Behind Developer Contributions on Conflicting Merge Scenarios

Gustavo Vale
Department of Computer Science
Saarland University
Saarbrücken, Germany
vale@cs.uni-saarland.de

Eduardo Fernandes
The Maersk Mc-Kinney Moller Institute
University of Southern Denmark
Odense, Denmark
edmf@mmmi.sdu.dk

Eduardo Figueiredo
Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte, Brazil
figueiredo@dcc.ufmg.br

Sven Apel
Department of Computer Science
Saarland University
Saarbrücken, Germany
apel@cs.uni-saarland.de

***Abstract*—*Context:* The success of Open Source Software (OSS) projects typically depends on simultaneous contributions of several developers. These contributions often affect the same changing source files and may lead to merge conflicts when integrated. Previous studies investigated the reduction of conflicting merge scenarios. However, empirical evidence on the involvement of OSS contributors in conflicting merge scenarios is scarce. *Objective:* We aim to fill this gap with a large-scale quantitative study with the goal of understanding: 1) the extent in which OSS contributors are involved in conflicting merge scenarios; 2) characteristics of these contributors; and 3) characteristics of changing source files. *Method:* We collect both contributor data and contribution data from 66 popular GitHub projects and analyze data of 2 972 distinct contributors who were involved in at least one conflicting merge scenario. We rely on both descriptive and inferential statistics to address our research questions. *Results:* About 80% of the analyzed contributors are involved in only one or two conflicting merge scenarios. Additionally, 42 out of the 66 projects had its top-one contributor as the one mostly involved in conflicting merge scenarios. Finally, only a small set of changing source files are involved in conflicting merge scenarios. *Conclusions:* We advocate that training the typically small group of contributors involved in conflicting merge scenarios could significantly reduce the number of merge conflicts.**

***Index Terms*—open source software, project contributor, merge conflict, mining software repositories, quantitative analysis**

I. Introduction

Contemporary software development greatly depends on simultaneous contributions of several developers [29], [36], [43]. Such observation particularly stands in the open source software (OSS) development [19], [33], which has GitHub as a prominent enabling platform. We refer to all GitHub users who have contributed to an OSS project (by either communicating with other users or changing source files) as contributors [49]. Users whose contribution is highly frequent (in this work, we consider users who are responsible for 80% of all contributions to an OSS project) are called *top contributors*. The remaining users are called *occasional contributors*.

Contributions may occur at two distinct levels of the OSS project development: *merge-scenario level* and *project level* [48], [49]. The former refers to contributions in a merge scenario, while the latter refers to contributions on the whole project at the end of each merge scenario (i.e., at the merge commit). Given the distributed nature of contemporary software development, contributions often affect a specific changing source file simultaneously and may lead to merge conflicts [1], [23]. Whenever the integration of simultaneous contributions leads to merge conflicts, we have a *conflicting merge scenario*. Contributors who were involved in a conflicting merge scenario are called *conflicting contributors*.

Several empirical studies investigated conflicting merge scenarios over the last decade [1], [23], [35], [48], [49]. While some studies focused on either predicting [35] or resolving [23], [48] conflicting merge scenarios, others investigated characteristics of merge conflicts [1], [49]. Still, little has been empirically done in terms of investigating the involvement of OSS contributors in conflicting merge scenarios. We advocate that understanding social aspects of conflicting contributors (and their activity on changing source files) can help decide which OSS contributors to instruct (and when to do it) with the purpose of avoiding conflicting merge scenarios.

In this paper, we present a large-scale quantitative study aiming at investigating what is *behind developer contributions on conflicting merge scenarios*. We are particularly interested in the three following aspects of developer

This work was partially supported by CNPq (grant 290136/2015-6) and CNPq/FAPEMIG (grants 150391/2023-4 and PPM-00651-17).

contributions. First, we assess the extent in which OSS contributors are involved in conflicting merge scenarios. Second, we look at the top contributors to understand their involvement in conflicting merge scenarios. Third, we characterize changing source files typically involved in conflicting merge scenarios.

To achieve our goals, we systematically collect both contributor data and contribution data from 66 popular GitHub projects. From a total of 25 397 distinct contributors, we analyzed 2 972 (11.70%) contributors who are involved in at least one conflicting merge scenario. We rely on both descriptive and inferential statistics to address our research questions. We summarize below our main study findings and their potential implications.

- **80% of the conflicting contributors were involved in only one or two conflicting merge scenarios.** Thus, a small group of conflicting contributors (only 20%) is involved in the majority conflicting merge scenarios. We advocate that training this specific small group could significantly reduce the number of merge conflicts.
- **64% of the 66 OSS projects under analysis had their top contributor as the one involved in more than 50% of the conflicting merge scenarios.** This result emphasizes the role of top contributors in the success of OSS projects with respect to the overall project quality.
- **A small set of changing source files was involved in conflicting merge scenarios.** Additionally, our results suggest that the most changed source files are affected by trivial, minor changes. Thus, we believe that contribution rules defined for each OSS project could avoid merge conflicts caused by these trivial, minor source file changes.

Replication Package: We have made the replication package of our quantitative study available online [47] for consultation. This package contains the framework to collect data, scripts to perform the analyses, and the data used in the analyses.

The remainder of this paper is structured as follows. In Section II, we provide our operational definitions of top contributors and occasional contributors which are relevant to the rest of the paper. In Section III, we describe our study design. In Sections IV to VI, we present our study results for each of our three research questions. In Sections VII and VIII, we discuss threats to the study validity and related work. Finally, in Section IX, we conclude this paper and suggest future work.

II. Top and Occasional Contributors

In this section, we summarize the operational definitions of top contributors and occasional contributors used in this paper. We rely on this classification instead of core and peripheral developers because, as suggested by a previous study [24], we consider that these terms better represent high-frequency or low-frequency contributors, respectively. Our definitions classify developers at distinct levels of granularity, project, and merge scenario as follows.

Top and occasional contributors at project level classification: Top and occasional developers at project level are classified based on their code contributions to the entire OSS project at the end of each merge scenario (i.e., at the merge commit). We followed the five steps below.

1) We compute, for each merge commit (we consider the `git checkout` command), the authorship of each line of code in the entire OSS project.
2) We sum up the lines of code each developer contributed, thereby mapping each developer with an unique identifier (*key*) to an object with the developer's information (*value*). This object includes an attribute informing the number of lines of code this developer changed in the entire project at the moment of the merge commit.
3) We compute the total number of lines of code in the project by summing all developer contributions.
4) We generate a list of developers by code contributions in descending order (i.e., developers who contributed the most appear at the top of the list).
5) We extract the developers at the top of the list whose sum of their contributions represent 80% of all contributions at the moment of the merge commit. These developers are the *top contributors*, while the others are the *occasional contributors*. The rationale behind this threshold is due to previous work [10], [33], [39], [44] which observed that code contributions typically follows a *Zipf* distribution (which implies that the top 20% of contributors are responsible for 80% of the contributions).

Top and occasional contributors at merge-scenario level classification: Top and occasional developers at merge-scenario level are classified based on their code contributions to a merge scenario. The classification at merge-scenario level is similar to the one at the project level; only the first step differs. Instead of measuring the authorship of each developer in the entire project, we measure the code contribution of each developer in the merge scenario. In other words, for each merge commit, we measure only the lines of code changed between the base commit and the merge commit. Hence, *top contributors* at the merge-scenario level are the developers who contribute to 80% of the changed lines of code in the merge scenario, while all other developers are *occasional contributors*.

The distinction between project level and merge-scenario level is crucial. The developer roles at *project level* provide a general view of the code contributions, while the developer roles at *merge-scenario level* provide a fine-grained, focused view on the code changes in a merge scenario as well as on merge conflicts.

III. Study Design

We describe our study design as follows. In Section III-A, we introduce our study goal and research ques-

tions. In Sections III-B and III-C, we explain the dataset extraction process and analysis procedures, respectively.

A. Study Goal and Research Questions

We rely on the Goal-Question-Metric template [5] to systematically define our study goal as follows: *analyze* OSS project contributors involved in conflicting merge scenarios; *for the purpose of* acquiring empirical evidence on characteristics and activities performed by the contributors; *with respect to* 1) how often contributors are involved in conflicting merge scenarios and the extent of such involvement, 2) key characteristics of the contributors involved in conflicting merge scenarios, and 3) characteristics of the contributions in terms of changed source files; *in the context of* popular GitHub projects.

We defined the three research questions (RQs) below with the purpose of driving our study.

RQ_1: *To what extent open source project contributors get involved in conflicting merge scenarios?* – We are particularly interested in the distribution of each developer role rather than a general analysis. Thus, RQ_1 is decomposed into the two sub-questions below.

- **$RQ_{1.1}$:** *How often do contributors get involved in conflicting merge scenarios?*
- **$RQ_{1.2}$:** *What is the proportion of involvement in conflicting merge scenarios by conflicting contributors?*

RQ_2: *How often are top contributors or top conflicting contributors involved in conflicting merge scenarios?*

RQ_3: *What are the main characteristics of the changed source files in conflicting merge scenarios?*

B. Dataset Extraction Process

We computed the number of stars to collect the 100 most popular GitHub projects [8]. We relied on GitHub as it is very popular in industry and largely investigated in the literature [12], [15], [19], [41], [42], [45], [48]. We limited our analysis to Git repositories because detecting merge scenarios is straightforward [35], [49]. To define our set of subject projects, we applied the following four filters [25]. **Filter 1:** filter out projects whose main file extension does not correspond to a programming language. Our study targets actual software projects. **Filter 2:** filter out projects with less than two commits per month in the last six months. Our study targets active GitHub projects. **Filter 3:** filter out projects for which we cannot reconstruct at least 50% of all merge scenarios. Our study targets projects that mostly adopt a three-way merge pattern [19], [28]. Including projects that adopt other patterns could bias our analysis. **Filter 4:** filter out less popular JavaScript projects to balance our set of subject projects in terms of the number of projects by programming language since it prevents analysis biases.

Our strategy for acquiring the merge scenario data consists of the following five steps. **Step 1:** for each selected repository, we clone its subject project. **Step 2:** we identify merge scenarios by filtering commits with multiple parent commits (merge commits are identified in Git when the number of parent commits is greater than one). **Step 3:** for each merge commit, we retrieve a common ancestor for both parent commits (i.e., the base commit). **Step 4:** we (re)merge the parent commit of the source branch into the parent commit of the target branch by using the standard `git merge` command and retrieve measurement data by comparing the changes that occurred from the base commit until the merge commit. **Step 5:** we store all data and repeat Steps 3 to 5 for each merge scenario found in Step 2.

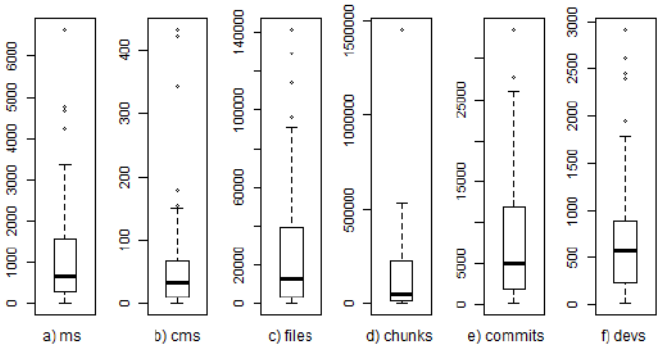


Fig. 1. Distribution of Project Metrics for the 66 Subject Projects

Our set of subject projects has 66 GitHub projects. These projects are developed in 12 different programming languages, including C++, Java, JavaScript, and Python. Our set has a total of 78 740 merge scenarios, which involve more than 1.5 million changed source files, 10.4 million chunks, and 3 950 conflicting merge scenarios. Bootstrap [46], Lantern [16], React [14], Redis [2], and TypeScript [32] are examples of subject projects. The full list of subject projects and additional content can be found in our supplementary repository [47].

In Figure 1, we depict the distribution of project metrics extracted from our set of subject projects. The metrics include the number of merge scenarios (*ms*), conflicting merge scenarios (*cms*), source files (*files*), *chunks*, *commits*, and developers (*devs*). We found a total of 25 397 distinct contributors; 2 972 out of these contributors (11.70%) were involved in at least one merge conflict.

C. Data Analysis Procedures

RQ_1 Analysis: To answer $RQ_{1.1}$, we create a table with all OSS contributors who are involved in at least one merge conflict. After, we group the contributors into five categories according to the number of conflicting merge scenarios they are involved in: one, two, three to five, six to ten, and more than ten conflicting merge scenarios. These values were arbitrarily chosen to support the data visualization given its power law distribution. In our case, power law distribution means that several merge scenarios have few contributors and few merge scenarios have several contributors. As we know the developer roles in each merge

scenario, we normalize each developer role to compare the number of developers who contribute in each group. To answer $RQ_{1.2}$, we also group contributors and analyze the distribution over the subject developer roles. However, we are interested in comparing their total of contributions with respect to their conflicting contributions. For instance, we investigate how often the contributions of a certain contributor lead to merge conflicts.

RQ_2 Analysis: Our RQ_2 analysis is two-fold. First, we analyze the share of conflicting merge scenarios of the top contributors by project. Top contributors are those who contribute to majority merge scenarios. Second, we analyze the share of conflicting merge scenarios of top conflicting contributors. Top conflicting contributors are those who are involved in the highest number of conflicting merge scenarios in the subject projects.

RQ_3 Analysis: We rely on manual and qualitative procedures to address RQ_3. We start by obtaining the list of top conflicting contributors from the five subject projects with the highest number of conflicting merge scenarios. After, we manually inspect each conflicting merge scenario. Our goal is to capture factors that may have led to the merge conflicts. For short, we perform three analyses: 1) we explore the numbers of merge scenarios and conflicting merge scenarios in these projects; 2) we discuss the impact of project rules on merge conflicts; and, 3) we analyze some changed files related to conflicts.

IV. Conflicting Merge Scenarios per Contributor (RQ_1)

In this section, we address RQ_1 by discussing the extent in which OSS contributors are involved in conflicting merge scenarios. In Section IV-A, we provide the results regarding how often contributors are involved in conflicting scenarios, while, in Section IV-B, we present the results on the proportion of involvement in conflicting merge scenarios in contrast to all merge scenario contributions.

A. Contributors in Conflicting Merge Scenarios ($RQ_{1.1}$)

Overall Results. In Figure 2, we depict the distribution of contributors by the number of conflicting merge scenarios they were involved in. Such distribution is presented in terms of five groups: one, two, three to five, six to ten, and more than ten conflicting merge scenarios. We present both the absolute number of contributors and the percentages with respect to the total of 2 972 contributors involved in at least one conflicting merge scenario. We discuss below our main findings.

Our data suggests that about 80% of the contributors were involved in only one or two conflicting merge scenarios. Indeed, we found that majority contributors are involved in either one conflicting merge scenario (62.6%) or two conflicting merge scenarios (17.3%). On the one hand, this result is expected if we consider the nature of globally distributed development [31] and the inherent complexity of modern OSS projects involving several

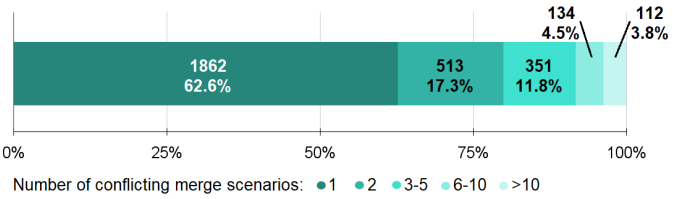
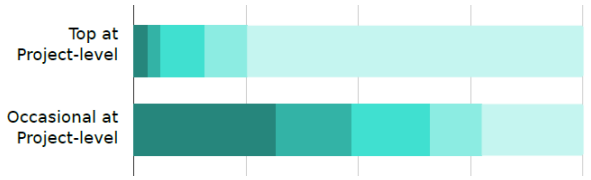
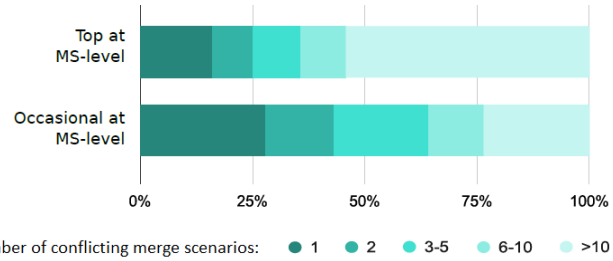


Fig. 2. Distribution of Contributors by the Number of Conflicting Merge Scenarios They Were Involved in



(a) Project-Level Contributors



(b) Merge Scenario-Level Contributors

Fig. 3. Distribution of Contributors by the Number of Conflicting Merge Scenarios They Participate

contributors [30]. On the other hand, it contrasts with past work assumptions on the relationship between merge conflicts and the little inexperience of new contributors with a specific OSS project [20]. Interestingly, however, we see in data of Figure 2 that approximately 20% of the contributors are involved in more than three conflicting merge scenarios. We also highlight that only 3.8% of the contributors are involved in more than ten conflicting merge scenarios. This result can be explained because, even in large OSS projects with hundreds of contributors, a very small group of contributors is responsible for most tasks related to maintaining and evolving a project.

Results by Developer Role. In Figure 3, we show the distribution of the investigated developer roles by the number of conflicting merge scenarios. That is, we compute the distribution of contributors by their respective top or occasional roles (see Section II). We expect this new perspective will help us understand the nature of those contributors who are more often involved in conflicting merge scenarios. We discuss below our main observations.

Project-level contributions. In Figure 3(a), we show that 74.61% of the top contributors are involved in more than ten conflicting merge scenarios, while only about 20% of the occasional contributors were involved in such amount of scenarios. This result is explainable since top contribu-

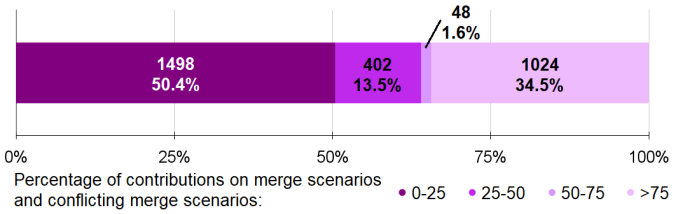


Fig. 4. Number of Developers Related to Conflicting Merge Scenarios

tors make most of the key decisions on the project maintenance and evolution. On the other hand, approximately 50% of the occasional contributors were involved in one or two conflicting merge scenarios. This finding is interesting given the occasional nature of many contributions [37], [38], which may be fine-grained and less likely to generate conflicts.

Merge-scenario-level contributions. In Figure 3(b), we show similar trends especially for occasional contributors. Slightly more than 50% of top contributors are involved in more than ten conflicting merge scenarios, against less than 25% for occasional contributors. This is an expressive percentage, although less expressive than the 75% of top contributors at project level. We speculate that such drop in percentages may be because we are looking at a fine-grained level and the changes performed by top contributors are more specific, like fixing a bug or introducing a new feature. On the other hand, about 40% of the occasional contributors at merge-scenario level were involved in one or two conflicting merge scenarios.

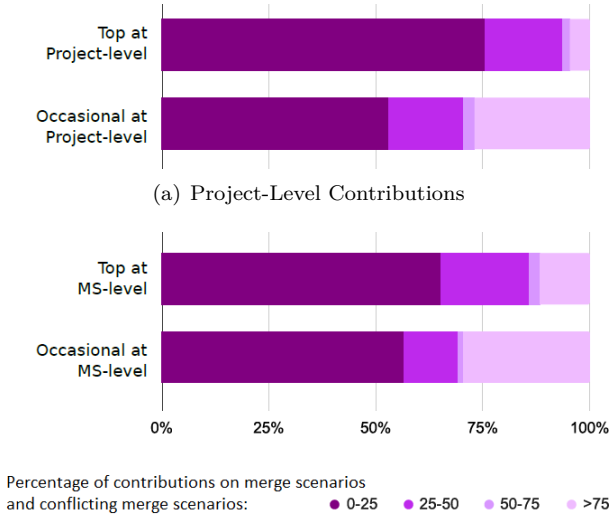
$RQ_{1.1}$ Finding: Overall, approximately 80% of OSS contributors are involved in a very small number of conflicting merge scenarios, i.e., one or two scenarios. With the analysis by developer roles, we see that top contributors are often involved in more than 10 conflicting scenarios while occasional contributors are often involved in less than 5 conflicting scenarios.

B. Proportion of Involvement in Conflicting Merge Scenarios ($RQ_{1.2}$)

In this section, we investigate the rate among all contributions and the conflicting contributions for the 2 972 contributors that are involved in merge conflicts.

In Figure 4, we show this rate divided into four groups: up to 25%, between 25% and 50%, between 50% and 75%, and greater than 75%. The first group ($\leq$ 25%) and last group ($>$ 75%) are the ones with more contributors. To illustrate the first group, a developer of the NETDATA project contributed to 1 085 merge scenarios and only two of them resulted in merge conflicts, i.e., this developer has a conflicting rate of 0.18%.

Regarding the first group ($\leq$ 25%), we find that 484 contributors have a very small conflicting rate (i.e., up to 5%). These are probably top contributors (at project level and merge-scenario level) that skip conflicts. Regarding



(a) Project-Level Contributions

(b) Merge Scenario-Level Contributions

Fig. 5. Rate of Conflicting Contributions by the Amount of Merge Scenario Contributions at Project-Level and Merge-Scenario Level

the last group ($>$ 75%), we found that 1 019 contributors introduce merge conflicts into all their contributions. This finding represents 34.3% of the contributors involved with merge conflicts and 4% of all subject contributors.

In Figure 5, we present the share of conflicting contributions by the amount of merge scenarios contributions at project- and merge-scenario-level. As we observe in this figure, most contributors from all developer roles belong to the group of contributors with conflicting rate less than 25%. On the other hand, we see that occasional (at project level and merge-scenario level) developers have more than 25% of developers in the group with a rate of conflicting merge scenarios above 75%. Hence, we assume that most contributors with a very high conflicting rate do not often contribute to the project. Looking at our data, we observed that the code contributions of the occasional developers whose rate of conflicting contributions equals 100% vary from 1 to 25 merge scenarios. In fact, 997 of them contributed to up to 5 merge scenarios.

$RQ_{1.2}$ Finding: About 50% of the contributors involved in merge conflicts have a rate of 25% of their contributions into conflicts. Most of these developers are top contributors at project- and merge-scenario levels. Surprisingly, 34% of conflicting contributors have merge conflicts in all of their contributions. Given the small number of contributions and supported by our data, we see that most of them are occasional contributors.

V. Top Contributors and Top Conflicting Contributors (RQ_2)

In this section, we provide an overview of the top contributors (Section V-A) as well as of the top conflicting contributors of each project (Section V-B).

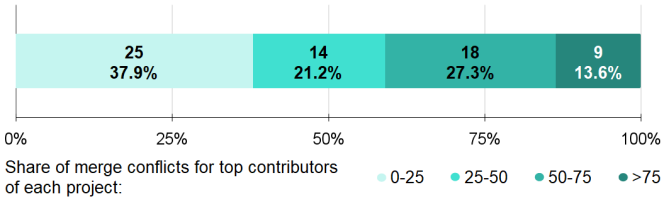


Fig. 6. Share of Conflicting Merge Scenarios for the Top Contributor of each Subject Project

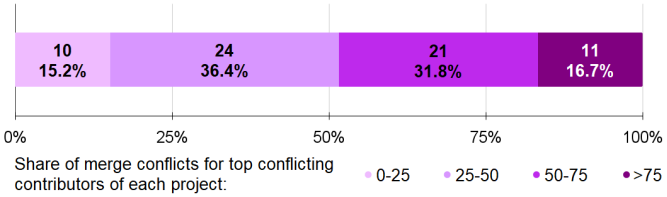


Fig. 7. Share of Conflicting Merge Scenarios for the Top Conflicting Contributor of each Subject Project

A. Most Active Contributors

In Figure 6, we present the percentage of conflicting merge scenarios for top contributors of each subject project. We can see in this figure that 37.9% of the top contributors participated in up to 25% of the conflicting merge scenarios. We also see that 21.2% of the top contributors were involved in 25-50% of the conflicting merge scenarios, 27.3% of the top contributors participated in 50-75% of the conflicting merge scenarios, and 13.6.% of these top contributors participated in more than 75% of the conflicting merge scenarios. In other words, in 27 out of 66 projects the top contributors were involved with more than 50% of the conflicting merge scenarios.

In 12 projects, the top contributors participate in more than a thousand merge scenarios. However, in only four projects, these developers are involved with the most conflicting merge scenarios in their project. One example is a BOOTSTRAP [46] project contributor. She contributed to 3 167 merge scenarios and was involved in 306 conflicting merge scenarios. It represents participation in 72.5% of the conflicting merge scenarios of this project. We further detail this case in Section VI while answering RQ_3.

B. Most Active Conflicting Contributors

In Figure 7, we depict the percentage of conflicting merge scenarios for top conflicting contributors of each subject project. For short, we observe in this figure that 15.2% of the contributors participated in 25% of the conflicting merge scenarios while 36.4%, 31.8%, and 16.7% of these contributors were involved with 25-50%, 50-75%, and 75% of the conflicting merge scenarios, respectively. In other words, in 32 out of 66 projects the top conflicting contributor is involved with more than 50% of the conflicting merge scenarios.

In a few cases, top conflicting contributors participated in more than 100 conflicting merge scenarios. For instance, in the case of project D3 [11] the same developer is involved with 364 conflicting scenarios. It represents 84.26% of the conflicting merge scenarios of that project.

Based on a manual verification, we compare the contributors that represent both charts of Figures 6 and 7. We see that they are the same developers in 42 projects; i.e., the developer that contributes to more merge scenarios in the project is also the one involved with the majority of conflicting merge scenarios. Interestingly, in two projects, the top contributor is involved with the same number of conflicting merge scenarios than other contributors. As the participation of top contributors and the top conflicting contributors in the majority of conflicting scenarios often happens, it is interesting to look deeper at their contributions aiming at identifying coding practices that lead to merge conflicts. We perform this analysis in Section VI.

RQ_2 Finding: In 42 out of 66 projects, the top contributor is also the top conflicting contributor. In 39.4% of the projects, the top contributors participate in the majority of conflicting merge scenarios in their project. Similarly, in 48.49% of the projects, the top conflicting merge scenario contributors are involved with the majority of the conflicting merge scenarios that happened in their respective projects. In other words, in 27 and 32 projects, the top contributor and top conflicting contributors are involved with more than 50% of the conflicting scenarios, respectively. It may be an indicator that these contributors follow bad practices which make them participate in the majority of the merge conflicts of their projects.

VI. Analysis of Conflicting Changes (RQ_3)

Looking at our data, we found three projects (CS-NOTES, MARDOWN-HERE, SYSTEMS-DESIGN-PRIMER) that the top conflicting contributors participated in all conflicting merge scenarios. However, since the number of conflicting merge scenarios is small (i.e., up to 37) in these cases, we decided to focus on the five projects with more than 1 thousand conflicting merge scenarios: D3, BOOTSTRAP, METEOR, WEBPACK, FREECODECAMP. To preserve the identity of the top conflicting contributors of these projects, we anonymously nickname them C1, C2, C3, C4, and C5.

We organize this section according to the three analyses described in Section III-C to answer RQ_3 with data of these 5 contributors.

A. Contributions of Top Conflicting Contributors

In Table I, we present the overall contributions of the top five conflicting contributors considering their absolute and relative numbers of conflicting merge scenarios. We organize Table I into three major parts. In the first one (Project), we show the overall number of merge scenarios

TABLE I
Overview Contributions of the Top-Five Conflicting Contributors

	Project		Contributor		Conflict	
Id	**#MS**	**#CMS**	**#MS**	**#CMS**	**#CMS**	**%CMS**
C1	1 076	432 (40.15%)	866 (80.48%)	397 (91.90%)	364 (84.26%)	42.03
C2	6 665	422 (6.33%)	3 167 (47.52%)	358 (84.83%)	306 (72.51%)	9.66
C3	2 737	345 (12.60%)	841 (30.73%)	178 (51.59%)	134 (38.84%)	15.93
C4	2 486	132 (5.31%)	1 563 (62.87%)	116 (87.88%)	89 (67.42%)	5.69
C5	4 665	108 (2.31%)	998 (21.39%)	100 (92.59%)	90 (93.33%)	9.02

***#MS**: number of merge scenarios, **#CMS**:number of conflicting merge scenarios, **%CMS**: percentage of CMS by MS of the contributor.*

(#MS) and conflicting merge scenarios (#CMS) in the five selected projects. In the second part (Contributor), we depict the #MS and #CMS that the contributor participated in. That is, she changed at least one line of code in these merge scenarios. Finally, in the third part (Conflict), we indicate #CMS that the contributor was actually involved in conflicting code. That is, their committed code was responsible for triggering a merge conflict in the respective merge scenario.

For instance, the project of C1 has 432 conflicting merge scenarios, but this contributor only participated in 397 of them. From these, C1 committed conflicting code into 364 merge scenarios. Moreover, we see 1 076 merge scenarios in her project and C1 contributed to 866 them. Hence, she contributed to 80.48% of the merge scenarios of this project. C2, C3, C4, and C5 participate in 47.52%, 30.73%, 62.87%, and 21.39% of the merge scenarios of their projects, respectively. Only C1 and C4 participated in the majority of the merge scenarios in their projects. Therefore, the high participation of these contributors on conflicting merge scenarios does not necessarily come only from high contributions to the majority of merge scenarios in the project.

Focusing on the percentages of columns #CMS (Contributor), we note that all contributors changed more than 50% of conflicting merge scenarios. Similarly, all contributors, except C3, committed code causing conflicts in the majority of the conflicting scenarios, as we see in the #CMS (Conflict) column of Table I. This is an indicator that these developers indeed influence the number of merge conflicts.

Aiming at finding out whether these contributors are isolated cases or recurrent practices of several developers in these projects, we compare the rate of conflicting merge scenarios in the project (3rd column) and the rate of conflicting merge scenarios of the contributor (last column in Table I). For instance, we see that the rate of conflicting merge scenarios of C1's project is 40.15%. However, C1 has a higher rate of conflicting merge scenarios (42.03%) than its project. In fact, all contributors presented in Table I have a higher rate of conflicting merge scenarios than the general average data of their respective projects. In the case of C5, while the project rate of conflicting merge scenarios is 2.31%, she has a rate of 9.02%; i.e., four times higher. These results suggest that these five contributors have relevant impact on the high percentages of merge conflicts in their projects. Therefore, a deeper analysis in their code changes is needed to uncover what these specific developers do.

RQ$_3$ Finding 1: Top conflicting contributors are not always involved with most merge scenarios in their respective projects, although they participated in most conflicting merge scenarios. In fact, their committed code is responsible for more merge conflicts that the average rate of conflicting merge scenarios in their projects which suggests that their coordination is crucial to the project success.

B. Project Contribution Rules

Previous work [1], [3], [27] have shown that several merge conflicts arise from formatting or from the location of code changes in a file. An easy way to minimize merge conflicts due to formatting and the location of the changes is through the definition of contribution rules. Contribution rules normally define the contribution process as well as the code style. Aiming at identifying whether these five projects have defined contribution rules, we looked at their *README.md* file searching for links to other files or definitions of contribution rules. Except for project D3, we found contribution rules for the other four projects. Furthermore, we observe that developers often defined contribution rules in a file named *CONTRIBUTING.md*. Aiming at finding out if the rate of conflicting merge scenarios increased or reduced after creating this file, we get the date this file was merged to the main branch for the first time and compare the number of merge scenarios with the number conflicting merge scenarios before and after the creation of this file.

In Table II, we present a summary of this analysis for the four projects with contribution rules defined. As we see in this table, most merge scenarios were created after the creation of the contribution rules file for all projects. For two of them, the creation of contribution rules dramatically reduced the share of conflicting merge scenarios from 14.96% to 3.73% and from 5.82% to 0.83%. For the other two cases, it does not seem to have an impact on the conflicting merge scenarios rate. Note that

TABLE II
COMPARING CONFLICTING MERGE SCENARIOS BEFORE AND AFTER THE CREATION OF CONTRIBUTION RULES

Project Name	#MS before	#MS after	%CMS before	%CMS after
bootstrap (C2)	1 544 (231)	5 121 (191)	14.96%	3.73%
meteor (C3)	526 (51)	2 211 (294)	9.70%	13.30%
webpack (C4)	99 (5)	2 387 (127)	5.05%	5.32%
freeCodeCamp (C5)	1 391 (81)	3 274 (27)	5.82%	0.83%

***#MS**: number of merge scenarios, **%CMS**: percentage of conflicting merge scenarios. The numbers in parenthesis stand for the number of conflicting merge scenarios.*

TABLE III
OVERVIEW OF CHANGED FILES IN PROJECTS OF TOP-FIVE CONFLICTING CONTRIBUTORS

Cont.	#Dist. Files	#Dist Files with Confl.	#File versions	#File versions with Confl.
C1	1 824	79 (4.33%)	39 202	603
C2	5 930	224 (3.78%)	142 533	1 717
C3	7 020	313 (4.46%)	114 757	1 129
C4	4 636	153 (3.30%)	92 266	275
C5	2 876	72 (2.50%)	52 875	138

***Cont.**: Contributor, **#Dist. Files**: number of distinct files, **#Dist. Files with Confl.**: number of distinct files with merge conflicts, **#File versions**: number of file versions, **#File versions with Confl.**: number of file versions with merge conflicts*

we evaluate neither the quality of the contribution rules nor the number of contributors in these two time frames; we only check if the file with contribution rules exists or not. Hence, other factors may influence the emergence of merge conflicts (limitations of this approach are discussed in Section VII). Anyway, we believe that adding contribution rules helps to avoid the emergence of simple merge conflicts and, consequently, the general number of merge conflicts.

RQ_3 Finding 2: The analysis of contribution rules diverges among projects. However, for most projects, we observed that the contribution rules may reduce the emergence of merge conflicts.

C. Changed Files

We have the feeling that some files are conflict-prone because they change more often than other files or because they are somehow more central to the project. We investigated these assumptions in 4 ways.

First, in Table III, we present an overview of distinct files (i.e., files with different paths and names), all file versions (i.e., all versions of distinct files) for the five projects that C1-C5 contribute to as well as the number of distinct files and file versions with merge conflicts. A new version of a file arises when one or more developers change it in a merge scenario. To illustrate the data of Table III, look at the project of C1. In this project, there are 1 824 distinct files of which 79 have merge conflicts. From these distinct files, we find 39 202 versions of which 603 have merge conflicts. The rate of merge conflicts in distinct files is 4.33%, 3.78%, 4.46%, 3.30%, 2.50% for the projects that C1, C2, C3, C4, and C5 contribute to, respectively. Hence, if all files change equally and considering an average of 4% conflict rate, the chance of emerging merge conflicts would be of around 1 in each 25 files changed. Looking at the number of file versions (*#File versions* column), we see that files often change. For instance, the 5 930 distinct files of C2's project changed 142 533 times. If all files change equally (i.e., each file changed 24.90 times), the chance would be of at least one merge conflict in each file. However, as the 1 717 merge conflicts were distributed over only 224 files, if they change equally, each of these files would have around 8 merge conflicts. Therefore, the main conclusion we can draw from this table is that files change often and merge conflicts are concentrated in a few files.

Second, in Table IV, we present the top-three files changed for the projects of C1-C5 with the number of merge scenarios emerging conflicts in these files and the percentage it represents. To illustrate, the two most changed files for the project that C1 contributed to (*d3.js* and *d3.min.js*) changed in 732 and 715 merge scenarios and had merge conflicts in 70 and 291 of them, respectively. It means that these files changed in 68.03% and 66.45% (732 and 715 out of 1 076) of the merge scenarios of this project and in 9.56% and 40.70% of the times they changed, merge conflicts emerged. Therefore, we see that these files have a greater chance to change and to arise merge conflicts than the average of files of this project (9.56% and 40.70% against 4.33%). On the other hand, the third most changed file for the same project (*package.json*) had only 9 conflicts in the 425 merge scenarios that it changed (2.12%). From the top three files most change for the project of C5, only the first one have merge conflicts. This table supports our understanding that files that often change usually have more merge conflicts than files that change less frequently. However, only the number of changes may lead to wrong conclusions and the outcome may change from project to project.

Third, in Table V, we present the top-three conflicting files for the projects of C1-C5 with the number of merge scenarios emerging conflicts in these files and the percentage it represents. To illustrate, the top three conflicting files for the project that C2 contribute to (*bootstrap-1.2.0.css*, *bootstrap.css*, and *bootstrap-1.0.0.css*) have merge conflicts in 77.78%, 21.71%, and 25.93% of the times they changed, respectively. On the other hand, the top-three conflicting files for the C5 project cause merge conflicts in less than 2.00% of the times they changed. Note that only 5 files appear in both Table IV and Table V and that despite the majority of files in these five projects are JAVASCRIPT files, only 5 out of 15 files of Table V are JAVASCRIPT files. With these analyses, we conclude that frequently changed files are more conflict-prone than

TABLE IV
TOP-THREE FILES CHANGED OVER TIME FOR PROJECTS OF C1-C5

Cont.	Top-3 files	#MS	#CMS	%CMS
C1	d3.js	732	70	9.56
	d3.min.js	715	291	40.70
	package.json	425	9	2.12
C2	docs/index.html	2 384	66	2.77
	dist/css/bootstrap.min.css	2 008	52	2.59
	dist/css/bootsatrap.css	1 928	137	7.11
C3	packages/meteor-tool/package.js	595	69	11.60
	.../meteor-release-experimental.json	544	54	9.93
	packages/webapp/package.js	497	24	4.83
C4	package.json	1 030	39	3.79
	lib/Compilation.js	651	7	1.08
	lib/Parser.js	435	4	0.92
C5	package.json	826	9	1.09
	seed/.../basic-javascript.json	677	0	0.00
	server/boot/user.js	479	0	0.00

***Cont.**: Contributor, **#MS**: number of merge scenarios, **#CMS**: number of conflicting merge scenarios, and **%CMS**: percentage of conflicting merge scenarios over all merge scenarios*

TABLE V
TOP-THREE CONFLICTING FILES OVER TIME FOR PROJECTS OF C1-C5

Cont.	Top-3 files	#MS	#CMS	%CMS
C1	d3.min.js	715	291	40.70
	d3.v2.min.js	203	88	43.35
	d3.js	732	70	9.56
C2	bootsatrap-1.2.0.css	216	168	77.78
	bootsatrap.css	631	137	21.71
	bootsatrap-1.0.0.css	324	84	25.93
C3	packages/meteor-tool/package.js	595	69	11.60
	.../meteor-release-experimental.json	544	54	9.93
	packages/babel-compiler/packages.js	396	41	10.35
C4	package.json	1 030	39	3.79
	test/.../StatsTestcases.tests.js.snap	260	14	5.38
	yarn.lock	379	8	2.11
C5	seed/challenges/basic-javascript.json	471	9	1.91
	package.json	826	9	1.09
	README.md	444	7	1.58

***Cont.**: Contributor, **#MS**: number of merge scenarios, **#CMS**: number of conflicting merge scenarios, and **%CMS**: percentage of conflicting merge scenarios over all merge scenarios*

others, but that the results may vary from project to project. Hence, merge conflict prediction strategies may have better performance when learning with previous merge scenarios of a project, i.e., there is no silver bullet strategy across projects. However, we also argue that the performance will depend on the specific characteristics of a project (e.g., life-cycle, domain, type of project, and programming language).

Fourth, with the knowledge acquired from previous discussion, we compute the Spearman's rank correlation between the number of times each file changed and the number of times code changes caused merge conflicts for all subject projects. Spearman's rank correlation is more adequate for our analysis than Pearson's correlation because our data do not have a normal distribution [22]. We find a Spearman's rank correlation of 0.26 with 99% significance level (p-value $< 2.2^{e-16}$). Since some files are more conflict-prone than others (see discussion of the 2nd and 3rd investigations above), we refine our analysis including only files with merge conflicts. In this analysis, we find a Spearman's rank correlation of 0.55 with 99% significance level (p-value $< 2.2^{e-16}$). Moreover, we believe that other data refinements, artificial intelligence techniques, and analysis considering characteristics of each project may increase even more this correlation.

RQ3 Finding 3: The analysis on changed files reveals that files changing more often are conflict-prone. In addition, we also found that merge conflicts are concentrated in a few files and argue that better predictions of merge conflicts can be achieved by considering historical and characteristics of each project.

VII. THREATS TO VALIDITY

We discuss below possible construct, internal, conclusion, and external threats to the study validity [50].

Construct Validity. Construct validity concerns inferring the result of the experiments to the concept or theory [50]. For instance, we used only metrics based on frequency of changes to classify developers between top and occasional contributors. This poses a threat that the metrics do not accurately capture actual scenarios of distributed software development. However, we believe this threat does not invalidate our main findings since existing evidence [28], [48] indicates that these metrics accurately reflect the developers' perception. Additionally, to answer RQ_3, we focus only on the existence of contribution rules in a specific file (*CONTRIBUTING.md*). For instance, we did no evaluate neither the quality of these contribution rules. A further study could investigate other characteristics of the contribution rules, such as its extension and clarity.

Internal Validity. Threats to internal validity are influences that can affect the independent variable to causality [50]. In our case, this threat may refer to the chosen dataset. For instance, since we selected projects from different programming languages, one or a few languages could have dominated our dataset. To minimize this threat, we excluded less popular JavaScript projects until they do not represent more than 50% our dataset. We excluded 6 projects with this filter. Another threat is the choice of only 5 subject projects to answer RQ_3. We selected them because they are large (i.e., 22.39% of all investigated merge scenarios and 36.43% of all conflicting merge scenarios). Hence, we investigated 1 439 out of 3 950 conflicting merge scenarios. It brings a confidence level of 99% within ±3% margin of error that we are measuring a significant amount of conflicting merge scenarios.

Conclusion Validity. Threats to the conclusion validity are concerned with issues that affect the ability to draw the correct conclusion between the treatment and the outcome [50]. In our study, a potential threat to conclusion validity is the reliability of the data extraction, since not all information was clear to answer the research questions. As a result, some data had to be inferred and sometimes

cross-discussions among the paper authors took place to reach a common agreement.

External Validity. Threats to external validity are conditions that limit our ability to generalize the results of our paper [50]. Our data come from Git and GitHub platforms and we restricted our analyses to projects following the three-way merge pattern. Therefore, we cannot generalize our results to other platforms, projects, development patterns, and developers. While more research is needed to allow generalization, we select and analyze a largely used platform and a high number of software projects from various domains, programming languages, sizes, and coordination practices.

VIII. Related Work

In this section, we discuss studies related to merge conflicts and developer classification to support the organizational structure of open-source software (OSS).

Merge conflicts have a negative effect on project's objectives compromising the project success, especially when arising frequently [21], [40]. Hence, researchers have investigated, for instance, merge strategies [3], [4], prediction strategies [9], [21], [26], awareness tools [6], [26], [40], tried to understand types of code changes related to conflicts [1], [17], [28], and strategies to efficiently resolve merge conflicts [18], [23], [34], [48]. For instance, Ji et al. [23] empirically investigated merge conflicts and resolutions in Git rebase scenarios. Their results suggest that rebases are often performed (about 8% of pull requests have rebases and 41% have two or more rebases). Several rebases are performed for the purpose of reducing reviewing changes in pull requests. About 25% of rebases involve textual conflicts, and approximately 29% of conflict rebases involve the introduction of new tokens. This study is particularly interesting because it emphasizes how complex the conflicting merge scenarios can be in practice, and how important it is to assist developers in resolving merge conflicts. As another example, Gonzalez and Fraternali [18] investigated merge conflict resolution, but now through the proposal and evaluation of a strategy that extends Git Rerere (REuse REcorded REsolution) with novel features. Git Rerere was designed to automatically resolve conflicts that are similar to previously solved conflicts. Their results suggest that the tool can resolve about 49% of the conflicts generated during the merge process, with most solutions being similar or the same as solutions manually performed by developers. This study is particularly interesting because it suggests that several merge conflicts have limited size, i.e., they involve one or two lines of code, and their solutions are significantly simple. Such finding is inline with our discussion regarding the importance of project contribution rules to resolve simple merge conflicts (see Section VI-B).

On the other hand, researchers have classified developers aiming at understanding the organizational structure of OSS [7], [10], [13], [24], [33], [39], [44]. For instance, Mockus et al. [33] found empirical evidence for the Mozilla browser and the Apache Web server that a small number of developers are responsible for approximately 80% of the code modifications. Their approach consists of counting the number of commits made by each developer and then computing a threshold at the 80% percentile. Although using a different approach, their result is inline with our results and also inline with previous work [13], [39], [44]. As another example, Joblin et al. [24] empirically classified developers into core and peripheral to model the organizational structure using network metrics (e.g., degree- and eigenvector-centrality) and analyzed how the set of core developers changed over time.

Despite the number of studies exploring merge conflicts and developer roles separately, we lack studies specifically focused on the contribution degree of different types of OSS developers to the occurrence of conflicting merge scenarios. We fill this gap performing three analyses first getting an overview on the topic and later looking deeper at the impact of code contribution patterns of developers of 5 projects as well as to their source code changes and the influence of project contribution rules on the emergence of merge conflicts. Our results bring an understanding of which developer roles and source code changes are related to merge conflicts. Furthermore, we provide implications and directions to researchers and practitioners avoiding conflicting merge scenarios.

IX. Conclusion

In this paper, we presented a large-scale quantitative study which investigated what is behind developer contributions on conflicting merge scenarios. Our results suggest that 62.6% of the contributors are involved only once with merge conflicts and only 3.8% are involved with more than 10 merge conflicts. Based on these results, we argue that training this small group of contributors could significantly reduce the number of merge conflicts. We also found that in 42 projects out of 66 the top contributors is also the top conflicting contributors.

As future work, we plan to expand our analysis to include fine-grained measures like the size of conflicting files or compare the conflicting lines of code by top-contributors and include data from proprietary software systems. Another possible direction for future work is to survey developers with a large rate of conflicting contributions to investigate their perception of bad practices that cause merge conflicts.

References

[1] P. Accioly, P. Borba, and G. Cavalcanti, "Understanding semi-structured merge conflict characteristics in open-source Java projects," *Empirical Software Engineering (EMSE)*, vol. 23, pp. 2051–2085, 2018.

[2] Antirez, "Redis," https://github.com/antirez/redis, 2023, [Online; accessed 14-August-2023].

[3] S. Apel, O. Leßenich, and C. Lengauer, "Structured merge with auto-tuning: Balancing precision and performance," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2012, pp. 120–129.

[4] S. Apel, J. Liebig, B. Brandl, C. Lengauer, and C. Kästner, "Semistructured merge: Rethinking merge in revision control systems," in *Proceedings of the SIGSOFT Symposium and the European Conference on Foundations of Software Engineering (ESEC/FSE)*, 2011, pp. 190–200.

[5] V. Basili and H. Rombach, "The TAME project: Towards improvement-oriented software environments," *IEEE Transactions on Software Engineering (TSE)*, vol. 14, no. 6, pp. 758–773, 1988.

[6] J. Biehl, M. Czerwinski, G. Smith, G. Robertson, and B. Bailey, "FASTDash: A visual dashboard for fostering awareness in software teams," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2007, pp. 1313–1322.

[7] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," in *Proceedings of the Joint Meeting of the European Software Engineering Conference and the SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2008, pp. 24–35.

[8] H. Borges and M. T. Valente, "What's in a GitHub star? Understanding repository starring practices in a social coding platform," *Journal of Systems and Software (JSS)*, vol. 146, pp. 112–129, 2018.

[9] Y. Brun, R. Holmes, M. Ernst, and D. Notkin, "Proactive detection of collaboration conflicts," in *Proceedings of the SIGSOFT Symposium and the European Conference on Foundations of Software Engineering (ESEC/FSE)*, 2011, pp. 168–178.

[10] K. Crowston, K. Wei, Q. Li, and J. Howison, "Core and periphery in free/libre and open source software team communications," in *Proceedings of the Hawaii International Conference on System Sciences (HICSS)*, 2006, pp. 118a:1–118a:7.

[11] D3, "D3," https://github.com/d3/d3, 2023, [Online; accessed 14-August-2023].

[12] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: Transparency and collaboration in an open software repository," in *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW)*, 2012, pp. 1277–1286.

[13] T. Dinh-Trong and J. Bieman, "The FreeBSD project: A replication case study of open source development," *IEEE Transactions on Software Engineering (TSE)*, vol. 31, no. 6, pp. 481–494, 2005.

[14] Facebook, "React," https://github.com/facebook/react, 2023, [Online; accessed 14-August-2023].

[15] E. Fernandes, A. Chávez, A. Garcia, I. Ferreira, D. Cedrim, L. Sousa, and W. Oizumi, "Refactoring effect on internal quality attributes: What haven't they told you yet?" *Information and Software Technology (IST)*, vol. 126, p. 106347, 2020.

[16] Getlantern, "Lantern," https://github.com/getlantern/lantern, 2023, [Online; accessed 14-August-2023].

[17] G. Ghiotto, L. Murta, M. Barros, and A. van der Hoek, "On the nature of merge conflicts: A study of 2,731 open source Java projects hosted by GitHub," *IEEE Transactions on Software Engineering (TSE)*, vol. 46, no. 8, pp. 892–915, 2020.

[18] S. L. Gonzalez and P. Fraternali, "Almost Rerere: Learning to resolve conflicts in distributed projects," *IEEE Transactions on Software Engineering (TSE)*, vol. 49, no. 4, pp. 2255–2271, 2022.

[19] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: The contributor's perspective," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2016, pp. 285–296.

[20] G. Gousios, A. Zaidman, M.-A. Storey, and A. Van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2015, pp. 358–368.

[21] M. Guimarães and A. Silva, "Improving early detection of software merge conflicts," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2012, pp. 342–352.

[22] H. Z. Jerrold, "Significance testing of the Spearman rank correlation coefficient," *Journal of the American Statistical Association (JASA)*, vol. 67, no. 339, pp. 578–580, 1972.

[23] T. Ji, L. Chen, X. Yi, and X. Mao, "Understanding merge conflicts and resolutions in Git rebases," in *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 70–80.

[24] M. Joblin, S. Apel, C. Hunsen, and W. Mauerer, "Classifying developers into core and peripheral: An empirical study on count and network metrics," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2017, pp. 164–174.

[25] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. German, and D. Damian, "The promises and perils of mining GitHub," in *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 92–101.

[26] B. Kasi and A. Sarma, "Cassandra: Proactive conflict minimization through optimized task scheduling," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2013, pp. 732–741.

[27] S. Larsén, J.-R. Falleri, B. Baudry, and M. Monperrus, "Spork: Structured merge for Java with formatting preservation," *IEEE Transactions on Software Engineering (TSE)*, vol. 49, no. 1, pp. 64–83, 2023.

[28] O. Leßenich, J. Siegmund, S. Apel, C. Kästner, and C. Hunsen, "Indicators for merge conflicts in the wild: Survey and empirical study," *Automated Software Engineering (ASE)*, vol. 25, pp. 279–313, 2018.

[29] J. Linåker, H. Munir, K. Wnuk, and C.-E. Mols, "Motivating the contributions: An open innovation perspective on what to share as open source software," *Journal of Systems and Software (JSS)*, vol. 135, pp. 17–36, 2018.

[30] W. Mahmood, M. Chagama, T. Berger, and R. Hebig, "Causes of merge conflicts: A case study of ElasticSearch," in *Proceedings of the International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS)*, 2020, pp. 1–9.

[31] S. Matthiesen, P. Bjørn, and C. Trillingsgaard, "Implicit bias and negative stereotyping in global software development and why it is time to move on!" *Journal of Software: Evolution and Process (S:EP)*, vol. 35, no. 5, p. e2435, 2023.

[32] Microsoft, "Typescript," https://github.com/microsoft/TypeScript, 2023, [Online; accessed 14-August-2023].

[33] A. Mockus, R. Fielding, and J. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002.

[34] N. Nelson, C. Brindescu, S. McKee, A. Sarma, and D. Dig, "The life-cycle of merge conflicts: processes, barriers, and strategies," *Empirical Software Engineering (EMSE)*, vol. 24, no. 5, pp. 2863–2906, 2019.

[35] M. Owhadi-Kareshk, S. Nadi, and J. Rubin, "Predicting merge conflicts in collaborative software development," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–11.

[36] R. Padhye, S. Mani, and V. Sinha, "A study of external community contribution to open-source projects on GitHub," in *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 332–335.

[37] P. Rigby, D. German, L. Cowen, and M.-A. Storey, "Peer review on open-source software projects: Parameters, statistical models, and theory," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 4, pp. 1–33, 2014.

[38] G. Robles, J. González-Barahona, C. Cervigón, A. Capiluppi, and D. Izquierdo-Cortázar, "Estimating development effort in free/open source software projects by mining software repositories: A case study of OpenStack," in *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 222–231.

[39] G. Robles, J. Gonzalez-Barahona, and I. Herraiz, "Evolution of the core team of developers in libre software projects," in *Proceedings of the International Working Conference on Mining Software Repositories (MSR)*, 2009, pp. 167–170.

[40] A. Sarma, D. Redmiles, and A. van der Hoek, "Palantir: Early detection of development conflicts arising from parallel code changes," *IEEE Transactions on Software Engineering (TSE)*, vol. 38, no. 4, pp. 889–908, 2012.

[41] L. Singer, F. Figueira Filho, B. Cleary, C. Treude, M.-A. Storey, and K. Schneider, "Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators," in *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW)*, 2013, pp. 103–116.

[42] M.-A. Storey, A. Zagalsky, F. Figueira Filho, L. Singer, and D. German, "How social and communication channels shape and challenge a participatory culture in software development," *IEEE Transactions on Software Engineering (TSE)*, vol. 43, no. 2, pp. 185–204, 2016.

[43] A. Svyatkovskiy, S. Fakhoury, N. Ghorbani, T. Mytkowicz, E. Dinella, C. Bird, J. Jang, N. Sundaresan, and S. Lahiri, "Program merge conflict resolution via neural transformers," in *Proceedings of the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2022, pp. 822–833.

[44] A. Terceiro, L. R. Rios, and C. Chavez, "An empirical study on the structural complexity introduced by core and peripheral developers in free software projects," in *Proceedings of the Brazilian Symposium on Software Engineering (SBES)*, 2010, pp. 21–29.

[45] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in GitHub," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2014, pp. 356–366.

[46] Twbs, "Bootstrap," https://github.com/twbs/bootstrap, 2023, [Online; accessed 14-August-2023].

[47] G. Vale, E. Fernandes, E. Figueiredo, and S. Apel, "Behind developer contributions on conflicting merge scenarios - supplementary website," https://gustavovale.github.io/behind-developer-contributions-on-conflicting-merge-scenarios/index.html, 2023, [Online; accessed 14-August-2023].

[48] G. Vale, C. Hunsen, E. Figueiredo, and S. Apel, "Challenges of resolving merge conflicts: A mining and survey study," *IEEE Transactions on Software Engineering (TSE)*, vol. 48, no. 12, pp. 4964–4985, 2021.

[49] G. Vale, A. Schmid, A. Santos, E. De Almeida, and S. Apel, "On the relation between GitHub communication activity and merge conflicts," *Empirical Software Engineering (EMSE)*, vol. 25, pp. 402–433, 2020.

[50] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, 1st ed. Springer Science & Business Media, 2012.