Identifying Technical Debt through Code Comment Analysis

Mário André de Freitas Farias[1,2], Methanias Colaço[5], Rodrigo Oliveira Spínola[3,4] and Manoel G. de Mendonça Neto[2]
[1]*Federal Institute of Sergipe, Lagarto, Brazil*
[2]*Department of Post-Graduate in Computer Science, Federal University of Bahia, Salvador, Brazil*
[3]*Fraunhofer Project Center at UFBA, Salvador, Brazil*
[4]*Salvador University, Salvador, Brazil*
[5]*Federal University of Sergipe, Itabaiana, Brazil*

1 RESEARCH PROBLEM

The term Technical Debt (TD) is being increasingly used to discuss technical compromises admitted by the development team during the phases of the software life cycle. Thus, this metaphor defines the Trade-off between internal tasks you choose do not perform at present, and the risk of causing future problems (Izurieta *et al.,* 2012). Currently, this may include immature software artifacts such as issues in the software design and in the software architecture, incomplete or insufficient documentation, incomplete design specification, insufficient code comment, lack of adequate testing, or inadequate technology (Alves *et al.,* 2014).

The identification of TD is the first step to effectively manage TD, it is necessary to identify TD items in the project before prioritizing them and select those which should be paid or not (Guo *et al.,* 2014). The term "TD item" represents an instance of TD for the purpose of this study.

Recent systematic reviews (Li *et al.,* 2014) (Alves *et al.,* 2016) reported that code quality analysis techniques have been frequently studied to support the identification of TD. In this sense, automatic analysis tools have used software metrics extracted from the source code to identify TD items by comparing values of software metrics to predefined thresholds (Mendes *et al.,* 2015).

Li *et al.* (Li *et al.,* 2014) analyzed and classified 29 different tools. Only one (FxCop) takes .NET assemblies as input, one (RE-KOMBINE) takes requirements and solutions as input, and one (CLIO) takes compiled binaries as input. Most 86% of the tools take source code as input, some of these tools are mentioned as follows: (i) SIG Software Analysis Toolkit is used to calculate code properties to identify code TD (Nugroho *et al.,* 2011), (ii) Resource Standard Metrics calculates source code metrics and analyzing code quality to find style violations and logic problems. This tool can identify design and code TD, and (iii) CodeVizard is a tool for detecting design TD thought code smells identification (Zazworka and Ackermann, 2010).

Although these tools and techniques have shown useful to extract structural information and support the automated management of some types of debt, they do not cover its real meaning and human factors (e.g., tasks commented as future work), taking aside them on it (Zazworka *et al.,* 2013) (Potdar and Shihab, 2014). Thus, they may not point to a relevant TD or not report a piece of code which is really considered a TD by developers. Besides, some types of debt are undetectable by tools and may not be directly identified using only metrics collected from the source code (Farias *et al.,* 2015).

In this sense, pieces of code that need to be refactored to improve the quality of the software may continue unknown. In order to complement the TD identification with more contextual and qualitative data, human factors and the developers' point of view should be considered.

2 OUTLINE OF OBJECTIVES

2.1 General Objective

Our objective is to propose an approach to support and automate the identification and management of different TD types through code comment analysis by considering the developers' point of view.

2.2 Complementary Objectives

- Perform a systematic mapping study with the aim of investigating how research is being conducted in the mining software repositories field. This allowed us to identify the main approaches with focus on comment analysis;

Farias, M., Colaço, M., Spínola, R. and Neto, M.
Identifying Technical Debt through Code Comment Analysis.
In *Doctoral Consortium (DCEIS 2016)*, pages 9-14

- Analyze and categorize contextualized terms, combinations, and patterns to understand how they may be combined to support the identification and management of different TD types through the comments analysis;
- Create a structure that systematically allows combining terms providing a large vocabulary to support the TD identification;
- Develop an automated tool in order to quickly analyze developer's comments embedded in source codes;
- Propose and plan a family of experiments. From them, we intend to evaluate and evolve our approach with the purpose of characterizing its overall accuracy and factors affecting the identification of TD through code comment analysis.
- Improve our proposed model, vocabulary, and tool;

3 STATE OF THE ART

3.1 Code Comments

Comments are a generic type of task annotation where developers insert documentation directly into source code (Storey *et al.,* 2008). These annotations are richness of semantic information written in natural language.

Despite there are different syntaxes and types of comments according to the programming language, they are divided into two categories: (i) inline comments, which only permit the insertion of one line of comment, and (ii) block comments, which permit the insertion of several lines. Developers write comments in a sublanguage of English using a limited set of verbs and tenses, and personal pronouns are almost not used (Davis and Bowen, 2001).

Code comments and the source code itself are an important documentation to help the software comprehension (Souza *et al.,* 2006). These descriptions may reveal important information, such as the reason for adding new lines to source code, knowing about the progress of a collective task, or even why relevant changes were performed. Thus, comments may be used to describe issues that may require work in the future, notice emerging problems and what decisions need to be taken about them (Maalej and Happel, 2010) (Shokripour *et al.,* 2013).

3.2 Code Comment Analysis

When the source code is well commented, we can understand what a piece of code does, what issues it has, and whether it needs to be fixed or improved, without needing to analyze its implementation. In general, comments are used by developers to understand unfamiliar software because comments are written in natural language (Freitas *et al.,* 2012). Comments provide an important set of information which may help to understand software features, and make easier software comprehension.

In fact, comments have been used to describe issues that may require future work, emerging problems and decisions taken about those problems (Maalej and Happel, 2010). These descriptions facilitate human readability and provide additional information that summarizes the developer context.

In this sense, authors have conducted experiments using code comments as data source in several research works in order to discuss and analyze their importance on the software comprehension.

In (Storey *et al.*,, 2008) the authors described an empirical study that explored how code comments play a role in how developers deal with software maintenance tasks, investigating how comments may improve processes and tools that are used for managing these tasks. In similar approach, (Maalej and Happel, 2010) analyzed the purpose of work descriptions and code comments aiming to discuss how automated tools can support developers in creating them.

Some research works from Mining Software Repository (MSR) have focused on code comments. Yang and Tan (Yang and Tan, 2012) proposed an approach that analyses the word context in code comments. The main idea is to discover semantically related words. Many words that are semantically related in software development process are not semantically related in English. In this same sense, Howard *et al.* (Howard *et al.,* 2013) also presented an approach to augment natural language thesauri with code-related terms.

(Freitas *et al.,* 2012) presents an approach to locate problem domain concepts on comments, and identify the relevant code chunks associated with them. The authors also introduce Darius, a tool to implement their proposal for Java programs. Darius identifies and extracts inline, bock, and Javadoc comments and provides some metrics. They concluded that higher level source entities tend to have comments oriented for problem domain information, whereas comments of lower level source entities tend to include more program domain information.

In other work, Gupta *et al.* (Gupta *et al.,* 2013) suggested a Part-of-speech tagging of program

identifiers to understand how a program element is named. Considering identifiers and comments, (Salviulo *et al.,* 2014) performed an experiment with students and young professional developers in order to understand how they perceive comments and identifier names.

3.3 Using Code Comments to Identify TD

More recently, code comments have been explored with the purpose of identifying TD. Potdar and Shihab (Potdar and Shihab, 2014) analyzed code comments to identify text patterns and TD items. For that, the authors used the *srcML toolkit* (Maletic *et al.,* 2002), a command line tool that parses source code into XML file, to extract the comments. In this step, the authors considered all types of comments. This decision may bring a lot of unnecessary effort because it considers comments that are not important to TD scope, such as license and Auto-generated comments. After the data extraction, the authors identified comments that indicate TD.

They read more than 101K code comments, and organized 62 text patterns that were used to quantify how much TD exist in four different projects (Eclipse, Chromium OS, ArgoUML, and Apache http). Their findings show that 2.4 - 31.0% of the files in a project contain self-admitted TD. In addition, the most used text patterns were: (i) “is there a problem” with 36 instances in the Eclipse, (ii) “hack” with 17 instances in the ArgoUML, and (iii) “fixme” with 20 instances in the Apache, and 761 instances in the Chrominum OS.

In another TD identification approach, Maldonado and Shihab (Maldonado and Shihab, 2015) evolved the work of Potdar and Shihab (Potdar and Shihab, 2014), proposing four simple filtering heuristics to eliminate comments that are not likely to contain technical debt. For that, they read 33K code comments from five open source projects (Apache Ant, Apache Jmeter, ArgoUML, Columba, and JFreeChart). Their findings showed that self-admitted technical debt can be classified into five main types: design debt, defect debt, documentation debt, requirement debt, and test debt. According to the authors, the most common type of self-admitted TD is design debt (between 42% and 84% of the classified comments).

In the same sense, Farias *et al.* (Farias *et al.*, 2015) proposed a model aiming to support the detection of different types of debt through code comment analysis.

These research works provide preliminary indication that comments can be effectively used to support TD identification. However, the factors that may affect its accurate usage are still unknown.

4 METHODOLOGY

Research works in software engineering have widely been conducted with focus on quantitative analysis. In general, this type of study analyzes treatment of variables, control groups, and statistical data (Wohlin *et al.,* 2012).

Differently from quantitative analysis, qualitative one appears to be unusual in software engineering approaches. Using this method is possible to achieve aspects behind the problem under study, analyze data, and suggest conclusions to which other methods would be blind (Segal *et al.,* 2005).

In order to include the combining of quantitative and qualitative approaches as complementary methods, we will use triangulation methodology to analyze how code comment analysis supports the TD identification.

Triangulation is a research strategy described as a convergent methodology with multiple operationalisms (Campbell and Fiske, 1959). The main idea is to analyze evidences from different sources, be collected using different methods, have different forms, be analyzed using different methods, or come from a different study altogether (Shull *et al.,* 2008). In this methodology, researchers can improve conclusions on their judgments through collecting and analysis of different data considering the same phenomenon (Jick 1979).

4.1 Overview

Figure 1 presents an overview of the research. We explore code comments using a systematic mapping study, and a family of experiment in order to propose methods and techniques to support the identification and management of different TD types. With the results of the studies, we expected advance the set of knowledge on how improve the TD identification process through code comment analysis. We briefly describe each step following the numbers of Figure 1.

To begin with, we performed a systematic mapping study (1) in order to understand the mining software repositories area and to identify its current targets and gaps, regarding mainly source codes and comments analysis (Farias et *al*., 2016). We identified some important studies on usage of comments for software comprehension, and some

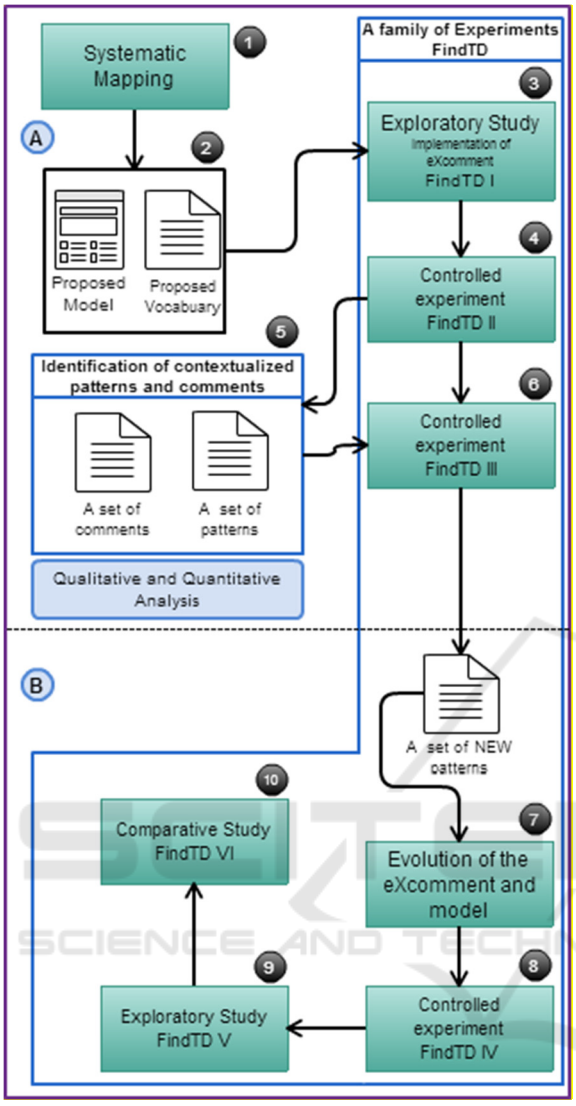

Figure 1: Overview of Research.

techniques and tools used to extract and analyze comments.

Second, we developed a Contextualized Vocabulary Model for identifying TD on code comments (CVM-TD) (2). CVM-TD uses word classes and code tags to support TD identification. The model provides a structure that systematically allows combining terms creating a large vocabulary on TD (Farias *et al.,* 2015).

Next, our work proposes a family of experiment called *FindTD* composed by five experiments, two exploratory studies, and three controlled experiments. A family of experiments involves not only replications, but variations among the experiments (Basili *et al.,* 1999). In this respect, we intend to perform experimental variations in order to evaluate and evolve our set of knowledge on the proposed model and techniques to identify TD using code comments.

In the first experiment (*FindTD I*), an exploratory study (3) was performed to characterize the feasibility of the proposed model to support the detection of TD through code comments analysis. For that, we developed a tool to extract comments from the software code, the *eXcomment*. This tool extracts and filters candidate comments from source code using the contextualized vocabulary provided by the model.

Following, the promising initial outcomes motivated us to further evaluate CVM-TD with other data sources. Thus, we performed a controlled experiment (*FindTD II*) (4). Therefore, in this evaluation we extend Farias *et al.* (Farias *et al.,* 2015) with an additional quantitative study. We analyzed the use of CVM-TD with the purpose of characterizing its overall accuracy when classifying candidate comments and factors that influence the analysis of the comments to support the identification of TD in terms of accuracy. For each candidate comment listed in a form, the participants chose "Yes" or "No", and their level of confidence on their answer. They used an ordinal scale of one to four to represent the confidence. Besides, for each comment marked as yes, they should highlight the pieces of text that was decisive for giving this answer (set of comment patterns).

Our findings indicate that CVM-TD provided promising results considering the accuracy values. We observed that many comments had high agreement, and almost 60% of comments filtered by terms that belong to the vocabulary (candidate comments) proposed in (Farias *et al.*, 2015) were identified as good indicators of TD (Farias *et al.,* 2016).

Next, we designed *FindTD III* (6) from insights of *FinTD II*, by changing the setup and controlling other variables. Our main goal in this experiment is to analyze the set of comment patterns identified and classified in previous experiment. We intend to carry out a qualitative analysis in order to improve the model and the vocabulary, identifying the most important patterns, and the relationship between comment patterns and TD types. To do this, we will perform a coding to group patterns into different comment indicators.

Coding is a method that enables researchers to organize and group similar data into categories or themes, attaching labels or codes to different segments - the beginning of themes. Information from different sources can be easily sorted and compared. A theme is an outcome of coding, categorization, and analytic reflection, not

something that is, in itself coded (Cruzes and Dyba, 2011) (Ellsberg and Heise, 2005).

This experiment will provide us inputs to improve CVM-TD, resulting a new release of the contextualized vocabulary. We also intend to develop new features in *eXcomment.* This feature is associated with the new vocabulary to quickly support the interpretation of comments (7).

After this study, we planned to perform a controlled experiment (8). In this experiment, besides evaluating the new release of the contextualized vocabulary and tool, we expect to compare the overall accuracy when classifying candidate comments between two groups, one using the tool to analyze comments and another one without the tool.

In *FindTD V* (9), we expected to perform an exploratory study in the software industry. In this study, we purpose to compare patterns and TD items identified into open source code and closed code developed in an industrial environment.

The last one is the *FindTD VI* (10). We intend to compare our approaches to different tools that use metrics extracted from the source code to identify TD items.

Our methodology might set some limitations on what can be experimented. The first considers the power of the proposed vocabulary. It is possible that the set of terms and combinations used by our model and vocabulary are simply too many to be studied. An alternative would be to limit the studies to a very specific context and software. Other risk involves the effort to carry out all studies because of the difficulty of performing experiments in this area.

5 EXPECTED OUTCOME

In the context of our empirical investigation, we are interested in findings that help us to comprehend how code comments analysis can support the identification and management of different TD types, considering the developers' point of view.

We hope to develop a rich contextualized vocabulary and a tool to support the TD identification through comment analysis. We believe this approach can improve methods of identifying and classifying TD items, analyzing code comments.

6 STAGE OF THE RESEARCH

In accordance with proposed methodology described in Section 4 and shown in Figure 1, we purpose a systematic mapping study and a family of experiment in order to discuss our goals. Figure 1 is broken down into two parts. The tasks that have already performed were organized on top of the figure (part A), and the tasks that will be performed in the future were organized on bottom of the figure (part B).

In this sense, we have: (i) performed the systematic mapping study, (ii) developed a Contextualized Vocabulary Model, (iii) performed the first exploratory study and its analysis, (iv) performed the first controlled experiment and its analysis, and (v) designed the *FindTD III* and carried out the experiment. Currently, we are analyzing data from this study, using qualitative methods.

REFERENCES

Alves, N.S.R. et al., 2016. Identification and Management of Technical Debt: A Systematic Mapping Study. *Information and Software Technology*, 70, pp.100–121.

Alves, N.S.R. et al., 2014. Towards an Ontology of Terms on Technical Debt. In *Sixth International Workshop on Managing Technical Debt (MTD)*. pp. 1–7.

Basili, V.R., Shull, F. and Lanubile, F., 1999. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4), pp.456–473.

Campbell, D.T. and Fiske, D.W., 1959. Convergent and discriminant validation by the multitrait-multimethod matrix. *Psychological Bulletin*, 56(2), pp.81–105.

Cruzes, D.S. and Dyba, T., 2011. Recommended Steps for Thematic Synthesis in Software Engineering. *2011 International Symposium on Empirical Software Engineering and Measurement*, (7491), pp.275–284.

Davis, C.G. and Bowen, L.L., 2001. The language of comments in computer software : A sublanguage of English. , 166(00), pp.1731–1756.

Ellsberg, M. and Heise, L., 2005. *Researching Violence Against Women. A PRACTICAL GUIDE FOR RESEARCHERS AND ACTIVISTS*, Washington: World Health.

Farias, M. et al., 2015. A Contextualized Vocabulary Model for Identifying Technical Debt on Code Comments. In *Seventh International Workshop on Managing Technical Debt*. pp. 25–32.

Farias, M.A. de F., Novais, R., et al., 2016. A Systematic Mapping Study on Mining Software Repositories. In *ACM SAC*.

Farias, M.A. de F., Silva, A.B., et al., 2016. Investigating the Use of a Contextualized Vocabulary in the Identification of Technical Debt : A Controlled Experiment. In *18Th International Conference on Enterprise Information System - ICEIS (Accepted)*.

Freitas, J.L., Da Cruz, D. and Henriques, P.R., 2012. A comment analysis approach for program

comprehension. *Proceedings of the 2012 IEEE 35th Software Engineering Workshop, SEW 2012*, pp.11–20.

Guo, Y., Spínola, R.O. and Seaman, C., 2014. Exploring the costs of technical debt management – a case study. *Empirical Software Engineering*, 1, pp.1–24.

Gupta, S. et al., 2013. Part-of-speech tagging of program identifiers for improved text-based software engineering tools. *IEEE International Conference on Program Comprehension*, pp.3–12.

Howard, M.J. et al., 2013. Automatically mining software-based, semantically-similar words from comment-code mappings. *IEEE International Working Conference on Mining Software Repositories*, pp.377–386.

Izurieta, C. et al., 2012. Organizing the technical debt landscape. *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, pp.23–26.

Jick, T.D., 1979. Mixing Qualitative and Quantitative Methods : Triangulation in Action Mixing Qualitative and Quantitative Methods : Triangulation in Action *. *Qualitative Methodology*, 24(4), pp.602–611.

Li, Z. et al., 2014. A systematic mapping study on technical debt. *The Journal of Systems and Software*, 101, pp.193–220.

Maalej, W. and Happel, H.-J., 2010. Can development work describe itself? *7th IEEE Working Conference on Mining Software Repositories (MSR)*, pp.191–200.

Maldonado, E.S. and Shihab, E., 2015. Detecting and Quantifying Different Types of Self-Admitted Technical Debt. In *7th International Workshop on Managing Technical Debt*. pp. 9–15.

Maletic, J.I., Collard, M.L. and Marcus, A., 2002. Source Code Files as Structured Documents. In *Proceedings. 10th International Workshop on*. pp. 289–292.

Mendes, T.S. et al., 2015. VisMinerTD - An Open Source Tool to Support the Monitoring of the Technical Debt Evolution using Software Visualization. In *17th International Conference on Enterprise Information Systems*.

Nugroho, A., Visser, J. and Kuipers, T., 2011. An Empirical Model of Technical Debt and Interest Software Improvement Group. *Proceeding of the 2nd working on Managing technical debt MTD 11*, p.1.

Potdar, A. and Shihab, E., 2014. An Exploratory Study on Self-Admitted Technical Debt. In *IEEE International Conference on Software Maintenance and Evolution*. pp. 91–100.

Salviulo, F. et al., 2014. Dealing with Identifiers and Comments in Source Code Comprehension and Maintenance : Results from an Ethnographically-informed Study with Students and Professionals. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering. ACM*. p. 48.

Segal, J., Grinyer, A. and Sharp, H., 2005. The type of evidence produced by empirical software engineers. *ACM SIGSOFT Software Engineering Notes*, 30(4), pp.1–4.

Shokripour, R. et al., 2013. Why So Complicated ? Simple Term Filtering and Weighting for Location-Based Bug Report Assignment Recommendation. , pp.2–11.

Shull, F., Singer, J. and Sjoberg, D., 2008. *Guide to Advanced Empirical Software Engineering*, Springer.

Souza, S.C.B. et al., 2006. Which documentation for software maintenance? *Journal of the Brazilian Computer Society*, 12(3), pp.31–44.

Storey, M. et al., 2008. TODO or To Bug : Exploring How Task Annotations Play a Role in the Work Practices of Software Developers. In *ICSE: International Conference on Software Engineering*. pp. 251–260.

Wohlin, C. et al., 2012. *Experimentation in Software Engineering*,

Yang, J. and Tan, L., 2012. Inferring semantically related words from software context. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. Ieee, pp. 161–170.

Zazworka, N. et al., 2013. A case study on effectively identifying technical debt. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering - EASE '13*. New York, New York, USA: ACM Press, pp. 42–47.

Zazworka, N. and Ackermann, C., 2010. CodeVizard: A Tool to Aid the Analysis of Software Evolution. *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2(4), pp.63:1–63:1.