Exploring the Relationship between Technical Debt and Lead Time: An Industrial Case Study

Bhuwan Paudel, Javier Gonzalez-Huerta, Ehsan Zabardast, Eriks Klotins
Software Engineering Research Lab SERL,
Blekinge Institute of Technology, Karlskrona, Sweden
{bhuwan.paudel, javier.gonzalez.huerta, ehsan.zabardast, eriks.klotins}@bth.se

***Abstract*—Background: Software companies must balance fast delivery and quality, a trade-off that often introduces technical debt and wastes developer's time. Technical debt tends to increase as software evolves, which is assumed to slow down development and maintenance activities. However, the potential relationship between technical debt and lead time lacks empirical evidence.**

Objective: This paper reports an empirical study to explore the potential relationship between technical debt and lead time in resolving Jira tickets. We further aim to measure the extent to which technical debt can explain the variation in lead time.

Method: We conducted an industrial case study to explore this relationship in six components, each of which was analyzed individually. Technical debt was measured using SonarQube and normalized with the component's size. Lead times to resolve Jira tickets were collected from Jira and averaged monthly.

Results: The study found little to no correlation between technical debt and lead time to resolve Jira tickets in five components, with technical debt explaining a variation in lead time ranging from 0% to 41%. However, it is less than 30% in most of the components.

Conclusion: Technical debt alone does not fully explain the variation in lead time. There should be some other confounding variables (e.g., size and complexity of the changes, number of teams involved, priorities, component ownership) affecting lead time or a residual effect, i.e., interest, that might manifest later. Further investigation into those confounding variables is essential.

***Index Terms*—Technical Debt, Lead Time, Industrial Study, Case Study**

I. Introduction

The relentless demand for new features forces software development organizations to rethink their development practices. The delivery of high-quality software and support, as well as the fast value to the customer, should be one of the core priorities of software organizations. However, this can only be achieved if the software quality is maintained, significantly impacting the overall software development lifecycle [1].

Software companies usually invest their time and effort into refactoring and improvement activities to identify and remove technical issues that could potentially improve software quality [2], although these refactoring activities do not always succeed in removing technical debt items, but rather might introduce new ones [3]. Different studies (e.g., [4], [5], [6], [7], [8], [9], [10]) have reported that software development time is often wasted due to technical debt in terms of extra effort required for additional development and maintenance activities. Therefore, technical debt may directly affect software development by reducing development speed and increasing maintenance costs now or in the future [11].

In pursuing faster time-to-market and rapid delivery of new features and products, software companies have often deprioritized maintaining code quality [5], [12], preferring release speed. The consequences of this trend from the software quality perspective have yet to be explored [13]. Software systems must keep evolving and adapting to the environment [14]. However, doing so can increase complexity and decrease quality if not appropriately maintained [14], [15]. In addition, some empirical evidence (e.g., [16], [17]) reports an association that technical debt tends to increase over time due to the increased size and/or complexity with reduced quality of software systems. Consequently, managing technical debt is becoming a major task that plays a significant role in maintaining software quality [18]. When technical debt accumulates and code quality declines, the time to work on new features gets shortened, and the time required to implement them is extended because the code will be more complicated to change [19].

Manual approaches become unrealistic and painful as complexity and size grow more and more [20]. Data-driven approaches are promising, but we still need more evidence. A few studies (e.g., [2], [5]) have initiated a data-driven approach to investigate the impact of code quality and technical debt on the lead time. However, further exploration in other contexts, such as at different levels of abstraction and industrial evidence, is needed. In this paper, we take a step in providing such evidence by exploring the potential relationship between technical debt and lead time. To the best of our knowledge, this is the first case study on a proprietary, industrial setting on this topic, thus demonstrating industrial relevance while aligning with the research community's and practitioners' interests.

In this study, we define the lead time as the sum of time developers spend in different statuses associated with resolving Jira tickets, which includes issues resolution and developing new features (i.e., aggregated sum of time in progress, code review, and testing). This approach to measuring lead time differs from previous studies (e.g., [2], [5]), and we believe it helps to obtain the actual time developers spend resolving Jira tickets more accurately because technical debt can also hinder time spent during the code review and testing. Additionally, the technical debt measure was from SonarQube, normalized by the component's size to get technical debt density (TDD).

TDD is a metric that has been employed in several other studies (e.g., [16], [21], [22]).

Our results suggest that although technical debt, in theory, can be a factor that might have a certain relationship and affect lead time to some extent, it could not entirely explain the variation in lead time across all components studied. Moreover, our study raises concerns that other confounding variables might be at play, such as the size of the changes made, complexity, number of teams involved, and component ownership. These findings indicate the need for further investigation with more fine-grained analyses.

The rest of the paper is organized as follows. Section II explains the technical debt and lead time estimation process. Section III presents the related work. Section IV outlines the study design, from the research question to data analysis. Section V reflects the study's results, and Section VI provides a discussion. Section VII identifies threats to the validity of our research, and Section VIII concludes the study.

II. Technical Debt and Lead Time Estimation

A. Technical Debt (TD)

Technical debt (TD) is a metaphor widely used to reason about the consequences of poor software development practices and design decisions [19], [23]. Cunningham [24] initially introduced the concept of TD to discuss the potential long-term negative consequences of sub-optimal solutions adopted during software development. TD will eventually have to be repaid but generate interest in the long run [3], [25].

The survey results from the study [26] found that no single metric effectively identifies the leading indicators of TD [11]. Teams must select and adjust metrics based on context, as the existence of TD is context-dependent. Lenarduzzi et al. [2] mentioned that estimating TD with a precise value is difficult due to its complex phenomenon. However, it is possible to estimate several of its properties.

The use of static code analysis tools in the industry and academia for assessing code quality and understanding TD has been gaining traction [23], [27]. SonarQube[1] is one of the most used open-source static code analysis tools for measuring software code quality [28]. Furthermore, our partner company has been using it to assess code quality and leverage insights to enhance software quality for years. It is an automatic static analysis tool with several built-in rules. Details about the SonarQube rules are publicly available in the documentation [29]. Additionally, users can add own rules in case they lack built-in rules [30]. SonarQube systematically scans the source code to ensure it adheres to a specific set of coding rules, and its violations are reported as TD issues, using different metrics such as lines of code or code complexity [31].

SonarQube creates "issues" when it detects violations of predefined coding rules or quality gates. These issues are categorized as *code smells*, *bugs*, or *vulnerabilities*. Code smells symbolize issues related to maintainability, while bugs and vulnerabilities indicate reliability and security concerns, respectively [31]. It also classifies these issues based on their severity score from lowest to highest, such as info, minor, major, critical, and blocker.

Based on the type of rules violated, SonarQube estimates the remediation cost in terms of development time using the SQALE (Software Quality Assessment Based on Lifecycle Expectations) method [32]. This remediation cost represents the development time required to address the technical debt items (i.e., Sonar issues), and it may be constant for identified TD items for some rules [29].

We collected the Sonar remediation time for each issue as a metric for TD for the past two years (2022-2023). The TD per component is calculated by adding up the TD present in each file for the respective component. We normalized the TD per component with the corresponding component's size, called technical debt density (TDD), which is the independent variable for this study. The concept of TDD has been used in many previous studies, such as [16], [21], [22]. TDD enables us to reason about the evolution and the impact external factors might have on the accumulation of TD [21].

B. Lead Time

Jira[2] is an issue-tracking and agile project management tool for software development teams developed by Atlassian. Jira tickets include feature development and other software maintenance activities. These other software maintenance activities might consist of the resolution of issues such as bugs, code smells, or vulnerabilities that SonarQube might have identified or others directly created by developers or testers. However, as opposed to the study [2], which focuses only on the resolution of technical debt items (TDIs), we measure lead time for every Jira ticket, including the development of new features, as well as the aforementioned maintenance activities. It is important to note that Jira provides two types of time during tickets resolution [5]: First, cycle time (time from the beginning to the end of a specific action, i.e., time when tickets are marked as "In Progress" status until tickets are moved to "Done" status). Second, lead time covers the entire duration from receiving a request until its completion, including any waiting time. It also includes the "Waiting in the Backlog" time.

In addition, Jira provides the *Status Time Reports*, providing how much time is spent on each status. We based our lead time measurement on the status time report, which is how the company tracks lead time. As shown in Figure 1, lead time was calculated as the sum of time spent by developers in three different statuses associated with resolving each individual Jira ticket. This allows us to accurately track developers' time spent on actual development work by eliminating waiting time in the backlogs.

The lead time includes: **In Progress:** Total time the task (i.e., Jira ticket) is in development. **In Code Review:** Total time the task is in the code review process, as reported in Jira. **In Testing:** Total time the task is under test after it has gone through the code review process. We gathered all Jira tickets

[1]https://www.sonarsource.com/products/sonarqube/

[2]https://www.atlassian.com/software/jira

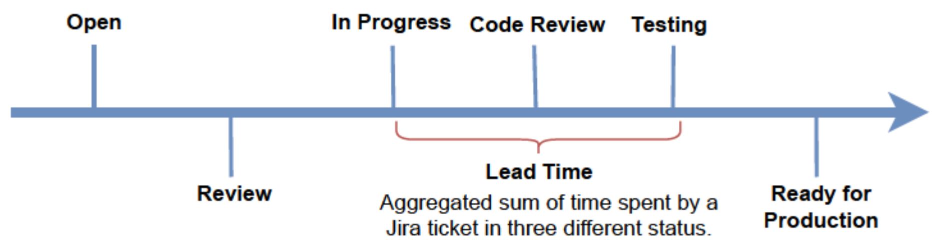

Fig. 1: Lead time calculation for Jira Tickets.

and lead time data for each component. We then created a map of the tickets and their corresponding lead time data using ticket-id. This helped us get the lead time data for all Jira tickets for specific components. We use this calculation for the lead time as it is the way the studied organization defines lead time and how it is reported in internal dashboards and indicators at the team level.

III. Related Work

Besker et al. [1] found a linear relationship between quality issues and the amount of time wasted. It means that every time developers face quality issues, more software development time is wasted. As mentioned in Tornhill and Borg [5], up to 42% of developers' time is spent managing TD, where most expenses are incurred due to the low-quality code costing software companies around $85 billion annually [33]. In a systematic literature review by Lenarduzzi et al. in Table 7 [7] reported that TD leads to developers' wasted time (effort), backed up by different studies (e.g., [6], [8], [9], [10]).

Participants in a study by Tom et al. [19] reported that one of the first impacts of TD noticed is a slowdown in development speed simply because developers need to spend more time refactoring TD issues. Liu et al. [34] suggest that the refactoring effort can be reduced by 17.64 to 20% when bad smells are detected and resolved in time. In addition, resolving Jira issues in code in the alert category takes longer time than in the healthy code, potentially doubling the time to market [5]. In contrast, TD sometimes can be used strategically by trading off quality with productivity, such as speeding up the development of new features, yielding the benefit of faster time to market [35], [36], and increasing the business value. However, if the interest rate of TD is too high, this strategy may become less effective or even detrimental. Therefore, software managers need to carefully weigh the potential benefits and interests of incurring TD when it comes to planning software projects [35].

Tornhill and Borg [5] investigated the relationship between code quality on a file level and the cycle time in development. They found that code quality significantly impacts development time. According to their findings, developers spend relatively more time resolving issues in low-quality source code. Their result shows that development time for Jira issues in the alert category code is, on average, 124% more than in the healthy category code and 78% more than in the warning category code. Poor source code quality can increase uncertainty in the time required to resolve issues. Conversely, high-quality source code can reduce this uncertainty [5], [37].

Similarly, Lenarduzzi et al. [2] initiated a data-driven approach to estimate the TD and its impacts on the variation in the lead time for resolving Jira issues with the aim of improving the estimation and prioritization of removing TD based on its impact. However, study [2] did not find a meaningful correlation between lead time and violations.

Our study addresses the gaps in previous studies. Unlike the study by Tornhill and Borg [5], we do not rely solely on 'time-in-development' as we think TD can also hinder time spent during the code review and testing process. Lenarduzzi et al. [2] focuses only on the resolution of technical debt items while we analyzed lead time for all Jira tickets. In the study [2], authors calculated lead time simply from the inducing and fixing commits' timestamps reported by SonarQube. First, this approach might be inaccurate since the fixing commit reported by SonarQube might be the merge commit squashing several other commits, and it is hard to distinguish whether this is the actual lead time to fix the TD item. The method we used to calculate lead time, particularly the time developers spent in three different statuses, as mentioned in Section II-B, provides a more accurate measurement technique for lead time and can yield better and more empirical evidence.

Finally, we conducted separate analyses for each component. This approach eliminates the influences of some other variables, such as the component's size and complexity, the component's owner, and other contextual factors that could arise while putting all components together.

IV. Methodology

A. Study Design

This study considers each component as an individual case and analyzes them separately. Figure 2 below illustrates our empirical study design based on the case study guidelines defined by Runeson and Höst [38].

Our goal is to explore whether a relationship exists between the TD in a specific component, as measured by a static analysis tool, normalized by the component's size, i.e., technical debt density (TDD), and the time taken to resolve Jira tickets associated with that component. Our primary focus is to derive the relationship between the amount of TD density

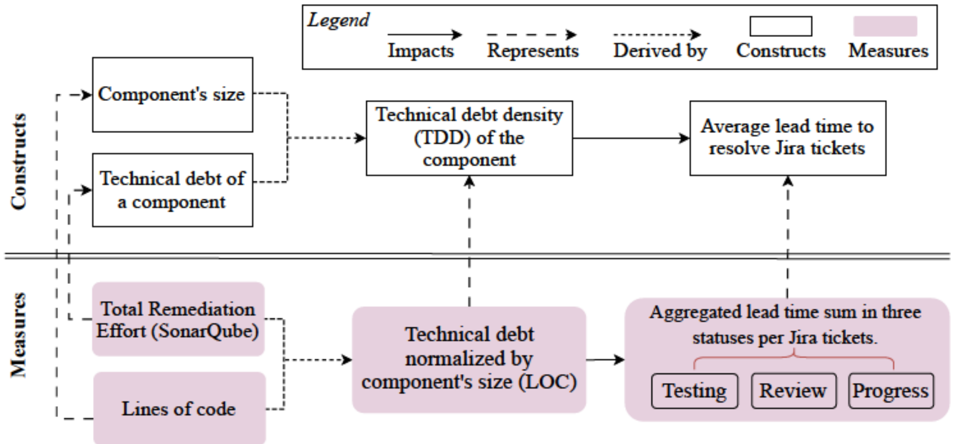

Fig. 2: Illustration of the study design.

in a component and the average lead time to fix a Jira ticket in that particular component. Based on this goal, this study investigates the following research question: **RQ:** ***How much of the variation in lead time to resolve the Jira tickets can be explained by technical debt?***

We hypothesize that, to some extent, there exists a correlation between TD present in a particular component at a given point in time and the average lead time required to resolve Jira tickets in that component at that point in time. If we can establish a correlation between these two variables, we should be able to determine how much of the variation in lead time can be explained by TDD alone. Therefore, we first examine the relationship between these two variables and then assess the strength of their relationship by calculating the R-squared value. This value explains the percentage of variation in lead time that TDD can explain. Furthermore, we plan to conduct a Granger Causality test to examine whether a causal relationship exists between TDD and lead time if we first find the correlation between them.

B. Context

We conducted this study with one of our industry collaborators, and the collected TD datasets are from SonarQube and lead time from Jira. The case has been chosen for convenience as the company has been using these tools to assess its internal quality metric since 2021. The company is a large and well-matured software development company that mostly develops products for financial and accountancy services (i.e., the Fintech area). Java is the primary programming language, and other languages such as Kotlin, TypeScript, and JavaScript are also used. For this study, we initially started with ten components and later excluded those with less than twenty months of active development in the last two years (2022-2023). The details of selected components for this study after exclusion are presented in Table I.

C. Data Collection

We collected data from two primary sources: SonarQube and Jira. The company has utilized these tools for years to track its metrics. TD data was collected from the Company's SonarQube endpoints at the beginning of each month for the past two years (2022-2023). The dataset includes information about violated rules, status, issue types, severity, size, complexity, number of violations, remediation effort (TD), and the files and components involved. Additionally, this data contains the timestamp of when the issue was created, the updated date, and the fixed timestamp if it has been resolved.

We strictly included sonar issues with an *Open* status as we intend to calculate the total TD present in a specific month and component. Additionally, we removed the sonar issues marked as *Resolved*, *Fixed*, *Closed*, and *False Positives* (which are set manually by developers). We then added the remediation effort (TD) of all sonar issues throughout the month in that component to get the total TD value (minutes) for the components over a specific month. SonarQube assigns the remediation effort to each sonar issue in terms of development time, which is the metric for TD. A detailed explanation of how SonarQube assigns the remediation time to sonar issues is provided in Section II-A.

TD was then normalized by the size of the corresponding components and multiplied by a thousand to get TDD per thousands of lines of code (TDD per KLOC), which is the independent variable in this study.

Although it would be possible to assess and perform the analysis on individual Jira tickets and the TD at that point in time, given the variability of individual data points, we opted for a higher-level aggregated view to capture general trends. Many other factors might explain the lead time for an individual Jira ticket, e.g., size, complexity, number of teams involved, ownership of the component, and other contextual factors such as task priority. Therefore, the role of TD on a single Jira ticket seems very far-fetched, although we plan to tackle this analysis in further work. So, considering the monthly average lead time value for Jira tickets for each component helps mitigate the biases of such factors as mentioned above since it will allow us to observe trends on TDD and lead time. In addition, we want to investigate the impact of

TDD on lead time from a bigger perspective, as both of them seem to be context-dependent.

TABLE I: Description of the selected components (The component's names are obfuscated to maintain anonymity).

Components	Individual Jira Tickets	Data Points (Monthly)	Avg. Monthly Lead Time (Days)	Avg. Monthly TDD (Min/KLOC)
Pangolins	78	23	9.18	187.45
Mongoose	98	22	3.41	96.77
Zorilla	211	24	4.14	59.45
Shark	194	22	3.82	57.27
Sloth	729	23	6.93	91.49
Penguin	157	23	7.73	269.49

We gathered all the Jira tickets for the years 2022 and 2023. We only consider this period because reliable lead time data became available at the company starting in early 2022 despite the components having a longer development history. Lead time data was collected separately from the Jira endpoint for each Jira ticket. This data includes the issue key, creation time, and the time developers spend in various statutes like time in progress, open, review, testing, and more. However, based on the company's way of tracking lead time, as mentioned in Section II-B, we calculate the lead time as an aggregated sum of time spent by tickets in *progress*, *code review*, and *testing*.

The dataset used in this study can not be disclosed publicly due to confidentiality and proprietary restrictions.

D. Data Analysis

We started our analysis by visualizing TDD and Lead time data to observe the nature of their distribution. Some of the Jira tickets were linked with multiple components, so we first removed those tickets as we analyzed each component separately, and it would then be easy to attribute due to which component the lead time is introduced. In addition, a few of the lead time data points were notably high. The reason behind those long lead times could be low-priority tickets, controversy, long code-review processes, pull-request re-scoping, and rejections. We discussed these extreme cases with the development team, and upon their recommendation after review, we treated them as outliers and removed them.

Figure 3 and 4 show that both the TDD and lead time data distribution are positively skewed to one-tail, even after removing outliers. We carried out a Shapiro–Wilk test [39] to confirm the normality of data distribution. We then tried to transform the data into a more normally distributed form by applying the square root transformation and re-evaluating normality after transformation using the Shapiro-Wilk test. We also plotted the TDD per thousand lines of code versus the monthly average lead time to resolve Jira tickets for each component, as shown in Figure 5, with each figure corresponding to a different component.

The correlation between the two variables can be measured using different correlation tests, such as Pearson, Spearman, and Kendal correlation coefficient, which were also adopted in a recent study [2]. Pearson correlation is a statistical measurement of the strength and direction of linear correlation between two variables [40]. Spearman and Kendall's rank correlation are non-parametric correlation metrics [41] that measure the monotonic (but not necessarily linear) relationship between the two variables [42]. Spearman correlation measures the degree of monotonic association between two variables, and Kendall correlation compares the ranking of the values between the variables [2], [42]. However, due to the non-normal distribution nature of our data and the expected monotonic relationship between TDD and lead time, we chose to perform the Spearman correlation test for this study. We also calculated the p-values to check whether the correlations were statistically significant. If the p-value is less than 0.05, the null hypothesis should be rejected, indicating that the correlation is statistically significant or vice versa.

Correlation coefficient [43] ranges in the interval $[-1, +1]$, where 0 indicates no linear relationship and $+1$ indicates a perfect positive linear relationship, which means that as one independent variable increases, the dependent variable also increases. On the other hand, -1 indicates a perfect negative linear relationship, meaning that the dependent variable decreases as the independent increases. Values between 0 and 0.3 (0 and -0.3) show a weak positive (negative) linear relationship. Likewise, values between 0.3 and 0.7 (-0.3 and -0.7) indicate a moderate positive (negative) linear relationship, while values between 0.7 and 1.0 (-0.7 and -1.0) suggest a strong positive (negative) linear relationship [43].

In order to answer our research question and test hypothesis, we then calculate the coefficient of determination, also called R$-$squared (R^2) [43]. The R$-$squared value indicates the percentage of variation in the dependent variable (lead time) explained by the independent variable (TDD). In addition, we also tried to fit different polynomial regression models with order 1,2,3 and calculated the RMSE (root mean squared value) for each order. The fitted models are visualized for each component and illustrated in Figure 6.

We further plan to explore the potential temporal relationship between the TDD and lead time by conducting a Granger Causality test [44] if we first observe the significant correlations between TDD and lead time. Granger Causality is a statistical method to assess the dependence relation between temporal time series that checks whether past values of one variable can be used to predict the future value of another variable [45]. If the p-value is less than 0.05, it rejects the null hypothesis, indicating strong evidence that one variable causes the changes in the other and vice versa.

V. Results

We began our analysis by gathering all Jira tickets for 2022-2023. The total number of Jira tickets over that period was 17,544. After removing Jira tickets that were linked with multiple components, 15,592 tickets were left. We started with ten components for this study; their total Jira tickets were 1735. Out of those components, we further analyzed only six that met our selection criteria (i.e., having more than twenty months of active development in the last two years).

TABLE II: Results of the different tests performed in this study.

Component's Name	Spearman's Correlations	P-Values for Correlations	R-squared (R^2) Values			RMSE			Shapiro-wilk test for Lead Time
			Linear	Quadratic	Cubic	Linear	Quadratic	Cubic	
Pangolins	0.08	0.73	0.00	0.07	0.11	6.29	6.04	5.94	0.09
Mongoose	0.18	0.43	0.08	0.29	0.41	2.29	2.01	1.83	1.01×10^{-5}
Zorilla	0.66	0.00047	0.31	0.36	0.37	1.95	1.88	1.85	2.33×10^{-5}
Sloth	-0.28	0.19	0.17	0.26	0.27	1.14	1.08	1.07	5.06×10^{-6}
Shark	-0.48	0.02	0.12	0.12	0.13	1.80	1.80	1.78	0.0010
Penguin	-0.16	0.46	0.04	0.05	0.05	5.81	5.78	5.78	3.72×10^{-8}

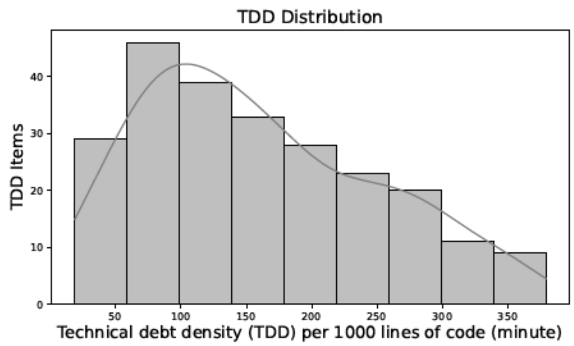

Fig. 3: TDD distribution for selected components.

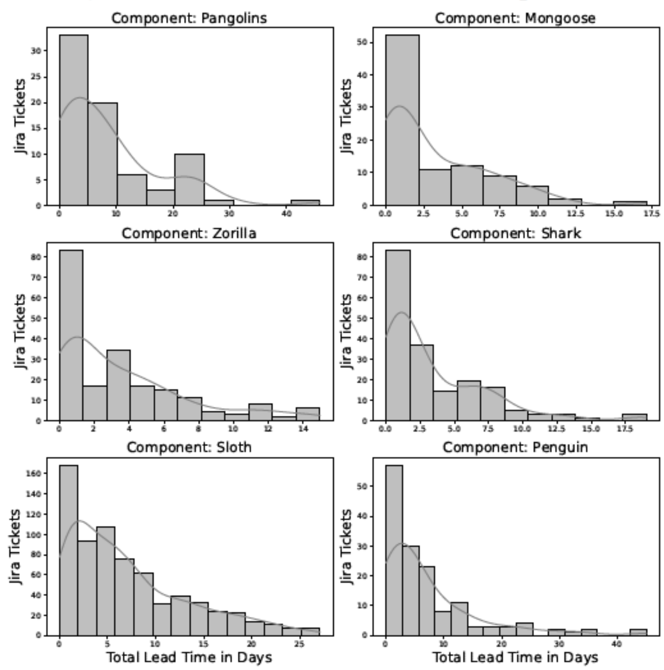

Fig. 4: Lead time (in days) distribution after removing outliers.

Figure 3 and the Shapiro-Wilk test with a very small p-value of $6.467 \times e^{-7}$ show that the TDD data does not follow a normal distribution. At the same time, lead time data had some outliers, so we removed the top five percentile of lead time data points as explained in Section IV-D. We then checked the distribution of lead time data, which was also not normally distributed as tested by the Shapiro-Wilk test and as shown in Figure 4. So, we first transformed both TDD and lead time data into square root. However, data still remained non-normally distributed, as tested by the Shapiro-Wilk test. The p-values for TDD (0.0035) and lead time data (see Table II) were significantly less than 0.05, leading to the rejection of the null hypothesis. Hence, we opted to work with non-transformed data to maintain the readability of results in the figures and the nature of data distribution.

Before calculating the correlation coefficient, we plotted each component's TDD and lead time data. Figure 5 illustrates the graphical representation of two variables over two years. After visualization, we calculated the Spearman correlation coefficient to test the correlation and association between the two variables for each component. Three out of the six components showed positive correlations, while the others showed negative correlations. However, only Zorilla showed a moderate positive correlation with a correlation coefficient of 0.66, which is statistically significant with a low p-value of 0,00047 that rejects the null hypothesis. The Pangolins and Mongoose indicated weak positive correlations, yet they are not statistically significant, as indicated by their high p-values. On the other hand, Sloth, Shark, and Penguin showed negative correlations. The correlation coefficients and their corresponding p-values for all components are given in Table II.

Once we calculated the correlation coefficients, we conducted the regression analysis to determine the percentage of variation in lead time that TDD can explain. To do this, we calculated the R-squared value, also known as the coefficient of determination for different orders 1,2 and 3. The R-squared value ranges from 0 to 1. A value of 0 means none of the variation is explainable, while a value of 1 means 100% of the variation is explainable. The scatter plot of TDD and lead time for each component, along with the fitted lines for each order, were then visualized and shown in Figure 6. Additionally, we calculated the root mean squared (RMSE) value for each order to decide which order fits the most. A higher R^2 with a smaller RMSE value indicates that the TDD can explain more variation in lead time.

Table II indicates that the R-squared values for linear and quadratic models for the component Pangolins were very low. Moreover, the RMSE value is high for both regression models. However, the R-squared value was 0.11 with 5.94 RMSE for the cubic regression. Similarly, for the components Mongoose and Zorilla, R-squared values for linear regression were 0.08 and 0.31, with RMSE values of 2.29 and 1.95, respectively. In contrast, cubic regression had values of 0.41 and 0.37, with RMSE values of 1.83 and 1.85, respectively. Despite negative correlations, the other three components (Sloth, Shark, and

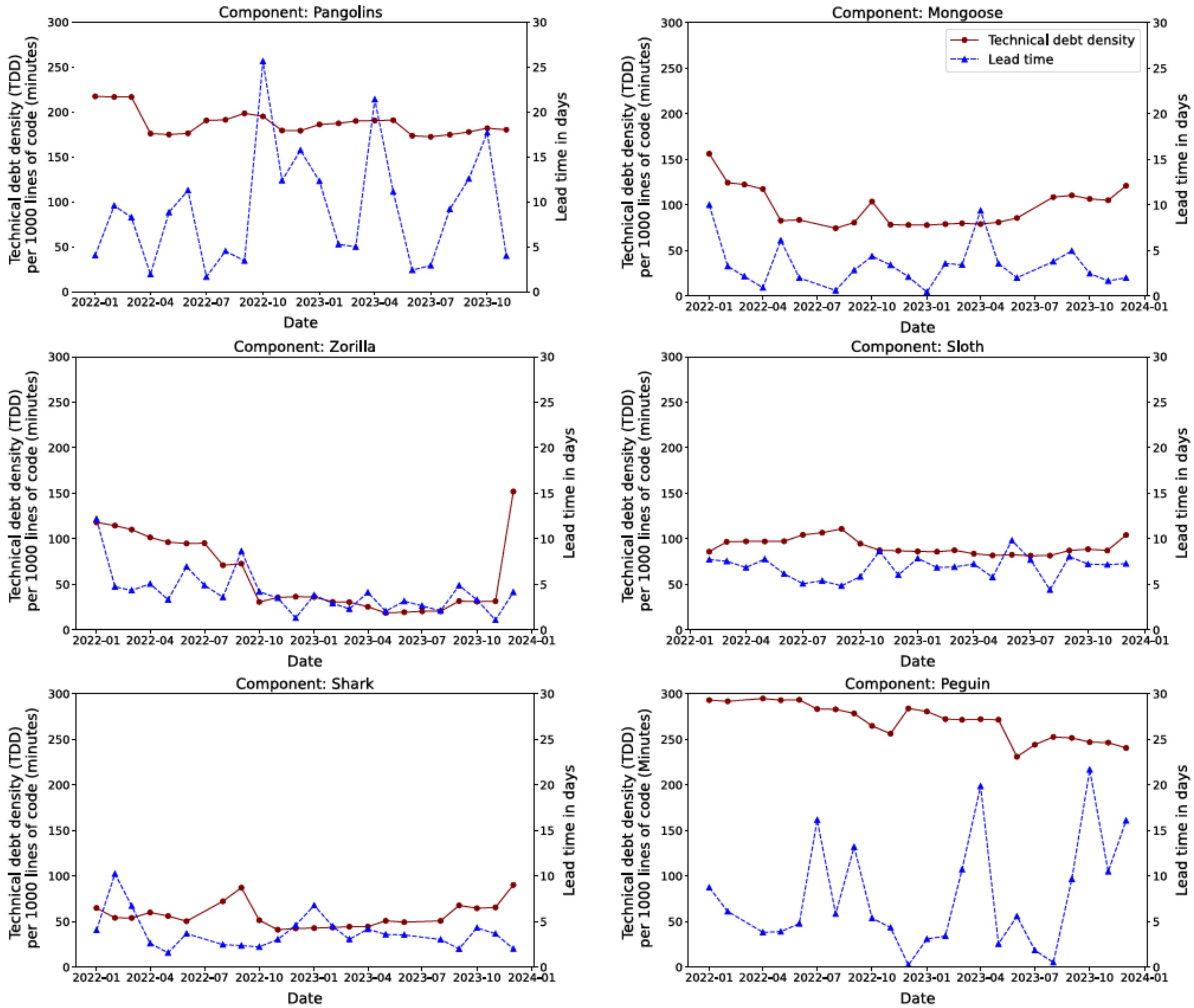

Fig. 5: Technical debt density (TDD) per KLOC (minutes) vs average monthly lead time (days) for the components. The legend on the top-right graph applies to all.

Penguin) had R-squared values of 0.17, 0.12, and 0.04 for the linear regression model, while the cubic regression values were 0.27, 0.13, and 0.05, respectively. Further detailed results can be found in Table II.

Finally, conducting a causality test to examine any causal relationship between the TDD and lead time seems inappropriate in this case study, as we found no significant correlations between them in five of the six components studied.

VI. Discussion

We answer our RQ by investigating the potential relationship between TD and lead time to resolve the Jira tickets. We further determine the extent to which technical debt density can explain the variation in lead time. To answer this question, we analyzed six individual components of a large-scale company separately.

We conducted a quantitative analysis and found little to no significant correlations between TDD and lead time in five out of six components, which aligns with the finding of a quantitative study [2]. Only one component “Zorilla” showed a significant correlation with a correlation coefficient of 0.66 and an R-squared value of 0.37, indicating that TDD can explain up to 37% of the variation in lead time in this particular component. Overall, this shows that TDD alone does not seem to affect lead time. Companies may be able to overlook the TD, which might not impact your lead time right now, but you may have to address that debt later, or it could result in suboptimal system behavior [2].

Lenarduzzi et al. [7] in their review among other studies (e.g., a review by Tom et al. [19]) report that TD contributes to developer’s wasted time, leading to increased lead time. This claim is based on a number of studies: [6], [8], [9], [10]. Interestingly, these studies mainly focus on the subjective assessment of TD and its impact. Studies focusing on a quantitative assessment like ours and Lenarduzzi et al. [2] do

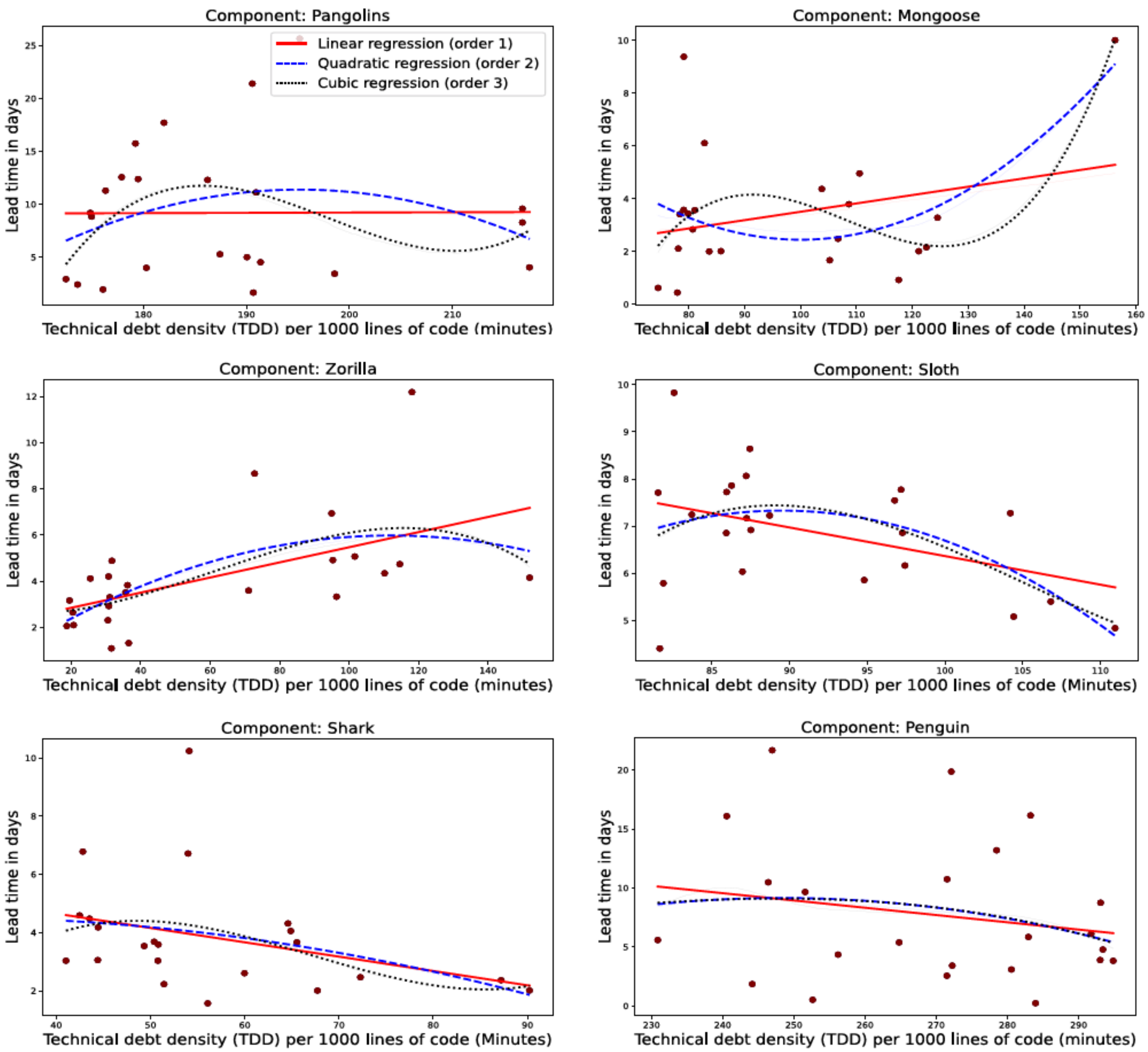

Fig. 6: Scatter plot of the technical debt density (TDD) per KLOC (minutes) and average monthly lead time (days) for each component. The lines represent the different regression models for order 1,2,3. The legend on the top-left graph applies to all.

not show the same relationship between TD and increased lead time.

The reason behind this contradiction among two different types of studies or not being able to find significant correlations between TD and lead time in our study could be:

- The subjective biases of study participants could inflate the relationship between TD and lead time as both variables are context-dependent.
- There might be an assessment bias between the static analysis tools such as SonarQube and practitioners. Practitioners may have a broader interpretation of TD than what SonarQube can capture. For example, sonarQube might not be able to address practitioner-relevant dimensions of TD, such as architectural debt or team-specific challenges. This can be handled by conducting surveys or interviews with developers involved or using alternative tools.
- The quantitative data might not be granular enough to discover the relationship. The analysis at a file or code level would yield a better result.
- Current TD may impact lead time in the future (with a lag), as we could not associate TD as the sole factor impacting lead time in the same month. It means that TD might have a residual effect that could manifest later. Conversely, lead time could also be a source of TD or compromised code quality. For example, software developers might use sub-optimal solutions to be faster. Further studies should examine time-lagged effects using

advanced statistical techniques like cross-correlation or time-series analysis.

- There might be other confounding variables affecting this relationship, such as priority, functional content, team ownership, size, and complexity of changes, and teamwork. In addition, other factors, such as whether a given Jira ticket is handled *internally* by the team owning the component if it is dealt with by another team different from the owner or if it is a joint effort by different teams.

Our findings also provide insight into the relationship between TDD and lead time in resolving Jira tickets, which is either very weak or nonexistent if TD and lead time are aggregated monthly. TDD vs lead time is not always explainable and positive but somewhat inexplicable and negative in some components. Furthermore, the R-squared and RMSE values indicate that this relationship is more cubic than linear across all components studied, as shown in Table II. This suggests that the relationship between TDD and lead time is complex as it varies better in cubic rather than linear ways.

Hence, our results suggest that further exploration in various contexts or across different components with more granularized data, such as a file or code level, is necessary to understand this relation. Furthermore, it would be worthwhile to investigate how the type of analyses reported in this paper can be done with alternative methods for measuring TD beyond static analysis tools such as SonarQube, as well as how to handle the false positive issues. In addition, conducting qualitative investigations, such as interviews or surveys with employees, might provide valuable context and insights into how technical debt affects developers' time in practice. This study paves the way for further research into investigating the impact of such factors on lead time.

VII. Threats to Validity

Our study, like other empirical studies, is subject to validity threats. This section discusses threats to validity from four aspects as defined by Runeson and Höst [38]: construct validity, internal validity, external validity, and reliability.

Construct validity refers to the extent to which measures studied in the study represent what researchers intend to measure and what is being studied in accordance with the research question [38]. To calculate the TDD, we relied on SonarQube, which is one of the most used and highly adopted tools to measure the TD [27], [28] and has been used for some other related studies such as the study [2]. We have normalized TD by the component size, eliminating the potential impacts of external factors such as size. Moreover, we have only considered the TD for the issues marked as *Open* and removed other issues marked as *Resolved*, *Fixed*, *Closed*, and *False Positives*. Developers manually have to mark false positive issues themselves. This ensures that we obtain an accurate representation of the technical debt, as reported by SonarQube, at the start of each month. However, we acknowledge that some False Positives issues may have been mishandled. The lead time for each ticket is calculated based on the time developers spend on three statuses in Jira: time in progress, code review, and testing. This approach assures that we only considered the actual time developers spent resolving a Jira ticket. We then calculated the average lead time for Jira tickets in a month, eliminating the threat of known and unknown external factors affecting lead time, such as size, complexity, and ownership of the changed code.

Internal validity concerns the possibility of other factors affecting the main factors being studied. In this study, the major concern regarding internal validity is the correlation between the lead time and the TD. We have found a mixed correlation ranging from weak to moderate for different components. However, we acknowledge that several confounding variables we could not consider in this study might influence this correlation. We tried to mitigate this threat by normalizing TD and averaging lead time on a monthly basis. Nevertheless, further studies should be conducted to investigate the impact of such internal and external factors.

External validity concerns the generalizability of findings. This case study is based on a large-scale financial product development company and may not represent the entire software development landscape. The main programming language used for the studied components is Java. SonarQube provides a unique set of TD issues for each programming language. Moreover, we do not rely solely on statistical generalization when interpreting data. Instead, we analyzed and discussed each component separately.

Reliability is concerned with reducing the errors and biases in a study, ensuring data and analysis are independent of the specific researchers. In order to mitigate this threat, we had multiple discussions with our partner company regarding the data collection, analysis, and interpretation of the results. We reported our results using descriptive statistics and correlation models. Additionally, we also applied polynomial regression with order 1,2,3. We used standard Python packages to perform all statistical analyses. Furthermore, we engaged in pair programming and discussions to ensure the quality of our findings and increase our confidence in the results.

VIII. Conclusion and Future Work

This study explores the relationship between TDD and lead time for resolving Jira tickets (feature development and other software maintenance activities) and the extent to which the TDD explains the variation in lead time to resolve Jira tickets. The exploration of this relationship allows us to identify components that are most affected by TDD in terms of lead time, thus enabling better prioritization and informed decisions for software maintenance.

In this work, we analyzed six components to study the impact of TD (measured as TDD) on lead time to resolve the Jira tickets in an industrial case. We did not observe a significant correlation, and TDD alone can not explain all the variations in lead time in most of the components studied. So, the results indicate that TD might not be the only factor that affects the lead time. However, the TD introduced in a given month may affect lead time in the future, i.e., TD, as expected, might have an accumulative impact that could manifest later, as *interest*. Future studies should examine time-lagged effects using advanced statistical techniques like cross-correlation or time-series analysis.

Additionally, there could be other confounding factors that influence lead time. However, exploring those factors does not seem feasible at the moment because the company started collecting data with traceability between Jira tickets and code commits after we started our analysis. Therefore, we plan to conduct further research to investigate other potential confounding factors that might affect this relationship, such as the number of affected lines of code for a Jira ticket, who owns the code and who made the change, the complexity of the changes made, functional content, team dynamics, and task priorities. We are currently exploring such variables.

We also plan to explore this relationship at a more granular level, such as the file level, by including some of the above-mentioned potential confounding variables. Furthermore, incorporating developers' perspectives or organizational factors through interviews or surveys would provide valuable qualitative insights.

ACKNOWLEDGEMENT

This research was supported by the KKS Foundation through the KKS SERT Research Profile project (Ref. 2018010).

REFERENCES

[1] T. Besker, A. Martini, and J. Bosch, "Time to pay up: Technical debt from a software quality perspective." in *CIbSE*, 2017, pp. 235–248.

[2] V. Lenarduzzi, A. Martini, N. Saarimäki, and D. A. Tamburri, "Technical debt impacting lead-times: An exploratory study," in *2021 47th euromicro conference on software engineering and advanced applications (seaa)*. IEEE, 2021, pp. 188–195.

[3] E. Zabardast, J. Gonzalez-Huerta, and D. Šmite, "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)*. IEEE, 2020, pp. 376–384.

[4] T. Besker, A. Martini, and J. Bosch, "Software developer productivity loss due to technical debt—a replication and extension study examining developers' development work," *Journal of Systems and Software*, vol. 156, pp. 41–61, 2019.

[5] A. Tornhill and M. Borg, "Code red: the business impact of code quality - a quantitative study of 39 proprietary production codebases," in *Proceedings of the International Conference on Technical Debt*, ser. TechDebt '22. ACM, 2022-08-16, pp. 11–20.

[6] T. Besker, A. Martini, and J. Bosch, "The pricey bill of technical debt: When and by whom will it be paid?" in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2017, pp. 13–23.

[7] V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, and F. Arcelli Fontana, "A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools," *Journal of Systems and Software*, vol. 171, p. 110827, 2021.

[8] T. Besker, A. Martini, and J. Bosch, "Impact of architectural technical debt on daily software development work — a survey of software practitioners," in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2017, pp. 278–287.

[9] A. Martini, E. Sikander, and N. Madlani, "A semi-automated framework for the identification and estimation of architectural technical debt: A comparative case-study on the modularization of a software component," *Information and Software Technology*, vol. 93, pp. 264–279, 2018.

[10] A. Martini, F. A. Fontana, A. Biaggi, and R. Roveda, "Identifying and prioritizing architectural debt through architectural smells: A case study in a large software company," in *Software Architecture*, C. E. Cuesta, D. Garlan, and J. Pérez, Eds. Cham: Springer International Publishing, 2018, pp. 320–335.

[11] P. Avgeriou, I. Ozkaya, A. Chatzigeorgiou, M. Ciolkowski, N. A. Ernst, R. J. Koontz, E. Poort, and F. Shull, "Technical debt management: The road ahead for successful software delivery," in *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, 2023, pp. 15–30.

[12] C. Giardino, N. Paternoster, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software development in startup companies: the greenfield startup model," *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 585–604, 2015, publisher: IEEE.

[13] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "Do faster releases improve software quality? an empirical case study of mozilla firefox," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 2012, pp. 179–188.

[14] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.

[15] L. Yu and A. Mishra, "An empirical study of lehman's law on software quality evolution," *International Journal of Software and Informatics*, 2013.

[16] G. Digkas, A. Chatzigeorgiou, A. Ampatzoglou, and P. Avgeriou, "Can clean new code reduce technical debt density?" *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1705–1721, 2022.

[17] G. Digkas, M. Lungu, A. Chatzigeorgiou, and P. Avgeriou, "The evolution of technical debt in the apache ecosystem," in *Software Architecture: 11th European Conference, ECSA 2017, Canterbury, UK, September 11-15, 2017, Proceedings 11*. Springer, 2017, pp. 51–66.

[18] P. Avgeriou, P. Kruchten, R. L. Nord, I. Ozkaya, and C. Seaman, "Reducing friction in software development," *IEEE Software*, vol. 33, no. 1, pp. 66–73, 2015, publisher: IEEE.

[19] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498–1516, Jun. 2013.

[20] E. Klotins and T. Gorschek, "Continuous software engineering in the wild," in *Software Quality: The Next Big Thing in Software Engineering and Quality*. Springer, 2022, pp. 3–12.

[21] E. Zabardast, J. Gonzalez-Huerta, and F. Palma, "The impact of forced working-from-home on code technical debt: An industrial case study," in *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2022, pp. 298–305.

[22] M. A. Al Mamun, A. Martini, M. Staron, C. Berger, and J. Hansson, "Evolution of technical debt: An exploratory study," in *2019 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, IWSM-Mensura 2019, Haarlem, The Netherlands, October 7-9, 2019*, vol. 2476. CEUR-WS, 2019, pp. 87–102.

[23] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure it? manage it? ignore it? software practitioners and technical debt," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015-08-30, pp. 50–60.

[24] W. Cunningham, "The WyCash portfolio management system," *ACM Sigplan Oops Messenger*, vol. 4, no. 2, pp. 29–30, 1992, publisher: ACM New York, NY, USA.

[25] T. Besker, A. Martini, and J. Bosch, "A systematic literature review and a unified model of ATD," in *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2016, pp. 189–197.

[26] C. Jaspan and C. Green, "Defining, measuring, and managing technical debt," *IEEE Software*, vol. 40, no. 03, pp. 15–19, 2023.

[27] V. Lenarduzzi, A. Sillitti, and D. Taibi, "A survey on code analysis tools for software maintenance prediction," in *Proceedings of 6th International Conference in Software Engineering for Defence Applications*, ser. Advances in Intelligent Systems and Computing. Springer, 2020, pp. 165–175.

[28] V. Lenarduzzi, N. Saarimäki, and D. Taibi, "The technical debt dataset," in *Proceedings of the 15th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2019, pp. 2–11.

[29] Sonarqube rules. [Online]. Available: https://rules.sonarsource.com/

[30] R. Alfayez, R. Winn, W. Alwehaibi, E. Venson, and B. Boehm, "How SonarQube-identified technical debt is prioritized: An exploratory case study," *Information and Software Technology*, vol. 156, p. 107147, 2023-04.

[31] M. T. Baldassarre, V. Lenarduzzi, S. Romano, and N. Saarimäki, "On the diffuseness of technical debt items and accuracy of remediation time when using SonarQube," *Information and Software Technology*, vol. 128, p. 106377, 2020-12.

[32] J.-L. Letouzey, “The sqale method for evaluating technical debt,” in *2012 Third International Workshop on Managing Technical Debt (MTD)*, 2012, pp. 31–36.

[33] “The Developer Coefficient,” Sep. 2018. [Online]. Available: https://stripe.com/en-se/newsroom/stories/developer-coefficient

[34] H. Liu, Z. Ma, W. Shao, and Z. Niu, “Schedule of bad smell detection and resolution: A new way to save effort,” *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 220–235, 2012.

[35] Y. Guo and C. Seaman, “A portfolio approach to technical debt management,” in *Proceedings of the 2nd Workshop on Managing Technical Debt*, ser. MTD ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 31–34.

[36] Z. Li, P. Avgeriou, and P. Liang, “A systematic mapping study on technical debt and its management,” *Journal of Systems and Software*, vol. 101, pp. 193–220, 2015.

[37] M. Borg, A. Tornhill, and E. Mones, “U owns the code that changes and how marginal owners resolve issues slower in low-quality source code,” in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2023, pp. 368–377.

[38] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Apr. 2009.

[39] E. González-Estrada and W. Cosmes, “Shapiro–wilk test for skew normal distributions based on data transformations,” *Journal of Statistical Computation and Simulation*, vol. 89, no. 17, pp. 3258–3272, 2019.

[40] J. Benesty, J. Chen, and Y. Huang, “On the importance of the pearson correlation coefficient in noise reduction,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 4, pp. 757–765, 2008.

[41] C. Croux and C. Dehon, “Influence functions of the Spearman and Kendall correlation measures,” *Statistical Methods & Applications*, vol. 19, no. 4, pp. 497–515, Nov. 2010.

[42] M.-T. Puth, M. Neuhäuser, and G. D. Ruxton, “Effective use of spearman’s and kendall’s correlation coefficients for association between two measured traits,” *Animal Behaviour*, vol. 102, pp. 77–84, 2015.

[43] B. Ratner, “The correlation coefficient: Its values range between +1/-1, or do they?” *Journal of Targeting, Measurement and Analysis for Marketing*, vol. 17, no. 2, pp. 139–142, Jun. 2009.

[44] C. W. J. Granger, “Investigating causal relations by econometric models and cross-spectral methods,” *Econometrica*, vol. 37, no. 3, pp. 424–438, 1969.

[45] I. Winkler, S. Haufe, A. K. Porbadnigk, K.-R. Müller, and S. Dähne, “Identifying granger causal relationships between neural power dynamics and variables of interest,” *NeuroImage*, vol. 111, pp. 489–504, 2015.