Ownership vs Contribution: Investigating the Alignment Between Ownership and Contribution

Ehsan Zabardast, Javier Gonzalez-Huerta, and Binish Tanveer
Software Engineering Research Lab (SERL)
Blekinge Institute of Technology, Karlskrona, Sweden
ehsan.zabardast@bth.se, javier.gonzalez.huerta@bth.se, binish.tanveer@bth.se

***Abstract*—Software development is a collaborative endeavour. Organisations that develop software assign modules to different teams, i.e., teams own their modules and are responsible for them. These modules are rarely isolated, meaning that there exist dependencies among them. Therefore, other teams might often contribute to developing modules they do not own. The contribution can be, among other types, in the form of code authorship, code review, and issue detection. This research presents a model to investigate the alignment between module ownership and contribution and the preliminary results of an industrial case study to evaluate the model in practice. Our model uses seven metrics to assess teams' contributions. Initial results suggest that the model correctly identifies misalignment between ownership and contribution. The detection of misalignment between ownership and contribution is the first step towards investigating the impact it might have on the faster accumulation of Technical Debt.**

I. Introduction

The development of software-intensive products and services has been evolving towards "componentising" architectures. Nowadays, big software development organisations tend to build their software products as a constellation of components often developed by different teams. One example of this trend is the wide adoption of the *microservices* architectural style [1], [2].

Alignment between architecture and organisation, therefore, plays a critical role [1], [3] in the development of large scale software systems. According to Conway's Law: Organisations that design systems tend to produce designs that mimic the communication structures of these organisations [4].

When we "componentise" (e.g., build the system using microservices), the team constellation should be adapted to minimise communication overhead. Although there are different approaches towards ownership and autonomy, software development organisations usually rely on weak ownership principle [5], which boils down to a given team owning a set of components (or microservices). Ideally, that team is the owner and responsible for that component and the main (or sole) contributor to the code. However, when examining real-world cases, that is seldom the case [6], [7]. In reality, a component can be providing support to several business streams; it can be developed by different teams and require the intervention of specialised teams to ensure non-functional requirements (e.g., security and reliability). Therefore, although the component will continue to be the responsibility of a team, the degree of contribution to the component can vary a lot.

The degree of alignment between the code ownership (i.e., the fact that the team is appointed as officially responsible for the quality of a given component or service [8]) and contribution (i.e., the extent to which that team is the main *contributor* for that particular product or service) can impact the effectiveness and efficiency of all the teams involved. We hypothesise that *the misalignment between team ownership and contribution can increase communication overhead and can make, for example, the code review time or the lead time to implement code changes longer*. We also hypothesise that *the misalignment may impact the pace of TD accumulation in services in which this misalignment is more acute*. In some cases, the *owner* team might only be holding the final responsibility of acting as gate (quality) keepers. In other words, the team members are not implementing the changes but only making sure that the code introduced in the codebase adheres to certain quality criteria and coding standards.

Let us suppose the owner team, after some time, loses control over the code of a component they own. In that case, it might also lose (partially) the ability to judge the appropriateness of the new code being committed, or just be cluttered by the number of changes and either not respond in time (longer lead time) or limit the extent of the code review. Therefore, properly analysing ownership and contribution seems crucial to properly "componentise" the architecture at scale effectively. However, we have started describing the disease, but we have not yet arrived at the actual symptoms. How can we adequately define contribution? Contribution can be in for of volume of code being authored, the number of commits, the code complexity, the number of created tickets, and other factors.

In this paper, we present a model that aims to distinguish between the team, which is the "official" or "formal" owner of a component, vs the team or teams which are the main contributors. The combination of the metrics and their visualisation enables the intuitive and easy interpretation of the state of contribution. While creating the model we have considered the research on both industrial (e.g., [6], [7]) and OSS (e.g., [9]). We have combined the metrics and methods they use to create our model to understand the alignment between ownership and contribution. In the following, we describe the proposed model in terms of the Goal-Question-Metric approach [10]. **Goal:** Analysing the software development projects to measure developer contributions to identify the alignment or mis-

alignment between ownership and contribution. And how the alignment impacts technical debt. **Questions:** How we can identify misalignment between ownership and contribution? Does the misalignment impact the growth of technical debt? **Metrics:** Number of commits, code complexity, code churn [11], number of tickets (e.g., Jira), ticket complexity, number of pull requests, and pull requests complexity.

Although there are resemblances with the code ownership in OSS development, we want to highlight that we are studying the role of *team* code ownership and its alignment with *team* contribution. We present a case study (Section II) that firstly led to the creation of a model to calculate the proportion of contribution to a component (Section III) and secondly to validate the model (Section IV). We present the initial results (Section V) and discuss their implications (Section VI). Finally, we present the related work (Section VII) and conclude the paper with our future work plan (Section VIII).

II. Industrial case study

We conducted a case study as part of an ongoing collaboration with an industrial partner. This section describes the goals, sample, data collection, and analysis.

1) Goal: The case study had two goals. The first goal was to analyse the software development process to measure developer contributions to identify misalignment between ownership and contribution. Furthermore, we were interested in investigating whether the misalignment between ownership and contribution impacts the accumulation of TD. This analysis led to the creation of a model (details in Section III) to assess the alignment between ownership and contribution. The second goal of the case study was to validate the developed model with the relevant stakeholders (details in Section IV).

2) Sample and Population: We conducted the case study with a large company that has chosen to remain anonymous. The company develops smart banking and financial solutions. The company is involved in a project profile collaboration with the research team and the components were selected based on the availability and convenience. The company wanted to improve their solutions/products and ways of working and hence was willing to participate in the study and learn from its results. The company employs agile practices and DevOps with autonomous teams working with practices like Scrum and Kanban. It uses a microservice architecture. Two teams A and B (with five and four developers respectively), were selected by convenience and availability. The teams worked on the main/legacy components and faced the challenges related to faster accumulation of TD and resolving pull requests.

3) Design: The case study was conducted using bi-weekly workshop meetings with these two teams for about eight months (Mar. 2021 – Nov. 2021). The research team (the authors) and the product manager, the product owner, and a few developers (whose number ranged from 2-3 depending on availability and requirement of the meeting) always participated in the meetings. Each meeting was arranged for about 30 to 45 minutes. During the first 20 minutes, the research team presented their findings and conducted focus groups for the rest of the time. They asked questions to identify metrics to capture the proportion of contribution and the root causes of the misalignment between ownership and contribution. The feedback from the teams then derived the next meeting sessions. Till mid of Nov. 2021, the feedback of the focus groups led to the creation of the model whereas the last focus group session held in end of Nov. 2021 was used to validate the developed model.

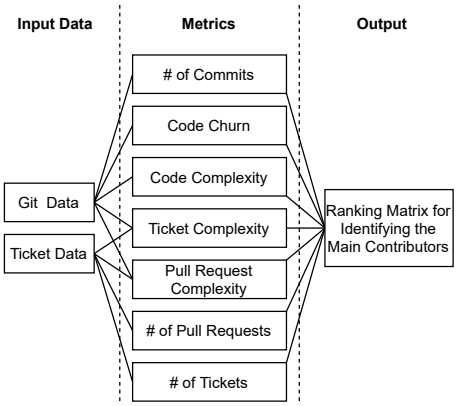

Fig. 1. The Ownership and Contribution Model. The input data is fed to seven metrics and the model creates the contribution matrix.

4) Data Collection and Processing: Through the focus groups (held from Mar. until Nov. 2021), the research team gathered data regarding formal component ownership. The research team also collected data from product code, version control (git), and issue tickets (Jira) to calculate team contribution. The data was collected through APIs provided by the applications used by the company, e.g., BitBucket API and Jira API. The collected data was pre-processed to remove anomalies (incomplete, inconsistent data). Relevant metrics like size of systems (in LOC), number of commits, code complexity, code churn, number of tickets, ticket complexity, number of pull requests, and pull request complexity.

III. The OCAM Model

The purpose behind the Ownership and Contribution Alignment Model (OCAM) is to calculate the proportion of the contribution of teams to a component. Since the contribution can come from different sources, the model considers contributions from code production, issuing tickets, and code reviews, as suggested by Bass et al. [12, pp. 355-356], e.g., code, issue, and pull request complexity respectively. The model is created in an iterative process through a case study (see Section II) while consulting professional developers from a software development organisation.

The input data for the model is extracted from git and ticket system APIs. The model uses seven metrics to calculate the proportion of contribution for each metric for specified time duration and ranks the contributors. Finally, the model creates the contribution ranking matrix (see Fig. 1). The ranking matrix consists of metrics (rows) and teams (columns). A number is assigned to each cell in a row with the team's rank

in that particular metric (row). A lower rank shows a higher proportion of contribution. These metrics are the number of commits, code churn, code complexity, number of tickets, ticket complexity, number of pull requests, and pull request complexity. There are no weights for the metrics in the model. The metrics can be calculated for individual developers or teams. The metrics are described in Table I. The flexible nature of the model allows for removal of the metrics in case the data for calculating them are not available, i.e., any metric can be disregarded in the process in case of data limitation. Similarly, other metrics that can improve the quality of the model can be included in the model. The final contribution matrix will be created based on the calculated metrics.

TABLE I
THE OCAM METRICS AND THEIR DESCRIPTIONS. THE METRICS CAN BE CALCULATED FOR INDIVIDUAL DEVELOPERS OR TEAMS.

Metric	Description
# of Commits	The total number of commits that were pushed to the repository in a period of time.
Code Churn [11]	The amount of code changed in a period of time, i.e., the ratio of written code in the codebase.
Code Complexity	Cyclomatic complexity [13] of the written code in a specific time.
# of Tickets	The total number of accepted tickets created in a period of time.
Ticket Complexity	The complexity of the accepted tickets created in a specific time. The complexity is calculated by examining the cyclomatic complexity of changes identified by `git diff` to the code in response to the tickets.
# of Pull Requests	The total number of pull requests created in a period of time.
Pull Request Complexity	The complexity of the pull requests created in a specific time. The complexity is calculated by examining the cyclomatic complexity of changes to the code identified by `git diff` in response to the pull requests.

Data was gathered continuously during the research and was presented to the team in every meeting. The purpose was to fine-tune the collection and interpretation of the data. Each metric in the model is calculated separately. The contributions are ranked for each metric. Once the metrics are collected, the contribution matrix is created and presented as a heatmap. Fig. 2 illustrates an example output of the model and represents a snapshot of a component. By investigating the heatmap, we can understand the distribution of the contribution to a component. In this hypothetical example, four teams contribute to a component owned by Team C. However, the majority of the code is written by Team A. The misalignment of contribution and ownership is revealed by investigating the heatmap. Though the component is owned by Team C, they are the main contributors only for two metrics. If two teams have the same number, they are ranked equally. Lastly, in a case where two teams end up with similar contributions but different ranks for the metrics, their contributions remain the area of contribution. Such cases need to be individually investigated for each category of metrics or further for each individual metric.

IV. MODEL VALIDATION

After the model creation process, we assessed the model in practice with the help of the industrial partner. In particular, we

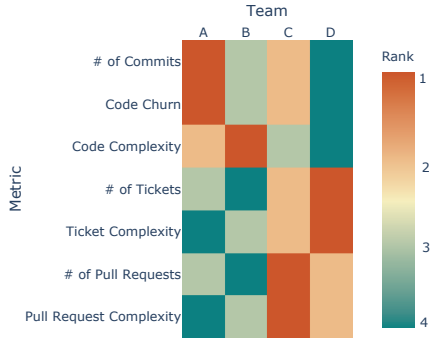

Fig. 2. Example heatmap created by OCAM on a component developed by four teams. The component is owned by team C. 1:4 Most/Least Contribution.

are interested in testing whether the model can meaningfully identify the proportion of contributions to a component by answering the following research question. **RQ:** *To what extent can the proposed model correctly identify developer contributions to a component?*

We collected data from 267 components, developed by the company, as input for OCAM to investigate the alignment between ownership and contribution. We used the metrics provided in the model to create the ranking matrix for identifying the main contributors for each component. In order to investigate the impact of the misalignment between ownership and contribution on TD, we used SonarQube to calculate effort to repay TD (in minutes) for each component. We used SonarQube because it is widely used in both industrial and open-source systems [14] and has been used in other research studies, e.g., Zabardast et al. [15].

To validate the model, we designed a focus group session with participation of two product managers, a product owner, and four developers. The results of the analysis were presented and discussed among the participants to validate the model and its accuracy. During the focus group, the participants were asked two questions for each presented component. *1. Was the detection of misalignment correct or not?* and *2. What is the reason behind the misalignment?*

V. INITIAL RESULTS

The model has identified 193 cases of misalignment, i.e., components where the owner team is not the main contributor. The final results of the analysis of 267 components were presented in a focus group with the participation of the authors and two product managers, a product owner, and four developers from teams A and B. Ten components from the output of the model were selected randomly to check for the validity of the model. The model correctly detected the misalignment in *all* the selected components. Each case was discussed separately between the researchers and participants.

Furthermore, we want to examine whether the misalignment between ownership and contribution impacts the accumulation of TD or not. We selected five components developed by two different teams (A and B). We extracted the amount of TD

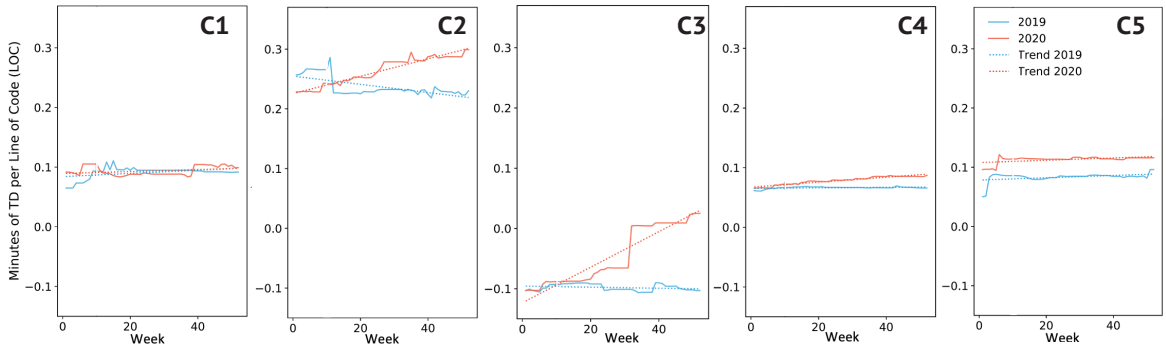

Fig. 3. Technical debt density growth in 2019 and 2020 in five selected components.

density per week during 2019 and 2020[1]. TD density is the total amount of TD on a component normalised by its size [16], [17]. TD density allows us to compare the growth of TD with its relation to the growth of system size. TD density trends for these five components are presented in Fig. 3. TD density in components C2 and C3 increases much faster than the rest. The OCAM identified misalignment between ownership and contribution for components C2 and C3, which turned out to have a different pattern when it comes to the accumulation of TD. The results of this analysis were presented to teams.

VI. Discussion and Practical Implications

OCAM provides an objective way to assess the contribution of developers from various perspectives. The results can be used in an informative way, i.e., to clarify the state of ownership and contribution to the developing teams. The results can help the developing teams approach development tasks more efficiently. Raising awareness regarding misalignment between ownership and contribution can help team understand why TD is increasing faster in some components, and therefore increase the level of quality required on code that is merged, or even find an explanation on why they are cluttered by the number of Pull Requests or bug reports they receive. The initial results presented in this paper are the first steps to investigate such factors further. We postulate that early identification of such misalignment can help organisations act in time to prevent faster accumulation of TD. Moreover, the model provides valuable insights for managers when investigating problems that stem from the organisational structure.

A. Limitations and Threats to Validity

Construct Validity We use different tools for measurements that reflect the construct of our study. We are aware of the limitations imposed by the tools. We have selected widely-adopted, industrial *de-facto* standard tools, such as SonarQube, to measure the metrics used for this study. Ownership and contribution alignment is a complicated and multifaceted phenomenon to study and there are many dimensions to consider while studying it. While OCAM presents metrics from different categories, there might be other metrics not considered on this study, or the ones being considered might not reflect the ownership vs contribution alignment. We created the model to be extensible to mitigate the model's limitations.

Internal Validity As stated above, the problem under study is complex, and there might be other factors that might explain some of the findings. We have used a methodological triangulation [18] to mitigate those threats, using quantitative and qualitative sources. We continue working on better understanding these factors to minimise their impact on the results.

External Validity Another threat to the study is the generalisability of the results into different contexts out of the investigated cases. We are aware that our results are strictly applicable to the studied case. In this study, we are focusing on analytical generalisability. The goal is to identify potential signals for misalignment between ownership and contribution.

Reliability Reliability is concerned with the data and the analysis being independent of the specific researchers. This is the most significant threat to the validity of our study. By including the company in focus group sessions for interpreting the results, we have tried to mitigate this threat.

VII. Related Work

1) Developer Contribution Metrics: There are many metrics derived from version control systems such as git that are used to calculate the amount of developer contribution. de Bassi et al. [19] provide a collection of code quality metrics. The authors categorise them in four groups of *Complexity Metrics*, *Inheritance Metrics*, *Size Metrics*, and *Coupling Metrics*. They successfully evaluate the individual contributions using quality metrics. Similarly, Diamantopoulos et al. [20] analyse the contribution of project collaborators in 3000 most popular Java projects on GitHub. The authors investigate 19 metrics in 2 categories of *Development* and *Operations*, e.g., "commits authored" and "issues participated" respectively for each category. Parizi et al. [21], present a git-driven solution to measure team members' contribution during the development. They propose five metrics including *number of commits*, *number of merge pull requests*, *number of files*, *total lines of code*, and *time spent* to measure contribution. The provided metrics by the authors do not consider the difficulty of the project. However, the authors note that the difficulty should always be considered when evaluating the contribution of developers.

[1]The company uses SonarQube in their development process.