


Understanding and Analyzing Factors that Affect Merge Conflicts from the Perspective of Brazilian Software Developers

Barbara Beato Ribeiro  [Universidade Federal do Estado do Rio de Janeiro | barbara.ribeiro@edu.unirio.br]

Catarina Costa  [Universidade Federal do Acre | catarina.costa@ufac.br]

Rodrigo Pereira dos Santos  [Universidade Federal do Estado do Rio de Janeiro | rps@uniriotec.br]

Abstract

Merge conflicts are very common in collaborative software development, which is supported mainly by the use of branches that can be potentially merged. In this context, several studies have proposed mechanisms to avoid conflicts whenever possible and some identified factors that lead to conflicts. In this article, we report on an investigation of factors that can lead to conflicts or that can somehow reduce the chances of conflict from the developers' perspective. To do so, based on related work, we conducted two empirical studies with Brazilian software developers to both understand and analyze factors that affect merge conflicts. Firstly, we conducted survey research with 109 software developers to understand how they use branches, the occurrence of conflicts and the resolution process, and factors that can lead to or avoid conflicts. Results showed that the use of branches is very common and mostly has the purpose of creating a new feature or fixing a bug. According to the participants, in most projects, developers have the autonomy to create new branches and sometimes conflicts happen. The main factors that can lead to conflicts are "the time a branch is isolated" and "lack of communication". On the other hand, the factors cited as good practices to avoid conflicts were "improve team communication" and "less branching duration". Secondly, we conducted a field study based on interviews with 15 software developers to analyze those factors to understand better what leads to or avoids conflicts in a merge. Finally, this work allowed us to conclude that communication with the team, checking code updates, shorter branch duration, and management are important for software developers, especially when they think about what increases and decreases merge conflicts.

Keywords: *Version Control, Merge Conflicts, Survey Research, Field Study, Software Developers*

1 Introduction

Version Control Systems (VCS) allow the creation of parallel branches in a simplified way. However, there is a cost regarding merge conflicts, which are common in collaborative software development. Developers usually combine the work they have performed in parallel and may have changed the same parts of a specific file. Although the solution is frequently present in one or both conflicting versions, it does not necessarily mean that it is a trivial task (Ghiotto et al., 2018).

Conflict resolution might degrade the quality of the merged code and requires a deeper understanding of the program's structure and goals (Shihab et al., 2012; Brindescu et al., 2020a). The person in charge may not have all the necessary knowledge to make the best decision or not feel comfortable making decisions by himself/herself over source code that was coded by other developers (Shihab et al., 2012; Costa et al., 2014). In some cases, it may be necessary to verify the knowledge of developers in the changes made in the branches to choose one or more developers to resolve the conflict (Costa et al., 2019).

In this context, recent studies (Leßenich et al., 2018; Owhadi-Kareshk et al., 2019; Dias et al., 2020; Menezes et al., 2020, 2021; Vale et al., 2020) have investigated factors, indicators and attributes that can lead to merge conflicts. Such studies have found evidence that some factors can impact merge conflicts more than others. Therefore, we decided to use this knowledge as a reference to verify the software developer's perspective in relation to factors that can lead or

help to avoid merge conflicts. As such, based on related work, we conducted two empirical studies to both understand and analyze factors that affect merge conflicts.

Firstly, we conducted survey research with 109 Brazilian software developers to understand the way they use branches, the occurrence of conflicts and the resolution process, and factors that can lead to or avoid merge conflicts. The following three research questions guided our survey:

- RQ1 (Branches): How often are branches created in software projects?
- RQ2 (Merge Conflicts): What factors lead to merge conflicts?
- RQ3 (Resolve Conflicts): Which practices do developers generally adopt to avoid merge conflicts?

We found that the main factors that can lead to conflicts are "the time a branch is isolated" and "lack of communication". This communication refers to the awareness of parallel changes: sometimes developers forgot to communicate what they were changing, resulting in two developers changing the same functionality or something very close. On the other hand, the factors cited as good practices to avoid conflicts were "improve team communication" and "less branching duration". Others mentioned by the participants were "divide the work among the team", "small changes", and "frequent commits". We also identified that the main reasons to create a branch are "create new features" and "bug fixes", and participants mentioned that developers create branches "frequently".

Secondly, we conducted a field study based on interviews

with 15 Brazilian software developers to analyze those factors to obtain a better understanding of what leads to or avoids merge conflicts. The following two new research questions guided our field study:

- RQ4 (Produce conflicts): How do the factors identified in the survey research mostly contribute increasing merge conflicts?
- RQ5 (Avoid conflicts): How do the factors identified in the survey research mostly contribute decreasing merge conflicts?

We deepened the factors highlighted in the survey and observed that most software developers agree with them and went through some situation that reinforces their opinion. Furthermore, time of experience was mentioned, highlighting that the experience can modify the software developer's perception regarding the question and, that the technology itself could evolve in this time, improving the work.

This article is an extended version of a conference paper (Costa et al., 2021) in which we answered the first three research questions, focused on the characterization of software developer's perceptions of factors related to merge conflicts. We complement our previous work by adding two new research questions analyzing how developers see these factors and if and how they contribute to increasing and/or decreasing the chances of a merge conflict occurring.

This article is organized as follows. We explain the merge conflict scenario and discuss related work in Section 2. In Section 3, we describe the research method. We present the studies conducted in this work, as well as their results and findings in Sections 4 and 5. Discussion and implications are presented in Section 6. Section 7 refers to threats to validity and credibility. Finally, Section 8 concludes this paper with some final remarks and opportunities for future work.

2 Background

In this section, we discussed the concepts of merge conflicts and other works that also investigated factors or attributes that can lead to conflicts.

2.1 Merge Conflicts

Textual or physical conflicts occur due to simultaneous modifications (e.g., addition, removal or editing) over the same physical parts of a file (e.g., same line) by several developers. Direct conflicts are detected by a VCS and require resolution from a developer or a project team. Figure 1 shows an example of a conflicting chunk detected by Git where each part of the chunk has a version of a function to sum two values in Python programming language. In this case, a developer in charge must choose one of the versions, since they have the same intention.

Ghiotto et al. (2018) verified how developers resolved conflicting chunks across 2,731 Java projects. The authors found that the resolution of conflicting chunks is frequently present in one of the versions and three quarters of the conflicting

```

<<<<<< HEAD
def sum(x, y):
    result = x + y
    return result
=====
def sum(value1, value2):
    return value1 + value2
>>>>>>

```

Figure 1. Conflict detected by VCS

chunks were resolved by choosing one of the versions - version 1 (50%) or version 2 (25%). In some cases, it was necessary a concatenation (3%), or a combination (9%), or even a new code (13%). This does not necessarily mean that it is a trivial task, the person in charge must understand the conflicting intentions and generate a single version. Vale et al. (2021) investigated the influence of some factors on conflict resolution time and found that the number of chunks, lines of code, conflicting chunks, developers involved, conflicting lines of code, conflicting files, and the complexity of the conflicting code influence the merge conflict resolution time.

Accioly et al. (2018) found that merge conflicts happened in 9.38% of their data set. The authors also mentioned that merging branches is not likely to be a simple task, since one needs to understand and merge contributions performed by different developers, probably working on different assignments (Accioly et al., 2018). Menezes et al. (2020) found that merge conflicts happened in 7.11% of their data set, but the number of merge conflicts is more than 20% in some projects. Kasi and Sarma (2013) analyzed a set of projects and found that merge conflicts ranged from 7.6% to 19.3%. In the study conducted by Brun et al. (2011), 17% of merge operations required human assistance to resolve a textual conflict.

As conflicts can be common, their consequences can be a problem for the quality of some projects. As mentioned by Brindescu et al. (2020a), this situation can affect the code quality, given that developers can follow an established process of peer review of code submissions. However, a solution with a lower quality can be produced during the resolution of the merge.

In fact, merge conflicts are widely discussed in the literature. Some works (Sarma et al., 2008; Brun et al., 2011; Sarma et al., 2011; Guimarães and Silva, 2012; Estler et al., 2013) aim to prevent conflicts by monitoring workspaces and notifying developers of the potential conflicts. Such approaches are important initiatives, but they do not guarantee conflict-free merges, mainly due to the adoption of branches. Others (Cavalcanti et al., 2015; McKee et al., 2017; Accioly et al., 2018; Ghiotto et al., 2018) try to characterize merge conflicts in order to learn more about the topic and support initiatives that help to reduce the number of conflicts. On the other hand, researchers (Leßenich et al., 2018; Owhadi-Kareshk et al., 2019; Dias et al., 2020; Menezes et al., 2020, 2021; Vale et al., 2020) have started looking at factors, attributes and indicators that can lead to or avoid conflict more recently.

Table 1. Related Work

Attributes	(Leßenich et al., 2018)	(Owhadi-Kareshk et al., 2019)	(Vale et al., 2020)	(Dias et al., 2020)	(Menezes et al., 2020)	(Menezes et al., 2021)
Abstract Syntax Tree (AST) nodes changed	X	–	–	–	–	–
Changed chunks	X	–	X	–	–	X
Changed files	–	X	X	X	X	X
Changed files in both branches (intersection)	X	X	–	–	X	X
Changed lines of code	X	–	X	X	–	X
Changes inside class declarations	X	–	–	–	–	–
Commit density	X	X	–	–	–	X
Commits	X	X	X	X	X	X
Communication measures	–	–	X	–	–	–
Developers	X	X	X	X	X	X
Duration	X	X	X	X	X	X
Files with merge conflict	–	–	–	X	X	X
Length of commit messages	–	X	–	–	–	–
Merge conflict occurrence	X	–	X	X	X	X
Modularity	–	–	–	X	–	–
Predefined keywords in commit messages	–	X	–	–	–	–
Programming language	–	–	–	–	–	X
Self-conflict	–	–	–	–	X	X

2.2 Related work

The studies (Leßenich et al., 2018; Owhadi-Kareshk et al., 2019; Dias et al., 2020; Menezes et al., 2020, 2021; Vale et al., 2020) that investigated factors, attributes or indicators that may lead to conflicts analyze timing and size attributes of merge scenarios, such as commits, committers, lines of code, files, and others. These studies and the factors, attributes or indicators are summarized in Table 1.

Leßenich et al. (2018) investigated indicators to predict the number of merge conflicts. Such indicators were inferred from a survey with 41 developers. In the survey, developers mentioned what causes merge conflicts: formatting changes, large-scale refactoring, structural changes in long-living forks, and import statements. Next, the authors conducted an empirical study with 163 open source projects, including 21,488 merge scenarios. They investigated the correlation of some indicators (commits, files, chunks, lines of code, developers, and others) with the number of conflicts. For example, they explored the commit density, with the hypothesis that “many commits within a small time span are more likely to produce conflicts than the same number of commits over longer time spans”. They did not observe any strong correlation with the number of conflicts and rejected this hypothesis. In fact, they found that no indicator analyzed in work can predict the number of merge conflicts, as suggested by the survey.

Owhadi-Kareshk et al. (2019) also investigated if conflict prediction is feasible. So, they designed a classifier for predicting merge conflicts. The authors conducted an empirical study with 744 open source projects, including 267,657 merge scenarios, written in seven programming languages. They created and used a set of potentially predictive features for merge conflicts based on the literature on software merging. Similarly to the work of Leßenich et al. (2018), they

also investigated the commit density, with the intuition that “lots of recent activity may increase the chance of conflicting changes”. Moreover, they did not find a correlation between their feature sets and conflicts, but they were able to indicate merge scenarios that are not likely to have conflicts.

Dias et al. (2020) investigated the effect of modularity, size, and timing of developer’s contributions on merge conflicts. The authors conducted an empirical study with 125 open source projects, including 73,504 merge scenarios, written in two programming languages. They found that “conflict occurrence significantly increases when contributions to be merged are not modular”. They also mentioned that “conflict occurrence increases when contributions to be merged have more developers, commits, and changed files” and “contributions developed over longer periods of time are more likely associated with conflicts”.

In a previous study, we also investigated size and timing attributes that can lead to conflicts (Menezes et al., 2020). We conducted an empirical study with 80 open source projects, including 182,273 merge scenarios, written in ten programming languages. We performed statistical tests and mined association rules. We found that some attributes in the branch that is being integrated (branch 2) have more influence than the same attributes in the other branch. For example, committers, commits, and changed files in branch 2 have a large impact on the occurrence of merge conflicts. Timing attributes, commits in branch 1, and changed files in branch 1 have a small influence. It is relevant to mention that this work calculated the metrics (except the timing attributes) by branch. The timing attributes were calculated by merge scenario, as well as the other attributes of the other works described here. Menezes et al. (2021) verified more attributes (chunks, changed lines of code, commit density, programming language) in a second study. The attributes that presented a

higher relation to the occurrence of merge conflicts were changed files, commits, and committers in the branch B2 (as in the first study), and changed lines of code in B2.

Vale et al. (2020) investigated the role of communication activity in the occurrence or avoidance of merge conflicts. The authors conducted an empirical study with 30 open source projects involving 19,000 merge scenarios. They mined and linked contribution (Git) and communication (Github) data. They quantified the amount of Github communication in merge scenarios (the communication of all active contributors - awareness-based, means of pull requests, and related issues - pull-request-based, and the communication mapped to artifacts that have been changed in the merge scenario - changed-artifact-based). The authors found no significant relation between communication measures and number of merge conflicts. They also performed a multivariate analysis using merge scenarios' characteristics, such as size, number of developers, and duration. Against their expectations, they did not find a strong correlation between the size of merge scenario code changes and the occurrence of merge conflicts.

Finally, related work investigated similar attributes, although they reached different results. It is worth mentioning that they used different analysis techniques, projects and languages, and some attributes are calculated differently as well. However, the important implication of such related studies to the present work is the possibility of gathering some knowledge and investigating the developers' perspective through a qualitative method focused on empirical studies addressing characteristics of open source projects.

3 Research Method

Based on related work identified as the first step of this work (Section 2), we conducted two empirical studies to both understand and analyze factors that affect merge conflicts. Firstly, we conducted survey research with 109 software developers to understand the way they use branches, the occurrence of conflicts and the resolution process, and factors that can lead to or avoid conflicts. Secondly, we conducted a field study based on interviews with 15 software developers to analyze those factors to obtain a better understanding of what contributes to increase or decrease merge conflicts.

We conducted survey research with Brazilian software developers. The survey aimed to collect opinions on the actions that software developers usually take when they need to create or work in branches and merge code files. The study was directed to software developers who used any VCS to coordinate changes in their projects. Next, we performed a field study with 15 developers based on conducting interviews. The field study aimed to deepen and detail the answers obtained in the survey research. These studies allowed us to organize a discussion and point out implications to researchers and practitioners in the field.

4 Understanding Factors that Affect Merge Conflicts

In this section, we present details on the survey planning and execution, as well as information about the survey participants. Finally, we answer our first three research questions.

4.1 Planning and Execution

We adopted the following steps to run the survey based on the principles presented by Pfleeger and Kitchenham (2001): (1) setting specific and measurable objectives, (2) planning and scheduling the survey, (3) preparing the data collection instrument, (4) validating the instrument, (5) selecting participants, (6) analyzing the data, and (7) reporting the results.

We planned and constructed our questionnaire from the first three research questions presented in Section 1 and based on the factors mentioned in related work (Leßenich et al., 2018; Owhadi-Kareshk et al., 2019; Dias et al., 2020; Menezes et al., 2020; Vale et al., 2020), mainly in the survey provided by Leßenich et al. (2018). This questionnaire was divided into three sections: (1) basic information and professional experience, (2) use of branches, and (3) merge conflicts. Our previous work and survey responses in Portuguese are publicly available on Github¹.

We performed a pilot with four software development practitioners aiming at validating the questionnaire and estimating response time. Based on the answer and suggestions, we adjusted and improved the questionnaire. We sent out the questionnaire to developers via email, together with some contextual information such as the research objective, expected knowledge in version control, and estimated time to answer (5 minutes). As we used mailing lists and asked developers to share the survey with colleagues, we cannot compute a response rate. Open and closed questions were used in the survey. The questions included in the survey are:

1. Age (Less than 24 years old, Between 25 and 34 years old, Between 35 and 44 years old, Between 45 and 54 years old, More than 55 years old);
2. Level of education (High school, Technical education, Bachelor's degree, Specialization degree, Masters' degree, PhD);
3. Job sector (Private Sector, Public Sector, Both, Self-employed);
4. Experience (Between 1 and 5 years, Between 6 and 10 years, Between 11 and 15 years, Between 16 and 20 years, More than 20 years);
5. Average size of the project teams (Between 1 and 5 people, Between 6 and 10 people, Between 11 and 15 people, More than 15 people);
6. Version control tools (Clear Case, CVS, Git, Jazz, Mercurial, PVCS Version Manager, RSC, Subversion, Team Foundation Server, Visual Source Safe, Others: <OPEN FIELD>);
7. Branch creation frequency (Rarely, Sometimes, Frequently, Very Frequently, Always);

¹<https://github.com/catarinacosta/macTool/blob/master/SurveyAnswers-SBES2021.xlsx>

8. Reason (Test, Bug fixes, Release, New features, Refactoring, Others: <OPEN FIELD>);
9. Branch creation policy (Developers have autonomy to create new branches, Only the project manager or the person who maintains the software, The team decides, Others: <OPEN FIELD>);
10. Conflicts frequency (Rarely, Sometimes, Frequently, Very Frequently, Always);
11. Factors that contribute to the occurrence of conflicts (Number of changed files, Number of changed lines, Number of commits, Number of developers, Branching duration, Lack of communication, Developer working in several branches, Others: <OPEN FIELD>);
12. Time to resolve a merge conflict (Some hours - less than 24 hours, Some days - 1 to 6 days, One week, More than a week);
13. Difficulty in resolving a merge conflict (Very easy, Easy, Medium, Difficult, Very difficult);
14. Practices to avoid conflicts (Team communication, Less Branching duration, Small changes, Frequent commits, Divide the work among the team, Others: <OPEN FIELD>).

We adopted the card sorting approach (Spencer, 2009; Zimmermann, 2016) to analyze the answers to the open-ended questions (in this questionnaire, optional questions 6, 8, 9, 11, and 14, in which the participants could enter other data) and obtained some answers not listed in the initial survey options. To do so, we grouped similar responses to the open-ended questions into codes. The coding was performed by two researchers who discussed the codes and categories and then were reviewed by another researcher with 10 years of experience in qualitative studies. An example of the coding is presented in Figure 2, in which the codes are first extracted and the categories emerge after checking the similarity.

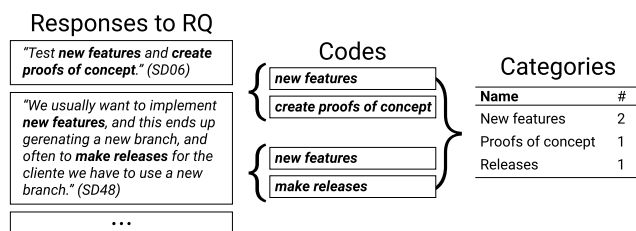


Figure 2. Example of coding

4.2 Results

From the 109 Brazilian software developers that answered the questionnaire, 38.5% are between 25 and 34 years old, and 33% are between 35 and 44 years old. Less than 24 years old are 12.8%, and between 45 and 54 years old are 11%. Finally, more than 55 years old are only 5%. 35.8% have Bachelor's degree, Masters' degree (29.4%), or Specialization degree (22%).

We asked participants where they worked and how much experience in software development they had. Regarding the experience as a developer, 27.5% have between 11 and 15

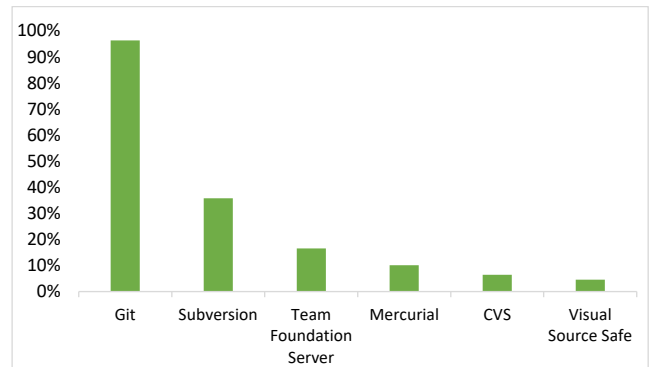


Figure 3. Experience with version control systems

years of experience, and also 27.5% have between 6 and 10 years of experience. Moreover, 23.9% have between 1 and 5 years of experience, 11% have more than 20 years of experience, and 10.1% have between 16 and 20 years of experience.

Additionally, we asked participants on the number of people in the last project they participated (or on average in their career). 38.5% answered that they worked in teams from 1 to 5 members, 36.7% worked in teams from 6 to 10 members, and 15.6% worked with more than 15 members. Finally, 9.2% answered that they worked in teams from 11 to 15 members.

We also wanted to identify which tools developers used to adopt for version control. In this question, the participant was allowed to mark more than one answer. 105 (96.3%) developers marked that they have experience with Git, 39 (35.8%) have experience with Subversion, and 18 (16.5%) have experience with Team Foundation Server. Mercurial, CVS, and Visual Source Safe were also mentioned. The developers were free to include other types of VCS not listed in the questionnaire, but no one answered anything different from the list. The information is shown in Figure 3.

4.2.1 RQ1 (Branches): How often are branches created in software projects?

We asked participants how often they create branches on software development. In the case of named branches, we believe that there is a scenario that may be more likely to conflict and be more complex to resolve. We would like to know the reason for the creation as well as the policies adopted to do so. Respondents could answer: rarely, sometimes, frequently, very frequently, or always. The prevalent answer was "always" (45.9%). Developers also chose "very frequently" (19.3%), and "frequently" (15.6%). Therefore, we can say branching is a very common practice among the participants. Results are shown in Figure 4. We also verified that developers create branches "always" in projects of private companies (64.2%) more than in government projects (26.5%).

We verified the main reasons for creating branches. In this question, developers were allowed to mark more than one answer. 94 (86.2%) participants answered that the main reason is "to create new features", 81 (74.3%) answered "to fix bug", and 46 (42.2%) mentioned "refactoring". "Test" (35.7%) and "Release" (35.7%) were also chosen by 39 respondents as

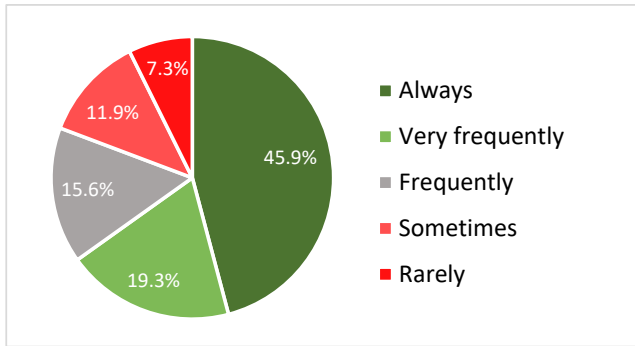


Figure 4. Frequency of branching

the main reasons. Participants could also use the open field to write other reasons. Two developers mentioned “proof of concept”, and one mentioned “enable the collaboration of different people”. The results are shown in Table 2.

Some software developers (hereafter referred to as SD) used the open field not to add a new reason, but to explain the selected reasons:

*“We usually want to implement **new features** and this ends up generating a new branch, (...) many times to **make releases** for the client we have to use a new branch.” (SD48)*

*“**Refactoring** is what we do most in the private company I work for.” (SD59)*

Others developers also mentioned a different reason:

*“**Test new features and create proofs of concept.**” (SD06)*

*“For different people to be able to participate in the **collaborative development.**” (SD83)*

Moreover, we evaluated policies adopted by the participants’ projects for creating a new branch. They responded that developers have “autonomy to create new branches” (68.8%). In contrast, others answered that the “team decides when a new branch will be created” (23.9%). Only 7.3% marked “only the project manager or the person who maintains the software”. Participants could also use the open field for other response options. Four developers mentioned the “use of Git Flow”, a set of guidelines and a tool for creating and standardizing the use and name of branches in a project. One developer also pointed that “branches are automatically created by code review and pipeline systems”, and another mentioned that the policy is “not use branch”. The results are shown in Table 3.

Some developers used the open field not to add a new policy, but to explain the selected policy, as exemplified in the following:

*“The **developers create** as many branches as they think it is necessary, but each one is responsible to constantly update the branch and integrate with the work of the others or exclude it if it does not have a well-defined purpose.” (SD48)*

Table 2. Reasons for creating branches

Reasons	#	%
New features	94	86.2%
Bug fixes	81	74.3%
Refactoring	46	42.2%
Testing	39	35.7%
Release	39	35.7%
Reasons also mentioned		# -
Proof of Concept	2	-
Enable the collaboration of different people	1	-

Table 3. Policies for creating branches

Policies	#	%
Developers have autonomy to create new branches	75	68.8%
Team decides when a new branch will be created	26	23.9%
Only the project manager or the person who maintains the software	8	7.3%
Policies also mentioned		# -
Git Flow	4	-
Automatically created (by code review and pipeline/continuous delivery systems)	1	-
Commit to a master branch (no branch)	1	-

*“**The team** always discuss when it is really worth creating a new branch, managing new branches is difficult and if we are not in control something can be wrong.” (SD87)*

Two developers selected the “team decides when a new branch will be created” and mentioned the “Git Flow strategy”. Two developers selected that “the developers have autonomy to create new branches”, and also mentioned the “Git Flow”. As such, although the projects adopt a similar strategy and tool, some projects give more autonomy to members and others prefer to discuss each decision in depth:

*“We use **Git Flow**, where both developers and managers have responsibilities when creating branches.” (SD108)*

*“A production and a development branch, based on the concept of **Git Flow.**” (SD28)*

Answer to RQ1: Branches are created frequently. Developers have the autonomy to decide when to create the branch. The main reasons are to create new features and bug fixes.

4.2.2 RQ2 (Merge Conflicts): What factors lead to merge conflicts?

We found that the use of branches is very common. However, as mentioned by Shihab et al. (2012), such level of isolation sometimes implies a cost of having to resolve integration conflicts. To measure how often merge conflicts occur, we asked

developers to estimate the frequency: rarely, sometimes, frequently, very frequently, or “all the time”. For 45% of the participants, conflicts occur “sometimes”. The second most chosen option was “frequently” (24.8%). In turn, the third most chosen option was “rarely” (16.5%). It is important to mention that for 13.8%, conflicts occur “very frequently”. This measure leads us to conclude that conflict occurrence is common to be between “sometimes” and “frequently”. Results are shown in Figure 5. We also found that conflicts are more common in government projects (52.9% of developers working for the government scored “frequently” or “very frequently”) than in projects of private companies (24% of developers working for private companies selected the option “frequently” or “very frequently”).

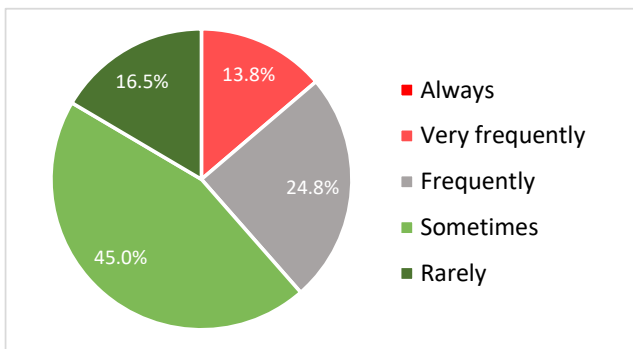


Figure 5. Frequency of conflict occurrence.

We also checked the factors that lead to the occurrence of conflicts. In this question, participants were allowed to mark more than one answer. 76 (69.7%) developers marked the option “branching duration”, i.e., the time a branch is isolated. The “lack of communication” among a team’s members was also chosen by 64 (58.7%), and the “number of changed files” was cited by 53 (48.6%). The “number of developers” was also chosen by 42 (38.5%). Developers could also use the open field for other response options. Five developers said that “not synchronizing the repositories” can lead to conflicts. Three developers mentioned the “difference in the code formatting” as a reason that can lead to conflicts and two developers mentioned “tasks not mapped correctly”. Results are shown in Table 4.

Some developers used the open field to explain the selected factors that can lead to conflicts and also to add more factors. They selected and mentioned factors such as “time between the branch and the merge” and “lack of communication”, but they rather mentioned the fact that “repositories are not kept up to date” and “complex functionalities” can lead to conflicts, as exemplified next:

“Generally the longer the time between the branch and the merge, the more files tend to be changed (...), resulting in greater possibilities of conflicts. Another point that influences is the non-practice of constant rebase, leaving the branch out of date with respect to its origin (usually the master). Complex functionality can also influence branches that take longer to merge.” (SD04)

“The lack of communication is the worst of them. Because if the team communicates daily, one

Table 4. Factors that lead to conflicts

Factors	#	%
Branching duration	76	69.7%
Lack of communication	64	58.7%
Number of changed files	53	48.6%
Number of developers	42	38.5%
Number of lines of code	31	28.4%
Same developers in many branches	28	25.6%
Number of commits	24	22.0%
Factors also mentioned		
Do not keep repositories up to date	5	-
Code formatting	3	-
Tasks not mapped correctly	2	-
Coupling level of the code	1	-
Complex features	1	-
Long time to deploy	1	-
Many features in development	1	-
Tasks not correctly mapped/broken into small pieces	1	-
Technical debt	1	-

knows what the others are up to, and conflicts are mitigated/reduced. If conflicts are not easy, you need more communication or more frequent integration (minor merge).” (SD55)

“Whenever there’s a conflict, it is because developers forgot to communicate what they were changing, resulting in two developers changing the same functionality or something very close.” (SD16)

Some developers mentioned unlisted factors, such as the difference in the “code formatting”, “tasks not mapped correctly” and “technical debt”. As conflicts occur in modifications in the same code region, criticism of minor issues such as code writing and style is understandable:

“Lack of configuration in the editors, which change between indentation with tab/space, amount of space, line break... just opening and saving the file, and lack of style in the code, where each one writes the code in a different way, and another developer adds/removes spaces, parentheses (...) this causes a change in one line to reflect the entire file.” (SD26)

“Tasks not mapped correctly/broken into small enough parts correctly that lead to interfering with the same pieces of code. Accumulated technical debt that requires changes in many places, for example regarding code formatting, use of depreciated techniques, etc...” (SD76)

Answer to RQ2: Conflicts sometimes occur. The main reasons that can lead to merge conflicts are the time a branch is isolated and lack of communication.

4.2.3 RQ3 (Resolve Conflicts): Which practices do developers generally adopt to avoid merge conflicts?

As conflicts are common, we asked participants about the difficulty in resolving a conflict and which practices they be-

lieve may contribute to avoiding merge conflicts. To verify the time to resolve a conflict, we asked developers to estimate the duration: some hours, some days, one week, or more than a week. Most of them (80.7%) answered that they spent “less than 24 hours” to resolve the conflict. Some of them (17.4%) answered that they spend “some days” (1 to 6) to resolve the conflict. Only 2 (1.9%) participants answered “one week”.

We verified the difficulty level in resolving a merge conflict from their perspective, as Greiler et al. (2022) cite: “factors may impact a specific developer’s experience and depends on his/her personal, team, organization, and project contexts”. Some developers answered “easy” (32.1%) and “medium” (32.1%) and some of them answered “very easy” (22.9%). The results about the time to resolve the conflict and the level of difficulty are shown in Table 5.

Table 5. Time to resolve a conflict and difficulty level

Time to resolve conflict	#	Difficult level	#
Less than 24 hours	88	Very easy	25
		Easy	31
		Medium	27
		Difficult	5
		Very difficult	-
Some days (1-6)	19	Very easy	-
		Easy	4
		Medium	6
		Difficult	9
		Very difficult	-
One week	2	Very easy	-
		Easy	-
		Medium	2
		Difficult	-
More than one week	0	Very difficult	-
		-	-

Finally, we investigated practices to avoid conflicts. Developers were allowed to mark more than one answer. The two most frequent answers that may contribute to reducing merge conflicts occurrence were: “improve team communication” by 78 (71.5%) participants and “less branching duration” by 75 (68.8%) participants. These factors really seem to be very important, given that “branching duration” and “lack of communication” were the most cited factors that can lead to conflicts. Participants also selected “divide the work among the team” (57.7%), “small changes” (54.1%), and “frequent commits” (52.2%) as good practices to avoid conflicts.

Developers could use the open field for other response options. Some participants informed that they do “not use new branches”, do commits directly on the main branch, “adopt code style” and “Git Flow tool”, and always keep the “workspace branch up to date with the remote repository”. Results are shown in Table 6.

Moreover, some developers used the open field to explain the selected factors that can avoid conflicts or even add unlisted factors:

“Always check for code updates in the master / trunk / main.” (SD34)

“Improve communication channels and also use

Table 6. Factors to avoid conflicts

Factors	#	%
Improve team communication	78	71.5%
Less branching duration	75	68.8%
Divide the work among the team	63	57.7%
Small changes	59	54.1%
Frequent commits	57	52.2%
Factors also mentioned	#	-
Do not use branches	3	-
Adopt code style tool	2	-
Keep the branch up to date with the master/trunk/main	2	-
Git Flow	2	-
Adopt awareness tool	1	-
Architecture patterns (more cohesion and less coupling)	1	-
Branch by task	1	-
Continuous integration	1	-
GUI to interact with repository	1	-
Frequent deploy	1	-
Feature flags	1	-
Keep only experts	1	-
Language syntax	1	-

other awareness tools to know what each one is changing.” (SD68)

“I encourage people on my team to avoid branches as much as possible and implement techniques such as feature flags for everyone to always work at master/main. Rather than dealing with conflicts, I would like ‘devs’ to become more experienced in trunk based development.” (SD31)

“Use of techniques like Git Flow.” (SD77)

“Adopt a tool that validates the code style.” (SD26)

Answer to RQ3: Developers take more than some hours to resolve a conflict since it is usually easy to do. The main factors to avoid conflicts refer to improve team communication and less time of isolation.

5 Analyzing Factors that Affect Merge Conflicts

The first study (survey research) was grounded (planned and constructed) on the factors for merge conflicts mentioned in solid related work, as mentioned previously, focusing on the Brazilian software developers’ perceptions. Based on the quantitative results, we decided to deepen the understanding of how the factors contribute to increasing or decreasing merge conflicts based on interviews in a qualitative study (field study). In this section, we present details on the planning and execution of this qualitative study with semi-structured interviews and we answer the two additional research questions.

5.1 Planning and Execution

We grounded this study on Singer et al. (2008)'s work. According to the authors, a field study aims to investigate practitioners in the context of any task or activity and, based on a specific technique, identify how they cope with their work in practice or how they solve some problems in their contexts. Based on Singer et al. (2008)'s recommendations, software developers with at least one year of experience were invited to answer a set of questions regarding two major aspects: how the factors identified in the survey research contributes to increasing or decreasing merge conflicts. These participants were invited from the researchers' networks and considered their availability to participate in an interview session.

We planned and constructed our interview questions (IQ) from the results of the first study (survey research) and focused on the last two research questions presented in Section 1. As such, we took the main factors pointed out by the software developers in the survey research and designed eight questions for the interview sessions for the field study: four questions about factors that contribute to increase merge conflicts (Table 4) and four questions about those that contribute to decrease merge conflicts (Table 6). The goal is to understand why these factors are so important and how time of experience (years of working, studying, dealing with merge conflicts and collaborating with other developers) influences and modifies the software developers' perceptions over the years. The eight grouped questions are listed below.

Regarding factors that contribute to increase merge conflicts, we asked the following questions:

- IQ1: Questions on the factors
 - Do you agree with the table presenting the factors that can lead to merge conflicts?
 - How was your experience in coping with merge conflicts early in your career?
 - How about coping with merge conflicts nowadays?
- IQ2: In your opinion, what makes the branching duration so important?
- IQ3: Questions on lack of communication
 - What is your opinion on the lack of communication?
 - Does the lack of communication affect other factors presented in the table?
- IQ4: Questions on negative effects
 - Which factor brings more negative effects? Why?
 - Has the time of experience changed your answer?

Regarding factors that contribute to decrease merge conflicts, we asked the following questions:

- IQ5: What is your opinion on the factors "lack of communication" and "branching duration" changing their positions in the table?
- IQ6: Questions on the team influence
 - How did communication with the team influence the way to avoid these conflicts?
 - Have you felt any improvement over time?

- IQ7: Questions on past experiences
 - Have you been able to cite or work on a project that covered any of these factors?
 - Was it a successful experience or not?
- IQ8: How do you see that "less branching duration" contributes to decrease merge conflicts?

A total of 15 software developers participated in the second study (hereafter referred to as FD from field study developer identifier). They were only Brazilians and answered about merge conflicts and their perceptions. The goal was to deepen the understanding of how branches are adopted, as well as conflict resolution in this context. With this in mind, an interview session of about 30 minutes was conducted with each software developer and recorded via the Zoom platform. All data and information were collected anonymously and treated specifically for academic purposes, as explained to the participants in the email invitation, informed consent form, and in the conversation before each interview.

As mentioned above, the interviewees were contacted by email. The main selection criterion was to have used some version control systems (e.g., Git, Subversion, Version Manager), which means that he/she has probably already faced some merge conflict. They were informed that they could withdraw at any time, they were allowed to not answer some questions (if they wanted to), and all video and sound data that would be collected would not be public (just collected to the study analysis purposes).

Before starting an interview, we asked about the developer's time of experience, how long he/she has been dealing with merge conflict situations (which is not necessarily linked to professional experience), what kind of industry sector he/she works in (or has worked to). In Table 7, the interviewees' time of experience is presented. From the 15 developers interviewed, 7 (46.6%) have about ten or more years of experience.

The first four questions of each interview referred to factors that contribute to increase merge conflict. From Tables 4 and 6, we could deepen this discussion by also understanding if the interviewees agree with, and/or would like to add more factors (or correlated factors). In turn, factors that contribute to decrease merge conflicts were covered by the last four questions. The goal was the same as the previous question, but we also would like to know if the software developers' time of experience affects their perceptions over time.

Firstly, a pilot was run with four software developers to verify the interview session protocol and duration. The pilot helped us to check the questions (if they were clear enough) as well as to the how long a slot would last in average to avoid stressing the interviewee and to lose the focus in complex questions.

5.2 Results

This section presents the results obtained from the interviews in the field study and the answers to the last two research questions of our work, as mentioned in Section 1. To do so, for each research question, we took as the main codes from the interviewees' answers the most frequent factors that contribute to increasing and decreasing merge conflicts based on

those reported in the first study (survey research). This strategy allowed us to get a better understanding of how those factors have an impact on practice.

Table 7. Interviewees' time of experience

Developer	Time of experience
FD01	Almost 6 years
FD02	4 years
FD03	6 years
FD04	2.5 years
FD05	12 years
FD06	7 years
FD07	24 years
FD08	14 years
FD09	3 years
FD10	10 years
FD11	12 years
FD12	1 year
FD13	14 years
FD14	4 years
FD15	15 years

5.2.1 RQ4 (Produce conflicts): How do the factors identified in the survey research mostly contribute to increase merge conflicts?

Regarding the factors that contribute to increase merge conflict and their order of importance, most of the software developers (9) agree (FD03, FD04, FD05, FD06, FD07, FD09, FD10, FD12, and FD14) with the table presented during the interview session (Table 4) and some (4) partly agree (FD01, FD02, FD08, and FD13). Only two interviewees (FD11 and FD15) declared that they do not completely agree with the table with the factors. The interviewees who did not completely agree with the table mentioned that maybe a factor should be considered as more meaningful for a specific context or scenario. "Number of lines of code" was highlighted by three software developers (FD03, FD10, and FD14) as something of greater importance. One interviewee reported an experience about one of the critical factors that contributes to increase merge conflicts:

"I had a lot of problems when I worked, even when the team was small (...) four people developing (...) It was like parallel editing of the same code and people did not have much experience in sharing code (what they did, what they edited...)." (FD11)

We invited the interviewees to comment a little bit more about their career in order to compare their beginning against their current perception. Six software developers (FD01, FD04, FD08, FD11, FD12, and FD14) explained that they did not work with either Git or other repositories early in their career. Additionally, they were running academic projects that were characterized as small and without merge problems:

"[starting with academic projects] is common in our career. As far as your projects are more

and more scaling, even the culture of the company where you work, you may have merges that end up being complicated to cope with." (FD14)

Four developers (FD7, FD10, FD13, and FD15) with more years of experience pointed out how important the evolution of version control tools is, especially for assisting situations regarding merge conflict resolution and for detecting conflicts as well. According to one of those interviewees:

"As time goes by (...), the existing tools started to carefully address this kind of activity (merge) (...) A diff not correctly done was complex for us at the first years of research and practice in version control systems (...) There was a free tool, but it was complicated to work with it considering a lot of existing bugs..." (FD07)

Team communication/behavior was mentioned by three developers (FD02, FD03, and FD06) as something noticed both early in their careers and also currently in their work. FD03 even highlighted that the size of a project - a factor mentioned direct and indirectly by more than one interviewees - also affects merge conflicts. FD06 pointed out that he noticed a programming language barrier in open source projects. On the other hand, long branch duration was referred to as important due to the changes made over time, i.e., how much code have been modified/moved in a project (FD01, FD04, FD09, FD12, and FD13). According to one of these interviewees:

"I believe that more long branches you have, more changes and modifications of code you have to cope with, implying in implementation of new features, deprecating other functions of the program and methods, and so on. As such, long branches bring bigger merge conflict problems..." (FD04)

The interviewees often argue their concern on not only what brings merge conflict problems in long branches, but also on why small branches would be more convenient. They mentioned some cases, such as a branch is outdated when it is compared to the main branch/another branch (FD05 and FD06), or a branch is associated with a sprint/short time interval (FD1, FD14, and FD15), as presented next:

"I believe that shorter branches (...) can decrease the number of merge problems." (FD04)

"... if you are running an agile method and stories that are better defined, broken into sub-tasks (...), for example, you do not have this problem, because (theoretically) you have a story in a certain sub-scope of the development of your project that will be somehow isolated." (FD01)

"...If we have a branch with a very long, very extensive time frame, (...) sometimes we cannot collect such an accurate feedback from the business area and this would be what we really need to change for production." (FD03)

Communication was called “essential” by one interviewee, “important” by another, and “fundamental” by a third one. The lack of communication was related to problems not only previously described in Table 4, but also declared as a behavioral problem, as exemplified next:

*“(…) because it leads to merge conflicts and frequently it makes **some behaviors in the software development** keep happening throughout the project and this affects code, functionalities, (…) the implementation of the project as a whole.” (FD02)*

Some developers (FD01, FD03, and FD09) mentioned that communication problems go beyond the technical aspect. This fact is highlighted in the following fragment:

*“**The lack of communication will lead to conflicts, not only in Git, but also any way of working.** Here it leads mainly (…) to cases in which you will end up messing with something that someone else was already working on…” (FD03)*

Moreover, agile methodology was also cited by 2 interviewees as a strategy to support communication in the software development project. This is pointed in the next fragment:

*“**It is clear the difference between those who use the [agile] methodology or not.**” (FD06)*

It is worth highlighting that the factors mentioned in Table 4 were also reinforced by the interviewees. One of them stated that:

*“**The lack of communication usually causes problems regarding the branching conflict, in merge, (…)** within the development team itself (…). This is critical for the understanding the time the story is started and that you are doing a part of a whole, (…) for example, not keeping the repository updated (…) will affect the parallel editing of the same code.” (FD01)*

Tools such as configuration management tools (FD10), project management tools (FD10), task organization tools (FD10), change tracking tool (FD05) and screen sharing tools (FD13) were cited as kinds of support for communication:

“...depending on the tool for change tracking, continuous change etc., I believe that (verbal) communication (…) helps you eliminate the problem a little bit. You can see ‘someone’ (…) touching exactly such and such point of the system (…) and you can verify where you can touch or not, and I believe that this impacts less on merge problems.” (FD5)

Communication problems can also lead to rework (FD11, FD12, FD13, and FD14), either by an added feature or by a change not communicated to the team:

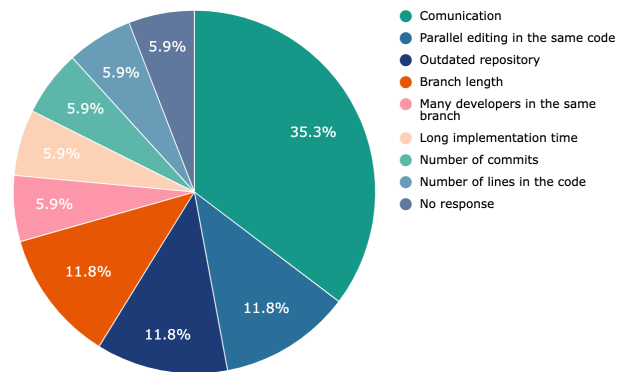


Figure 6. Factors that negatively affect merge conflicts

*“Lack of communication (…) **you end up having to redo what you did.** You thought it was right, but it was not what was supposed to be done. This generates so much rework for the developer, stress for the manager…” (FD11)*

When we asked about which factors they consider the most negative ones, nine different answers were obtained, as shown in Figure 6. Communication was the most mentioned factor in the opinion of six interviewees (FD02, FD04, FD06, FD09, FD11, and FD14).

*“...they [conflicts in merge] occur because the **distraction of several people**, myself included of course. It can lead to some error that will lead to a headache until we can solve it” (FD09)*

In this context, two problems were pointed out by two interviewees each: outdated repository (FD03 and FD07) and a long branch duration (FD8 and FD15). Some interviewees selected other problems: many developers in the same branch (FD05); long implementation time (FD07); number of commits (FD08); and number of lines of code (FD12). Only one of them (FD13) pointed out that it was hard to solve conflicts in the beginning of his career, but he would not see it as a problem at the present moment, but as something expected from the learning process over the years.

When the interviewees were asked to remember their experiences from the beginning of their career, only three of them (FD02, FD10, and FD11) believe they did not respond to the questions differently. The majority, 12 developers (FD01, FD03, FD04, FD05, FD06, FD07, FD08, FD09, FD12, FD14, and FD15), believe they would think differently on what contributes to increase merge conflicts over time.

As a conclusion, seven factors were perceived as the main problems regarding merge conflicts according to the interviewees, but that they have been rethought over time: number of modified files (FD01 and FD06); communication and duration of the branches (FD03); organization (FD04); tools used in the projects (FD07); lack of attention (FD09); and number of lines modified in the projects (FD14). When the interviewees were asked to compare their perception at the beginning of their career against their current perception, it is not clear if there is a pattern of “X-answers from the beginning changed to Y-answers at the present moment”. The interviewees mentioned situations they had experienced to justify the choice of

a factor that affected their work early in their career versus others impacting their current projects.

Answer to RQ4: The interviewees mostly agree with the factors that lead to merge conflicts, presented in Table 4. Long branches, software development methodology and communication problems were pointed out as some of the main factors to be considered in this context.

5.2.2 RQ5 (Avoid conflicts): How do the factors identified in the survey research mostly contribute to decrease merge conflicts?

When we asked the interviewees if communication should be in the first position in Table 4 (factors that lead to merge conflicts) as well as in the first position of Table 6) (factors that avoid merge conflicts), 10 interviewees (FD02, FD03, FD04, FD05, FD06, FD09, FD10, FD11, FD12, and FD13) agreed with the greater importance of this factor. This relevance is exemplified in the following fragment:

“...because communication is in fact very important and it impacts on several factors...” (FD03)

In addition, the interviewee FD05 emphasizes the effective stimuli and use of communication tools (e.g., email, chat, awareness-support systems etc.) contributes to decrease merge conflicts. The interviewee FD09 argued that good communication reduce rework and the interviewee FD6 mentioned that the problems are more related to human aspects in the software development process. As such, there was no specific type of communication specifically recommended in the interviews. The answers referred to talking more and better before working on a project, using some computational tools and applying agile methodology as a way to improve and keep communication frequent.

Some interviewees (FD07, FD08, FD14, and FD15) believe that “branch duration” is a major factor even when the goal is to decrease merge conflict. The interviewee FD08 mentioned that some factors in this context may be related to inexperience or “post-conflict” thinking. The interviewee FD14 raised a concern on to which extent we notice communication surrounding us:

“... I think it is normal to change your mind, and I think what happens is that you start thinking about the situations you have been through on the team, then you start thinking ‘if that guy had talked to me, it would be less torturous to resolve the conflict’...” (FD14).

The interviewee FD01 reported that communication was highlighted because of cultural reasons. In other words, it refers to the idea of pointing out that there is a problem and that “lack of communication” would be the main factor on this subject, being similar to “pointing the finger” to someone. By indicating communication as a strategy to decrease merge conflict, people feel more comfortable in communicating to each other:

“the communication) as something to avoid a problem rather than being the problem itself.”(FD01)

All interviewees mentioned the communication in the team, either based on an previous project (past experience), or on the one they are currently work on. In both scenarios, nine of them (FD01, FD02, FD05, FD06, FD07, FD09, FD10, FD11, and FD14) notices improvements regarding effective communication and its positive effects. As some suggestions for improving communication, some interviewees cited management, planning, and infrastructure:

“There are several factors that will influence that aspect of improving communication. I think the first one is management.” (FD03)

“There is also that question related to technological limitation. If we think about the current pandemic scenario, (...) several companies have adapted their infrastructure with resources to foster and ensure good communication.” (FD03)

“Communication works from the moment you plan how that communication is going to be done.” (FD04)

The improvements mentioned by the interviewees resulted in problem-solving (FD01), collaboration between team members (FD04), less rework (FD11) and less time doing merge (FD13). An example of this report is presented next:

“The fact that you have a person with more knowledge helps a lot who is there working on that project and who may not have enough knowledge, especially related to the business in which that project is inserted.” (FD04)

The interviewee FD15 mentioned that communication would help to resolve, rather than avoid, a merge conflict. This fragment is presented as follows:

“I don’t think there has been much change regarding this topic in the last few years (...) I think all of that is still a problem related to our inability to clearly record or summarize the developer’s intention at the time he/she writes a particular piece of code.” (FD15)

Finally, it is worth mentioning that not everyone may have faced a situation in which they realized that communication would be the key factor for avoiding a merge conflict. This is indicated in the following fragment:

“I did not have maturity on this subject before [i.e., thinking about communicating to avoid conflict]. So, if this ability is improved over time, I could only have the notion of its importance to avoid merge conflicts currently...” (FD04)

From the 15 software developers who were interviewed in our field study, 12 have some experience in implementing practices to address the factors they completely or partially agree with. Based on the factors listed in Table 6, communication (FD01, FD02, FD03, and FD09), task division (FD01, FD02, FD11, and FD15), repository up to date (FD01 and FD05), task-based branch (FD01 and FD05), branching strategy (FD01 and FD08), more frequent commits (FD01), small changes (FD01) and short branches (FD15) were the most prominent. DevOps culture was also mentioned (FD01 and FD13). Some interviewees also included some of the previously mentioned factors, e.g., training (FD04) and some support tools (FD05, FD07, FD08, FD09, FD12, and FD13), such as Trello, Discord, and VSCode. These tools were mentioned when the interviewees talked about aspects regarding communication, change history, standardization, and task division, as exemplified next:

“We standardized vscode as our IDE. There were many people who used other environments.” (FD13)

“... it required improvements. Communication should be a frequent concern. You cannot have communication only when there is a problem, i.e., it has to be a daily target.” (FD01)

Another factor mentioned by some interviewees was the branching duration (FD03 and FD08), especially because of the business. This was also mentioned by the interviewee FD11:

“...branching duration is directly linked to the business. What comes in and leaves depends on the business, the owner of the company, the client (...) and there is a little margin for negotiation.” (FD08)

Factors that contribute to decrease merge conflicts also mentioned by the interviewees refer to continuous integration (FD01 and FD08), project management environment (FD10), and more frequent commits (FD14). Branching duration was confirmed as an important factor to avoid conflicts by nine interviewees (FD04, FD05, FD06, FD07, FD10, FD11, FD13, FD14, and FD15). Other three (FD02, FD09, and FD12) did not know how to evaluate it, and the other three (FD01, FD03, and FD08) commented that it is not really the duration of a branch itself that prevents conflict.

In this regard, we found arguments for a shorter branching duration, such as the repository being up to date (FD04, FD07, and FD13), less time for code changing (FD05), less divergence (FD06), memory of what has been done and what is not affected (FD10), the speed of development (FD11), less chance of conflicts (FD13), and faster merges (FD14). The interviewee FD15 explained somehow the mentioned factors:

“The longer you are isolated in a branch, the more likely another developer will come and change the code that is in parallel with you (...). You will generally remember less and less about

it. Knowing how the code was before and having fewer developers working in the same code area as you are factors that help you solve (...) or avoid a merge conflict.” (FD15)

Answer to RQ5: The interviewees mostly agree with the factors that avoid merge conflicts, presented in Table 6. Communication (from simply conversations to those based on computational tools), team management, and infrastructure were pointed out as some of the main factors to be considered in this context.

6 Discussion and Implications

In this section, we present the main findings of this research on factors that affect merge conflicts based on a quantitative method.

1) Branches are very common and developers have the autonomy to create new branches:

Most software developers create branches frequently or all the time. The use of branches is very common according to the developers' perspective collected from the survey questionnaire, but no participant in the field study interviews mentioned if he/she did not use to do it. Only a few developers marked the option that they discuss the creation of new branches with their teams. In a large study at Microsoft, Shihab et al. (2012) identified that developers should be careful about branch creation, since it may lead to an increase in the likelihood of failures. They suggest aligning branching structure according to architectural structure and with the organizational structure of their teams (Shihab et al., 2012). As mentioned by Bird et al. (2011), branches do not come without a price, given that it is normally integrated into others at some point.

2) New features and fixing bugs are the main reasons for creating branches:

Our results confirm the findings of other studies. Zou et al. (2019) found similar results in their investigation with 2,923 projects developed on Github - branches are mainly used to implement new features, conduct version iteration, and fix bugs. Owahdi-Kareshk et al. (2019) and Vale et al. (2020) address that developers often use branches to add features or fix bugs. According to Bird et al. (2011), branches are created to implement a feature, perform a maintenance exercise, do continued maintenance on a subsystem, or fix several related bugs. Premraj et al. (2011) mentioned that branches help developers, architects, build managers, testers, and others people to change software artifacts. Additionally, the agile methodology was cited by some software developers in the field study interviews as one of the strategies to cope with the creation of a new branch without contributing to increasing or decreasing merge conflicts.

3) Branching duration and lack of communication are the main problems:

Based on the related work (Table 1), attributes related to the branching duration are very common, but only two studies mention the branch duration as an indicator of conflict.

Dias et al. (2020) and Menezes et al. (2020) found a relation between the duration of the merge scenario and the conflict occurrence. Dias et al. (2020) mentioned that “contributions developed over longer periods of time are more likely associated with conflicts”. Menezes et al. (2020) found that the timing attributes have a (small) impact on the conflicts. Vale et al. (2020) verified the relation between Github communication and the occurrence of merge conflicts. The authors found no significant relation between communication measures and number of merge conflicts. However, the communication recorded by the authors was based only on the communication extracted from Github. So, they extract the communication of all active contributors, means of pull requests, and related issues, and the communication mapped to artifacts that have been changed in the merge scenario.

The communication mentioned by the software developers who responded to the survey questionnaire is regarding the awareness of parallel changes. Sometimes developers forget to communicate what they are changing, resulting in two developers changing the same functionality or something very close. In the field study interviews, some software developers also suggested that keeping shorter branches is the best decision to avoid merging conflict problems, especially those related to developers’ communication and lack of memory on the changes performed in the project and team over time.

4) Most of the time, conflicts are not difficult:

Most conflicts offer no difficulty (medium or easy) and are resolved in some hours. Accioly et al. (2018), Ghiotto et al. (2018) and Pan et al. (2021) identified the most common conflict patterns and resolutions. Accioly et al. (2018) found that 84.57% of merge conflicts happen because developers modify the same lines or consecutive lines of the same method. Ghiotto et al. (2018) found that conflicting chunks generally contain all the necessary information to resolve them. Pan et al. (2021) found in their study on conflict resolution that 28% of changes are of 1-2 lines for both main and forked branches and 39.5% of the resolution strategies involved concatenating the main and the forked branch’s changes.

McKee et al. (2017) performed a survey and found nine factors and developers attempting to determine if the conflict is difficult, the complexity of conflicting lines of code and files, the knowledge in the area of conflicting code, and the number of conflicting lines were most cited. It is interesting to mention that some of these factors were used in some related work to predict conflict occurrence.

Brindescu et al. (2020b) also investigated the characteristics of merge conflicts that are associated with their difficulty. The authors found a subset of ten factors that can predict the difficulty of merging conflicts, including complexity, diffusion, size, and development pattern. The more experienced developers have appointed the improvement of the version control tool over time as a factor that has improved conflict resolution.

It is worth highlighting that the field study interviews also raised that the project’s size somehow influences the resolution of merge conflicts, especially in large projects (and large teams) where the chance of merge conflicts is higher. Moreover, when a developer is at the beginning of his/her career, he/she does not use to pay attention to this kind of situation, especially on the importance of communication in a project

(de Farias Junior et al., 2022).

5) Improve team communication and less branching duration can avoid conflicts:

As mentioned previously, Dias et al. (2020) and Menezes et al. (2020) found that timing measures have an influence conflict occurrence. So, we believe that a good practice is to pay attention to the isolation time and not postpone the merge so much. When developers are less isolated, the repositories are synchronized and people are aware of what other people are doing. Software developers in the survey questionnaire mentioned the importance of knowing what parts others are working on to avoid conflicts.

Communication and relation with merge conflict are investigated mainly in studies addressing awareness. Some specific studies (Sarma et al., 2008; Brun et al., 2011; Sarma et al., 2011; Guimarães and Silva, 2012; Estler et al., 2013) focus on the prevention of conflicts through awareness, i.e, detecting conflicts early. Basically, these tools monitor workspaces and inform developers of ongoing parallel changes in other workspaces. As also mentioned by some software developers in the field study interviews, it is relevant to improve communication channels and also use awareness tools to know what each one is changing. Moreover, other points related to the factors referred to having more and better conversations before starting a branch (or even a project), based on computational tools, such as Trello, as well as applying agile methodology as a strategy to reduce time span.

6) Qualitative analysis findings:

The answers to our survey questionnaire and field study interviews show that software developers also use a branch to create proofs of concept and Git Flow seems to be a good strategy to coordinate the use of branches. In addition, they suggest that not keeping the repository up to date can cause problems, so developers need to bring up the changes constantly. Attention regarding the code formatting is important. Accioly et al. (2018) noticed that part of the merge conflicts is simply caused by changes to code indentation or consecutive line edits. Regarding this problem, some software developers suggest adopting a code style tool. Furthermore, as good practices to avoid conflicts, some of them also mention the option of not using branches and adopting techniques such as feature flags. They also cited always communicating with the team and checking for code updates in the master/trunk as a good practice, as noticed in the field study interviews.

7 Threats to Validity and Credibility

This work applied a quanti-qualitative method. Therefore, there are two different empirical studies (survey research and field study) and each of them has specific threats and limitations. Each subsection below informs their threads as well as strategies to mitigate them.

1) Survey Research:

A) Protocol. We adopted some predefined answers to some closed questions, given that they were grounded on previous studies published in the literature (Owhadi-Kareshk et al., 2019; Leßenich et al., 2018; Dias et al., 2020; Menezes

et al., 2020; Vale et al., 2020). Moreover, we also leave an open field allowing a participant to comment on different factors not listed in the question. We develop the questionnaire very carefully. As it would be our main source for all sections of this study, we discussed and took a long time to construct our questionnaire. In addition to our experience on the subject, we spent a lot of time looking at the literature and building our survey based on the pieces of evidence from these studies and some similar initiatives (Condina et al., 2020; Kamei et al., 2020). We also conducted a pilot with four developers and asked for feedback on the questions, and whether they were understandable and relevant to the study.

B) Sample. The software developers who responded to the questionnaire were invited by email via contact lists and they were asked to share with their colleagues with experience in software development (snowballing invitation). We tried to make sure that only people with experience in the use of VCS answered the questionnaire, either in the invitation or in the survey description, or even in the question specifically referring to the use of any VCS. Such approach was important to avoid any participant with lack of experience or knowledge.

C) Context. We only had the participation of Brazilian software developers in our study. Results may not be generalized to the context of all software developers all over the world. Some results confirmed the findings presented in related work, but others require more in-depth investigation. In addition, according to Smith et al. (2013), high-quality research on the human side of software engineering requires real software developers, but getting high levels of participation remains a challenge for researchers. Nonetheless, it is relevant to emphasize that our results reflect the perspective of a large group (109 participants).

2) Field Study:

A) Protocol. We used the results from the survey research as the input for the questions prepared to the interview sessions, considering the main factors that affect merge conflicts according to the Brazilian software developers who answered the survey questionnaire. The developers who were interviewed were invited by email and they were requested to share with their colleagues with some experience in resolving merge conflicts (snowballing invitation). Only Brazilian software developers participated in the field study interviews. Therefore, the results may not be generalized, especially considering the interpretive validity of a qualitative study, i.e., the possibility, even without the researcher's intention, to put his/her perception instead of really understanding, to perceive what the interviewee meant.

B) Sample. Our intention was to have at least 20 interviewees, based on Guest et al. (2006)'s work regarding the occurrence of saturation with at least 12 interviews given that this research has "the aim is to understand common perceptions and experiences among a group of relatively homogeneous individuals". Moreover, Steglich et al. (2019) and Greiler et al. (2022) conducted field studies with software developers considering Guest et al. (2006)'s work and reinforce that the main important criteria is the saturation, i.e., when any new interview with relatively homogeneous individuals do not provide any new data or information. For example, Steglich et al. (2019) reached saturation with 11 interviews. In our study, 15 developers were able to participate in the pe-

riod when the field study was run. Based on the interviews, the saturation was obtained with 12 interviews and this is in accordance with Guest et al. (2006)'s work. It is important to remark that the main goal of our field study was to collect the Brazilian software developers' perceptions on merge conflicts in a qualitative setting and not through a large-scale, quantitative study based on software repository analysis.

C) Context. The same concern point out by Smith et al. (2013) is valid to the field study, i.e., "high-quality research on the human side of software engineering requires real software developers, but getting high levels of participation remain a challenge for researchers". This includes the software developers' feeling on how to proceed with the interview questions given the fear of leaking confidential information from their own projects and/or companies in which they work on. It is a critical barrier faced in field studies, given its qualitative, in-person nature (Singer et al., 2008), especially when requesting participation. Nonetheless, it is important to highlight that the result of this study reflects the vision of a group of Brazilian software developers and whose focus was on deepening the understanding the results from the previous survey research (Smith et al., 2013).

8 Conclusion

This research aimed to investigate factors that lead to or help to avoid merging conflicts. To do so, based on related work, we conducted two empirical studies to both understand and analyze factors that affect merge conflicts. Firstly, we conducted survey research with 109 software developers to understand the adoption of branches as well as the occurrence and resolution of conflicts. Results suggest that the main factors that can lead to conflicts are "the time a branch is isolated" and "lack of communication". On the other hand, the factors cited as good practices to avoid conflicts were "improve team communication" and "less branching duration". "Divide the work among the team", "small changes", and "frequent commits" were also marked many times by the participants of the survey research. Communication here refers to the awareness of parallel changes, considering the importance of knowing what others are working on. We also performed a qualitative analysis to extract codes and categories from open fields of five questions responded to the participants. We identified that Git Flow is a common strategy adopted to coordinate branches, synchronizing the repository constantly and paying attention to the formatting of the code to avoid conflicts.

Next, we conducted a field study based on interviews with 15 software developers to analyze those factors to obtain a better understanding of what contributes to increasing or decreasing merge conflicts. Results show that communication with the team, checking code updates, shorter branch duration and management (which comprises software development methodology, communication strategies and awareness-support systems) seem to be key policies, not only to merge conflict resolution, but also to decrease conflict. Moreover, the developers' time of experience can change their perception on the problems faced in this context and helps to avoid or resolve a merge conflict, besides the fact

that version control systems have evolved to a greater extent, being also an important support in this topic. Finally, this study allowed us to conclude that most of the software developers agree with the factors that lead to and the factors that avoid merge conflicts, and underlying problems and how to resolve them are still a concern for all them.

In future work, we intend to evaluate the application of some good practices suggested in this work. We could evaluate supporting processes and tools to improve communication, and reduce isolated work and other mentioned factors. Another opportunity is to perform a quantitative study based on mining software repositories in order to analyze some GitHub projects against some findings of this work, for example, through a study on the projects' branching duration, communication tactics, and merge conflict resolution. Finally, this work can be executed with software developers from other contexts (e.g., different cultures, countries, genders etc.) to produce other indications and allow systemic analyses.

Acknowledgements

We thank all the participants that answered our survey and interview. The author also thanks UNIRIO and FAPERJ (grant: 211.583/2019) for partial support.

References

- Accioly, P., Borba, P., and Cavalcanti, G. (2018). Understanding semi-structured merge conflict characteristics in open-source Java projects. *Empirical Software Engineering*, 23:2051–2085.
- Bird, C., Zimmermann, T., and Teterov, A. (2011). A theory of branches as goals and virtual teams. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 53–56.
- 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 International Conference on Software Engineering (ICSE)*, pages 801–811.
- 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 Symposium and the 13th European Conference on Foundations of Software Engineering (SIGSOFT)*, pages 168–178.
- Cavalcanti, G., Accioly, P., and Borba, P. (2015). Assessing semistructured merge in version control systems: A replicated experiment. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10. IEEE.
- Condina, V., Malcher, P., Farias, V., Santos, R., Fontão, A., Wiese, I., and Viana, D. (2020). An exploratory study on developers opinions about influence in open source software ecosystems. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*, pages 137–146.
- 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.
- Costa, C., Figueiredo, J. J., Pimentel, J. F., Sarma, A., and Murta, L. G. P. (2019). Recommending participants for collaborative merge sessions. *IEEE Transactions on Software Engineering*.
- Costa, C., Menezes, J., Trindade, B., and Santos, R. (2021). Factors that affect merge conflicts: A software developers' perspective. In *Brazilian Symposium on Software Engineering*, pages 233–242.
- de Farias Junior, I., Marczak, S., dos Santos, R. P., Rodrigues, C., and Moura, H. (2022). C2m: A maturity model for the evaluation of communication in distributed software development. *Empirical Software Engineering*.
- Dias, K., Borba, P., and Barreto, M. (2020). Understanding predictive factors for merge conflicts. *Information and Software Technology*, 121:106256.
- Estler, H. C., Nordio, M., Furia, C. A., and Meyer, B. (2013). Unifying configuration management with merge conflict detection and awareness systems. In *22nd Australian Software Engineering Conference (ASWEC)*, pages 201–210.
- 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*, 46:892–915.
- Greiler, M., Storey, M.-A., and Noda, A. (2022). An actionable framework for understanding and improving developer experience. *IEEE Transactions on Software Engineering*.
- Guest, G., Bunce, A., and Johnson, L. (2006). How many interviews are enough? *Field Methods*, 18:59–82.
- Guimarães, M. L. and Silva, A. R. (2012). Improving early detection of software merge conflicts. In *34th International Conference on Software Engineering (ICSE)*, pages 342–352.
- Kamei, F., Wiese, I., Pinto, G., Ribeiro, M., and Soares, S. (2020). On the use of grey literature: A survey with the brazilian software engineering research community. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*, pages 183–192.
- Kasi, B. K. and Sarma, A. (2013). Cassandra: Proactive conflict minimization through optimized task scheduling. In *35th International Conference on Software Engineering (ICSE)*, pages 732–741.
- 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.
- McKee, S., Nelson, N., Sarma, A., and Dig, D. (2017). Software practitioner perspectives on merge conflicts and resolutions. In *33rd IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 467–478.
- 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.
- Menezes, J. W., Trindade, B., Pimentel, J. F., Plastino, A., Murta, L., and Costa, C. (2021). Attributes that may raise the occurrence of merge conflicts. *Journal of Software Engineering*, 9:14.
- 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.
- Pan, R., Le, V., Nagappan, N., Gulwani, S., Lahiri, S., and Kaufman, M. (2021). Can program synthesis be used to learn merge conflict resolutions? an empirical analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 785–796. IEEE.
- Pfleeger, S. L. and Kitchenham, B. A. (2001). Principles of survey research: part 1: turning lemons into lemonade. *ACM SIGSOFT Software Engineering Notes*, 26:16–18.
- Premraj, R., Tang, A., Linssen, N., Geraats, H., and van Vliet, H. (2011). To branch or not to branch? In *Proceedings of the 2011 International Conference on Software and Systems Process*, pages 81–90.
- Sarma, A., Redmiles, D., and Van Der Hoek, A. (2008). Empirical evidence of the benefits of workspace awareness in software configuration management. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 113–123.
- 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.
- Shihab, E., Bird, C., and Zimmermann, T. (2012). The effect of branching strategies on software quality. In *12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 301–310.
- Singer, J., Sim, S. E., and Lethbridge, T. C. (2008). *Software Engineering Data Collection for Field Studies*, pages 9–34. Springer London, London.
- Smith, E., Loftin, R., Murphy-Hill, E., Bird, C., and Zimmermann, T. (2013). Improving developer participation rates in surveys. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 89–92.
- Spencer, D. (2009). *Card sorting: Designing usable categories*. Rosenfeld Media.
- Steglich, C., Marczak, S., De Souza, C. R., Guerra, L. P., Mosmann, L. H., Figueira Filho, F., and Perin, M. (2019). Social aspects and how they influence mseco developers. In *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 99–106.
- Vale, G., Hunsen, C., Figueiredo, E., and Apel, S. (2021). Challenges of resolving merge conflicts: A mining and survey study. *IEEE Transactions on Software Engineering*.
- 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. (2016). Card-sorting: From text to themes. In *Perspectives on data science for software engineering*, pages 137–141. Elsevier.
- Zou, W., Zhang, W., Xia, X., Holmes, R., and Chen, Z. (2019). Branch use in practice: A large-scale empirical study of 2,923 projects on github. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pages 306–317. IEEE.