \), 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 \( \alpha \), then a Long Method instance is identified. The parameter \( \alpha \) 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
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 \ 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 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

\[ \text{Table I}
\]

<table>
<thead>
<tr>
<th>Project</th>
<th>DECOR Prec</th>
<th>DECOR Recall</th>
<th>DECOR F-measure</th>
<th>TACO Prec</th>
<th>TACO Recall</th>
<th>TACO F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Cassandra</td>
<td>0.63 0.71</td>
<td>0.67 0.67</td>
<td>0.57 0.62</td>
<td>0.63 0.77</td>
<td>0.71 0.71</td>
<td></td>
</tr>
<tr>
<td>Apache Xerces</td>
<td>0.63 0.71</td>
<td>0.67 0.67</td>
<td>0.57 0.62</td>
<td>0.63 0.77</td>
<td>0.71 0.71</td>
<td></td>
</tr>
<tr>
<td>Eclipse Core</td>
<td>0.10 0.19</td>
<td>0.67 0.77</td>
<td>0.71 0.71</td>
<td>0.63 0.77</td>
<td>0.71 0.71</td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td>0.52 0.74</td>
<td>0.51 0.65</td>
<td>0.61 0.63</td>
<td>0.63 0.77</td>
<td>0.71 0.71</td>
<td></td>
</tr>
</tbody>
</table>

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 \ 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 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

![Fig. 1. TACO: Identification of Long Method smell.](image-url)

---

**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.