The Impact of Ownership and Contribution Alignment on Code Technical Debt Accumulation

Ehsan Zabardast, Javier Gonzalez-Huerta, Francis Palma, Panagiota Chatzipetrou

***Abstract*—Context: Software development organisations strive to maintain their effectiveness whilst the complexity of the systems they develop continues to grow. To tackle this challenge, software development organisations tend to be organised into small teams working with components that can be developed, tested, and deployed separately. In this scenario, organisations must design their software architecture and organisational structures in such a way that enables communication and minimises dependencies, as well as helps teams reduce code and architectural degradation. Ensuring that each small, independent team is responsible for the components they primarily contribute is one approach to achieving this goal.**

Objective: This article reports a study that aims at understanding the impact of ownership and contribution alignment (contribution alignment, for short) on accumulation of code technical debt (TD) and how abrupt changes in team constellation affect teams' effectiveness in managing TD.

Method: We have conducted an embedded case study in a software development company developing a very large software system, analysing ten components belonging to one team. During the studied period, the team was split into two, and the components owned by that team were distributed between the two new teams. We have collected archival data from the company's tools in their daily development operations.

Results: In most cases with high degrees of contribution alignment, we have noticed a negative correlation between contribution alignment and TD per line of code (TD Density) before the team split. In four components, this correlation is statistically significant. This means that a higher contribution alignment degree implies a lower TD Density. After the split, we observe a statistically significant negative correlation in three components. The positive correlation observed in the other five components could potentially be attributed to low contribution alignment, leading to difficulties in managing TD Density.

Conclusion: Our findings suggest that contribution alignment can be important in controlling TD in software development organisations. Making teams responsible for the quality of components they have more expertise over and minimising dependencies between teams can help organisations mitigate the growth of TD.

***Index Terms*—Ownership and contribution, Contribution degree, Technical debt, Case study**

I. Introduction

Large-scale software organisations tend to organise their systems as a constellation of components that are usually developed and maintained by different teams as a way of "componentising" their software architectures. Microservices architecture style [1], [2] is a widely adopted specific example. In a microservices architecture, applications are composed of many small, independent services that communicate with each other, and that are owned, developed, and maintained independently by different teams. This approach can help organisations to enhance maintainability, improve agility, and reduce time-to-market [3]. At the same time, introduce challenges, such as the need for communication and coordination between development teams, and to manage dependencies among microservices and the teams developing them [3].

Although there are several approaches to handling ownership, with different effects on teams' and individuals' autonomy, large-scale organisations usually rely on weak ownership principle [4], which consists of one team *owning* a component (or a microservice). The *owning* team is responsible for the quality of the *owned* component, and its members usually decide about code changes made by members of other teams through code reviews [4]. However, this responsibility does not necessarily mean they are the sole contributors to the code of that component, not even the *main* contributors.

Prior studies explored the relationship between the number of developers in open-source projects and their quality [5], [6]. However, having multiple authors who belong to multiple clusters might have a negative impact on quality, for example, the introduction of security flaws [6]. The quality decline associated with authors from different clusters can be attributed to the concept of *responsibility diffusion*. Responsibility diffusion [7] is a concept from social sciences that describes the lack of action when many people witness a criminal action. All witnesses tend to think someone else will take action (i.e., call the police), and yet, no one does. In software engineering, we associate the concept of responsibility diffusion with the lack of sense of responsibility when a given component (or code element) has multiple authors belonging to different organisations or clusters, and therefore no one will take corrective actions [8]. This article goes one step forward, claiming that this lack of sense of responsibility is also affected by the fact that an owning team does not have a major contribution to the owned code element or component, i.e., the ownership and contribution alignment degree is low.

In proprietary software systems, the alignment between the architecture and the organisation's communication structure becomes a critical factor to success in the development of large, very large, and ultra-large-scale[1] software systems [3], [11]. Conway [12] hypothesises that *"organisations that design systems tend to produce designs that mirror the communication structure of these organizations"*. It seems natural that when we componentise the architecture, the team constellation

E. Zabardast and J. Gonzalez-Huerta are with Blekinge Institute of Technology, Sweden.

F. Palma is with University of New Brunswick, Canada.

P. Chatzipetrou is with University of Örebro, Sweden.

[1]Based on the definitions by [9] and [10], respectively.

and the ownership should be adapted to minimise communication overhead and the dependencies between teams at the task level (i.e., a team depending on some other team's work to finish a task). Ideally, a team will be more productive if it is responsible for the components for which they are the main contributors. This aligns the team's responsibilities with their contribution, which might help reduce the average time to review and integrate new code [13], and help mitigating the accumulation of technical debt (TD). TD is a metaphor used to describe the long-term costs of maintaining and updating a software system due to using suboptimal or inefficient solutions during development [14], [15].

We hypothesise that a low contribution alignment degree can trigger faster accumulation of TD. Teams with low contribution to a given component might not perceive it as their responsibility and might not have enough knowledge to take corrective actions or decide on changes implemented by others.

To study this phenomenon, i.e., the impact that the degree of alignment between ownership and contribution might have on accumulation of TD, we have conducted a case study on a software company developing a very large software system (1.5 million LOC, developed by >20 teams). We have followed the evolution of ten software components (microservices) initially developed by one team over three years. During this period, the team was split into two, and the components owned by the team were distributed between the new teams. The degree of contribution alignment after the split changed substantially for some components since some of the main contributors were not in the owning team anymore.

The first goal of this case study is to investigate how sudden changes in team constellation might affect teams' effectiveness in managing TD in the components they are responsible for. Second, to analyse the impact of contribution alignment on accumulation of TD.

The remainder of the article is structured as follows: Section II describes the research methodology, by describing the case, the data collection, and the analysis methods. Section III presents the results. Section IV discusses the main findings and implications. Section V discusses and addresses the limitations and threats to validity. Section VI discussed the related work. Finally, Section VII presents the conclusion and future work.

II. Research Methodology

In this article, we are addressing the following research questions:

- **RQ1:** *How does the change in team structure impact the accumulation of code technical debt?*
- **RQ2:** *How does degree of ownership and contribution alignment impact the accumulation of code technical debt?*

To address the research questions, we have designed an embedded case study (type 2) according to the definition by Yin [16]. We conduct this study in an industrial setting and rely on the analysis of ten `Java` components developed by a company that has chosen to remain anonymous. We use archival data collected from the tools that the company uses during development. The research approach of this article is illustrated in Figure 1.

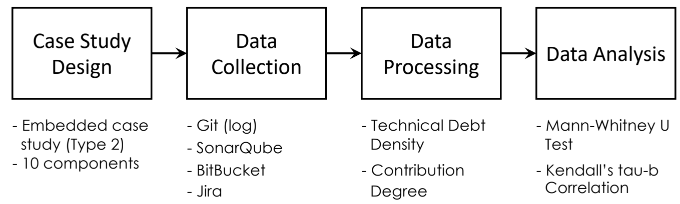

Fig. 1. The research method.

A. Case Study Design

The *context* of the case study is a large-scale software development company that develops cloud-based financial and accountancy services. It is a mature company in its development practices and has well-established, successful products on its portfolio, and it has been selected by convenience (availability and access). The company is interested in continuously improving its products and ways of working. Therefore, it was willing to participate in the study and learn from its results. The *case* is a software system developed by the company with 1.5 million LOC developed by >20 teams. *Units of analysis* are the ten components, i.e., microservices, that are part of this software system. Table I presents the details of the investigated components in this study.

TABLE I
Component information - size, number of commits, number of active development weeks, and the owning team.

Component	Size (LOC)	Commits	Active Weeks	Team
C1	18,244	1,019	122	Brown
C2	13,968	714	89	Gray
C3	12,290	872	105	Gray
C4	5,019	192	50	Brown
C5	11,001	691	70	Gray
C6	7,642	366	74	Brown
C7	31,708	1,136	130	Brown
C8	1,872	203	57	Gray
C9	11,994	434	78	Brown
C10	17,187	617	101	Brown
Total:	**130,925**	**6244**	-	-

The analysis period is three years from 2020 to 2022. The measurements are collected in weekly intervals, and we only consider data from the weeks where there was active development on the component. Choosing shorter intervals, e.g., days, or even individual commits, results in a high proportion of observations with `zero` values which might impact the variables under study.

In the following, we describe the main constructs and measurements used for the quantitative analysis of the data collected in this study. We use *Degree of Ownership and Contribution Alignment*, and *Technical Debt Density* ($TD_{Density}$) [17], [18] as the main constructs. Figure 2 illustrates the constructs used in this work.

- The *Degree of Ownership and Contribution Alignment* or *Contribution Degree* represents how much of the contribution to a component comes from the officially

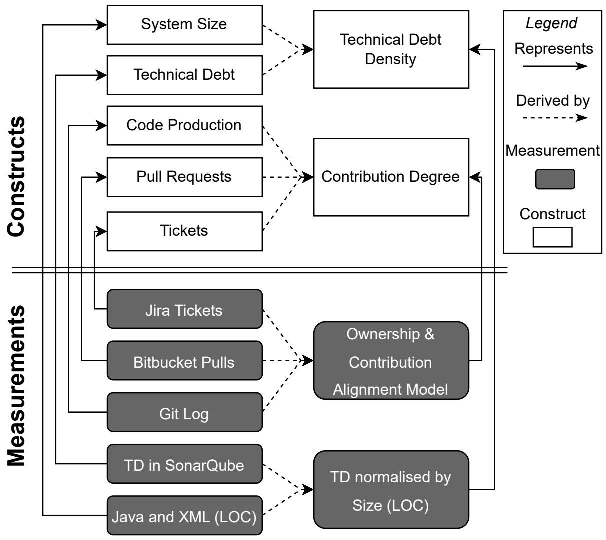

Fig. 2. Study constructs and measurements.

assigned owner. A 100% contribution degree means that all the contribution to a component is from the formally responsible team (owning team), as designated by the organisation. From this point on, we will refer to the *degree of ownership and contribution alignment* as *Contribution Degree*. We use the ownership and contribution alignment model (OCAM) [19] to calculate the *Contribution Degree*. In this article, we are using an extension of the original OCAM model reported in [19]. We use four measures out of the seven presented in the original model, but we compile them in a single metric with a ratio scale that allows us to use it as contribution degree, i.e., the ratio of contribution to a component by the owning team. The contribution degree is calculated using the contribution to `code production`, `pull requests`, and `tickets`. 'Code production' is the degree of contribution of a team both in terms of the number of commits and code churn [20]. 'Pull requests' is the degree of contribution of a team in terms of the number of created pull requests [13]. 'Tickets' is the degree of contribution of a team in terms of the number of tickets created on the ticket management system. The results obtained from applying the OCAM model to the components were validated with component owners and their managers.

- $TD_{Density}$ is the normalised amount of technical debt per line of code [17], [18]. $TD_{Density}$ is calculated by dividing the total amount of TD by the component size (LOC).

The two main variables used in this article are: *Contribution Degree* (See Equation 1) and $TD_{Density}$ (See Equation 6) $ContributionDegree(t)$ is defined as the contribution degree during a period:

$$Contribution.Degree(t) = \frac{C(t) + Ch(t) + P(t) + T(t)}{4} \tag{1}$$

With $C(t)$ being the contribution to the commits (See Equation 2), i.e., the percentage of commits done by the team on a specific component; $Ch(t)$ being the contribution to code churn (See Equation 3), i.e., the percentages of code written by the team in the code base of a specific component; $P(t)$ being the contribution to pull requests (See Equation 4), i.e., the percentage of pull requests created by the team on a specific component; and $T(t)$ being the contribution to tickets (See Equation 5), i.e., the percentage of tickets created by the team on a specific component.

$$C(t) = Commit_{Cont.Deg.}(t) = \frac{Commits_{team}}{Commits_{all}} \times 100 \tag{2}$$

$$Ch(t) = CodeChurn_{Cont.Deg.}(t) = \frac{CodeChurn_{team}}{CodeChurn_{all}} \times 100 \tag{3}$$

$$P(t) = PullRequest_{Cont.Deg.}(t) = \frac{PullRequests_{team}}{PullRequests_{all}} \times 100 \tag{4}$$

$$T(t) = Ticket_{Cont.Deg.}(t) = \frac{Tickets_{team}}{Tickets_{all}} \times 100 \tag{5}$$

$TD_{Density}(t)$ is defined as the amount of TD t normalised by the component size at a given point in time. It is operationalised as the total remediation time for all TD items present in the component on a given instant t, as reported by SonarQube, divided by the component size at t:

$$TD_{Density}(t) = \frac{\sum_{i=1}^{\#SystemTDI(t)}(RemeditionTime_i)}{Size(t)} \tag{6}$$

B. Data Collection and Processing

We used widely used tools such as `Git` and API endpoints to access the tools used by the company in their daily operations. We collected data from `Git`[2], `BitBucket`[3], `Jira`[4], and `SonarQube`[5]. We collected the developer team affiliation data from the company's API endpoint. Here, we present how the data was collected from each tool.

- `Git`: We used `Git Log` to collect the data regarding the component size and code production to calculate commit frequency and code churn. Component size is calculated using CLOC[6]. The historical data was collected by checking out the commits and calculating the size using CLOC. When calculating the size of the system, we include only the `Java` and `XML` files. We exclude the other files since technical debt is calculated based on `Java` and `XML` profiles.

[2] https://git-scm.com
[3] https://bitbucket.org
[4] https://www.atlassian.com/software/jira
[5] https://www.sonarsource.com/products/sonarqube/
[6] https://cloc.sourceforge.net

- `BitBucket`: We used the BitBucket API to collect the data regarding the creation of pull requests. The raw data was collected using the company API endpoint.
- `Jira`: We used the Jira API to collect the data regarding the creation of tickets. The raw data was collected using the company API endpoint.
- `SonarQube`: We used SonarQube to calculate the amount of code technical debt for each component. The data gathered from SonarQube are the issues detected by the quality profile and quality gate configured by the company. The data is collected via the company's SonarQube instance API endpoint. SonarQube has been used in similar studies on the topic of TD, e.g., [18], [21]–[27]. The tool provides the estimated remediation time (i.e., effort in minutes) required to resolve TD items (issues in the SonarQube terminology). The accumulated TD of a component at a point in time is the total remediation time for all the issues detected in its codebase. We considered TD as *repaid* when issues are tagged as 'fixed' or 'closed' or are removed from the component[7].

C. Data Analysis

We used the Shapiro-Wilk test as the normality test to analyse whether *contribution degree* and $TD_{Density}$ 'before' and 'after' the team split were normally distributed[8]. We decided to proceed with non-parametric tests as the data was normally distributed only in one case.

1) Mann-Whitney U Test for RQ1: To test if the change in team structure impacts the *contribution degree*, we used the Mann-Whitney U test [28], [29] to test to determine whether the contribution degree is different before and after the team split. In other words, we wanted to test whether the team split is a confounding factor that impacts the contribution degree. If the differences between the two periods are statistically significant, the analysis of the relationship between *contribution degree* and $TD_{Density}$ needs to be done separately.

Variables: The independent variable is the categorical groups of 'before' and 'after' the split for a given component. The dependent variable is the Contribution Degree for a given component *in that particular week*.

Hypothesis H_0: There is no difference in contribution degree between 'before' and 'after' teams split.
Hypothesis H_1: There is a difference in contribution degree between 'before' and 'after' teams split.

To test if the change in team structure impacts the accumulation of $TD_{Density}$, we used the Mann-Whitney U test. We used the test to examine whether the $TD_{Density}$ is different between 'before' and 'after' the team split.

Variables: The independent variable is the categorical groups of 'before' and 'after' the split for a give component. The dependent variable is $TD_{Density}$ for a given component *in that particular week*.

Hypothesis H_0: There is no difference of $TD_{Density}$ between 'before' and 'after' teams split.
Hypothesis H_1: There is a difference of $TD_{Density}$ between 'before' and 'after' teams split.

2) Kendall's tau-b Correlation Test for RQ2: To test whether there is an association between *Contribution Degree* and $TD_{Density}$ and the type of association, we used Kendall's tau-b correlation coefficient[9]. Kendall's tau-b correlation coefficient is a non-parametric measure of the strength and direction of association between *Contribution Degree* and $TD_{Density}$. Kendall's tau-b correlation coefficient can range from one (1) to minus one (-1). The closer a result is to one, the higher the level of association. The corresponding p-values for Kendall's tau-b correlation coefficients indicate the presence of a significant relationship between *Contribution Degree* and $TD_{Density}$. Kendall's tau-b correlation coefficient is more robust and efficient than alternative non-parametric tests (i.e., Spearman rank correlation), and it is preferred when we have smaller samples.

Variables: Contribution degree for a give component *in that particular week* and $TD_{Density}$ for a given component *in that particular week*.

Hypothesis H_0: There is no association between contribution degree and $TD_{Density}$.
Hypothesis H_1: There is an association between contribution degree and $TD_{Density}$.

D. Validation of the Results

During the execution of the case study, the research team meets the teams under study in recurring meetings every two weeks to report the plan and discuss this and other studies, as well as in focus groups in which we presented the results and collected their feedback. The results of the *contribution degree* were presented to the development team, the engineer leader and product owners to validate the accuracy of our calculations, and whether they reflected the events that occurred during the studied period (e.g., top contributors leaving the organisation).

Moreover, we gathered data regarding potential explanations for some sudden changes in $TD_{Density}$ during the studied period. In this latter case, to avoid introducing bias, we did not reveal the purpose of the analysis until presenting the full results. This way, we avoid the studied teams trying to defend suboptimal decisions. For this data gathering sessions, we sometimes visualised both non-normalised TD and $TD_{Density}$ to reason about its evolution.

III. Results

This section presents the results investigating the impact of ownership and contribution alignment on code technical debt for ten components. Table II presents the descriptive statistics for *Contribution Degree* and $TD_{Density}$.

[7] The company's SonarQube instance has been configured not to remove TDI records when they are removed without tagging them as fixed or closed, but to keep their introduction and removal dates to improve the reliability of the data.

[8] All the statistical tests reported in this subsection were conducted using `Python ver. 3.9.9` and `NumPy ver. 1.10.1`

[9] The correlation tests were conducted using `IBM SPSS Statistics ver. 28`

TABLE II
DESCRIPTIVE STATISTICS FOR CONTRIBUTION DEGREE AND $TD_{Density}$. THE TABLE IS SPLIT TO PRESENT THE DATA SEPARATELY FOR BEFORE AND AFTER THE TEAM STRUCTURE CHANGE.

Contribution Degree

	Before					**After**				
	N	Mean	STD	Min	Max	N	Mean	STD	Min	Max
C1	46	53.183	9.372	31.3	58.9	76	53.271	12.969	36.4	75.9
C2	32	68.516	10.502	52.1	82.3	57	41.853	8.535	32.8	82.4
C3	38	84.482	9.462	68.2	97.2	67	35.278	18.069	20.8	81.7
C4	10	38.240	2.370	36.0	42.5	40	19.230	12.550	2.8	32.8
C5†	-	-	-	-	-	70	83.651	3.672	73.9	89.3
C6	22	35.259	3.879	22.6	39.4	52	37.596	19.230	11.1	55.8
C7	45	42.258	5.502	29.2	45.3	85	40.935	15.336	18.0	62.9
C8†	3	-	-	-	-	54	31.924	32.504	7.3	81.4
C9	26	58.127	7.181	42.4	65.7	52	51.398	7.071	39.5	59.7
C10	36	64.533	2.632	57.9	66.9	65	47.646	8.094	28.8	55.8

Technical Debt Density

	Before					**After**				
	N	Mean	STD	Min	Max	N	Mean	STD	Min	Max
C1	46	0.221	0.006	0.211	0.231	76	0.205	0.004	0.193	0.213
C2	32	0.232	0.051	0.167	0.392	57	0.158	0.028	0.117	0.203
C3	38	0.093	0.011	0.080	0.126	67	0.099	0.014	0.079	0.112
C4	10	0.085	0.004	0.079	0.089	40	0.085	0.008	0.072	0.094
C5†	-	-	-	-	-	70	0.025	0.036	0.001	0.101
C6	22	0.109	0.007	0.100	0.123	52	0.138	0.006	0.128	0.154
C7	45	0.434	0.007	0.423	0.446	85	0.359	0.034	0.308	0.423
C8†	3	-	-	-	-	54	0.327	0.029	0.278	0.386
C9	26	0.084	0.009	0.074	0.108	52	0.095	0.013	0.073	0.119
C10	36	0.228	0.006	0.215	0.241	65	0.236	0.007	0.223	0.248

†No descriptive statistics due to lack or limited number of observations before team split.

As described in Section I, the owning team of the components went through a drastic change and split into two teams on week 9 in 2021 (week 61). Each new team took ownership of certain components. Team Brown took ownership of components C1, C4, C6, C7, C9, and C10, whilst Team Gray took ownership of components C2, C3, C5, and C8 (see Table I). There were other changes to teams' composition during the analysed period, i.e., members leaving a team or new members joining a team. The changes to the teams are illustrated in Figure 3.

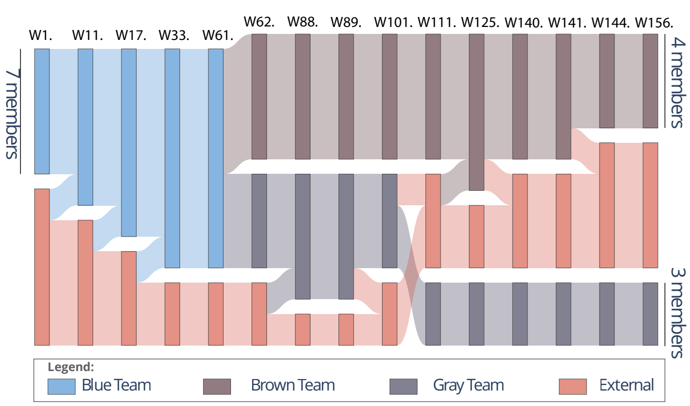

Fig. 3. Changes to the team during 2020 and 2022. The blue team split into two teams, team brown and team grey, on week 9 in 2021 (week 61).

Figures 4 and 5 present the box plots for the distribution of *Contribution Degree* and $TD_{Density}$ respectively. We observe: *Contribution Degree*: The medians of distributions are different for components *C2*, *C3*, *C4*, and *C10*.

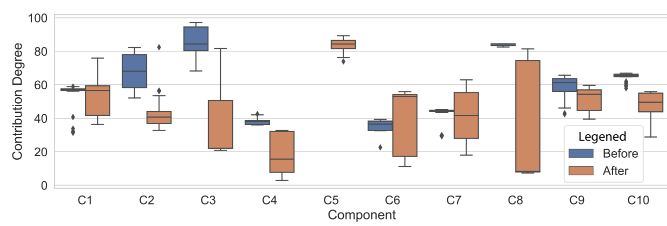

Fig. 4. Distribution of the *Contribution Degree* observations for components 'before' and 'after' the split.

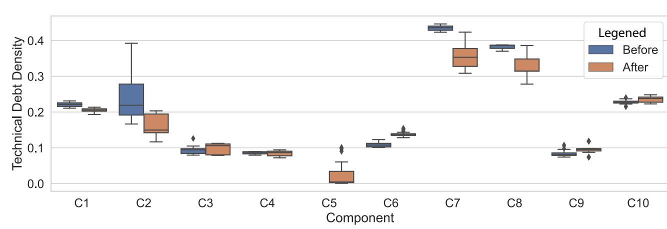

Fig. 5. Distribution of the $TD_{Density}$ observations for components 'before' and 'after' the split.

$TD_{Density}$: The medians of distributions are different for components *C1*, *C2*, *C3*, *C6*, *C7*, *C9* and *C10*.

Visual inspection of the components' changes to *Contribution Degree* and $TD_{Density}$ helps identify key events during the analysed period. These key events include: the change in team structure, i.e., the team splitting into two teams on week 9 in 2021 (week 61) and key members leaving a team. For example, a senior architect left Team Gray on week 37 in 2022 (week 141). Figure 6 illustrates the evolution of

Contribution Degree and $TD_{Density}$ for each component. The charts present the changes 'before' and 'after' the team split with blue and orange colours, respectively. The dashed lines illustrate the *Contribution Degree*, and the solid line illustrates $TD_{Density}$.

C1 [`Team Brown`]: The contribution degree for this component slightly increased before the team split. The split caused a sudden drop in contribution since some key developers for that component were allocated to Team Gray. Some weeks after that, the contribution degree went back to levels prior to the split and continued slightly growing. Conversely, $TD_{Density}$ slightly decreased before and after the team split.

C2 [`Team Gray`]: We observe that $TD_{Density}$ decreases as the contribution degree increases before and after the team split (the team keeping $TD_{Density}$ under control). As in *C1*, there is a major and sudden decrease in contribution degree after the team split. $TD_{Density}$ slightly increases after the split.

C3 [`Team Gray`]: We observe that $TD_{Density}$ slightly decreases as the contribution degree is high before the team split. As in most components, there is a major and sudden decrease in contribution degree when the team splits. And another major and sudden decrease in contribution degree was in week 101 when a product owner[10] left the team. We observe an increasing trend of $TD_{Density}$ during the period after the split.

C4 [`Team Brown`]: We observe that the contribution degree for this component decreases significantly after the team split (close to 0%). $TD_{Density}$ slightly decreases before the team split and increases significantly after the split.

C5 [`Team Gray`]: This component was created after the team split. Therefore, we cannot compare it similar to the other cases. We can observe that the contribution degree is very high from the owning team (80% and higher) from the beginning, meaning that the owning team is its main contributor. As observed in Figure 6, $TD_{Density}$ is very low at the beginning of the inception of the component. But it increases towards the end of the analysis, although the ownership levels remain in the same range.

C6 [`Team Brown`]: We observe that the contribution degree for this component decreases after the team split. And $TD_{Density}$ starts to increase after the contribution degree drops. The owning team increases the contribution degree around week 80. This coincides with $TD_{Density}$ slowly decreasing. There is a major and sudden decrease in contribution degree in week 141 when a senior architect left the team.

C7 [`Team Brown`]: This component has the highest amount of $TD_{Density}$ among the investigated components. The component is a fork of an existing open-source repository that the team developed further internally. We observe that $TD_{Density}$ decreases as the owning team increases its contribution degree. There is a major and sudden decrease in contribution degree in week 141 when a senior architect left the team.

C8 [`Team Gray`]: This component was created right before the team split. Contribution degree and $TD_{Density}$ stay do not change significantly before and after the team split. There is a major and sudden decrease in contribution degree in week 101 when a product owner left the team. $TD_{Density}$ started increasing after the product owner left the team.

C9 [`Team Brown`]: We observe that $TD_{Density}$ decreases as the contribution degree increases before the team split. There is a sudden decrease in contribution degree after the team split. The owning team increased their contribution after the team split. $TD_{Density}$ slightly increases after the split.

C10 [`Team Brown`]: We observe that $TD_{Density}$ has not changed substantially after the change. There is a decrease in contribution degree after the team split. However, the owning team has increased their contribution to the component after the split. There is a major and sudden decrease in contribution degree in week 141 when a senior architect left the team.

In order to test whether the change in the team structure significantly impacted contribution degree and $TD_{Density}$, we used the Mann-Whitney U test, as described in Section II-C. The test results show that there is a significant difference between *before* and *after* the team split in contribution degree and $TD_{Density}$ in most of the components (5 out of 8 for contribution degree and 7 out of 8 for $TD_{Density}$). Given that the team split is a confounding factor for analysing the association between contribution degree and $TD_{Density}$, we decided to separately perform Kendall's tau-b correlation test for the periods before and after the team split. The test results are presented in Table III. The statistically significant cases are highlighted in bold.

TABLE III
MANN-WHITNEY U TEST RESULTS FOR CONTRIBUTION DEGREE AND TECHNICAL DEBT DENSITY TO COMPARE THE DIFFERENCES BETWEEN *before* AND *after* THE TEAMS SPLIT. SIGNIFICANT RESULTS ARE PRESENTED IN BOLDFACE.

	Contribution Degree		**Technical Debt Density**	
	P-value	N	P-value	N
C1	0.571	122	**<0.001**	122
C2	**<0.001**	89	**<0.001**	89
C3	**<0.001**	105	**0.029**	105
C4	**<0.001**	50	0585	50
C5†	-	-	-	-
C6	0.195	74	**<0.001**	74
C7	0.922	130	**<0.001**	130
C8†	-	-	-	-
C9	**<0.001**	78	**0.001**	78
C10	**<0.001**	101	**<0.001**	101

†No statistical test due to lack or limited number of observations before team split.

The results from Kendall's tau-b correlation coefficient for components 1-10 are presented in Table IV. The results depict if there are any differences under two different occasions: before and after the team split. The statistically significant cases are highlighted in bold. The p-values less than 0.01 are identified with (§), and p-values that are less than 0.05 are identified with (†).

The results for the correlation (see Table IV), before the team split up, show that there is a statistically significant relationship between *contribution degree* and $TD_{Density}$ for components *C1*, *C2*, *C4*, and *C9*. For component *C5*, we have no observations before the team split. We have very few

[10]This product owner was a senior developer for over 10 years working on the same components in the same team.

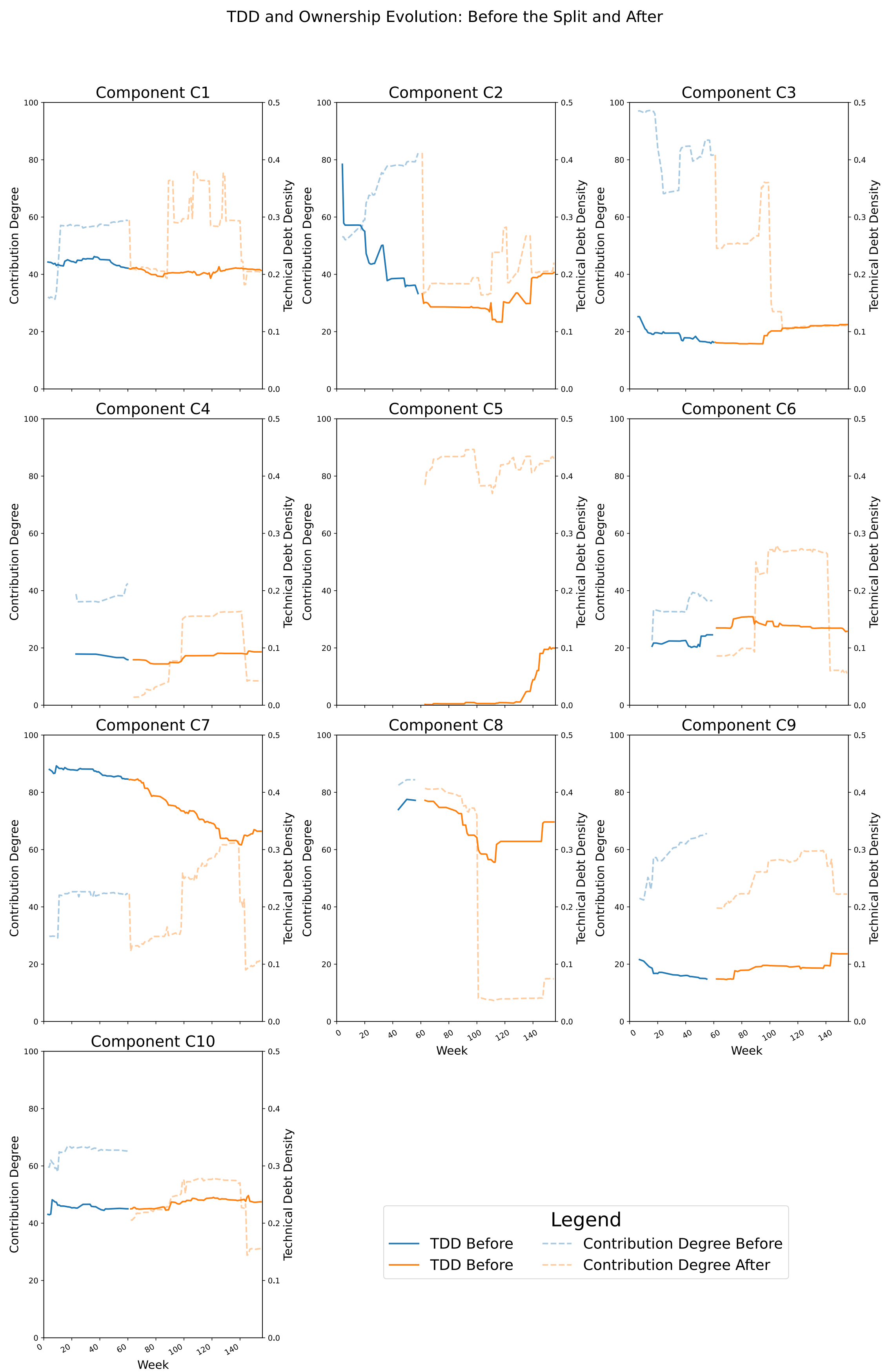

Fig. 6. The evolution of contribution degree and $TD_{Density}$ in ten components.

TABLE IV
TEST RESULTS - KENDALL'S τ. SIGNIFICANT RESULTS ARE IN BOLDFACE. MAGNITUDE OF ASSOCIATION IS CALCULATED BASED ON [30].

	Before				After			
	P-value	Kendall's τ	Magnitude	N	P-value	Kendall's τ	Magnitude	N
C1	**0.001**	**-0.332 §**	**Strong**	**46**	**0.013**	**-0.199 †**	**Moderate**	**76**
C2	**<0.001**	**-0.840 §**	**Strong**	**32**	**0.011**	**0.237 †**	**Moderate**	**57**
C3	0.410	0.093	Very Weak	38	**<0.001**	**-0.353 §**	**Strong**	**67**
C4	**0.037**	**-0.529 †**	**Strong**	**10**	**0.002**	**0.353 §**	**Strong**	**40**
C5*	-	-	-	-	0.073	0.148	Weak	70
C6	0.051	-0.301	Strong	22	**0.019**	**0.229 †**	**Moderate**	**52**
C7	0.309	-0.107	Weak	45	**<0.001**	**-0.464 §**	**Strong**	**85**
C8*	-	-	-	3*	**<0.001**	**0.770 §**	**Strong**	**54**
C9	**0.001**	**-0.918 §**	**Strong**	**26**	0.282	0.104	Weak	52
C10	0.743	0.039	Very Weak	36	**<0.001**	**0.454 §**	**Strong**	**65**

†Correlation is significant at the 0.05 level (2-tailed).
§Correlation is significant at the 0.01 level (2-tailed).
* No statistical test due to lack or limited number of observations before team split.

observations for component *C8*, i.e., 3 cases. Thus, we present no results for those two components before the team split.

On the other hand, in the results (see Table IV) for the correlation after the team split, a statistically significant relationship between contribution degree and $TD_{Density}$ was detected for all the components except for component *C5*, where the team did not split, and component *C9*.

Figure 7 illustrates the relationship between *contribution degree* (X axis) and $TD_{Density}$ (Y axis) for each component. We use Kendall's tau-b correlation coefficient (see Table IV) and Figure 7 to investigate each component. Note that the regression lines presented in the figure are used only for visual inspection. We are not using regression to draw conclusions nor predict contribution degree and $TD_{Density}$.

C1 [`Team Brown`]: There is a significant strong negative correlation between contribution degree and $TD_{Density}$ before the team split (P-Value: 0.001; Kendall's τ: -0.332) and a significant moderate negative correlation after team split (P-Value: 0.013; Kendall's τ: -0.199). Our observation and the test results suggest that as the contribution degree increases over time, $TD_{Density}$ decreases in the component.

C2 [`Team Gray`]: There is a significant strong negative correlation between contribution degree and $TD_{Density}$ before the team split (P-Value: < 0.001; Kendall's τ: -0.840) and a significant moderate positive correlation after team split (P-Value: 0.011; Kendall's τ: 0.237). We observe a different pattern after the team split. There is a significant moderate positive correlation between contribution degree and $TD_{Density}$; as contribution degree increases, $TD_{Density}$ increases too.

This increase in $TD_{Density}$ might be due to the context of this particular component. The component is related to handling *user authentication* in the system. There were issues integrating a third-party authentication application into the system and the increase in $TD_{Density}$ can be attributed to the development of new features.

Furthermore, we observe that as the contribution degree decreases to 40% and below, $TD_{Density}$ changes unexpectedly, i.e., other factors such as responsibility diffusion [8] and knowledge loss [31], [32] might have a bigger impact on $TD_{Density}$.

C3 [`Team Gray`]: There is no significant correlation between contribution degree and $TD_{Density}$ before the team split (P-Value: 0.410; Kendall's τ: 0.093). However, the results show a significant strong negative correlation after the team split (P-Value: < 0.001; Kendall's τ: -0.353). Our observation and the test results suggest a decrease in $TD_{Density}$ as the contribution degree increases only before the split.

Similar to *C2*, there is a decrease of contribution degree below 40% around week 100 in this component, after which $TD_{Density}$ starts increasing.

C4 [`Team Brown`]: There is a significant strong negative correlation between $TD_{Density}$ and contribution degree before team split (P-Value: 0.037; Kendall's τ: -0.529) and a significant strong positive correlation after team split (P-Value: 0.002; Kendall's τ: 0.353). Overall, the team's contribution degree is low in this component.

We observe a different pattern after the team split with a significant strong positive correlation between contribution degree and $TD_{Density}$. This increase in $TD_{Density}$ might be due to the context of this component, which is related to handling different *licenses* in the system. There has been a significant amount of new development in this component, and the increase in $TD_{Density}$ can be attributed to the development of new features while the owning team was gaining ownership.

We observe a decrease of contribution degree to near 0% after the team split, after which $TD_{Density}$ starts increasing.

C5 [`Team Gray`]: There is no significant correlation between $TD_{Density}$ and contribution degree since the creation of this component (P-Value: 0.073; Kendall's τ: 0.148).

C6 [`Team Brown`]: There is no significant correlation between $TD_{Density}$ and contribution degree before the team split (P-Value: 0.051; Kendall's τ: -0.301). However, the results show a significant moderate positive correlation after the team split (P-Value: 0.019; Kendall's τ: 0.229). There are two clusters for contribution degree observations —orange dots— after the team split (See Figure 7 Component C6). This left cluster, i.e., lower contribution degree, is related to the fact that a senior architect left the team. The increase of the $TD_{Density}$ after the team split is also associated with the same event.

C7 [`Team Brown`]: There is no significant correlation between $TD_{Density}$ and contribution degree before the team

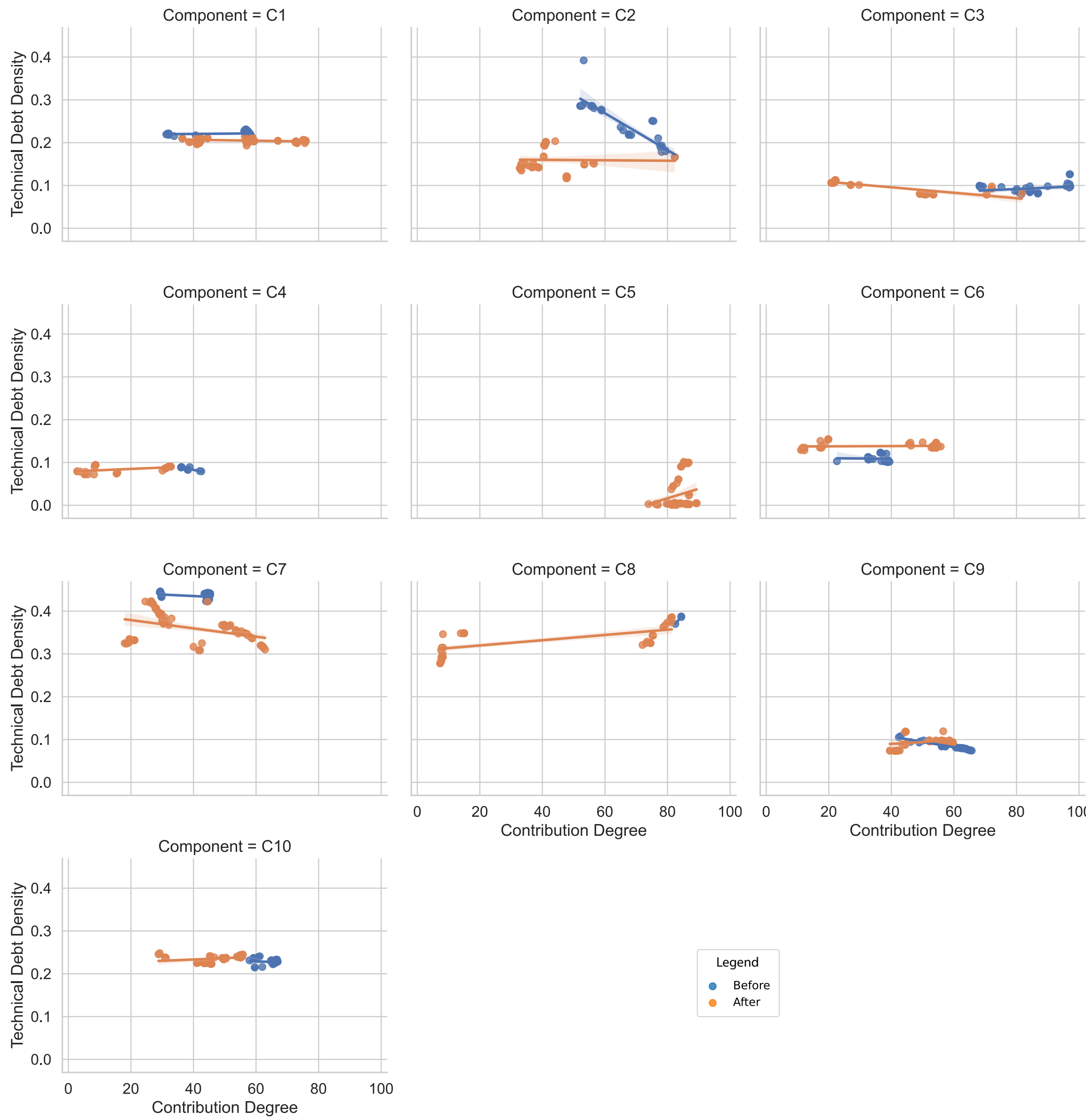

Fig. 7. Relationship between *Contribution Degree* (X axis) and $TD_{Density}$ (Y axis) for each component. The regression lines presented in the figure are only for visualisation. We are not using regression to describe the results nor prediction of contribution degree and $TD_{Density}$.

split (P-Value: 0.309; Kendall's τ: -0.107). However, there is a significant strong negative correlation after the team split (P-Value: < 0.001; Kendall's τ: -0.464). Our observation and the test results suggest that as the degree of ownership increases over time after the split, $TD_{Density}$ decreases in the component.

Similar to *C2*, *C3*, and *C4*, we observe a decrease of contribution degree to below 40% around week 145 in this component, after which $TD_{Density}$ starts increasing.

C8 [`Team Gray`]: There is a significant strong positive correlation between $TD_{Density}$ and contribution degree after the team split (P-Value: < 0.001; Kendall's τ: 0.770). There are two clusters for contribution degree observations —orange dots— after the team split (See Figure 7 Component C8). The left cluster, i.e., lower contribution degree, is related to the removal of a product owner from the team. The increase of the $TD_{Density}$ after the team split is also associated with the same event.

Similar to *C2*, *C3*, *C4* and *C7*, we observe a decrease of contribution degree to below 40% around week 100 in this component, after which $TD_{Density}$ starts increasing.

C9 [`Team Brown`]: There is a significant strong negative correlation between $TD_{Density}$ and contribution degree before the team split (P-Value: 0.001; Kendall's τ: -0.918). However, there is no significant correlation after the team split (P-Value: 0.282; Kendall's τ: 0.104). We observe a decrease of contribution degree to below 40% after the team split, after which $TD_{Density}$ starts increasing.

C10 [`Team Brown`]: There is no significant correlation between $TD_{Density}$ and contribution degree before the team split (P-Value: 0.743; Kendall's τ: 0.039). However, the results show a significant strong positive correlation after the team split (P-Value: < 0.001; Kendall's τ: 0.454). There is a significant strong positive correlation between contribution degree and $TD_{Density}$. This increase in $TD_{Density}$ is due to the context of this component. The component is related to handling *users* in the system. The team maintains three versions of the same functionality at the same time, and the increase in $TD_{Density}$ can be attributed to it.

IV. Discussion

This section discusses the insights gained from analysing the components and practical implications of the findings.

> **RQ1.** *How does the change in team structure impact the accumulation of code technical debt?*

The findings of this study suggest that altering the structure of a team, specifically by dividing it into two separate teams, has the potential to impact the accumulation of technical debt. Specifically, our analysis reveals that components *C2*, *C4*, *C6*, and *C10* experienced a reduction in their contribution degree to below 40%, which corresponded with an increase in $TD_{Density}$ in the subsequent weeks. Our observations are similar to the work presented by Nagappan et al. [33] on the impact of organisation structure on code quality.

The observations were further corroborated by the statistical tests conducted. However, it is worth noting that we cannot draw the same conclusion for component *C8*, as its introduction occurred several weeks prior to the team split and thus was not subject to the same impact. These findings highlight the importance of carefully considering the potential effects of team restructuring on technical debt accumulation and the need for continued monitoring and analysis to ensure optimal team performance and productivity.

> **RQ2.** *How does degree of ownership and contribution alignment impact the accumulation of code technical debt?*

The results presented in Section III suggest that contribution degree might impact the accumulation of $TD_{Density}$. We observe that before the split that in four components (*C1*, *C2*, *C4*, *C9*)[11] the contribution degree has a negative correlation, meaning that the higher the contribution degree, the lower $TD_{Density}$. After the split, we observe eight components for which the contribution degree seems to impact $TD_{Density}$. In components *C1*, *C3*, and *C7*, we observe a statistically significant negative correlation between contribution degree and $TD_{Density}$; the higher the contribution degree, the lower $TD_{Density}$. However, we also observe that for components *C2*, *C4*, *C6*, *C8*, and *C10*, there is a positive correlation, which might seem counter-intuitive. The first clarification is that a lower contribution degree does not mean not incurring in TD since many other factors might be impacting its accumulation, but what we observe, after inspecting Figures 6 and 7, that $TD_{Density}$ tend to grow together with contribution degree when the contribution degree is at levels below 40% (except for the case of *C5*), which might mean that controlling TD effectively is harder when the work in that component is mainly done by developers from outside the team.

One aspect to be considered when analysing the results after the split is that the change in trend on $TD_{Density}$ might manifest with a few weeks of lag. In the first weeks after the split, the owning team might restrain themselves from making big changes, or the changes we observe might have been under development before the split and therefore have a different impact on TD.

In the case of *C5*, There might be several explanations, the first being the fact that during the first weeks of development for a component created from scratch, $TD_{Density}$ is more easily kept under control, and when the functionality starts to grow $TD_{Density}$ grows with it. We can also explain with the *Technical Credit* concept. Experienced teams or individual developers might have *credit* to incur in TD, similar to the financial concept of *credit*, which allows a customer to loan money from a financial institution. Higher ownership might be linked with technical credit, which we will study in further work. C8 does not seem to follow the same pattern because *C8* is created a few weeks before the split, but not from scratch. Instead, this component was created as a migration of a service in a monolith architecture to a microservices.[12]

We observe that when a key contributor (senior architect and senior developer) leaves the team, the impact of "architectural knowledge vaporisation" [31], [32] might be manifested as an increase in technical debt. This result is aligned with the findings by Bird et al. [34], which suggest that high levels of top contributors result in higher quality. We observed that when a senior developer left the team in Week 101, the contribution degree dropped more than 50%, in components *C3* and *C8*, resulting in an increase in $TD_{Density}$. Similarly, we observe the same when a senior architect left the team in Week 141, the contribution degree dropped more than 50% in components *C6* and *C7*.

Our observations suggest that the events that cause major changes to contribution degree, i.e., the decrease of contribution degree, might impact the accumulation of $TD_{Density}$. As reported by [35], software quality is directly linked to the volume of collaboration and commitment of the development team. Therefore, the impact of a major decrease in contribution degree can be manifested as the faster accumulation of technical debt (i.e., $TD_Density$ growth over time).

Implications

- In the process of assigning component ownership to teams, it is crucial to take into account the contribution degree and its impact on the accumulation of TD to each component. The changes in contribution degree can arise from a number of factors, including organisational changes —changes to team constellation— and attrition —team members leaving a team or the company—,

[11] For components *C5* and C8 where there is no analysis before the split since *C5* was a component created after the team split and *C8* only a few weeks before, and therefore without enough data points to draw meaningful conclusions.

[12] Most of the components under study are part of a migration from a monolith to a microservices architecture, but they count on a long development history. However, C8 is the only component that results from the migration of functionality from the monolith that occurred during the period under analysis.

among others. Neglecting to consider these factors when making component ownership decisions can lead to increase in TD impacting the development effectiveness and efficiency. Therefore, it is imperative for managers and team leaders to carefully evaluate the contribution degree to each component.

- In the context of organisational restructuring, when the creation of new teams involves splitting existing ones, a key consideration is assigning, when possible, components to the sub-team that exhibits the highest degree of contribution. This rationale may be based on a number of factors, such as the distribution of expertise and skills among team members, the complexity of the component, or the urgency of the task at hand. By assigning components in this manner, organisations can aim to minimise the impact of the uncontrolled accumulation of TD.
- The contributions of the senior members of the team are important to preserving the quality of the code [36]. Attrition is unavoidable and members of the team might leave due to a variety of reasons. Companies and development teams should understand the degree of contribution of individual team members to try to preserve knowledge and avoid architectural knowledge vaporisation.
- In general, high contribution degree levels seem to be a good way to manage TD effectively. We hypothesise that a high contribution degree is a way to fight back responsibility diffusion. We might have many other teams contributing to a particular component, but if the owning team is also the main contributor, they might put more effort into caring about the code base and performing TD mitigation activities, e.g., refactorings [37].
- Ownership and contribution alignment and its implications (i.e., TD) seem to be phenomena that need to be studied further, especially in proprietary software development. This might also help us further understand the potential impact of organisational changes.
- We observed that experienced and skilled members of teams, such as senior architects and senior developers, tend to make decisions that may manifest as technical debt. These individuals possess a high degree of expertise and can take on more debt due to their seniority and major contributions. Hence, the term *Technical Credit* can be used to describe their ability, potential, and magnitude of making sub-optimal decisions that may result in TD.

V. Limitations and Threats to Validity

Different types of threats to validity might impact the results of our research. *Construct validity* concerns the selection of measurements that reflect the constructs. The study constructs and measurements used in this study are presented in Section II. We use several measurements from different tools including `Git`, `BitBucket`, `Jira`, and `SonarQube`. We are aware of the limitations of static analysis tools (e.g., SonarQube) in detecting TD items [38], however first, SonarQube is the tool (and construct) that the studied company (and many others) uses for reasoning about technical debt and code and architectural degradation. Second, it has been used in other studies approaching the evolution of Technical Debt [18], [21]–[26]. We use $TD_{Density}$ [17] and *contribution degree*. The former has been used in other studies to investigate the impact of external factors such as Clean Code [22] and the latter is an extension of the model presented in [19]. We do not distinguish between the TD items when conducting the analysis, i.e., all TD items are considered to have the same weight in the calculations. The repaid TD is calculated through the issues flagged as 'fixed' and 'closed'. However, there might be cases where TD has been removed by deleting the files that might not be included in our analysis.

We mitigated the potential threats regarding the calculations of measures and the selection of cases in consensus with the heads of development in the studied company.

Internal validity concerns the extent to which the results are free from error, i.e., the results represent the truth. A major threat to the validity of this study is the presence of confounding factors that, together with ownership and contribution misalignment, might impact the results presented in this work. We present the results from correlation tests, i.e., the results presented in this article do not aim to provide a cause-and-effect relationship between the studied variables. We mitigate this threat to validity by providing contextual information regarding the components to find other possible factors impacting our observations. There might be other non-studied factors that can explain the observations. Additional studies are required to better understand the other factors impacting the growth of $TD_{Density}$.

External validity concerns the generalisability of the results. The study results presented in this article only apply to the components investigated in this case study. The case study results should not be considered generalisable based on cases. We use statistical tests strictly to reason about each specific case.

Reliability, as one of the biggest threats to validity in this study, concerns the collected data and the performed analysis to be independent from individual researchers. We mitigated this threat to validity by consistently and frequently involving the developing teams —holding bi-weekly meetings— throughout the study design and interpreting the results.

Finally, *conclusion validity* refers to the degree to which the conclusions drawn from a statistical analysis are accurate and reliable [39]. It is concerned with whether the statistical inferences are based on a sound and robust data analysis. To mitigate this threat, we used appropriate statistical methods and considered potential confounding variables that could affect the results, i.e., team split. We also collected an adequate sample size representative of the population being studied.

VI. Related Work

There are several research works that scrutinise the relationship between team or organisation structure and software quality [5], [6], [13], [33], [34], [36], [40], [41], some of which directly put the focus on the relationship of ownership and software quality, e.g., [13], [34], [36], [41]. The concept of $TD_{Density}$ [17], [18], [42]–[44] has been used as a construct to reason about the evolution of code and architectural degradation [45]. The degree to which developers contribute to a particular project has also been addressed in several studies, e.g., [35], [46]–[48] as a way to characterise the degree in which developers participate in software projects.

In the following subsections, we summarise relevant works in each of these areas to position the contribution of our study in relation to the existing literature on the field.

A. Organisations, Team Structure and Quality

The quality of any product is strongly affected by organisation structure [40]. This is empirically shown by Nagappan et al. [33] on the relationship between organisational structure and software quality via eight organisational complexity metrics from the code viewpoint. Examples include "*the absolute number of unique engineers who have touched a binary and are still employed by the company*" and "*the total number of unique engineers who have touched a binary and have left the company as of the release date of the software system*" [33]. The authors develop a model to predict the failure-proneness and achieve a precision of up to 86.2% and recall of up to 84%, compared to other traditional metrics, including code churn and code complexity. Thus, organisational and owner metrics are more effective in software quality estimation.

There also have been studies analysing the impact that the number of developers might have on the quality of Open-Source Systems (OSS) (e.g., [5], [6]). Schweik et al. [5] opposes two laws (or principles): Brooks' law [40] against Linus' law [49]. On the one hand, Brooks' law states that adding more (human) resources to a software project tends to make it late. On the other hand, Linus' law [49] named after the creator of the Linux operating system, states that "with enough eyeballs, all bugs are shallow". In other words, adding more developers (in OSS) encompasses higher quality. In [5], authors report the impact of the number of developers on the "survivability" of OSS projects, using the latter as a proxy of its quality. Meneely and Williams [6] did a similar study, in this case just focusing on Linus' law, specifically regarding the number of vulnerabilities. The main result is that the presence of developers from different clusters seems to have a negative impact on quality in the form of more vulnerabilities present in the system.

B. Software Ownership and Quality

Bird et al., [34] examine the relationship between various (software) ownership measures (e.g., *the number of low-expertise developers* and *proportion of ownership for the top owner* - although in this case, the authors address individual contributors and not teams) and software failures, and find that the measures of ownership are associated with pre-release faults and post-release failures. In particular, the evidence of correlation is significant for minor contributors, i.e., those who contribute less and infrequently on a module. Our *contribution degree* calculation uses the degree of contribution of the owning team, instead of the top contributor, as the metric to reason about the degree of alignment between ownership and contribution, and its impact on $TD_{Density}$

Thongtanunam et al. [13] argue that code ownership comes with responsibility, and developers who author the majority of changes to a module are presumably the owners. However, contribution may come in other forms, e.g., in modern code review by criticising code changes authored by other developers. In our *contribution degree* calculation, we consider code review information. The authors also show that 67%-86% of the developers who contribute to a module do not author code changes, i.e., contribute by reviewing code. Among them, only 18%-50% are core team members. Thus, the authors suggest that code review should be included in code ownership estimation. While evaluating the relationship between review-specific and review-aware code ownership and defect-proneness, the authors find that developers with low traditional and review ownership are more prone to post-release faults. These results aligned with our findings that suggest that higher ownership might help teams keep $TD_{Density}$[13].

Moreover, Posnett et al. [36] argue that low ownership of a module (i.e., too many contributors) can affect code quality. The authors define a metric called DAF (Developer's Attention Focus, which measures the focus of a developer on some activities) and show that more focused developers introduce fewer defects than less-focused developers. In contrast, files receiving narrowly focused activity are more likely to have faults than others.

C. Technical Debt Density in Source Code

In an exploratory study, Al Mamun et al. [17] investigates the ability of the '$TD_{Density}$ trend' metric to estimate the evolution of technical debt (TD). The '$TD_{Density}$ trend' is the slope of two consecutive '$TD_{Density}$' measures. Using the $TD_{Density}$ trend' metric, the authors observe that a file has the highest level of $TD_{Density}$ at the initial stage of its revisions, and $TD_{Density}$ decreases as the file size increases.

In another study, Digkas et al. [18] examine the relationship between the amount of technical debt in new code and the evolution of technical debt in a system. The authors argue that TD grows in absolute numbers as software systems evolve. In contrast, the density of TD (i.e., TD divided by lines of code, also known as normalised TD) may decline due to refactoring or the development of new artefacts with less TD. As their findings, the authors report that among the three major types of code changes (insertion, deletion, and modification), the contribution of code modification (i.e., refactoring) is strongly related to the change in the $TD_{Density}$.

Levén et al. [43] investigate the causal relationship between the existing $TD_{Density}$ of a system and developers' tendency to introduce new TD during the system's evolution (using the Broken-Window Theory). The findings suggest that existing TD affects developers' tendency to introduce new TD during further system development.

Arvanitou et al. [44] investigate how the accumulation of TD can be explained in (scientific) software development. To do that, authors first identify software engineering practices used to develop scientific software and the most common causes of introducing TD and then map them. Findings suggest

[13]One of the dimensions used by SonarQube, which includes bugs, as detected using static analysis, which is different from the number of bug reports in ticket management systems.

that the scientists that develop scientific software lack certain software engineering practices and introduce TD in scientific software. To minimise the TD in scientific software, the authors recommend reusing software libraries and process improvement methodologies and working in pairs (i.e., applying eXtreme Programming), which would minimise about half of the causes of introducing TD.

D. Developer Contribution Metrics and Quality

Developers' contributions can be estimated and likewise can be linked to the quality of the source code. De Bassi et al. [35] argue that software quality can be directly linked to the volume of collaboration and commitment in the development team. The authors evaluate 20 quality metrics related to complexity, inheritance, and size that can measure team members' participation regarding their source code contribution to the project. In the end, those metrics are analysed based on positive, negative, and no influence on source code quality, including maintainability, testability, and understandability.

Parizi et al. [47], utilising git-driven technology and its features, estimate and visualise a team member's contribution using a set of metrics, including *the number of commits*, *number of merge pull requests*, *number of files*, and *total lines of code*. For computing these metrics, authors primarily rely on git logs as inputs to extract the performance data in combination with the total time spent on a project each day.

In another work, Oliveira et al. [48] define source code ownership by a developer when they contributed most to a source code file. The authors study several code-based and commit-based developer productivity metrics, including *source lines of code by time* (SLOC/Time, code-based) – the larger the source code created by the developer, the higher their productivity; and *commits performed by time* (Commits/Time, commit-based) – the higher the number of commits, the higher is their productivity. Using those metrics, the authors explore whether and to what extent developer productivity metrics have a relationship with developers' productivity. Authors find that code-based metrics better explain productivity than commit-based metrics.

The literature shows that in software development, team structure, and the developers' contribution seem to impact quality. However, what is not known is whether the misalignment between ownership and contribution (i.e., low values of *contribution degree*) affects TD. This article aims to bridge this gap by studying the relationship between the TD and the misalignment between ownership and contribution. To measure the misalignment between ownership and contribution, we include several perspectives, including contributions to code reviews as suggested by [13], and measure contribution degree based on the degree of contribution by the owning team, as suggested by [34].

VII. Conclusion

In this article, we present the results of an industrial case study on the impact of ownership and contribution alignment on code technical debt. We investigated ten components during a three-year period (2020 to 2022) from a large software development company that develops web-based financial and accountancy services.

Study results reveal that prior to the team structure's change, a negative correlation was found between contribution degree and $TD_{Density}$ in the majority of cases where high levels of contribution degree were present. This negative correlation means that higher contribution degree correlate with lower $TD_{Density}$ was statistically significant in four components, indicating that heightened contribution degree was associated with decreased $TD_{Density}$. Once the original team was divided into two, a statistically significant negative correlation was observed in three components, whereas five components exhibited a positive correlation that may be attributed to low levels of contribution degree, thereby suggesting the team's inability to effectively manage $TD_{Density}$.

The results suggest that, when assigning the components' ownership responsibility to teams, it is important to consider the contribution degree of each team to make sure that the team receiving this responsibility is ready to take the required actions to preserve the components' $TD_{Density}$ under control. Additionally, the contributions of senior team members might be crucial to mitigating the accumulation of code TD and preserving knowledge when attrition occurs. High levels of contribution degree can help manage TD effectively by restraining responsibility diffusion and encouraging the owning team to invest more effort in mitigating TD. The study highlights the need for further research on the ownership and contribution alignment and their impact on TD in proprietary software development, which may help organisations better understand the potential effects of organisational changes.

However, both concepts, ownership and contribution degree alignment and TD, are complex phenomena, and other factors might impact the changes in the accumulation of TD. In addition, we are aware that the results are not generalisable, but the aim of this paper is also to raise awareness of the importance of aligning ownership and contribution degree.

We believe the results presented in this article need to be strengthened with future replications of the study. Finally, we plan to investigate the concept of TD credit in other cases.

Acknowledgements

This research was supported by KK foundation through the SHADE KK-Hög project 2017/0176 and KKS Profile project SERT 2018/010 at Blekinge Institute of Technology, Sweden.

References

[1] M. Fowler, "Microservices," 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html
[2] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, pp. 24–35, 2018.
[3] S. Newman, *Building Microservices*. O'Reilly, 2015.
[4] M. Fowler, "Code ownership," 2006. [Online]. Available: https://martinfowler.com/bliki/CodeOwnership.html
[5] C. M. Schweik, R. C. English, M. Kitsing, and S. Haire, "Brooks' versus linus' law: An empirical test of open source projects," in *9th Annual International Digital Government Research Conference*, 2008, pp. 423–424. [Online]. Available: http://ossmole.
[6] A. Meneely and L. Williams, "Secure open source collaboration: An empirical study of linus' law," in *16th ACM Conference on Computer and Communications Security*. Association for Computing Machinery, 2009, pp. 453–462.

[7] J. M. Barley and B. Latanfi, "Bystander intervention in emergencies: Diffusion of responsibility," *Journal of Personality and Social Psychology*, vol. 8, pp. 377–383, 1968.

[8] A. Tornhill, *Your Code As A Crime Scene*. The Pragmatic Bookshelf, 2015.

[9] T. Dingsøyr, T. E. Faegri, and J. Itkonen, "What is large in large-scale? a taxonomy of scale for agile software development," in *International Conference on Product-Focused Software Process Improvement*, vol. 8892, 2014, pp. 273–276.

[10] L. Northrop, P. Feiler, R. P. Gabriel, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Shcmidt, K. Sullivan, and K. Wallnau, "Ultra-large-scale systems the software challenge of the future ultra-large-scale systems: The software challenge of the future," Software Engineering Institute, Carnegie Mellon University, Tech. Rep., 2006.

[11] S. Baškarada, V. Nguyen, and A. Koronios, "Architecting microservices: Practical opportunities and challenges," *Journal of Computer Information Systems*, vol. 60, pp. 428–436, 9 2020.

[12] M. E. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[13] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 1039–1050.

[14] W. Cunningham, "The wycash portfolio management system," *ACM SIGPLAN OOPS Messenger*, vol. 4, no. 2, pp. 29–30, 1993.

[15] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering (dagstuhl seminar 16162)," in *Dagstuhl reports*, vol. 6, no. 4. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[16] R. K. Yin, *Case study research and applications: Design and Methods*, 6th ed. Sage, 2018.

[17] 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.

[18] 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, 2020.

[19] E. Zabardast, J. Gonzalez-Huerta, and B. Tanveer, "Ownership vs contribution: Investigating the alignment between ownership and contribution," in *19th IEEE International Conference on Software Architectures*. IEEE, 2022.

[20] J. C. Munson and S. G. Elbaum, "Code churn: A measure for estimating the impact of code change," in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. IEEE, 1998, pp. 24–31.

[21] 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.

[22] G. Digkas, M. Lungu, P. Avgeriou, A. Chatzigeorgiou, and A. Ampatzoglou, "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)*. IEEE, 2018, pp. 153–163.

[23] V. Lenarduzzi, N. Saarimaki, and D. Taibi, "On the diffuseness of code technical debt in java projects of the apache ecosystem," in *2019 IEEE/ACM international conference on technical debt (TechDebt)*. IEEE, 2019, pp. 98–107.

[24] V. Lenarduzzi, F. Lomio, H. Huttunen, and D. Taibi, "Are sonarqube rules inducing bugs?" in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 501–511.

[25] 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.

[26] M. Siavvas, D. Tsoukalas, M. Jankovic, D. Kehagias, and D. Tzovaras, "Technical debt as an indicator of software security risk: a machine learning approach for software development enterprises," *Enterprise Information Systems*, vol. 16, no. 5, p. 1824017, 2022.

[27] 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)*. IEEE, 2022, pp. 298–305.

[28] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[29] E. L. Lehmann and H. J. D'Abrera, *Nonparametrics: statistical methods based on ranks.* Holden-day, 1975.

[30] B. Botsch, "Significance and measures of association, scopes and methods of political science," 2011. [Online]. Available: https://polisci.usca.edu/apls301/Text/Chapter%2012.%20Significance%20and%20Measures%20of%20Association.htm

[31] J. Bosch, "Software architecture: The next step," in *Software Architecture: First European Workshop, EWSA 2004, St Andrews, UK, May 21-22, 2004. Proceedings 1*. Springer, 2004, pp. 194–199.

[32] D. Tofan, M. Galster, and P. Avgeriou, "Reducing architectural knowledge vaporization by applying the repertory grid technique," in *Software Architecture: 5th European Conference, ECSA 2011, Essen, Germany, September 13-16, 2011. Proceedings 5*. Springer, 2011, pp. 244–251.

[33] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: an empirical case study," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 521–530.

[34] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code! examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 4–14.

[35] P. R. De Bassi, G. M. P. Wanderley, P. H. Banali, and E. C. Paraiso, "Measuring developers' contribution in source code using quality metrics," in *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*. IEEE, 2018, pp. 39–44.

[36] D. Posnett, R. D'Souza, P. Devanbu, and V. Filkov, "Dual ecological measures of focus in software development," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 452–461.

[37] A. Sundelin, J. Gonzalez-Huerta, K. Wnuk, and T. Gorschek, "Towards an anatomy of software craftsmanship," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 1, pp. 1–49, 2021.

[38] V. Lenarduzzi, F. Pecorelli, N. Saarimaki, S. Lujan, and F. Palomba, "A critical comparison on six static analysis tools: Detection, agreement, and precision," *Journal of Systems and Software*, vol. 198, p. 111575, 4 2023.

[39] D. I. Sjoberg and G. R. Bergersen, "Construct validity in software engineering," *IEEE Transactions on Software Engineering*, pp. 1–1, 2022.

[40] F. P. Brooks, "The mythical man-month," *Datamation*, vol. 20, no. 12, pp. 44–52, 1974.

[41] F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 491–500.

[42] M. BenIdris, H. Ammar, and D. Dzielski, "The technical debt density over multiple releases and the refactoring story," *International Journal of Software Engineering and Knowledge Engineering*, vol. 31, no. 01, pp. 99–116, 2021.

[43] W. Levén, H. Broman, T. Besker, and R. Torkar, "The broken windows theory applies to technical debt," *arXiv preprint arXiv:2209.01549*, 2022.

[44] E.-M. Arvanitou, N. Nikolaidis, A. Ampatzoglou, and A. Chatzigeorgiou, "Practitioners' perspective on practices for preventing technical debt accumulation in scientific software development." in *ENASE*, 2022, pp. 282–291.

[45] E. Zabardast, J. Frattini, J. Gonzalez-Huerta, D. Mendez, T. Gorschek, and K. Wnuk, "Assets in software engineering: What are they after all?" *Journal of Systems and Software*, vol. 193, p. 111485, 11 2022.

[46] G. Gousios, E. Kalliamvakou, and D. Spinellis, "Measuring developer contribution from software repository data," in *Proceedings of the 2008 international working conference on Mining software repositories*, 2008, pp. 129–132.

[47] R. M. Parizi, P. Spoletini, and A. Singh, "Measuring team members' contributions in software engineering projects using git-driven technology," in *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2018, pp. 1–5.

[48] E. Oliveira, E. Fernandes, I. Steinmacher, M. Cristo, T. Conte, and A. Garcia, "Code and commit metrics of developer productivity: a study on team leaders perceptions," *Empirical Software Engineering*, vol. 25, pp. 2519–2549, 2020.

[49] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, pp. 23–49, 1999.