Refactoring Debt: Myth or Reality? An Exploratory Study on the Relationship Between Technical Debt and Refactoring

Anthony Peruma
axp6201@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Eman Abdullah AlOmar
eman.alomar@mail.rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Christian D. Newman
cnewman@se.rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Mohamed Wiem Mkaouer
mwmvse@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Ali Ouni
ali.ouni@etsmtl.ca
ETS Montreal, University of Quebec
Montreal, Quebec, Canada

Abstract

To meet project timelines or budget constraints, developers intentionally deviate from writing optimal code to feasible code in what is known as incurring *Technical Debt* (TD). Furthermore, as part of planning their correction, developers document these deficiencies as comments in the code (i.e., self-admitted technical debt or SATD). As a means of improving source code quality, developers often apply a series of refactoring operations to their codebase. In this study, we explore developers repaying this debt through refactoring operations by examining occurrences of SATD removal in the code of 76 open-source Java systems. Our findings show that TD payment usually occurs with refactoring activities and developers refactor their code to remove TD for specific reasons. We envision our findings supporting vendors in providing tools to better support developers in the automatic repayment of technical debt.

1 Introduction

In 1992, Ward Cunningham coined the debt metaphor when explaining to product stakeholders the need to keep improving the quality of their software through refactoring [10]. Since then, *Technical Debt* (TD) has become a reference to any non-optimal code shipped to production with the promise of enhancing it in the next cycle, i.e., developers *owe* users better software in a future release. In this context, *Self-Admitted Technical Debt* (SATD) refers to developers deliberately admitting the existence of TD in their system by keeping track of it through inline documentation, which typically manifests in the form of source code comments that describe an anomaly of a code element (e.g., class, method, etc.) [15, 23].

Since SATD is a declaration of the need to update the system, several studies have analyzed its removal to understand better how developers address the issues mentioned in the comments, and consequently manage and reduce TD [12–14, 18, 26]. Yet, little is known about the extent to which refactoring contributes to the removal of the issues documented in the SATD comments. For example, in Listing 1, the comment mentions the need to rename a method name as per its current implementation. Therefore, the fix of this SATD was performed through the application of the *Rename Method* refactoring, where the method `getVertexName()` was renamed to `getName()`, with the commit message "Replace confusing names on Vertex API".

```
1 - public String getVertexName()
2 - // FIXME rename to getName()
3 + public String getName()
```

Listing 1: Code diff showing the renaming of a method due to its associated technical debt comment and the removal of the same comment [4].

1.1 Goal & Research Questions

The goal of this paper is to explore the co-occurrence of refactorings with the removal of SATD. We also distill the types of debt being addressed by extracting topics from the text of the comments. To this end, we answer the following Research Questions (RQs):

- **RQ1: To what extent do developers refactor code when removing technical debt?** This RQ gives us insight into the volume of occurrence of technical debt in our dataset and examines the types of refactorings that developers frequently apply.
- **RQ2: What motivates a developer to refactor their source code to remove technical debt?** Since SATD are indicators of shortcomings in the code, this RQ examines the repayment categories for which developers refactor their code.

2 Experiment Design

Figure 1 outlines the experiment for our study. In the following subsections, we describe the elements and activities that were part of our methodology. Our replication package is available at [1].

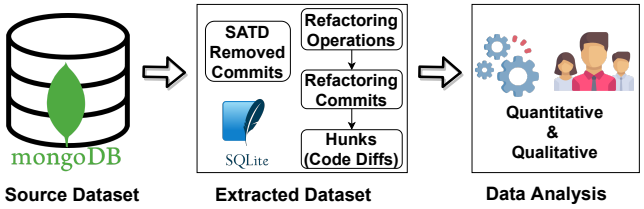

Figure 1: Overview of our experiment design.

2.1 Source Dataset

In this study, we utilize the *SmartSHARK MongoDB Release 2.1 (full version)* dataset [28]. The dataset contains the commit details of 77 open-source Java projects that are part of the Apache ecosystem and utilize GitHub as their project repository. Furthermore, the dataset also contains the changed hunks (including diffs) of each

Table 1: Summary of the extracted data.

Item	Value
Count of total projects	77
Count of commits in all projects	366,322
Count of projects with SATD removed	76
Count of commits with SATD removed	13,259
Count of refactoring operations	703,260
Count of commits with refactorings	67,582
Count of refactoring commits with SATD removed	7,341

file and utilizes RefDiff [25] and RefactoringMiner [29] to mine refactoring operations. Furthermore, the authors of the dataset indicate if a commit includes the removal of SATD by analyzing the code comment diffs in the source files associated with the commit. The authors examine the comment diffs for the presence or removal of the terms: "TODO", "FIXME", and "XXX".

2.2 Extracted Dataset

Since the source dataset contains an abundance of data, some of which is not related to our study, we built custom scripts to extract data pertinent to our study (i.e., commits, hunks, refactorings) from the source dataset into an SQLite database for analysis. First, we extract all commits with a label showing SATD removal. Next, we extract all refactoring operations. However, due to the use of two refactoring mining tools, there are duplicate operations in the source data. Hence, our next step removes all duplicates by comparing the refactoring descriptions. After that, we select all commits associated with a refactoring operation. Finally, we extract all hunks (i.e., code diffs) in files associated with the extracted refactoring commits. Table 1 is a summary of the extracted data.

2.3 Data Analysis

Our analysis of the extracted data follows a mixed-methods approach, where we collect and analyze both quantitative and qualitative data [27]. This approach presents us with the opportunity to provide representative samples from the dataset to complement our findings. Our quantitative approach utilizes well-established statistical measures and custom code/scripts on our dataset to report trends and patterns. In Section 3, we elaborate in detail on our analysis approach to answering each research question.

3 Experiment Results

In this section, we report on the findings of our experiments by answering our RQs. To answer our RQs, **we utilize SATD as a proxy for technical debt**, and hence, the removal of such comments indicates that developers removed technical debt code from the same source code file. For each RQ, we first explain the primary motivation(s) and the approach we undertake to produce the results, then we present our findings.

RQ1: To what extent do developers refactor code when removing technical debt?

RQ1 provides a deep-dive quantitative-based examination of refactoring operations co-occurring with technical debt removal. To this extent, this RQ looks into the volume of occurrence of technical debt in our dataset and examines the types of refactorings that developers frequently apply.

Table 2: Statistical summary of technical debt removal related to files in a commit and applied refactoring operations.

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Files in a refactoring commit having technical debt removed					
1	5	12	54.56	25	7593
Files in a refactoring commit without technical debt removed					
1	2	4	10.04	10	2315
Refactoring operations in a file with technical debt removal					
1	1	2	3.278	4	96
Refactoring operations in a file without technical debt removal					
1	1	1	2.774	3	161

Approach: A limitation of the source dataset is that it only provides the commit associated with a refactoring operation and not the actual source file to which the refactoring is applied. Therefore, our examination is limited to the commit level. First, we look at all occurrences of refactoring commits having technical debt removed, and then (to mitigate false positives) we narrow our analysis to only refactoring commits containing a single file with technical debt removed. We opt for this decision as it allows us to analyze cases where the removed TD and the refactoring are co-located in the same code. When analyzing commits with one file, we select two groups of refactored files– those that have technical debt removed and those that do not.

Analysis 1: Volume of refactoring and technical debt removal. First, we observe that 76 of the 77 projects in our dataset contain technical debt removal and the application of refactoring operations. We encounter 7,341 commits having refactorings and technical debt removal. This is around 55.37% of all commits labeled as having technical debt removed in the source dataset (with and without refactoring). As our study is on the relationship between technical debt removal and refactoring, we focus on the 7,341 refactoring commit instances. In contrast, 60,241 (or 89.14%) refactoring commits do not show the removal of technical debt. Our dataset contains 395 (or 5.38%) refactoring commits with only one file that exhibits the removal of technical debt. Conversely, there are 10,845 (or 18%) refactoring commits with only one file that does not have technical debt removed. Our analysis also includes measuring the association between debt repayment and refactoring by performing an odds ratio (OR) test for each of the 76 systems. All 76 systems have an OR > 1, with the value of 74 systems being statistically significant (i.e., p-value < 0.05), showing *a greater likelihood of debt repayment through refactoring activities.*

Next, we look at the volume files and refactoring operations. Since a commit can contain more than one file, an examination of commits shows the median number of files with technical debt removal in a refactoring commit is 12. In contrast, the median value in a refactoring commit without technical debt removal is 4. The complete statistical summary is in Table 2. Finally, we compare the differences between the number of refactoring operations in each file type (i.e., with and without technical debt removal). We utilize the non-parametric Mann Whitney U test as the data is not normally distributed. In this test, our null hypothesis (H_0) is that the distribution of refactoring operations for the two groups is equal. The results of the test yield p-value < 0.05, thereby rejecting H_0

Table 3: Distribution of top 10 refactoring operations on commits containing a single file with and without SATD removal.

Commit with one file with SATD removal				Commit with one file without SATD removal			
Refactoring Operation Type	**Total Count**	**Percentage**	**Avg. Operation Counts Per File**	**Refactoring Operation Type**	**Total Count**	**Percentage**	**Avg. Operation Counts Per File**
Extract Method	289	22.32%	2.16	Change Variable Type	6,519	21.67%	2.39
Change Variable Type	234	18.07%	2.27	Extract Method	6,188	20.57%	2.36
Rename Variable	141	10.89%	1.60	Rename Variable	2,648	8.80%	1.54
Rename Method	126	9.73%	1.66	Rename Method	2,473	8.22%	1.78
Rename Parameter	99	7.64%	1.90	Extract Attribute	2,459	8.17%	3.01
Extract Attribute	96	7.41%	5.05	Extract Variable	2,336	7.76%	1.32
Inline Method	58	4.48%	2.07	Change Return Type	2,054	6.83%	2.22
Extract Variable	54	4.17%	1.17	Rename Parameter	1,672	5.56%	1.81
Rename Attribute	51	3.94%	1.59	Rename Attribute	896	2.98%	1.43
Change Return Type	43	3.32%	1.48	Inline Method	591	1.96%	1.70
Other Operations	104	8.03%	*N/A*	*Other Operations*	2,248	7.47%	*N/A*

and shows that a difference between the population is statistically significant.

Analysis 2: Refactoring operations typically co-occuring with technical debt removal.

Shown in Table 3 are the frequently occurring refactoring operations developers apply to a file when removing SATD and also the operations that are frequent in source files where there is no removal of SATD. Comparing these two categories shows that the top ten refactoring operations are the same, even though the percentage frequency of occurrence for some differ. Furthermore, we observe that the files without technical debt removal undergo more types of refactoring operations; 32 types against 21. Looking at the average number of operations in a file, we observe developers frequently apply the *Extract Attribute* operation to the class followed by *Change Variable Type* and *Extract Method*. Next, we apply a Fisher's Exact Test to examine a statistically significant association between the average number of refactoring operations, of each refactoring type, applied to a file with and without technical debt. We utilize this test since we are comparing two categorical groups. In this test, variable independence is our null hypothesis (H_0). The results yield a p-value > 0.05, resulting in the acceptance of H_0. Therefore, showing the developers apply the different refactoring operations to improve the quality of the code regardless if the developer is addressing technical debt concerns in the code.

Summary for RQ1. The repayment of technical debt is an activity that developers often perform when refactoring code by applying a variety of refactoring operations, with *Extract Attribute* frequently applied multiple times in a file. Furthermore, there is a significant difference in the volume of refactorings developers apply when removing technical debt compared to general maintenance activities.

RQ2: What motivates a developer to refactor their source code to remove technical debt?

The previous RQ shows that removing technical debt is often associated with a refactoring activity. However, we lack context around the rationale. As SATD-related comments indicate shortcomings in the existing code, removing such comments is usually an indicator that developers correct the deficiency. Hence, an analysis of these comments provides insight into the issues the developer is correcting by performing one or more refactoring operations. To this end, this RQ constructs a grouping of *technical debt categories developers resolve through refactoring*.

Approach: First, we extract bigrams frequently occurring in SATD comments, following an approach similar to Peruma et al. [22]. We utilize bigrams as they provide more context for the terms and thereby help in reducing false presumptions. Additionally, we also extract the refactoring-specific terminology in these comments. Our extraction process involves programmatically examining each source file associated with a refactoring commit and extracting the set of removed SATD comments. These comments are available in the diff hunks of the source dataset. We then normalize the text using a series of pre-processing activities, including converting the text to lowercase, expanding contractions, removing non-alphabetic characters, digits, ASCII punctuation characters, and stopwords (standard English terms and terms specific to the dataset).

Table 4 shows the top ten frequently occurring bigrams in our dataset, while Table 5 shows the dataset's top ten frequently occurring (stemmed) refactoring terms. Our analysis of these extracted bigrams and terms results in us proposing the below categories as rationales for technical debt repayment via refactoring.

Error Handling: Improving error handling is a frequent area of technical debt that developers address when refactoring their code. This is evident by the frequent occurrence of the bigrams: 'catch block', 'error handling', 'exception handling', and 'throw exception'. Looking at the removed comments, we observe text such as "TODO: We could use a better strategy for error handling" and "TODO: Fix exception handling". This shows that developers knowingly write code prone to errors or utilize generic (or auto-generated) error handling and then make necessary corrections when refactoring their code in later revisions of the codebase. For example, in Listing 2, the developer removes code that always returns a null value.

```
4 public Object nextElement()
5 {
6     return null;// TODO: check exception handling
7 }
```

Listing 2: Removal of erroneous code [7].

Code & Structural Improvements: This category comprises of two sub-categories– 1) Clean-up Activities and 2) Design Improvements. Clean-up activities can include removing temporary code or renaming identifiers, such as in the case the developer renames an identifier (e.g., "TODO - Rename variable to commandId" [5]). From

Table 4: Top ten frequent bigrams in SATD comments

Bigram	Count	Percentage
catch block	1,037	6.03%
make thisdefault	89	0.52%
make sure	76	0.44%
get rid	63	0.37%
error handling	59	0.34%
pessimistic read	56	0.33%
exception handling	51	0.30%
throw exception	50	0.29%
implement needed	42	0.24%
add repository	40	0.23%
Others	15,637	90.91%

Table 5: Top ten frequent (stemmed) refactoring terms in SATD comments.

Term	Count	Percentage
add	74	20.32%
mov	66	18.13%
fix	51	14.01%
remov	48	13.18%
chang	26	7.14%
creat	25	6.86%
rewrit	11	3.02%
replac	10	2.74%
extend	6	1.64%
merg	6	1.64%
Others	41	%

prior studies [9, 20], we know that clean-up is an activity associated with refactoring. Design-level changes include the removing and moving code (such as moving methods, method extraction, etc.) and data types changes. Further, the bigram 'get rid' is mainly associated with code the developer needs to remove from the project. For instance, the comment "TODO get rid of this cast" is addressed by a Change Attribute Type operation [2]. From Table 5, the terms 'mov', 'remov', and 'chang' are indicators of design-level operations.

***Feature Updates**:* This category comprises of refactoring operations developers perform to incorporate feature changes in the system. For instance, the term'implement' is a placeholder for the developer to update the functionality, as in the example of "FIXME: our implementation is flawed...", in which the developer applies a Move Attribute refactoring in addition to a series of other code changes [3]. In another example, we encounter the removal of the comment "TODO: we can probably get a more efficient implementation..." with an Extract Attribute refactoring [6]. A similar placeholder term developers utilize is 'add', like in the example "TODO: Add method to extract...", which is resolved by an Extract Method operation combined with other changes [8].

Summary for RQ2. Analysis of SATD comments show that developers refactor to repay technical debt related to error handling, code optimization, and system features.

4 Threats To Validity

Though our dataset is limited to Java systems, the 76 systems are well established and widely utilized open-source systems. Furthermore, as we utilize an existing dataset, our analysis is limited to the data points contained within the source dataset. This includes the dataset authors' mechanisms/tools to identify SATD comments in the code (i.e., presence of the terms "TODO", "FIXME", and "XXX") and mine refactoring operations. Furthermore, since the dataset does not provide a straightforward mechanism to map SATD comments to an identifier, our analysis of SATD is limited to the file and commit level and not at the identifier level. That said, the features, volume, and age of data are more extensive and more recent than similar datasets [12, 16].

5 Discussion & Conclusion

As an exploratory study, our research aims to understand the extent of the relationship between technical debt repayment and code refactoring, and our preliminary findings show promise in further investigating these trends. Furthermore, we provide direction to research areas for supporting developers in designing and maintaining their code.

Our RQ1 findings confirm prior work showing that the removal of technical debt is frequently associated with refactoring actions [12, 19, 24]. In contrast, we observe differences in the types of refactoring operations that frequently occur. That said, when comparing the results from prior studies, such as [12] and [30], we observe a slight consensus between the reported frequent operations. While it can be argued that the cause can be due to the datasets and mining tools, further research in this area is warranted, especially since we did not observe a significant difference between the operations applied to remove technical debt and general refactoring.

Our RQ2 results provide interesting insight into the association of refactoring and technical debt repayment and support a further study to create a more formal and exhaustive set of causes. However, we see specific parallels with prior work. For instance, the category of repayment associated with features is also mentioned by [31], while we share code & structural improvements and error handling with [11]. Additionally, our categories also share similarities with the refactoring motivation taxonomy proposed by [9].

Below, we discuss how the findings from our RQs support the community through a series of takeaways.

Takeaway 1: Support for robust error handling. Our findings of developers improving error handling in their code is an opportunity for tool and IDE vendors to provide automated support for detecting such shortcomings in the code. Furthermore, developers should also understand that auto-generated try-catch blocks must be customized as per the project specifications.

Takeaway 2: Improving the accuracy of refactoring recommendation tools/models. With the repayment of technical debt frequently occurring with refactoring activities, refactoring recommendation tools/models, such as identifier renaming [17] and appraisal [21], can improve their accuracy by considering the occurrence of such SATD comments in their recommendation approach. Furthermore, there is a research opportunity for defining a standard set of composite refactorings to repay common types of debt since repayment usually involves applying complex changes [31].

References

[1] [n. d.]. https://doi.org/10.5281/zenodo.6345275.
[2] [n. d.]. agent/durability/NaiveFileWALDeco.java. https://github.com/apache/flume/commit/516e4da.

[3] [n. d.]. apache/struts2/components/Form.java. https://github.com/apache/struts/commit/840adfe.

[4] [n. d.]. apache/tez/dag/api/Vertex.java. https://github.com/apache/tez/commit/cfe9a42.

[5] [n. d.]. core/data/EntityIdentifier.java. https://github.com/apache/fineract/commit/892afd4.

[6] [n. d.]. repository/stats/DefaultRepositoryStatisticsManager.java. https://github.com/apache/archiva/commit/479f10c.

[7] [n. d.]. src/main/java/org/bouncycastle/asn1/x509/TBSCertList.java. https://github.com/apache/directory-kerby/commit/a475542.

[8] [n. d.]. util/featuregen/GeneratorFactory.java. https://github.com/apache/opennlp/commit/528033b.

[9] Eman Abdullah AlOmar, Anthony Peruma, Mohamed Wiem Mkaouer, Christian Newman, Ali Ouni, and Marouane Kessentini. 2020. How we refactor and how we document it? On the use of supervised machine learning algorithms to classify refactoring documentation. *Expert Systems with Applications* (November 2020), 114176. https://doi.org/10.1016/j.eswa.2020.114176

[10] Ward Cunningham. 1992. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger* 4, 2 (1992), 29–30.

[11] Georgios Digkas, Mircea Lungu, Paris Avgeriou, Alexander Chatzigeorgiou, and Apostolos Ampatzoglou. 2018. How do developers fix issues and pay back technical debt in the Apache ecosystem?. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 153–163. https://doi.org/10.1109/SANER.2018.8330205

[12] Martina Iammarino, Fiorella Zampetti, Lerina Aversano, and Massimiliano Di Penta. 2019. Self-Admitted Technical Debt Removal and Refactoring Actions: Co-Occurrence or More?. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 186–190. https://doi.org/10.1109/ICSME.2019.00029

[13] Martina Iammarino, Fiorella Zampetti, Lerina Aversano, and Massimiliano Di Penta. 2021. An empirical study on the co-occurrence between refactoring actions and Self-Admitted Technical Debt removal. *Journal of Systems and Software* 178 (2021), 110976.

[14] Valentina Lenarduzzi, Terese Besker, Davide Taibi, Antonio Martini, and Francesca Arcelli Fontana. 2021. A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. *Journal of Systems and Software* 171 (2021), 110827.

[15] Valentina Lenarduzzi, Antonio Martini, Davide Taibi, and Damian Andrew Tamburri. 2019. Towards surgically-precise technical debt estimation: early results and research roadmap. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*. 37–42.

[16] Valentina Lenarduzzi, Nyyti Saarimäki, and Davide Taibi. 2019. The Technical Debt Dataset. In *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering* (Recife, Brazil) *(PROMISE'19)*. Association for Computing Machinery, New York, NY, USA, 2–11. https://doi.org/10.1145/3345629.3345630

[17] Guangjie Li, Hui Liu, and Ally S. Nyamawe. 2020. A Survey on Renamings of Software Entities. *ACM Comput. Surv.* 53, 2, Article 41 (apr 2020), 38 pages. https://doi.org/10.1145/3379443

[18] Willian Oizumi, Ana C Bibiano, Diego Cedrim, Anderson Oliveira, Leonardo Sousa, Alessandro Garcia, and Daniel Oliveira. 2020. Recommending Composite Refactorings for Smell Removal: Heuristics and Evaluation. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*. 72–81.

[19] Boris Pérez, Camilo Castellanos, Darío Correal, Nicolli Rios, Sávio Freire, Rodrigo Spínola, and Carolyn Seaman. 2020. What Are the Practices Used by Software Practitioners on Technical Debt Payment: Results from an International Family of Surveys. In *Proceedings of the 3rd International Conference on Technical Debt* (Seoul, Republic of Korea) *(TechDebt '20)*. Association for Computing Machinery, New York, NY, USA, 103–112. https://doi.org/10.1145/3387906.3388632

[20] Anthony Peruma. 2019. A Preliminary Study of Android Refactorings. In *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 148–149. https://doi.org/10.1109/MOBILESoft.2019.00030

[21] Anthony Peruma, Venera Arnaoudova, and Christian D. Newman. 2021. IDEAL: An Open-Source Identifier Name Appraisal Tool. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 599–603. https://doi.org/10.1109/ICSME52107.2021.00064

[22] Anthony Peruma, Steven Simmons, Eman Abdullah AlOmar, Christian D. Newman, Mohamed Wiem Mkaouer, and Ali Ouni. 2021. How Do I Refactor This? An Empirical Study on Refactoring Trends and Topics in Stack Overflow. *Empirical Software Engineering* 27, 1 (23 Oct 2021), 11. https://doi.org/10.1007/s10664-021-10045-x

[23] Aniket Potdar and Emad Shihab. 2014. An exploratory study on self-admitted technical debt. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 91–100.

[24] Boris Pérez, Camilo Castellanos, Darío Correal, Nicolli Rios, Sávio Freire, Rodrigo Spínola, Carolyn Seaman, and Clemente Izurieta. 2021. Technical debt payment and prevention through the lenses of software architects. *Information and Software Technology* 140 (2021), 106692. https://doi.org/10.1016/j.infsof.2021.106692

[25] Danilo Silva and Marco Tulio Valente. 2017. Refdiff: detecting refactorings in version histories. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 269–279.

[26] Jie Tan, Daniel Feitosa, and Paris Avgeriou. 2022. Does it matter who pays back Technical Debt? An empirical study of self-fixed TD. *Information and Software Technology* 143 (2022), 106738.

[27] A. Tashakkori, C. Teddlie, and C.B. Teddlie. 1998. *Mixed Methodology: Combining Qualitative and Quantitative Approaches*. SAGE Publications.

[28] Alexander Trautsch, Fabian Trautsch, and Steffen Herbold. 2021. MSR Mining Challenge: The SmartSHARK Repository Mining Data. arXiv:2102.11540 [cs.SE]

[29] Nikolaos Tsantalis, Matin Mansouri, Laleh Eshkevari, Davood Mazinanian, and Danny Dig. 2018. Accurate and efficient refactoring detection in commit history. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 483–494.

[30] Ehsan Zabardast, Javier Gonzalez-Huerta, and Darja Šmite. 2020. Refactoring, Bug Fixing, and New Development Effect on Technical Debt: An Industrial Case Study. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 376–384. https://doi.org/10.1109/SEAA51224.2020.00068

[31] Fiorella Zampetti, Alexander Serebrenik, and Massimiliano Di Penta. 2018. Was Self-Admitted Technical Debt Removal a Real Removal? An In-Depth Perspective. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. 526–536.