Understanding the relationship between Technical Debt, New Code Cost and Rework Cost in Open-Source Software Projects: An Empirical Study

Judith Perera
University of Auckland, New Zealand
jper120@aucklanduni.ac.nz

Ewan Tempero
University of Auckland, New Zealand
e.tempero@auckland.ac.nz

Yu-Cheng Tu
University of Auckland, New Zealand
yu-cheng.tu@auckland.ac.nz

Kelly Blincoe
University of Auckland, New Zealand
k.blincoe@auckland.ac.nz

ABSTRACT

Making sub-optimal design decisions during software development leads to the accumulation of Technical Debt (TD) in software projects. There are tools to identify TD Items in software code through static code analysis. However, quantifying TD to support decision-making on whether to keep taking on TD or if it is time to refactor TD is a difficult task, and proposed approaches for this still lack consensus. Prior work observed that TD Interest could be further decomposed into constituents 'New Code Cost' and 'Rework Cost', which gives an interesting direction of research to explore TD quantification in terms of these costs. Therefore, through our empirical study, we plan to explore the relationship between TD, New Code Cost and Rework Cost in Open-Source Software Projects. This paper reports on an initial motivating study, our plan for future work and implications for researchers.

CCS CONCEPTS

• **Software and its engineering → Software design tradeoffs**;

KEYWORDS

technical debt management, mining software repositories

ACM Reference Format:
Judith Perera, Ewan Tempero, Yu-Cheng Tu, and Kelly Blincoe. 2023. Understanding the relationship between Technical Debt, New Code Cost and Rework Cost in Open-Source Software Projects: An Empirical Study. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE '23), June 14–16, 2023, Oulu, Finland.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3593434.3593490

1 INTRODUCTION

Software companies make sub-optimal decisions with respect to the design and implementation of a software product either deliberately or inadvertently [3]. These sub-optimal decisions lead to the accumulation of Technical Debt (TD) in the form of TD Interest. In prior work, Perera et al., [6] observed that the TD Interest can be further decomposed into constituents New Code Cost and Rework Cost. Through this work, we plan to study the concept of TD Interest in terms of New Code Cost and Rework Cost by examining their relationship with TD Items in open-source software code.

A TD Item is an artifact in the software code that represents a sub-optimal decision made with respect to the design of the software product. According to Avgeriou et. al., a TD Item could be for example, a Code Smell, Design Smell, or an Architectural Smell which can make future changes costly or impossible [1]. In this study, we consider 'Code Smells' as the TD Items under investigation with respect to our research question: ***RQ: Is there a relationship between TD Items, New Code Cost and Rework Cost?***

However, distinguishing between New Code Cost and Rework Cost is a non-trivial problem on its own. Therefore, we begin by examining what constitutes writing New Code and doing Rework — Added and Deleted Lines of Code. We refine the research question for our initial study as follows: ***RQ1: Is there a relationship between Code Smells, Added and Deleted Lines of Code?***

To conduct our study, we extract commit history data from GitHub, Issue Tracking data from Jira, and Code Smells from SonarQube[1] (a TD identification tool widely used in academia as well as industry). Then we perform statistical tests between the variables of our interest (See Section 3 Methodology). The novel contributions of this paper are, preliminary results of our on-going research, study design and methodology, plan for our future work and implications for researchers.

The rest of the paper is structured as follows. Section 2 summarizes the most relevant related work, Section 3 describes the study design and methodology. Section 4 reports preliminary results. Section 5 discusses the interpretation of results, threats to validity, future work, and implications for researchers. The paper concludes in Section 6.

2 BACKGROUND AND RELATED WORK

2.1 TD Interest Decomposition as 'New Code Cost and Rework Cost'

The decomposition of TD Interest into New Code Cost and Rework Cost for software code-related types of TD (Code, Design, and Architectural TD) was introduced in Perera et al., [6]. As reported by the authors, the concepts received multiple mappings from the analyzed quantification approaches in their Systematic Mapping Study. For example, 'Impact of god class on quality attributes (i.e., defect

EASE '23, June 14–16, 2023, Oulu, Finland

likelihood and change likelihood)' from one study and 'Penalty incurred by Debts' from another study were mapped to 'New code cost associated with TD' and 'Rework cost associated with TD'. The observations made by Perera et al., motivate an interesting line of research to explore TD Interest decomposition in terms of New Code and Rework costs. Their work is based on the Conceptual Model introduced in their study which identified New Code Cost and Reworks Cost as constituents of TD Interest. We perform an empirical study based on open-source data which will complement their work and suggest future research directions for researchers.

2.2 Empirical Studies investigating the relationship between TD and Lines of Code

2.2.1 Relation between TD Items and source-code changes. Sas et. al. [8], performed a study on 31 open-source Java systems to study if the frequency and size of changes correlated with the presence of Architectural Smells (ASs). They found that source-code artifacts such as files and packages that were affected by ASs changed more frequently than the artifacts that were not affected by ASs. They also found that the size of the change is significantly higher in smelly artifacts than in non-smelly ones. The metrics they employed in their study were: Change Has Occured (CHO), Percentage of Commits a Class has Changed (PCCC) and TACH which is also called the change size or code churn that is the sum of added, deleted and twice the changed lines. However, they did not study the different types of changes (added or deleted lines) individually. Furthermore, their study was limited to a selected set of ASs — Cyclic Dependency, Hub-Like Dependency, Unstable Dependency and God Component. Our study is different from theirs with respect to the study goal as well as in terms of the variables we study (more details in Section 3).

2.2.2 Relation between TD Density and New Code. Digkas et. al. [5], investigated the relation between the amount of TD Density of new code and existing code. They investigated whether additions, deletions, and modifications of methods are responsible for changes in the TD Density (the effort to remediate TD divided by lines of code — the remediation effort was measured by SonarQube). They propose writing clean new code as a complementary strategy to refactoring to reduce the TD density over time. However, their goal was different from ours, they studied the cleanness of the new code and if it has the potential to reduce TD. In our study, we examine the relationship between New Code Cost and Rework Cost associated with TD to determine if they can be constituents of TD Interest, i.e., the consequence of accumulating TD. Furthermore, we examine New Code and Rework with respect to different Issue Types such as New Features and Bugs. Bugs can be a form of extra work accumulated due to the presence of TD in the software system.

3 STUDY DESIGN AND METHODOLOGY

3.1 Research Question

Although our long term goal is to investigate the relationship between TD, New Code Cost and Rework Cost, that might require defining heuristics for New Code Cost and Rework Cost with respect to different granularities in software code (e.g., code level, design level, and architecture level). In this initial study, which is the first of a series of studies, we begin by examining the data as it is without defining heuristics for New Code Cost and Rework Cost. Therefore, we refined the research question for this study as follows: ***RQ1: Is there a relationship between Code Smells, Added and Deleted Lines of Code?***

As mentioned in Section 1, our selection of TD Items for this study are Code Smells reported by SonarQube[1]. However, in this initial study we do not investigate which files are specifically affected by the Code Smells and if they are prone to incur TD interest i.e., TD Interest Probability [2], this is to be incorporated in a future study. We rather focus on the difference of Code Smells between commits (Code Smells Diff) and the Density of Code Smells (Code Smell Diff divided by the total lines of code pertaining to that commit) in this study. We answer RQ1 by investigating if there is a relationship between Code Smell Diff and Code Smell Density with Added Lines and Deleted Lines of Code.

We followed recommendations given by Ralf et al. for repository mining to perform our study. Repository mining is, "a study that quantitatively analyzes a dataset extracted from a platform hosting of structured or semi-structured text (e.g a source code repository)" [7]. The units of analysis are source-code commits, Jira issues pertaining to the commits, and Code smells identified using SonarQube for the same set of commits. See Figure 1 for the Overview of the Methodology. We perform statistical tests on the dataset before and after classifying commits based on Issue Types obtained from Jira. Therefore, we define two sub-research questions to complement RQ1 as follows:

- ***RQ1.1: Is there a relationship between Code Smells Diff and Density with Added and Deleted Lines of Code?***
- ***RQ1.2: Is there a relationship between Code Smells Diff and Density with Added and Deleted Lines of Code for different Issue Types?***

3.2 Data Collection

3.2.1 Data Sources. We obtained data from 3 data sources: Commit history from projects on GitHub, Issue Tracker data from Jira, and Code Smells through static code analysis performed by SonarQube[1]. It is a tool used widely in academia as well as industry for the purpose of detecting Code Smells.

3.2.2 Tool Support. As illustrated in Figure 1, we utilize Pydriller — a Python Framework that supports mining software repositories [9] and SonarQube[1] as the main tools to help with the data extraction. The rest of the automation for data extraction and pre-processing tasks was done by writing Python scripts.

3.2.3 Selection of GitHub projects. We selected projects from the Apache Software Foundation as they have reputed standards regarding the maintenance of their projects as open-source software. They also maintain Jira as an Issue tracker and the commits usually refer to a Jira Issue Key, which means that we have access to issue tracker data pertaining to the commits. The criteria we used to select projects were: *the project should be active and last for a few years, should be written in the Java programming language, and should be of considerable size and complexity.* In this paper, we report on the initial study performed on one selected project — the Apache

[1] https://docs.sonarqube.org/

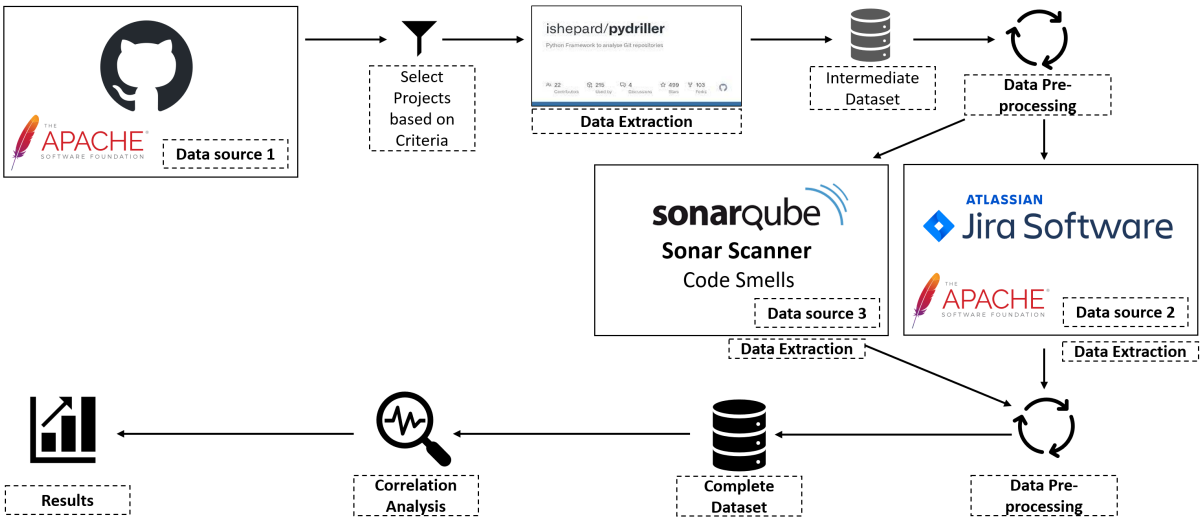

Figure 1: Overview of the Methodology

Hadoop Common Project[2] which is a collection of common utilities and libraries that support other Hadoop modules in the Apache Hadoop Framework.

3.3 Data Extraction, Pre-Processing and Creation of the Dataset

3.3.1 Data Extraction. Data that was gathered from each data source were compiled into a single dataset to be able to conduct the analysis. We first extracted the latest 500 consecutive commits for the Apache Hadoop Common Project at the time of conducting the study (February 2023). We then collected data related to the commits e.g., *commit hash, authors, the number of modified files, names of the files, type of modification, methods modified in those files, and lines of added and deleted code.* Then from Jira we extracted data such as the *Jira key, Issue Type, Issue Status, and Resolution.* Afterward, we extracted *Code Smells* for the commits under analysis via the SonarQube[1] tool by running the tool for each commit hash. The complete list of variables extracted from the different data sources can be seen in Table 1.

3.3.2 Data Pre-processing. Data was pre-processed at two stages prior to the analysis: once after extracting commits from GitHub and then prior to compiling the final dataset after extracting data from Jira and SonarQube. In the first pre-processing stage we removed the commit records that did not contain a Jira Issue Key since we had the need to classify commits based on the Issue Type in Jira — *New Feature, Bug, Improvement, Test, Wish, Task and Sub-task.* In the second stage of pre-processing, we aggregated the file-level data to the commit level, for example, the added and deleted lines of code. For Code Smells, we computed the Code-Smell-Diff which is the difference between the Code Smells in the code base for each commit that we analyzed. We also computed the Code Smell Density which is the Code Smell Diff divided by the Total Lines of Code that pertain to that commit. We then classified commits based on the Jira Issue Type. The motivation behind this was to understand if the relationships change depending on the Issue Type the developers worked on. We included commits where the linked Jira Issues were in Status 'Closed' with Resolution 'Fixed' and excluded any Duplicates.

3.3.3 Dataset. The complete list of variables in the dataset can be seen in Table 1. This includes the data extracted from the three data sources as well as data obtained via pre-processing (or aggregation).

3.4 Classification based on Issue Types

The commits in our dataset were classified according to the different Issue Types in Jira reported for the Apache Hadoop Common project (See Table 2). The highest percentage of Issues reported among the analyzed commits were Bugs. Although the main Issue Types were New Feature, Bug, Improvement, Test, Wish and Task, we also saw that some commits were done using the Jira Issue Key for sub-tasks as well. However, we focus on the main Issue Types: *'New Feature', 'Bug', 'Improvement' and 'Test'* in this paper. The reasoning behind this is to focus on the main development activities — Implementation, Bug Fixing, Improvements, and Testing.

3.5 Statistical Analysis

We performed statistical analysis to determine the relationships between Code Smell Diff and Code Smell Density with Added and Deleted lines of code for all commits and for different Issue Types after classifying commits based on the Issue Type (Results are discussed in Section 4, Tables 3 and 4). However, it is worth mentioning that we were interested only in the correlations and we did not intend to test causality in this study. First, we performed the Shapiro-Wilk test [4] to determine the Normality of our data. Since our data was not Normally Distributed, we then opted for the Spearman Rank Correlation test [4]. We interpret the correlations

[2]https://hadoop.apache.org/

Datasource	Variables
Github commit history/PyDriller	Commit Hash, Commit Message, Commit Author and Date, Parents, Num. of Modified Files, List of Mod. Files, per Mod. File: [Change Type, List of Modified Methods, NLOC, File Path, Added LOC, Deleted LOC]
Jira	Issue Key, Issue Summary, Issue Type, Issue Status and Resolution
SonarQube	Num. of Code Smells per Commit
Pre-processing	Num. of Modified Java Files, List of Mod. Java Files, per Mod. Java File: [Change Type, List of Modified Methods, NLOC, File Path, Added LOC, Deleted LOC], Issues with Status 'Closed' and Resolution 'Fixed, Code Smells Diff per Commit, Code Smell Density per Commit (Code Smell Diff/NLOC * 100)

Table 1: Variables in Dataset

Issue Type	Number of Commits	Percentage out of 359 Analyzed Commits	Min—Max Added LOC	Min—Max Deleted LOC	Min—Max Code Smell Diff	Min—Max Code Smell Density
New Feature	12	3.34	1 — 1702	0 — 147	(-6) — 29	(-0.00268) — 0.00994
Bug	**123**	**34.26**	0 — 921	0 — 131	(-10) — 13	(-0.01170) — 0.10811
Improvement	117	32.59	0 — 9318	0 — **13717**	(-23) — 62	(-0.04237) — 0.03396
Test	3	0.84	4 — 167	0 — 5	0 — 7	0 — 0.01666
Wish	2	0.56	1 — 28	1 — 2	0 — 0	0 — 0
Task	15	4.18	3 — 130	1 — 215	(-5) — 0	(-0.00267) — 0
Sub-task	87	24.23	0 — **11238**	0 —2709	(-38) — **550**	(-0.075) — **0.11017**

Table 2: Number and Percentage of Commits Classified based on Issue Types, Min and Max Added and Deleted Lines of Code, Min and Max Code Smell Diff and Code Smell Density | in *Italics* — Issue Types focused in this paper, in Bold — Highest or Max among all Commits | positive values indicate increase, minus values indicate decrease

as follows: 0–0.19 as negligible, 0.20–0.29 as weak, 0.30–0.39 as moderate, 0.40–0.69 as strong, and equal or greater than 0.70 as very strong [4]. The null and alternative hypotheses were formulated to test the relationship between Code Smell Diff and Code Smell Density with Added and Deleted lines of code. e.g., *Null Hypothesis (H0) — There is no relationship between Code Smell Diff and Added Lines of Code, Alternative Hypothesis (H1) — There is a relationship between Code Smell Diff and Added Lines of Code.* The results are considered statistically significant if the p-value is less than 0.05 (i.e., $p < 0.05$) implying that the null hypothesis can be rejected.

4 PRELIMINARY RESULTS

4.1 RQ1.1: Code Smell Diff, Density Vs Added, Deleted Lines (all Commits)

Table 3 shows the results obtained for the Spearman Rank Correlation [4] for the relationships between Code Smell Diff and Code Smell Density with Added Lines and Deleted Lines of Code, for all commits. Among the results obtained, all relationships had a *Negligible* correlation except for the relationship between Code Smells Diff and Added Lines of Code that had a *Weak (Positive)* correlation. Since $p < 0.05$, the correlation is statistically significant. Therefore, the null hypothesis can be rejected. This can be interpreted as, the difference of Code Smells (increase since positive correlation) being associated with the addition of lines of code to the code base and vice-versa regardless of the type of development by which code is being added to the code base, e.g., whether it is implementing new features, fixing bugs, doing an improvement, or writing new tests.

4.2 RQ1.2: Code Smell Diff, Density Vs Added, Deleted Lines (different Issue Types)

See Table 4 for the results obtained for the Spearman Rank Correlation [4] for different Issue Types (commits classified based on Issue types in Jira).

The *Strong, Positive* correlation between Code Smell Diff and Added Lines of Code for New Features can be interpreted as, the difference (increase) of Code Smells being associated with the addition of lines of code to the code base and vice-versa, during the activity of implementing new features. This can indicate that more Code Smells can be added to the codebase during the implementation of New Features with the addition of new code (increase of Added Lines). The association between Code Smells Diff and Added Lines can also be an indication that some extra work may be done by developers due to the presence of TD i.e., Code Smells in the code base resulting in more Added Lines in subsequent commits. However, this correlation is not statistically significant ($p > 0.05$). Therefore, the null hypothesis can not be rejected. This can be due to the nature of the project we have selected or due to the size of the dataset. We are interested to see results for this correlation for different GitHub projects and for a larger dataset in future work.

The *Strong, Negative* correlation between Code Smell Diff and Deleted Lines of Code for Features can be interpreted as, the reduction of Code Smells being associated with the deletion of lines of code to the code base and vice-versa, during the activity of implementing new features. This can indicate that the removal of Code Smells can happen during feature implementation while some code is being deleted due to the implementation. For Code Smell Density

Relationship	Spearman Rank Corr.	Strength	p-Value
Code Smell Diff — Added Lines	**0.21047**	**Weak (Positive)**	**0.00005**
Code Smell Diff — Deleted Lines	-0.14441	Negligible	0.00620
Code Smell Density — Added Lines	0.15512	Negligible	0.00325
Code Smell Density — Deleted Lines	-0.19596	Negligible	0.00019

Table 3: Spearman Rank Correlations between Code Smell Diff and Code Smell Density with Added Lines and Deleted Lines (all commits) | in Bold — p-Value < 0.05 (statistically significant result)

Issue Type	Relationship	Spearman Rank Corr.	Strength (Direction)	p-Value
New Feature	Code Smell Diff — Added Lines	0.49478	Strong (Positive)	0.10198
	Code Smell Diff — Deleted Lines	-0.41884	Strong (Negative)	0.17536
	Code Smell Density — Added Lines	0.27762	Weak (Positive)	0.38230
	Code Smell Density — Deleted Lines	-0.25808	Weak (Negative)	0.41801
Bug	Code Smell Diff — Added Lines	0.13035	Negligible	0.15072
	Code Smell Diff — Deleted Lines	-0.07983	Negligible	0.38010
	Code Smell Density — Added Lines	0.05021	Negligible	0.58129
	Code Smell Density — Deleted Lines	-0.11088	Negligible	0.22211
Improvement	**Code Smell Diff — Added Lines**	**0.36753**	**Moderate (Positive)**	**0.00004**
	Code Smell Diff — Deleted Lines	-0.05456	Negligible	0.56075
	Code Smell Density — Added Lines	**0.33016**	**Moderate (Positive)**	**0.00030**
	Code Smell Density — Deleted Lines	-0.09543	Negligible	0.30821
Test	Code Smell Diff — Added Lines	0.5	Strong (Positive)	0.66667
	Code Smell Diff — Deleted Lines	-0.5	Strong (Negative)	0.66667
	Code Smell Density — Added Lines	-0.5	Strong (Negative)	0.66667
	Code Smell Density — Deleted Lines	0.5	Strong (Positive)	0.66667

Table 4: Spearman Rank Correlations between Code Smell Diff and Code Smell Density with Added Lines and Deleted Lines (different Issue Types) | in Bold — p-Value < 0.05 (statistically significant result)

the correlations are *Weak, Positive*, and *Weak, Negative*, respectively for the relationships with Added Lines and Deleted Lines. The positive correlation between Code Smell Density and Added Lines indicates that the presence of Code Smells per line of code in the modified files in the commit is associated with the addition of lines of Code in the commit and vise-versa. The negative correlation between Code Smell Density and Deleted Lines indicates that the presence of Code Smells per line of code in the modified files in the commit is negatively associated with the deletion of lines of Code and vise-versa. However, since the results are not statistically significant the null hypothesis can not be rejected.

The *Moderate, Positive* correlation for the relationship between Code Smell Diff and Added Lines of Code and the relationship between Code Smell Density and Added Lines of Code for Issue Type 'Improvement' both indicate that there is a positive association between the variables (results are statistically significant since $p<0.05$). According to our dataset, the Issue Type 'Improvement' can indicate two types of improvements: improvements to Features and improvements to the code (i.e. Refactoring). During these improvements, either the introduction or the reduction of Code Smells could happen. For example, new Code Smells can be added to the codebase when improving Features. Although the intention of refactoring is to remove Code Smells, it can also possibly add new Code Smells. Hence, the positive correlations.

The *Strong, Positive* correlation between Code Smell Diff and Added Lines of Code for Tests can be interpreted as there is an association between the difference (increase) of Code Smells and the addition of Lines of code. Developers may need to write relatively more new Lines of Code to accommodate new Tests while there are more Code Smells. There can also be extra work for developers having to remove (delete) lines of code while there are more Code Smells as indicated by the *Strong, Positive* correlation between Code Smell Diff and Deleted Lines of Code. However, the number of commits classified as 'Test' Issue Type are only 3 in this dataset (See Table 2). Therefore, it is difficult to make conclusions based on a small sample size.

5 DISCUSSION

5.1 Results for Bug Issue Type

All correlations were Negligible for the Issue Type 'Bug' although Bugs were the most frequent in the dataset (123 commits). This can be explained using Table 2. See the Max values for Added and Deleted Lines of Code for the different Issue Types. Added Lines: for New Feature it is 1702, for Improvement it is 9318 and for Bug it is 921. Similarly, for Deleted Lines: for New Feature it is 147, for Improvement it is 13717 and for Bug it is 131. Usually, the amount of code changed during Bug fixes is less compared to the

amount of code changed during the implementation of features or improvements (i.e., the amount of code developers write or remove for Bug fixes is not as high as for New Features or Improvements). Therefore, Negligible correlations are expected for Bug Issue Type.

5.2 Interpretation of Results for RQ1

In this initial study, we explore the relationships between Code Smell Diff (increase or decrease of Code Smells between commits) and Code Smell Density (ratio of Code Smell Diff over Total Lines of Code pertaining to a commit) with Added and Deleted lines of code. The reasoning behind choosing to investigate Added and Deleted Lines of Code as a first study is because they are the building blocks of New Code and Rework. For any type of change made in the code either writing new code or doing rework will comprise of adding new lines of code or deleting existing lines of code. Furthermore, Git records the changes to software code in terms of added lines of code (+) and deleted lines of code (-).

Results for the Spearman Rank Correlation for all commits resulted in a *Weak Positive statistically significant correlation* between Code Smell Diff and Added Lines of Code indicating that Added Lines of Code are associated with the increase of Code Smells. Correlations for different Issue Types resulted in *Strong* correlations that are not statistically significant. However, *two correlations for issue Type 'Improvement' both Moderate and Positive were statistically significant* — the correlation between Code Smells Diff and Added Lines and the correlation between Code Smell Density and Added Lines. **These correlations indicate that there are associations between Code Smells Diff and Added Lines, and between Code Smell Density and Added Lines in our dataset**. These findings suggest that we should examine other GitHub projects as well as a larger sample of commits, perhaps spanning a longer time span to get better coverage of data.

5.3 Threats to Validity

We do not discuss the 'typical' threats to validity in this paper due to space limits and the study being on-going. We discuss an observation that might have affected our results. Although we collected and classified data for Issue Types New Feature, Bug, Improvement, Test, Wish, Task, and Sub-Task, in this paper we focused on the findings for the main Issues Types '*New Feature, Bug, Improvement and Test'*. We did not focus on the Issue Type 'Sub-task'. We understand that this choice might have affected the discussion of results as some developers might have committed their changes via sub-tasks instead of using the main Issue Key in the commit message even if their commit was pertaining to the main Jira Issue. This explains the highest Max values for Added Lines and Deleted Lines for Issue Type 'Sub-task' in Table 2. Our future work will consider linking sub-tasks to their main Issues.

5.4 Future Work and Implications for Researchers

In future work, we plan to expand our dataset by adding more projects following our project selection criteria given in Section 3.2.3. Furthermore, we plan to experiment with different heuristics for New Code and Rework and to incorporate the time spent working on an Issue reported in Jira so that we could compute the New Code Cost and Rework Cost in terms of effort spent by the developers. Thereby we can investigate if this could contribute towards the end goal of being able to make informed decisions regarding TD Management. Researchers can contribute to this goal by exploring different algorithms to compute the time or effort spent on New Code and Rework. Another way to extend our study is by defining New Code and Rework for different granularities e.g., design, architectural level, investigating the relationships with Design Smells and Architectural Smells. It is also worth examining which files are affected by the TD Items and incorporating TD Interest Probability (probability of an implementation being affected by a TD Item) [2].

6 CONCLUSION

We performed an empirical study on the commit history of an open-source software project analyzing commits, Jira Issues linked to commits, and Code Smells from SonarQube to examine the relationship between Code Smell Diff and Density with Added and Deleted lines of code. Preliminary results suggest that there is some association between Code Smells Diff and Density and Added Lines of Code as indicated by the statistically significant correlations for Issue Type 'Improvement' and between Code Smell Diff and Added Lines of Code for all commits. This shows that there is potential to continue this research goal.

REFERENCES

[1] Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, and Carolyn Seaman. 2016. Managing technical debt in software engineering (dagstuhl seminar 16162). In *Dagstuhl Reports*, Vol. 6.

[2] Sofia Charalampidou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Paris Avgeriou. 2017. Assessing code smell interest probability: a case study. In *Proceedings of the XP2017 Scientific Workshops*. 1–8.

[3] Ward Cunningham. 1992. The WyCash portfolio management system. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA* Part F1296, October (1992), 29–30. https://doi.org/10.1145/157709.157715

[4] Christine P Dancey and John Reidy. 2017. *Statistics without maths for psychology*. Pearson London.

[5] George Digkas, Alexander Chatzigeorgiou, Apostolos Ampatzoglou, and Paris Avgeriou. 2020. Can Clean New Code reduce Technical Debt Density? *IEEE Transactions on Software Engineering* 48, 5 (2020), 1705–1721.

[6] Judith Perera, Ewan Tempero, Yu-Cheng Tu, and Kelly Blincoe. 2023. Quantifying Technical Debt: A Systematic Mapping Study and a Conceptual Model. arXiv:2303.06535 [cs.SE]

[7] Paul Ralph, Sebastian Baltes, Domenico Bianculli, Yvonne Dittrich, Michael Felderer, Robert Feldt, Antonio Filieri, Carlo Alberto Furia, Daniel Graziotin, Pinjia He, Rashina Hoda, Natalia Juristo, Barbara A. Kitchenham, Romain Robbes, Daniel Méndez, Jefferson Seide Molléri, Diomidis Spinellis, Miroslaw Staron, Klaas-Jan Stol, Damian A. Tamburri, Marco Torchiano, Christoph Treude, Burak Turhan, and Sira Vegas. 2020. ACM SIGSOFT Empirical Standards. *CoRR* abs/2010.03525 (2020). arXiv:2010.03525 https://arxiv.org/abs/2010.03525

[8] Darius Sas, Paris Avgeriou, Ilaria Pigazzini, and Francesca Arcelli Fontana. 2022. On the relation between architectural smells and source code changes. *Journal of Software: Evolution and Process* 34, 1 (2022), e2398.

[9] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. Pydriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 908–911.