Journal of Software Engineering Research and Development, 2021, 9:14, doi: 10.5753/jserd.2021.1911

Attributes that may raise the occurrence of merge conflicts

José William Menezes [**Universidade Federal do Acre** | ***jose.william@sou.ufac.br***]
Bruno Trindade [**Universidade Federal do Acre** | ***bruno.trindade@sou.ufac.br***]
João Felipe Pimentel [**Universidade Federal Fluminense** | ***jpimentel@ic.uff.br***]
Alexandre Plastino [**Universidade Federal Fluminense** | ***plastino@ic.uff.br***]
Leonardo Murta [**Universidade Federal Fluminense** | ***leomurta@ic.uff.br***]
Catarina Costa [**Universidade Federal do Acre** | ***catarina.costa@ufac.br***]

Abstract

Collaborative software development typically involves the use of branches. The changes made in different branches are usually merged, and direct and indirect conflicts may arise. Some studies are concerned with investigating ways to deal with merge conflicts and measuring the effort that this activity may require. However, the investigation of factors that may reduce the occurrence of conflicts needs more and deeper attention. This paper aims at identifying and analyzing attributes of past merges with and without conflicts to understand what may induce direct conflicts. We analyzed 182,273 merge scenarios from 80 projects written in eight different programming languages to find characteristics that increase the chances of a merge to have a conflict. We found that attributes such as the number of changed files, the number of commits, the number of changed lines, and the number of committers demonstrated to have the strongest influence in the occurrence of merge conflicts. Moreover, attributes in the branch that is being integrated seem to be more influential than the same attributes in the receiving branch. Additionally, we discovered positive correlations between the occurrence of conflicts and both the duration of the branch and the intersection of developers in both branches. Finally, we observed that PHP, JavaScript, and Java are more prone to conflicts.

Keywords: *Version Control, Merge Conflicts, Conflict prediction*

1 Introduction

Software development normally involves collaboration among members of the project team. This collaborative development is supported by a Version Control System (VCS). Often, when there is a need to develop new features or fix bugs, developers choose to create a branch, which is a separate development line. This separate development line helps teams to focus on their tasks, without prematurely worrying about how it affects other parts of the software (Bird et al., 2011). However, the use of branches can cause problems, as changes made in different branches are usually merged, and direct and indirect conflicts may arise (Brindescu et al., 2020b; Costa et al., 2016; Sarma et al., 2011; Brun et al., 2011). According to Bird et al. (2011), the effort involved in the merge process is dependent on how much work went on in the branches.

Some studies investigate ways to deal with merge conflicts by proactively detecting changes that can lead to conflicts (Brun et al., 2011; Sarma et al., 2011), identifying merge characteristics (Accioly et al., 2018; Ghiotto et al., 2018; Vale et al., 2020), investigating the characteristics of difficult merge conflicts (Brindescu et al., 2020b), and examining the decisions usually made to resolve conflicts (Accioly et al., 2018; Ghiotto et al., 2018). However, only recently, some studies started to investigate factors that may induce the occurrence of conflicts. Dias et al. (2020) verify how seven factors related to modularity, size, and timing of developers' contributions affect conflict occurrence. Leßenich et al. (2018) analyze the predictive power of seven indicators, such as the number, size, and scattering degree of commits in each branch, to forecast the number of merge conflicts. In the same direction, Owhadi-Kareshk et al. (2019) investigate the predictive power of nine lightweight Git feature sets, such as the number of changed files in both branches, the number of commits and developers, and the duration of the development of the branch. Finally, Vale et al. (2020) investigate the role of communication activity and the number of modified lines, chunks, files, developers, commits, and days that a merge scenario lasts in the increase or reduction of merge conflicts.

Similar to Dias et al. (2020), Leßenich et al. (2018), Owhadi-Kareshk et al. (2019), and Vale et al. (2020), we assume that by analyzing attributes of past merges, it is possible to identify characteristics that may increase the chances of having a merge conflict. However, in addition to the attributes investigated by those authors (e.g., isolation, number of changed files, changed lines, commits, commit density, and developers), we analyzed some other attributes, such as the programming language, the frequency of one or more developers committing in both branches, and the existence of self-conflicts[1] (Zimmermann, 2007). As mentioned by Brindescu et al. (2020a), the changes in conflict are generally authored by two different developers, but merge conflicts can also happen between the edits of the same developer, in two different branches. Besides, in terms of the number of analyzed merges, our corpus is representative (it is only smaller than the corpus of Owhadi-Kareshk et al. (2019)), and our analysis of the number of commits, commit density, committers, and changed lines and files is performed by branch, not using averages. Finally, it is important to mention that some metrics with similar names in the related work are calculated differently.

Thus, our work aims at providing a more in-depth analysis of how a set of merge attributes can influence the occurrence of conflicts. To do so, we mined association rules from

[1] A self-conflict is a conflict among changes committed by the same developer.

182,273 merge scenarios extracted from 80 software projects hosted on GitHub, written in eight different programming languages. The following eight research questions guided the analysis:

- RQ1. How is the isolation of a branch related to the occurrence of merge conflicts? *Our intuition is that the longer the isolation time of the branches, the greater the likelihood of having conflicts.*

- RQ2. How is the number of commits related to the occurrence of merge conflicts? *Our intuition is that the greater the number of contributions in terms of commits in the branches, the greater the likelihood of having conflicts.*

- RQ3. How is the number of developers that performed commits related to the occurrence of merge conflicts? *Our intuition is that the greater the number of contributors in the branches, the greater the likelihood of having conflicts.*

- RQ4. How is the number of changed files related to the occurrence of merge conflicts? *Our intuition is that the greater the number of contributions in terms of changed files in the branches, the greater the likelihood of having conflicts.*

- RQ5. How is the number of changed lines related to the occurrence of merge conflicts? *Our intuition is that the greater the number of contributions in terms of changed lines in the branches, the greater the likelihood of having conflicts.*

- RQ6. How is the programming language related to the occurrence of merge conflicts? *We had no intuition about the programming language, but we would like to know if any language was more prone to have conflicts.*

- RQ7. How is the intersection of developers in both branches related to the occurrence of merge conflicts? *Our intuition is that the greater the number of contributors in both branches, the lesser the chances of having conflicts, because these developers are aware of the parallel changes*.

- RQ8. How prevalent is the occurrence of merge self-conflicts? *We had no intuition about the proportion of self-conflicts, but we would like to know if it is common in projects.*

The answers to these questions can provide insights on how software project teams' work may affect the occurrence or avoidance of merge conflicts. We found that the investigated attributes have a positive correlation with merges with conflicts. Notably, in the integrated branch, the number of changed files, the number of commits, the number of changed lines, and the number of committers have the strongest influence in the occurrence of conflicts among all attributes we analyzed. Surprisingly, having some developers committing in both branches also increases the chance of conflicts, but having no common developer or having exactly the same developers committing in both branches decreases the chance of conflicts. We also verified that three programming languages (PHP, JavaScript, and Java) are more prone to conflicts.

This paper is an extended version of a conference paper (Menezes et al., 2020) in which we answered six research questions, focused on the impact of the attributes time, commits, committers, changed files, intersection, and self-conflicts, in the occurrence of merge conflicts. This work complements our previous work by adding two new research questions and three new attributes, the number of changed lines, the commits density, and the programming language. Additionally, we detail the investigation of developer intersection and replace the old "some intersection" category by percentages of intersections in association rules. We also deep the analysis of self-conflicts, obtaining the number of self-conflicts per chunk instead of per file. After all, a file can have several pieces of conflicting code that are from the same or different developers. Hence, the analysis became more precise. In addition, we mine rules to verify the relation to the attributes in the occurrence of self-conflicts.

Besides this introduction, this paper is organized in 7 sections. In Section 2, we present the research steps followed. In Section 3, we present the results of our statistical analysis, the association rules, and the discussion about the self-conflicts. In Section 4, we present the answers to our research questions. In Section 5, we discuss threats to validity. In Section 6, we discuss the related work. Finally, in Section 7 we present the conclusion.

2 Materials and Methods

To answer the research questions presented in the introduction, we performed an exploratory study. The following steps, detailed in the following, compose our exploratory study: merge attributes definition, projects and merges selection, merges and attributes extraction, and data mining.

2.1 Merge Attributes Definition

The attributes were mainly derived from our research questions and defined in **Table 1**. We divided the attributes into **project attributes**, **merge attributes**, and **branch attributes**.

The **project attributes** are the predominant programming language, number of merges, number of analyzed merges (non-fast-forward), merges with conflicts, merges without conflicts, and self-conflicts.

The **merge attributes** are the information collected by the merge scenario, using the information present in both branches. The merge conflict occurrence (yes or no), timing metrics, and information about changes and developers in both branches.

The **branch attributes** are collected and presented by each branch[2] (B1 and B2). We do not adopt any aggregation of

[2]When referring to the identification of branches in merges, i.e., the distinction between branch 1 (B1) and branch 2 (B2), we borrow the reasoning of Chacon and Hamano (2009): "the first parent is the branch you were on when you merged, and the second is the commit on the branch that you merged in".

Table 1. Attributes

#	Attributes	Definition
		Project attributes
a)	Programming language	Predominant programming language.
b)	Total of merges	Number of merges in total.
c)	Analyzed merges	Number of three-way merges, not considering fast-forward merges.
d)	Merges with conflicts	Number of analyzed merges with conflicts.
e)	Merges without conflicts	Number of analyzed merges without conflicts.
f)	Merges with self-conflict	Number of analyzed merges with the same developer authoring both sides of at least one conflicting chunk.
		Merge attributes
g)	Merge conflict occurrence	Binary attribute (yes or no) indicating if the merge has conflicts.
h)	Branching-duration	The effective duration of development in the branches, from the first branch commit (min(B1,B2)) to the last branch commit (max(B1,B2)), in days.
i)	Total-duration	The total duration of isolation, from the common ancestor (base commit) to the merge commit, in days.
j)	Committers in both branches	Percentage of developers in both branches.
k)	Conflicting chunk	Number of conflicting chunks.
l)	Conflicting chunk by the same developers	Number of conflicting chunks authored by the same developer in both sides.
		Branch attributes (B1 and B2)
m)	Commit Density	Number of commits in the Branching-duration.
n)	Loc-churn	Number of lines changed (added + deleted) in each branch.
o)	Changed files	Number of changed files in each branch.
p)	Commits	Number of commits in each branch.
q)	Committers (commit authors)	Number of developers that authored commits in each branch.

values. The branch attributes are the number of commits, number of committers (commit authors), number of changed files, and loc-churn (number of lines changed: added + deleted).

The attributes are collected for merges with conflicts and merges without conflicts. They allow us to compare different characteristics between merges with and without conflicts. Some of these attributes are also mentioned in related work, such as the timing metrics, merge conflict occurrence, number of merge conflicts, number of commits, commit density, committers, and changed lines and files (Dias et al. (2020); Leßenich et al. (2018); Vale et al. (2020)).

2.2 Projects and Merges Selection

First, we decided to select projects developed in different and popular programming languages. Thus, we identified the top-8 programming languages present in the following surveys: GitHub[3] top active languages survey 2019, Stack Overflow[4] developer survey results in 2019, and TIOBE [5] Index 2019. The top-8 selected programming languages present in the three surveys were: JavaScript, Python, Java, PHP, C#, C++, C, and Ruby.

We selected the projects using the GitHub API. We used the following criteria: (1) popular projects (projects with more than 1,000 stars), (2) software projects, (3) number of merges greater than 100, (4) projects with a wiki or some documentation, and (5) a balanced amount of merges per project. After applying the first criterion, we initially selected 461 projects. After applying criteria 2 to 4, 279 projects remained. To obtain a balanced corpus in terms of the number of analyzed merges per project, we selected ten projects per programming language, where the number of analyzed merges was less than 5% of the total number of analyzed merges in the dataset (**Table 2**). For example, the Graal project is the project with the highest number of analyzed merges (7,064). However, it represents just 3.9% of our final dataset (182,273).

Table 2. General information of our dataset.

Programming Language	**Total Merges**	**Analyzed Merges**	**Conflicting Merges**
C	31,013	21,948	981
C#	31,468	21,148	2,003
C++	32,463	24,155	2,290
Java	32,989	24,109	2,519
JavaScript	31,542	21,803	2,613
PHP	31,208	22,371	3,376
Python	32,591	22,585	1,923
Ruby	37,001	24,154	2,114
Total	**260,275**	**182,273**	**17,819**

It is important to mention that although the total number of merges was initially 260,275 (**Table 2**), we removed 78,002 merges from the analysis: 74,293 fast-forward merges (i.e., merges with no changes in a branch, in which Git would be able to just moves the pointer forward, but due to the option `--no-ff`, a merge commit was created (Chacon and Hamano, 2009)), 37 merges with negative total-duration (i.e., merges

[3]`https://githut.info/`
[4]`https://insights.stackoverflow.com/survey/2019#technology`
[5]`https://www.tiobe.com/tiobe-index/`

in which the date of the common ancestor is more recent than the date of the merge, probably due to some clock misconfiguration in the developer computer), and 3,672 merges with only merge commits (merges in which all commits from both branches are merge commits).

2.3 Merges and Attributes Extraction

We have implemented a tool to extract the attributes. The tool and the dataset are publicly available on GitHub[6]. The tool was developed in Java to parse the log provided by Git, retrieving all merge commits. Then, it identifies the parent commits that were merged and navigated until the common ancestor, just before forking the history. Our tool also checks whether the merge resulted in conflicts.

Figure 1 shows a merge example composed of: a merge commit (C57), two parents commits that were merged (C55 and C56), and the common ancestor (C50). From these commits, it is possible to identify the commits within each branch. These commits are located between the common ancestor, and each of the parents commits merged (including the parent commits). The "feature" branch in the example has three commits (C51, C54, and C56), and the "master" branch also has three commits (C52, C53, and C55). By identifying all commits from the branches of each merge, our tool was able to collect all the attributes listed in Section 2.1.

In our example, to calculate the branching-duration, we check the date of the first branch commit (min(B1,B2)), "08 Aug 2020", and the date of the last branch commit (max(B1,B2)), "22 Aug 2020". So, the branching-duration was 14 days. In the attribute verification of committers in both branches, the tool would identify that Ana made changes to both branches. In the verification of the conflicting chunk by the same developers, Ana could also have been the author of a self-conflict. In the committers attribute verification, the "feature" branch has two committers (Lisa and Ana), and the "master" branch also has two committers (Ana and Tom). Three files were changed in the "feature" branch (A, B, and C), and two files (A and B) were changed in the "master" branch.

With the attributes of the 182,273 merge cases, we could conduct statistical analysis to understand the difference between the distributions of merges with and without conflicts. Additionally, we plotted graphs representing the probability of having a conflict in a merge (axis y) given that an attribute is higher than a value (axis x). We calculated this probability according to the Bayes theorem: $P(conflict|attribute > value) = P(conflict \cap attribute > value)/P(attribute > value)$.

2.4 Data Mining

In this step, we adopted a data mining technique called association rules extraction. In summary, an association rule R is a pair (X, Y) of two disjoint entity sets, X and Y. In the notation $X \rightarrow Y$, X is called antecedent, and Y is called consequent (Han et al. (2012)). The rules aim at finding associations or correlations, but, as said by Zimmermann et al.

[6]https://github.com/catarinacosta/macTool/

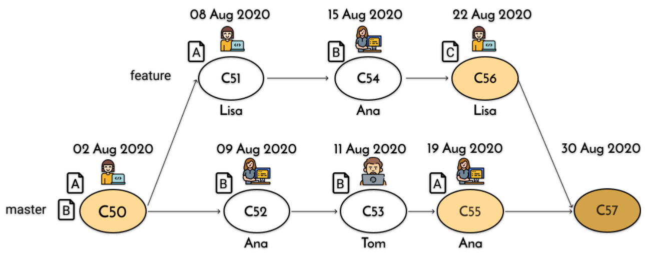


Figure 1. Merge example.

(2004), rules do not tell an absolute truth. They have a probabilistic interpretation based on the amount of evidence determined by two metrics (Agrawal et al., 1994): (a) $support$, the joint probability of having both antecedent and consequent, and (b) $confidence$, the conditional probability of having the consequent when the antecedent is present. Another measure of interest used is the (c) $lift$, which indicates how much the occurrence of Y increases given the occurrence of X. Han et al. (2012) explain that $lift(X \rightarrow Y) = confidence(X \rightarrow Y)/support(Y)$, where $lift = 1$ indicates that the antecedent (X) does not interfere with the occurrence of the consequent (Y), $lift > 1$ indicates that the occurrence of X increases the chances of the occurrence of Y, and $lift < 1$ indicates that the occurrence of X decreases the chances of the occurrence of Y.

We adopted the Knowledge Discovery in Databases (KDD) process (Fayyad et al. (1996)) to extract the association rules from our dataset: (a) data selection, (b) preprocessing, (c) transformation and data enrichment, (d) association rules extraction, and (e) results interpretation and evaluation.

After we selected and collected the projects and the attributes using our tool (step a), we removed instances (merge cases) with inconsistent values (step b), for example, merge cases with negative total-duration. These two initial steps were described in Section 2.1 to 2.3.

The discretization (step c) was performed through the supervised algorithm proposed by Fayyad and Irani (1992), available in the Weka[7] tool. This algorithm transforms numerical attributes into categorical ones, aiming at reducing the entropy of the original class distribution by finding ranges that maximize their class-related purity. In this study, the class attribute indicates the merge conflict occurrence.

For the association rules extraction (step d), we used R[8] with the Apriori (Agrawal et al. (1994)) algorithm and the Rattle[9]tool. In this study, our focus was on finding rules with the occurrence of conflict in the consequent (conflict=YES). However, due to the presence of conflicts being approximately 10% of our dataset, we lowered the support and confidence measures of interest considerably, to 0.01%.

Finally, we looked at all the association rules extracted that would help us answer the research questions (step e). In this step, we performed the analysis of the results.

[7]https://www.cs.waikato.ac.nz/ml/weka/
[8]https://cran.r-project.org/bin/windows/Rtools/
[9]https://rattle.togaware.com/

3 Results

This section answers the research questions posed in Section 1 according to the research process described in Section 2. Section 3.1 presents a statistical analysis of the merge attributes. Section 3.2 analyzes the extracted association rules. Section 3.3 presents the number of self-conflicts.

3.1 Statistical Analysis

In this section, we analyze the distribution of each merge attribute for merges with and without conflict to understand which attributes act more as an indicator of conflict. Hence, we divided the dataset of 182,273 merges into two subsets: one with 164,454 merges without conflicts and the other with 17,819 merges with conflicts.

Table 3 presents the comparison of the distributions. For comparing the distribution of the attributes, we first analyzed their normality using the Anderson-Darling test (Anderson and Darling, 1954). We chose this test due to the size of the distributions. We observed non-normality in all distributions at 95% confidence. Then, we applied the Mann-Whitney test (Mann and Whitney, 1947) for each pair of subsets, and we found statistically significant differences for all the distributions, except the number of committers in B1 $p\text{-}value = 0.323$. After calculating the mean of the statistically different distributions, we observed that merges with conflicts have higher values than merges without conflicts for all the attributes. Given these results and the non-normality of the distributions, we used Cliff's Delta (Macbeth et al., 2011) to calculate the effect size of these differences. We found four attributes with a large effect size (the ones related to B2) and five with a small effect size (the ones related to the time and most of the ones associated with B1).

For clarification, let us analyze the distributions of changed files in B2 from **Table 3** as an example. We started the analysis by applying the Anderson-Darling test for the distribution of changed files in B2 for merges without conflicts and obtained a $p\text{-}value < 10^{-15}$ rejecting the null hypothesis at 95% confidence (i.e., we found that this data are not from a population with a normal distribution). Then, we applied the same test for the distribution related to merges with conflict, and we also observed a $p\text{-}value < 10^{-15}$. Since both distributions are not normal nor paired, we compared them with a non-parametric test for unpaired data: Mann-Whitney. Once again, we observer a $p\text{-}value < 10^{-15}$, indicating that the distributions are statistically different from each other. Note in **Table 3** that both the average and the boxplot of the number of changed files in B2 for merges with conflicts (WC) are higher than the ones for merges without conflicts (WO). Finally, we used Cliff's delta to calculate the effect size of the difference between these distributions, and we obtained a magnitude of -0.57, which is classified as large (Romano et al., 2006).

After analyzing the distributions and observing a significant statistical difference in most of them, we applied the Bayes theorem to calculate the probability $P(conflict|attribute \geq x)$ and we variated x for values within the range of the boxplots presented in **Table 3** (i.e., between $max(Q1 - 1.5 \times IQR, minimum)$ and $min(Q3 + 1.5 \times IQR, maximum)$). **Figure 2** presents the distribution of probabilities for each numeric distribution. As expected, all probabilities start at around 10%, which represents the percentage of merge conflicts, but they grow at different rates. **Figure 2** highlights the probabilities in the medians of the distributions with and without merge conflicts and the probability in the last value of the interval. Note in **Figure 2**(e) that the $P(conflict|committers\ B1 \geq x)$ is 9% for $x = 2$ (the median for both merges with and without conflicts). It indicates that the probability had a small decrease in comparison to the starting point $(x = 1)$.

Continuing our example using the number of changed files in B2, note that the $P(conflict|changed\ files\ B2 \geq x)$ in **Figure 2**(h) starts at 9.8% when $x = 0$. Then, when x reaches the median number of changed files in B2 for merges without conflicts ($x = 2$), the probability is 13.5%. When x reaches the median number of changed files in B2 for merges with conflicts ($x = 19$), the probability is 29.5%. Finally, at the end of the interval ($x = Q3 + 1.5 \times IQR = 205$), the probability is 47.9%. As expected, in **Figure 2**, attributes with a large effect size (changed files in B2, commits in B2, changed lines in B2, and committers in B2) grow faster than attributes with a smaller effect size (changed lines in B1, branching-duration, changed files in B1, total-duration, and commits in B1).

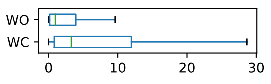
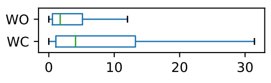
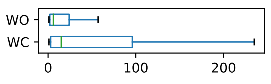
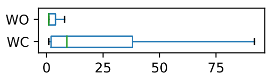
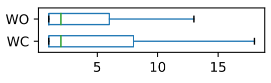
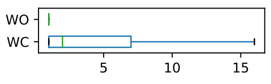
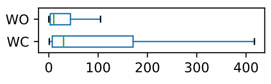
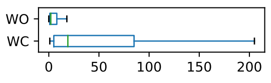
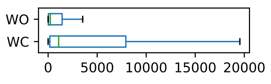
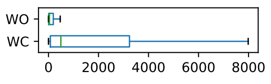
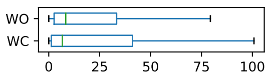
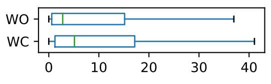
3.2 Association Rules

We used data mining to enrich our analyzes with association rules. **Table 4** presents the extracted association rules in which the antecedent is the range value of each attribute, obtained by the discretization process, and the consequent is the presence of conflicts. It also presents the three measures of interest used, support (Sup.), confidence (Conf.), and lift. In general, smaller attributes' values make the chance of merge conflicts decrease ($lift < 1$), while higher values make the chance of merge conflicts increase ($lift > 1$). For instance, for just one changed file in B2, the probability of merge conflicts decreases by 81% ($lift = 0.19$). However, for more than 30 changed files, the probability of merge conflicts increases by 243% ($lift = 3.43$).

Through all these analyzes, we observed that attributes related to B2 (i.e., the branch being integrated into B1) influence more the probability of merge conflicts than the other attributes, with the number of changed files, number of commits, number of changed lines, and number of committers in B2 being the attributes that influence the most, in this order. We observed that the attributes branching-duration and total-duration have a similar impact on the probability of merge conflicts and could be used interchangeably in most situations. We also verified the eight programming languages we selected regarding their influence on the occurrence of conflicts. Three languages (PHP, JavaScript, and Java) have shown a positive conflict dependency ($lift > 1$), which increases the chances of conflicts occurring (**Table 5**). We observe that, when using PHP, the probability of conflicts occurrence increases by 53% ($lift = 1.53$). On the other hand, when programming in C, the probability of having conflicts decreases by 54% ($lift = 0.46$).

Finally, we evaluated the intersection of developers, i.e., the number of developers working in both branches. Some

Table 3. Comparison of merge distributions with (WC) and without conflicts (WO)

Attribute	Anderson-Darling (p-value)		Mann-Whitney	Average		Cliff's Delta		Distribution
	WO	WC	(p-value)	WO	WC	Value	Meaning	
Branching-duration	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	6.26	18.53	-0.3	Small	
Total-duration	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	7.27	19.71	-0.27	Small	
Commits B1	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	73.51	252.91	-0.24	Small	
Commits B2	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	9.76	81.07	-0.53	Large	
Committers B1	$< 10^{-15}$	$< 10^{-15}$	0.32	8.52	21.31	-	-	
Committers B2	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	2.01	9.4	-0.48	Large	
Changed Files B1	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	100.8	425.44	-0.29	Small	
Changed Files B2	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	21.43	166.3	-0.57	Large	
Loc-churn B1	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	6666.95	32934.05	-0.33	Small	
Loc-churn B2	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	1623.7	13734.43	-0.51	Large	
Density B1	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	545.98	1074.17	0.07	Negligible	
Density B2	$< 10^{-15}$	$< 10^{-15}$	$< 10^{-15}$	35.21	51.53	-0.11	Negligible	

studies have already mentioned that developers may work in both branches (Costa et al., 2014, 2016; Zimmermann, 2007). According to Zimmermann (2007), many developers work at different places (e.g., home and office) or on different branches, and, at some point, they need to synchronize their changes. Costa et al. (2014) analyzed the number of merges in repositories according to three scenarios: the presence of the same developers in both branches, disjoint sets of developers, or some intersection of the developers. They found a significant number of merges with developers working in both branches. We also performed this analysis in our dataset, but we compared the numbers of merges with and without conflicts. **Figure 3** shows the number of merges cases with no intersection, with some intersection, and with all the developers in common for merges with and without conflicts. Since the number of merges with conflicts is much smaller than the number of merges without conflicts, we normalized both groups according to the total number of merges.

Then, we mined association rules to find the increase or decrease in the probability of merge conflicts. **Table 6** presents the results, which indicates that having some intersection (67% to 99%) increases the chance of conflicts by 265% ($lift = 3.65$), and having no intersection reduces the chances of conflict by 41% ($lift = 0.59$).

After extracting rules with only one attribute in the antecedent, and considering the multidimensional characteristics of an association rule (Lu et al. (2000)), we decided to analyze the combination of rules and understand if the combination of factors increases some measures of interest in the occurrence of conflict. The algorithm that brought the best results in the selection of attributes was InfoGainAttributeE-

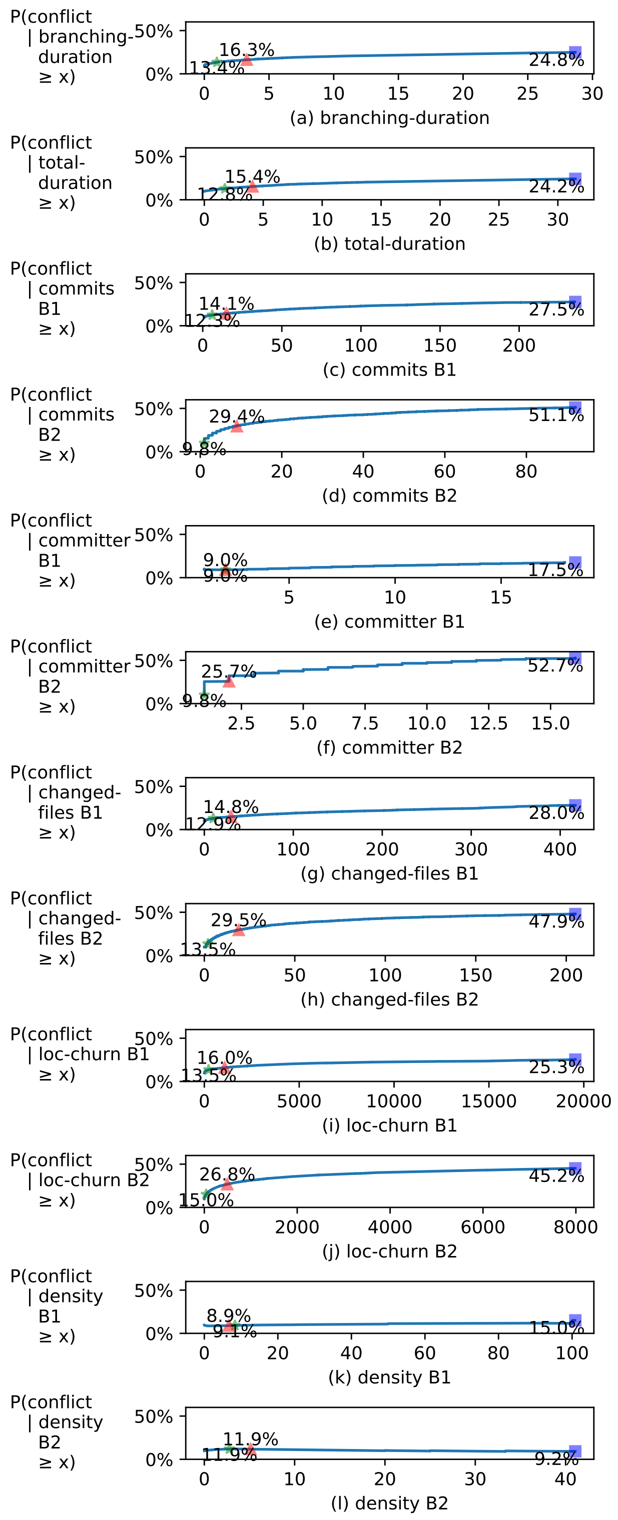


Figure 2. Probability of conflicts given that the attribute is greater than the value on the x axis. Green stars represent the probability on the median of the distributions without conflicts. Red triangles represent the probability on the median of the distributions with conflicts. Blue squares represent the probability on the maximum value.

val[10]. Six attributes (Branching-duration, Committers in B2, Intersection, Commits in B2, and Changed Files and Lines in B2) with the best classification were selected and ten combinations of attributes with the rules with the best measures of interest are presented in **Table 7**.

[10]weka.attributeSelection.InfoGainAttributeEval

Table 4. Measures of interest for the rules {attribute = range value} → {conflict=YES}

Attibute	Range Value	Sup. (%)	Conf. (%)	Lift
Branching-Duration (in days)	< 1	2.85	5.84	0.60
	1 – 7	3.84	10.87	1.11
	8 – 15	1.13	16.31	1.67
	16 – 30	0.77	17.93	1.84
	> 30	1.17	25.45	2.61
Total-Duration (in days)	< 1	2.22	6.01	0.62
	1 – 7	4.19	9.43	0.97
	8 – 15	1.27	15.06	1.54
	16 – 30	0.83	16.85	1.73
	> 30	1.24	24.20	2.48
Commits in B1	1	1.27	5.99	0.61
	2 – 5	1.98	7.60	0.78
	6 – 20	2.38	17.82	1.82
	> 20	4.37	15.01	1.54
Commits in B2	1	1.60	3.39	0.35
	2 – 5	2.33	7.76	0.79
	6 – 20	2.38	17.82	1.82
	> 20	3.46	36.92	3.78
Committers in B1	1 – 3	6.14	9.71	1.00
	4 – 10	1.51	6.88	0.70
	11 – 30	1.08	10.91	1.12
	> 30	1.04	21.00	2.15
Committers in B2	1 – 3	5.84	6.57	0.67
	4 – 10	2.19	29.51	3.02
	11 – 30	1.15	43.00	4.40
	> 30	0.59	59.12	6.05
Changed Files in B1	1 file	0.42	3.18	0.33
	2 – 5	1.55	6.85	0.70
	6 – 30	2.94	9.34	0.96
	> 30	4.85	14.90	1.53
Changed File in B2	1 file	0.60	1.89	0.19
	2 – 5	1.99	5.99	0.61
	6 – 30	3.07	13.55	1.39
	> 30	4.10	33.47	3.43
Loc-churn in B1	0 – 10	0.41	3.03	0.31
	11 – 100	1.44	9.25	0.64
	101 – 1000	2.94	9.32	0.95
	1001 – 10000	2.83	12.81	1.31
	> 10000	2.15	22.27	2.28
Loc-churn in B2	0 – 10	0.82	3.04	0.31
	11 – 100	1.95	5.52	0.57
	101 – 1000	3.04	12.13	1.24
	1001 – 10000	2.59	26.81	2.74
	> 10000	1.36	46.25	4.73
Density B1	0 – 5	4.33	10.82	1.11
	> 5 – 20	2.07	7.44	0.76
	> 20 – 40	0.84	8.72	0.89
	> 40	2.83	11.30	1.16
Density B2	0 – 5	4.95	8.39	0.86
	> 5 – 20	2.69	13.73	1.40
	> 20 – 40	0.82	11.44	1.16
	> 40	1.32	9.26	0.95

Considering the first rule in **Table 7**, when the Branching-duration is 16-30 days, the number of Committers in B2, is 4-

Table 5. Measures of interest for the rules {language} → {conflict=YES}

Language	Sup. (%)	Conf. (%)	Lift
C	0.55	4.47	0.46
C#	1.11	9.62	0.97
C++	1.27	9.49	0.96
Java	1.42	10.77	1.09
JavaScript	1.45	12.18	1.23
PHP	1.87	15.18	1.53
Python	1.04	8.43	0.85
Ruby	1.18	8.88	0.90

Table 6. Measures of interest for the rules related to the intersection of developers {intersection} → {conflict=YES}

% Intersection	Sup. (%)	Conf. (%)	Lift
0%	3.39	5.78	0.59
1% – 33%	4.91	17.83	1.83
34% – 66%	0.74	11.94	1.22
67% – 99%	0.13	35.68	3.65
100%	0.60	8.19	0.84

10, the intersection of developers is 26%-50%, and the number of Commits in B2 is greater than 20, then the probability of conflict occurrence increases by 850% ($lift = 9.50$). Please note the confidence and lift of this rule are greater when compared to the individual rules of each attribute.

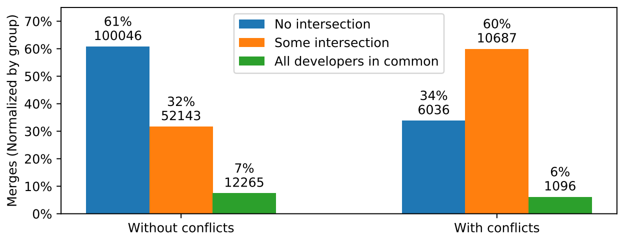


Figure 3. Intersection of developers in branches.

3.3 Self-Conflicts

We observed a significant number of developers intersection in **Figure 3**. So, we investigated conflicting chunks and commits that have been made by the same developer. We noticed something interesting: in some cases, a developer made parallel changes that resulted in a merge conflict. Zimmermann (2007) named this phenomenon as self-conflicts. We identified self-conflict cases in all 80 investigated projects. **Figure 4** summarizes the comparison between self-conflicts and conflicts inserted by different committers in each of the 80 projects, grouped by programming language, for merges with conflicts. In this analysis, we divided the number of conflicting chunks by the same developer by the total number of conflicting chunks.

We also decided to mine association rules about the attributes investigated in this study and their effect on the occurrence of self-conflicts. When looking at attributes such as time, the number of commits, committers, changed lines and

Table 7. Measures of interest for the rules combined

Antecedent	Sup. (%)	Conf. (%)	Lift
Branching-Duration = 16 – 30 ∧ Committers in B2 = 4 – 10 ∧ % intersection = 26% – 50% ∧ Commits B2 = 20	0,01	92,86	9,50
Branching-Duration = 30 ∧ Committers in B2 = 4 – 10 ∧ % intersection = 1% – 25% ∧ Changed File in B2 = 30 ∧ Loc-churn = 101 – 1000	0,02	90,91	9,30
Branching-Duration = 30 ∧ % intersection = 1% – 25% ∧ Commits B2 = 6 – 20 ∧ Changed File in B2 = 30 ∧ Loc-churn = 101 – 1000	0,01	89,47	9,16
Branching-Duration = 30 ∧ Committers in B2 = 4 – 10 ∧ % intersection = 1% – 25% ∧ Commits B2 = 6 – 20 ∧ Changed File in B2 = 30	0,02	89,29	9,14
Branching-Duration = 16 – 30 ∧ Committers in B2 = 4 – 10 ∧ % intersection = 26% – 50% ∧ Changed File in B2 = 30	0,01	87,50	8,96
Branching-Duration = 30 ∧ Committers in B2 = 4 – 10 ∧ Commits B2 = 6 – 20 ∧ Changed File in B2 = 30	0,02	87,10	8,91
Branching-Duration = 30 ∧ Committers in B2 = 4 – 10 ∧ % intersection = 1% – 25% ∧ Commits B2 = 6 – 20 ∧ Loc-churn = 101 – 1000	0,03	86,27	8,83
Branching-Duration = 16 – 30 ∧ Committers in B2 = 4 – 10 ∧ % intersection = 26% – 50%	0,01	85,00	8,70
Branching-Duration = 30 ∧ Committers in B2 = 11 – 30 ∧ % intersection = 1% – 25% ∧ Changed File in B2 = 30 ∧ Loc-churn = 101 – 1000	0,01	82,35	8,43

Branching-Duration = 30 ∧
Commits B2 = 6 – 20 ∧
Changed File in B2 = 30 ∧
Loc-churn = 101 – 1000
	0,01	81,82	8,37

files, intersection, commits density, and the programming language. Only the existence of developer intersection showed a strong influence on the occurrence of self-conflicts. Self-conflict logically only exists when a developer works in both branches. However, it is important to verify that there is a tendency for the chances of self-conflict to increase as the percentage of intersection increases (with a slight exception in the range of 67% - 99%, which reduces the chances by 1% compared to the range of 34% - 66%), as shown in **Table 8**.

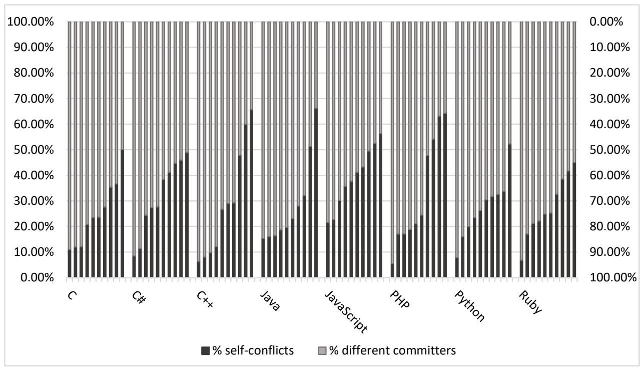


Figure 4. Conflicting chunks in projects grouped by programming language.

Table 8. Measures of interest for the rules related to the intersection of developers {intersection} → {self-conflict=YES}

% Intersection	Sup. (%)	Conf. (%)	Lift
0%	2.40	6.98	0.19
1% – 33%	23.06	45.93	1.27
34% – 66%	4.67	59.33	1.65
67% – 99%	0,70	59,18	1.64
100%	5.22	81.48	2.26

4 Discussions

In this section, we answer the research questions presented in Section 1 based on the results described in Sections 3.1, 3.2, and 3.3. In general, the results for B2 (ie, the branch that is integrated into B1 during the merge) demonstrated a greater impact on the occurrence of conflicts, mainly for the number of changed files, commits, changed lines, and committers. As the identification of B1 and B2 is based on the merge direction, it depends on the strategy adopted by the software project.

4.1 How is the isolation of a branch related to the occurrence of merge conflicts? (RQ1)

The isolation of the branches is mentioned by some studies (Bird et al. (2011); Costa et al. (2014); Dias et al. (2020); Leßenich et al. (2018)) as a factor that may contribute to the occurrence of conflicts. In our study, we measured the isolation of branches using two attributes related to time: the branching-duration and the total-duration. We calculated these attributes for each merge case (with conflicts and without conflicts), in days. In Section 3.1, we observed that both attributes have a very similar distribution, and they both present some impact on the occurrence of merge conflicts (effect sizes of -0.3 and -0.27 for branching-duration and total-duration, respectively).

After mining association rules, we noted that the probability of conflicts occurring decreases when the duration is very short (less than a day): 40% less for branching-duration ($lift = 0.60$) and 38% less for total-duration ($lift = 0.62$). When the time is medium (8-15 days), the chances of having a conflict increases by 67% ($lift = 1.67$) for branching-duration, and 54% ($lift = 1.54$) for the total-duration. So, the results indicate a positive dependence between the duration increase and the chances of having a conflict. The lift of very long duration (more than 30 days) suggests that the chances of having a conflict increases by 161% ($lift = 2.61$) for branching-duration and by 148% ($lift = 2.48$) for the total-duration.

Answer to RQ1: The branch-duration and total-duration have a small impact on the occurrence of merge conflicts (effect sizes of -0.3 and -0.27, respectively). Despite the small impact, the association rules indicate that the occurrence of conflict increase when time increases (lift close to 1 for durations of 1-7 days and lift > 2.4 for durations bigger than 30 days).

4.2 How is the number of commits related to the occurrence of merge conflicts? (RQ2)

To answer this question, we checked the amount of work done in terms of commits in each branch. In Section 3.1, we observed that both the number of commits in B1 and the number of commits in B2 have a positive impact on the occurrence of merge conflicts. However, the impact of commits in B2 is larger (effect size of -0.53) than the impact of commits in B1 (effect size of -0.24), indicating that the number of commits in B2 (i.e., the branch that is being integrated into B1) is a better predictor of conflicts than the number of commits in B1.

We analyzed how much more frequent the conflict becomes with the increase in the number of commits in both branches in **Figure 2** and **Table 4**. We can see that contributions with few commits in B1 and B2 have a negative dependency on the occurrence of conflicts. When the branch has only one commit, the occurrence of conflict decreases by 39% ($lift = 0.61$) for B1 and 65% ($lift = 0.35$) for B2. Having few commits (2-5) shows a decrease of 22% ($lift = 0.78$) for B1, and 21% ($lift = 0.79$) for B2. The lift of 1.54 for B1 and 3.78 for B2 when there are more than 20 commits indicates that the chances of having a conflict increase by 54% for B1 and by 278% for B2. By looking at the probability of having a conflict given the number of commits in **Figure 2**(c) and **Figure 2**(d), it is possible to see that this probability grows faster according to the number of commits in B2, reaching around 40% for 30 commits while reaching just around 16% for 30 commits in B1.

We also verified the commit density, i.e., the number of commits in B1 and B2 in relation to branching-duration. We noticed a significant difference between the impact of the number of commits and the number of commits divided by the branching-duration, the commit density. The impact of commit density in B1 (0.05) and B2 (-0.12) is negligible. When looking at the density association rules, we observed that unlike other attributes, there is no pattern of evolution when the value of the attribute increases, that is, when the number of commits in B1 or B2 divided by the branching-duration is greater. When the number of commits in B1 and B2 is between 0 to 5, the chance of having a conflict increases 11% for B1 ($lift = 1.11$) and decreases 14% for B2 ($lift = 0.86$). When the number of commits in B1 or B2 divided by the branching-duration is greater than 40, the chance of having a conflict increases 16% for B1 ($lift = 1.16$) and decreases 5% for B2 ($lift = 0.95$).

Answer to RQ2: The number of commit has a small impact for B1 (effect size of -0.24) and a large impact for B2 (effect size of -0.53) on the occurrence of merge conflicts. The association rules indicate that the chances of conflict increase when the number of commits increases (according to the ranges of commits, lifts in B1 range from 0.61 to 1.54 and lifts in B2 range from 0.35 to 3.78).

4.3 How is the number of developers that performed commits related to the occurrence of merge conflicts? (RQ3)

For this question, we checked the number of committers in each branch. We observed that the number of committers in B1 (i.e., the branch that receives the integration) does not seem to have a statistically significant impact on the probability of merge conflicts. On the other hand, we observed that the number of committers in B2 has a large impact on the occurrence of merge conflicts (effect size of -0.48). These differences can also be observed in **Figure 2**(e) and **Figure 2**(f), which present the probabilities of conflicts. While the probability barely grows according to the numbers of committers in B1 (from 10% for one committer to 11% for six committers), it has a considerable growth for the number of committers in B2 (from 10% for one committer to 40% for six committers). Hence, the number of committers in the branch that is being integrated (B2) seems to be a good indication of the possibility of merge conflicts.

Comparing the distributions of committers in B2 for merges with and without conflicts in **Table 3**, we noted that while merges without conflicts usually have a single committer in B2, conflicting merges seem to have more committers. The association rules in **Table 4** also indicate that when the number of committers is large, the chances of conflicts are higher. First, having few committers (1-3) in B1 does not imply more or fewer conflicts ($lift = 1.00$). However, there is a negative dependency when considering B2. In this case, the occurrence of conflict decreases by 33% ($lift = 0.67$). For a very large number of committers (i.e., more than 30 committers), we observed an increase in the chances of having a conflict by 115% for B1 ($lift = 2.15$) and 505% for B2 ($lift = 6.05$).

Answer to RQ3: The number of committers has no impact for B1 (p-value is 0.32) and a large impact for B2 (effect-size of -0.48) on the occurrence of merge conflicts. The association rules indicate that the chances of conflict increase when the number of committers increases, especially for B2 (lift goes from 0.67 for 1–3 committers to 6.05 for >30 committers).

4.4 How is the number of changed files related to the occurrence of merge conflicts? (RQ4)

For this question, we checked the amount of work done in terms of changed files in each branch. The results are similar to the ones related to the number of commits in B1 and B2, with changes on B2 influencing more the probability of merge conflicts (effect size of -0.57) than changes on B1 (effect size of -0.29). **Figure 2**(g) and **Figure 2**(h) present the distributions and the probabilities of conflicts according to the number of changed files in B1 and B2, respectively. The probability of a merge conflict after changes in 40 or more files in B1 is around 16%.

On the other hand, the probability after changes in the same number of files in B2 is approximately 36%. For the number of changed files, as expected, the association rules also confirmed that fewer changed files are less likely to cause conflicts. As shown in **Table 4**, a single changed file indicates lower chances of conflicts: 67% less for B1 ($lift = 0.33$) and 81% less for B2 ($lift = 0.19$). However, for many changed files (i.e., more than 30), we observed an increase of 53% for B1 ($lift = 1.53$) and 243% for B2 ($lift = 3.43$).

Answer to RQ4: The number of changed files has a small impact for B1 (effect-size of -0.29) and a large impact for B2 (effect-size of -0.57) on the occurrence of merge conflicts. The association rules indicate that the chances of conflict increase when the number of commits increases (> 30 files in B1 has lift 1.53; > 6 files in B2 has lift 1.39; > 30 files in B2 has lift 3.43).

4.5 How is the number of changed lines related to the occurrence of merge conflicts? (RQ5)

For this question, we checked the loc-churn, the total number of lines of code added and removed in each branch (Gousios and Zaidman, 2014; Nagappan and Ball, 2005; da Silva et al., 2020). We verified that changed lines on B2 influence more the probability of merge conflicts (effect size of -0.51) than changed lines on B1 (effect size of -0.33). This result is similar to the ones related to the number of changed files and commits.

We also verified that association rules involving changed lines of code have a negative conflict dependency for values less than 100 changed lines . Rules with values of 0-10 changed lines have their chances of conflict reduced by 69% ($lift = 0.31$) for B1 and B2. For changes involving 11-100 lines, the chances are reduced by 36% ($lift = 0.64$) for B1 and 43% ($lift = 0.57$) for B2. For modifications involving many changed lines, the chances of a conflict occurring are increased. For more than ten thousand lines of code, the chances increased by 128% ($lift = 2.28$) for B1 and 373% ($lift = 4.73$) for B2.

Answer to RQ5: The number of changed lines has a small impact for B1 (effect-size of -0.33) and a large impact for B2 (effect-size of -0.51) on the occurrence of merge conflicts. The association rules indicate that the chances of conflict increase when the number of changed lines increases (lift goes from 0.31 for 0 – 10 loc to 2.28 for > 10000 loc in B1 and 0.31 for 0 – 10 loc to 4.73 for > 10000 loc in B2).

4.6 How is the programming language related to the occurrence of merge conflicts? (RQ6)

For this question, we observed the eight programming languages adopted in the selected projects. As shown in **Table 5**, for five programming languages (C, C#, C++, Python, and Ruby), the chances of conflict occurrences decrease. The language C reduces the chances of conflicts in 54% ($lift = 0.46$). Python and Ruby also decreases the chances of conflicts, in 15% ($lift = 0.85$) and 10% ($lift = 0.90$), respectively.

On the other hand, three programming languages (PHP, JavaScript, and Java) selected for this study present a positive dependency to conflict occurrence. We observed that PHP increases the chances of conflict by 53% ($lift = 1.53$) and JavaScript increases in 23% ($lift = 1.23$). For projects written in Java, there is an increase of 9% ($lift = 1.09$) in the chances of a merge conflict.

Answer to RQ6: The association rules indicate that the chances of conflict increase when the project is written in PHP (53%), JavaScript (23%), and Java (9%).

4.7 How is the intersection of developers in both branches related to the occurrence of merge conflicts? (RQ7)

For this question, we checked the frequency of the committers in both branches and divided the merges into three groups: merges with no intersection, merges with some intersection, and merges with all developers in common. Contrary to our expectations, as presented in **Figure 3**, the intersection of developers does not decrease the chance of merge conflicts.

When we mined association rules related to the intersection of developers, we divided the merges into five groups: 0% (merges with no intersection), 1%-33%, 34%-66%, 67%-99%, and 100% (merges with all developers in common) (**Table 6**). We observed that having some intersection (67%-99%) increases the chance of conflict by 265% ($lift = 3.65$), while having no intersection decreases the probability of conflict by 41% ($lift = 0.59$). However, when the merge has all developers in common, the chance of conflicts also decreases by 16% ($lift = 0.84$). So, having all the developers or no developers in common seems to be better than having just one set of developers in common.

Answer to RQ7: The association rules indicate that having some intersection increases the chances of conflict (67% - 99% in 265%, 1% - 33% in 83%, and 34% - 66% in 22%).

4.8 How prevalent is the occurrence of merge self-conflicts? (RQ8)

Conflicts caused between commits of the same developer seem more common than we anticipated. Note that the percentage of self-conflicts in **Figure 4** ranges from 5.46% (of 3,152 conflicting chunks) in Yii2 project to 66.23% (of 835 conflicting chunks) in Vert.x project. Note also that ten projects had more than 50% of self-conflicts. When considering projects with more than 40% of self-conflicts cases, 22 projects are listed. We then decided to analyze a merge case (commit 456424) from the Elasticsearch project, and observed two examples of self-conflicts in a source-code file and in a debug file. Regarding the source-code file, in B1, the developer created an instance of a SearchResponse object with a parameter (commit 3a6429), and in B2, the developer performed validation and also created an instance of a SearchResponse object, but without parameters (commit d82faf). Regarding the debug file, the developer added several lines in both branches (commits 3a6429 and d82faf), possibly during execution in a test environment.

When we mined association rules related to the occurrence of self-conflicts, we verified when the merge involves all the developers in common, the chances of a self-conflict occurring are increased by 126% ($lift = 2.26$), as shown in **Table 8**. We analyzed other attributes, but none showed a strong influence (> 27%), with the exception of the intersection of developers.

Answer to RQ8: We identified self-conflicts in all 80 projects. The percentage of self-conflicts range from 5.46% (of 3,152 conflicting chunks) in Yii2 project to 66.23% (of 835 conflicting chunks) in Vert.x project.

5 Threats to Validity

As in any study, ours also has limitations. Our approach uses the committers' git ID (names and/or email addresses) to identify developers who committed in both branches. Developers may use multiple aliases, eventually generating inconsistencies (i.e., false negatives) in the results. We adopted the strategy to turn all letters in uppercase and remove all existing spaces to reduce this threat. We may have missed some cases when the aliases are lexically different, but in this case, the number of committers in both branches and self-conflicts would be higher.

We believe that a branch's isolation time is relative. Someone can create a branch and not commit to it for a while or someone can perform the branch's last commit and not merge for a while. Therefore, the measurement of the duration of a branch has limitations. We used two metrics of time to mitigate this threat: considering just the commits performed within the branches (branching-duration) and considering the merge commit (total-duration).

We are investigating only three-way merge scenarios integrating two branches, so we found and excluded 74,293 fast-forward merges. Different merge strategies may not have been considered, for example, the Git rebase, as it flattens the rich information of parallel development into a linear history. We also excluded 37 merge cases in which the time metrics were negative. Since the timestamp for each commit is generated on the developer's computer, if the computer's clock is wrong, the timestamp is recorded incorrectly. In a merge case (merge commit 1da7521) from the Elasticsearch project, for example, while the merge was committed on 2/8/2017, the common ancestor (commit 5ee82e4) of the parents' commits (commits 1ba5f8f and e761b76) was committed on 4/20/2018. Finally, we excluded 3,672 merges with only merge commits. For example, in a merge commit (197f57c) of the Osu project, we found just one commit in each branch, and these commits are also merge commits (commits 660afb4 and 436e155).

6 Related Work

Vale et al. (2020) investigated the role of communication activity in the increase or reduction of merge conflicts. They analyzed the history of 30 popular open-source projects involving 19,000 merge scenarios. The authors mined and linked contributions from Git and communication from GitHub data. They used bivariate and multivariate analyses to evaluate the correlations. In bivariate analysis, they found a weak positive correlation between GitHub communication activity and the number of merge conflicts. In the multivariate analysis, they discovered that GitHub communication activity does not correlate with the occurrence of merge conflicts. Thus, they investigated if it depends on the merge scenarios' characteristics, such as the number of modified lines, chunks, files, developers, commits, and days a merge scenario lasts. These variables are calculated by merge scenario (both branches). For example, the authors considered the sum of the number of developers in both branches. They found that there is no relation between the communication measures and the number of merge conflicts when considering these factors. They concluded that: (1) longer merge scenarios with more developers involve more GitHub communication, but not necessarily more merge conflicts, (2) the size of the changes of merge scenarios (in terms of numbers of files, chunks, and lines of code involved) is not sufficient to predict the occurrence of merge conflicts.

Leßenich et al. (2018) surveyed 41 developers and extracted a set of seven indicators (the number of commits, commit density, number of files changed by both branches, larger changes, fragmentation of changes, scattered changes across classes or methods, and the granularity of changes above or within class declarations) for predicting the number of conflicts in merge scenarios. They also checked additional indicators mentioned in the survey, i.e., whether the more developers contribute to a merge scenario, the more likely conflicts happen and whether branches that are developed over a long time without a merge are more likely to lead to merge conflicts. After determining the respective value for each branch, they compute the geometric mean of these values. To evaluate the indicators, the authors performed an empirical study on 163 open-source Java projects, involving 21,488 merge scenarios. They found that none of the indicators can predict the number of merge conflicts, as suggested by the developer survey. Hence, they assumed that these indicators are not useful for predicting the number of merge conflicts.

Owhadi-Kareshk et al. (2019) also investigated if conflict prediction is feasible. They verified nine indicators (the number of changed files in both branches, number of changed lines, number of commits and developers, commit density, keywords in the commit messages, modifications, and the duration of the development of the branch) for predicting whether a merge scenario is safe or conflicting. They adopted norm-1 as the combination operator to combine the indicators extracted for each branch into a single value. To evaluate the predictor, they performed an empirical study on 744 GitHub repositories in seven programming languages, involving 267,657 merge scenarios. Similar to related work, they did not find a correlation between the chosen indicators and conflicts, but using the same indicators, they designed a classifier that was able to detect safe merge scenarios (without conflicts) with high precision (0.97 to 0.98) using the Random Forest classifier.

Dias et al. (2020) also conducted a study to understand better how conflict occurrence is affected by technical and organizational factors. They investigated seven factors related to modularity, size, and timing of developers' contributions. They computed the geometric mean of the branch values for each factor. The authors analyzed 125 projects, involving 73,504 merge scenarios in GitHub repositories of Ruby (100) and Python (25) MVC projects. They found that merge conflict occurrence significantly increases when contributions to be merged are not modular in the sense that they involve files from the same MVC slice (related model, view, and controller files).

As previously discussed, Vale et al. (2020) and Owhadi-Kareshk et al. (2019) tried to predict the occurrence of merge conflicts. Complementary, Leßenich et al. (2018) tried to predict the number of merge conflicts. Vale et al. (2020) and Leßenich et al. (2018) did not find a strong correlation between the analyzed attributes and the occurrence and number of conflicts. Owhadi-Kareshk et al. (2019) also found no correlation between the indicators and conflicts, but were able to design a classifier for merge conflicts. Our study investigated some similar attributes to the ones evaluated by Vale et al. (2020) and Owhadi-Kareshk et al. (2019) (time metric, number of commits, committers, changed lines and files), and by Leßenich et al. (2018) (number of commits, commit density, and files in both branches), however, in our results the investigated attributes seem to have a positive correlation with merges with conflicts.

Similar to our results, Dias et al. (2020) found that more developers, commits, changed files, and contributions developed over long periods are more likely associated with merge conflicts. However, no evaluated attributes showed predictive power concerning the number of merge conflicts. They also investigated some similar attributes, as timing metrics, number of commits, committers, changed lines, and files. Although we did not check whether the contributions were modular or not, we added some attributes, such as the frequency of one or more committers in both branches and the verification of conflicting chunks and commits that have been made by the same developer. The extraction of association rules also showed us a tendency to merge conflicts when there is a longer duration, more commits, committers, and files changed.

It is worth mentioning that the attributes evaluated by the previous studies might not be computed in the same way, despite the attributes' name similarity. For example, the number of commits is presented in all the related work. Leßenich et al. (2018) reported the number of commits between the common ancestor and the merge as the geometric mean of both branches. Vale et al. (2020) report this number as the sum of commits performed in the two branches. Owhadi-Kareshk et al. (2019) used norm-1 (also a sum of absolute values) as the combination operator for the number of commits between the ancestor and the last commit in a branch. Dias et al. (2020) also used the geometric mean of the number of commits in each contribution. In our work, we decide to keep the information by branch, using no aggregate measure.

7 Conclusion

In this work, we analyzed 182,273 merge scenarios from 80 projects written in eight programming languages to understand which attributes impact on the occurrence of merge conflicts. While all attributes seem to have a positive influence on the probability of merge conflicts, some appear to have a more significant impact than others. The attributes that presented a higher relation to the occurrence of merge conflicts are changed files, commits, changed lines, and committers in the branch B2 (i.e., the branch that is integrated into B1 during the merge). These attributes in the branch B1 have a smaller impact (changed lines, changed files, and commits) or even no statistically significant difference (committers) on the occurrence of conflict. Both the branching-duration and the total-duration seem to have an impact comparable to the impact of attributes in B1. Despite some attributes presenting a smaller impact on merge conflicts when we compare the whole distributions, the association rules indicate that higher values of them increase the chances of conflicts by over 53%.

In addition to these attributes, we analyzed the impact of the selected programming language and the intersection of developers between branches on the occurrence of conflicts. Among the eight programming languages verified, PHP, JavaScript, and Java, have a positive conflict dependency, and PHP increases the chances of conflicts by 53%. Regarding the intersection of developers, we noticed that merges with one or more committers acting in both branches do not seem to reduce the chances of merge conflicts. Instead, having some intersection in the developers increases the chance of conflicts (1%-33% by 83%, 34%-66% by 22%, and 67%-99% by 265%). However, having all the developers or no developers in common reduces the chances of conflicts (41% and 16%, respectively). Finally, we analyzed how common it is for a single developer to make self-conflicts. We observed that all projects have self-conflicts with a huge variation on the proportion. While some projects have only 5.46% of self-conflicts, other projects have up to 66.23% of self-conflicts.

While some attributes have a large impact on the occurrence of merge conflicts, they may not be used as predictive attributes since the probability of having a conflict given the value of these attributes is relatively small. Nonetheless, these attributes can be used to elaborate policies and best practices to reduce the chances of merge conflicts. The adoption of recognized best practices such as frequent commits, small changes, continuous integration, among others, can be reinforced with attention to the number of developers involved and conflicting changes by the same developer.

As future work, we intend to increase the number of attributes and further investigate some of them by conducting a qualitative study on the programming language (what actually influences a language to have greater chances of conflicts, such as verbosity, developer freedom, among other aspects) and self-conflicts (if self-conflicts are evenly distributed among the project's committers or if some committers concentrate the majority of self-conflicts). We also would like to verify our results with some of the analyzed project communities. Finally, we intend to develop a tool that analyzes the project's history and measures these metrics from time to time to warn the project team.

Acknowledgements

This work was partially supported by CAPES (88882.464250/2019-01), CNPq, and FAPERJ.

References

Accioly, P., Borba, P., and Cavalcanti, G. (2018). Understanding semi-structured merge conflict characteristics in open-source Java projects. *Empirical Software Engineering*, 121:2051 – 2085.

Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *20th International Conference on Very Large Data Bases (VLDB)*, pages 487 – 499, San Francisco, CA, USA.

Anderson, T. W. and Darling, D. A. (1954). A test of goodness of fit. *Journal of the American statistical association*, 49:765–769.

Bird, C., Zimmermann, T., and Teterev, A. (2011). A theory of branches as goals and virtual teams. In *4th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 53 – 56, Waikiki, Honolulu, HI, USA.

Brindescu, C., Ahmed, I., Jensen, C., and Sarma, A. (2020a). An empirical investigation into merge conflicts and their effect on software quality. *Empirical Software Engineering*, 25:562 – 590.

Brindescu, C., Ahmed, I., Leano, R., and Sarma, A. (2020b). Planning for untangling: Predicting the difficulty of merge conflicts. In *42nd IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 801 – 811, Seoul, South Korea.

Brun, Y., Holmes, R., Ernst, M. D., and Notkin, D. (2011). Proactive detection of collaboration conflicts. In *19th*

ACM Special Interest Group on Software Engineering (SIGSOFT) Symposium and the 13th European conference on Foundations of software engineering (ESEC), pages 168 – 178, Szeged, Hungary.

Chacon, S. and Hamano, J. (2009). Pro git. *Berkeley, CA*, 1:509.

Costa, C., Figueiredo, J., Murta, L., and Sarma, A. (2016). Tipmerge: recommending experts for integrating changes across branches. In *24th International Symposium on Foundations of Software Engineering (FSE)*, pages 523 – 534, Seattle, WA, USA.

Costa, C., Figueiredo, J. J., Ghiotto, G., and Murta, L. (2014). Characterizing the problem of developers' assignment for merging branches. *International Journal of Software Engineering and Knowledge Engineering*, 24:1489 – 1508.

da Silva, D. A. N., Soares, D. M., and Gonçalves, S. A. (2020). Measuring unique changes: How do distinct changes affect the size and lifetime of pull requests? In *14th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS)*, pages 121 – 130, Natal, Brazil.

Dias, K., Borba, P., and Barreto, M. (2020). Understanding predictive factors for merge conflicts. *Information and Software Technology*, 121:106256.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine*, 17:37 – 37.

Fayyad, U. M. and Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87 – 102.

Ghiotto, G., Murta, L., Barros, M., and Hoek, A. V. D. (2018). On the nature of merge conflicts: a study of 2,731 open source Java projects hosted by GitHub. *IEEE Transactions on Software Engineering*, 48:892 – 915.

Gousios, G. and Zaidman, A. (2014). A dataset for pull-based development research. In *11th Working Conference on Mining Software Repositories (MSR)*, pages 368 – 371, Hyderabad, India.

Han, J., Kamber, M., and Pei, J. (2012). Data mining concepts and techniques (3rd edition).

Leßenich, O., Siegmund, J., Apel, S., Kästner, C., and Hunsen, C. (2018). Indicators for merge conflicts in the wild: survey and empirical study. *Automated Software Engineering*, 25:279 – 313.

Lu, H., Feng, L., and Han, J. (2000). Beyond intratransaction association analysis: mining multidimensional intertransaction association rules. *ACM Transactions on Information Systems (TOIS)*, 18:423 – 454.

Macbeth, G., Razumiejczyk, E., and Ledesma, R. D. (2011). Cliff's delta calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica*, 10:545–555.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 18:50–60.

Menezes, J. W., Trindade, B., Pimentel, J. F., Moura, T., Plastino, A., Murta, L., and Costa, C. (2020). What causes merge conflicts? In *34th Brazilian Symposium on Software Engineering (SBES)*, pages 203 – 212, Natal, Brazil.

Nagappan, N. and Ball, T. (2005). Use of relative code churn measures to predict system defect density. In *27th International Conference on Software Engineering (ICSE)*, pages 284 – 292, St. Louis, MO, USA.

Owhadi-Kareshk, M., Nadi, S., and Rubin, J. (2019). Predicting merge conflicts in collaborative software development. In *13th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1 – 11, Porto de Galinhas, Brazil.

Romano, J., Kromrey, J. D., Coraggio, J., and Skowronek, J. (2006). Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In *10th Annual Meeting of the Florida Association of Institutional Research (FAIR)*, pages 1 – 3, Florida, USA.

Sarma, A., Redmiles, D. F., and Hoek, A. V. D. (2011). Palantir: Early detection of development conflicts arising from parallel code changes. *IEEE Transactions on Software Engineering*, 38:889 – 908.

Vale, G., Schmid, A., Santos, A. R., Almeida, E. S. D., and Apel, S. (2020). On the relation between GitHub communication activity and merge conflicts. *Empirical Software Engineering*, 25:402 – 433.

Zimmermann, T. (2007). Mining workspace updates in cvs. In *4th International Workshop on Mining Software Repositories (MSR)*, page 11, Washington, DC, USA.

Zimmermann, T., Weisgerber, P., Diehl, S., and Zeller, A. (2004). Mining version histories to guide software changes. In *26th International Conference on Software Engineering (ICSE)*, pages 563 – 572, USA.