Towards Realistic SATD Identification Through Machine Learning Models: Ongoing Research and Preliminary Results

Eliakim Gama
University State of Ceará
Fortaleza, Ceará, Brazil
eliakim.gama@aluno.uece.br

Matheus Paixao
University State of Ceará
Fortaleza, Ceará, Brazil
matheus.paixao@uece.br

Mariela I. Cortés
University State of Ceará
Fortaleza, Ceará, Brazil
mariela.cortes@uece.br

Lucas Monteiro
University State of Ceará
Fortaleza, Ceará, Brazil
lucas.amaral@aluno.uece.br

ABSTRACT

Automated identification of self-admitted technical debt (SATD) has been crucial for advancements in managing such debt. However, state-of-the-arts studies often overlook chronological factors, leading to experiments that do not faithfully replicate the conditions developers face in their daily routines. This study initiates a chronological analysis of SATD identification through machine learning models, emphasizing the significance of temporal factors in automated SATD detection. The research is in its preliminary phase, divided into two stages: evaluating model performance trained on historical data and tested in prospective contexts, and examining model generalization across various projects. Preliminary results reveal that the chronological factor can positively or negatively influence model performance and that some models are not sufficiently general when trained and tested on different projects.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**.

KEYWORDS

Self-admitted Technical debt, Chronological Analysis, Machine Learning

ACM Reference Format:
Eliakim Gama, Matheus Paixao, Mariela I. Cortés, and Lucas Monteiro. 2024. Towards Realistic SATD Identification Through Machine Learning Models: Ongoing Research and Preliminary Results. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3663529.3663876

1 INTRODUTION

In the software development process, decisions favoring shortcuts often lead to Technical Debt (TD) accumulation. This notion was initially proposed by Cunningham in 1992 [2], associating TD with low-quality artifacts produced during the software development cycle to meet short-term demands. The literature offers various works [5] for TD accumulation management. Code debt, the primary type, arises from deadlines, competition, and cost pressures, leading developers to compromise code quality for short-term gains.

A commonly used approach to identify TD is analyzing comments in source code. Potdar et al. [4] conducted a study on SATD, where developers themselves acknowledge the existence of TD and express them through comments in the source code or other project management tools. Since then, several works have been dedicated to the automated identification of SATD [6] in various software artifacts, such as code comments, pull requests, commits etc.

Machine learning (ML) models, such as those utilized in identifying TD [1], aim to aid software teams in early detection and management of SATD. However, current literature experiments often employ a random selection process for training and testing, which doesn't accurately mirror real development environments. This methodology overlooks project-specific nuances and dynamics, potentially leading to outdated models ill-suited to modern practices and languages. Thus, using these models outside their original design contexts can greatly diminish their effectiveness.

Concerns arise about the generalization of SATD identification models beyond temporal data challenges. When developers lack sufficient data for training specific SATD identification models in new projects, they may resort to using datasets from previous or external projects. However, the effectiveness of these models trained with external data becomes uncertain, as they may struggle to capture the nuances of the new project context. This can significantly hinder their ability to accurately identify SATD.

This study aims to experimentally assess ML model performance when trained on historical data and tested in a future temporal context, reflecting developers' need for TD identification tools trained on diverse projects. Additionally, it seeks to analyze how well these models generalize to new projects by exploring the feasibility of training them with datasets from different projects. The study aims to address specific research questions:

(RQ1:) ***To what extent is the performance of the ML models affected when trained on historical data and evaluated in a future temporal setting?***

(RQ2:) ***To what extent is the generality of SATD models affected when trained and tested with cross-projects datasets?***

FSE Companion '24, July 15–19, 2024, Porto de Galinhas, Brazil

ACM ISBN 979-8-4007-0658-5/24/07
https://doi.org/10.1145/3663529.3663876

2 METHODOLOGY

2.1 Data Collection and Machine learning Models

In this research, we used the dataset from Maldonado *et al.* [3], a widely employed resource in SATD identification [6]. This dataset comprises ten open-source Java projects. However, this dataset does not contain chronological data regarding the insertion of SATD.

Therefore, to incorporate the temporal factor into our analysis, through bash commands, we supplemented the dataset by collecting insertion dates of source code comments, enriching it with temporal information. Only seven out of the ten projects had available chronological data for collection. The dataset used in this study includes the projects apache-ant-1.7.0 (Ant), argouml (Argo), emf-2.4.1 (EMF), hibernate-3.3.2 (Hibernate),jfreechart-1.0.19 (JFC), jruby-1.4.0 (JR), and sql12 (SQL). It spans from 1998 to 2021 and contains 37,414 SATD instances, i.e., source code comments and their labeled debt type. As an ML model for SATD identification, we used the two most used models to identify TD according to the systematic review carried out by Albuquerque *et al.* [1]. The selected models were Naive Bayes (NB) and Random Forest (RF).

2.2 Experimental Setup

2.2.1 Phase 1: Initially, we split each project's dataset into two temporal segments: a training set (70%) representing old data and a testing set (30%) representing newer data. The stratified division was done project-wise, with D1 (Train) containing older comments and D2 (Test) containing newer ones. This method improves model evaluation across different timeframes. We employ project-specific partitioning instead of random assignment to assess model performance within each project's timeline. We aim to reveal potential performance differences between training on historical data and testing on future data, illuminating temporal dynamics in software development. For comparison, we employ both **Random** (R) and **Chronological** (C) data partitioning for training and testing.

2.2.2 Phase 2: We aimed to assess the model's generalization across different projects, irrespective of chronological data order. To evaluate their adaptability, models were **trained** (Training) on one project's data and **tested** (Test) on another's. The percentage of data allocated for training and testing was standard, 70% and 30%, respectively. Results from this approach were compared to those from training and testing within the same project. By identifying and analyzing any differences in results, we gained insights into the models' effectiveness and generalization across diverse contexts.

3 PRELIMINARY RESULTS

(RQ1:) Due to space constraints, only a subset of results is presented. Table 1 shows the time gap between training and testing sets had varying impacts on different projects. For "argouml" and "sql12," this gap consistently decreased model performance metrics, while for "hibernate," it increased all metrics. These findings highlight the importance of considering the nature of data when choosing a validation strategy. Additionally, the RF model demonstrated good performance across all projects in random and chronological validation scenarios, indicating its robustness and suitability for classification tasks in diverse contexts. We believe that this difference is due to the fact that the majority of Hibernate data comes from the same year, making the data have a narrower chronological range. A more in-depth study of these nuances is necessary.

Table 1: Results with random and chronological data.

Project	Sample	Accuracy		Precision		Recall		F1 Score	
		NB	RF	NB	RF	NB	RF	NB	RF
Argo	R	90%	94%	88%	94%	90%	94%	88%	93%
	C	67%	91%	87%	89%	67%	91%	73%	90%
SQL	R	96%	98%	96%	97%	96%	98%	95%	97%
	C	86%	96%	95%	96%	86%	96%	89%	95%
Hibernate	R	85%	93%	81%	92%	85%	93%	80%	92%
	C	90%	98%	95%	97%	90%	98%	92%	97%

(RQ2:)

Analysis reveals various scenarios for model performance when trained on one project and tested on another, as shown in Table 2. In some cases, we observe a significant drop in algorithm performance, while performance remains relatively stable in others. The RF yields superior results compared to NB, regardless of the testing scenario, highlighting its robustness and generalization capability across different data contexts. These findings underscore the importance of careful model selection and testing strategy to ensure reliable and consistent results in practical applications.

Table 2: Results for cross-project data.

Trainig	Test	Accuracy		Precision		Recall		F1 Score	
		NB	RF	NB	RF	NB	RF	NB	RF
Ant	Argo	85%	72%	85%	78%	85%	75%	85%	79%
JR	EMF	98%	96%	98%	97%	98%	97%	98%	97%

4 FUTURE WORK

Our future goal is to evaluate existing models or tools for identifying SATD and apply our methodology to different datasets. The complete results are provided in our supporting package[1].

ACKNOWLEDGMENTS

This work received partial funding from CNPq-Brazil, Universal grant 404406/2023-8, and support from CAPES - Funding Code 001.

REFERENCES

[1] Danyllo Albuquerque, Everton Guimarães, Graziela Tonin, Pilar Rodríguez, Mirko Barbosa Perkusich, Hyggo O. Almeida, Angelo Perkusich, and Ferdinandy Chagas. 2023. Managing Technical Debt Using Intelligent Techniques - A Systematic Mapping Study. *IEEE Trans. Software Eng.* 49, 4 (2023), 2202–2220.

[2] Ward Cunningham. 1992. The WyCash Portfolio Management System. In *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum).* 29–30.

[3] Everton da S. Maldonado, Emad Shihab, and Nikolaos Tsantalis. 2017. Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt. *IEEE Trans. Software Eng.* 43, 11 (2017), 1044–1062.

[4] Aniket Potdar and Emad Shihab. 2014. An Exploratory Study on Self-Admitted Technical Debt. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014.* 91–100.

[5] Nicolli Rios, Manoel Gomes de Mendonça Neto, and Rodrigo Oliveira Spínola. 2018. A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information and Software Technology* 102 (2018), 117 – 145.

[6] Amleto Di Salle, Alessandra Rota, Phuong T. Nguyen, Davide Di Ruscio, Francesca Arcelli Fontana, and Irene Sala. 2022. PILOT: synergy between text processing and neural networks to detect self-admitted technical debt. In *TechDebt'22: International Conference on Technical Debt, May 17-18, 2022.* ACM, 41–45.

Received 2024-03-15; accepted 2024-04-26

[1] https://zenodo.org/records/11185473