Choosing the Right Git Workflow: A Comparative Analysis of Trunk-based vs. Branch-based Approaches

Pedro Lopes
Centro de Informática, Universidade Federal de Pernambuco
Recife, Pernambuco, Brazil
phls2@cin.ufpe.br

Paola Accioly
Centro de Informática
Universidade Federal de Pernambuco
Recife, Pernambuco, Brazil
prga@cin.ufpe.br

Paulo Borba
Centro de Informática
Universidade Federal de Pernambuco
Recife, Pernambuco, Brazil
phmb@cin.ufpe.br

Vitor Menezes
Centro de Informática
Universidade Federal de Pernambuco
Recife, Pernambuco, Brazil
vitorcardimmenezes@gmail.com

ABSTRACT

Git has become one of the most widely used version control systems today. Among its distinguishing features, its ability to easily and quickly create branches stands out, allowing teams to customize their workflows. In this context, various formats of collaborative development workflows using Git have emerged and gained popularity among software engineers. We can categorize such workflows into two main types: branch-based workflows and trunk-based workflows. Branch-based workflows typically define a set of remote branches with well-defined objectives, such as feature branches, a branch for feature integration, and a main branch. The goal is to migrate changes from the most isolated branch to the main one shared by all as the code matures. In this category, GitFlow stands out as the most popular example. In contrast, trunk-based workflows have a single remote branch where developers integrate their changes directly. In this range of options, choosing a workflow that maximizes team productivity while promoting software quality becomes a non-trivial task. Despite discussions on forums, social networks, and blogs, few scientific articles have explored this topic. In this work, we provide evidence on how Brazilian developers work with Git workflows and what factors favor or hinder the use of each model. To this end, we conducted semi-structured interviews and a survey with software developers. Our results indicate that trunk-based development favors fast-paced projects with experienced and smaller teams, while branch-based development suits less experienced and larger teams better, despite posing management challenges.

KEYWORDS

Git Workflow, Branch-based, Trunk-based, GitFlow

1 Introduction

Among existing version control systems (VCS), Git stands out as an increasingly popular tool for its open-source nature and for providing both quick and easy creation of development branches, allowing the implementation of new modules and features in isolation without affecting the primary code repository. This functionality promotes modular software development and facilitates task division. Thus, development teams can customize how they want to collaborate.

In this context, various workflows with Git became popular among software engineers/developers. In this work, we categorize such workflows into the following: *branch*-based and *trunk*-based. While branch-based workflows organize the software development cycle into well-defined stages using separate branches, trunk-based workflows integrate all code directly into the main branch.

GitFlow is the most popular example of a branch-based workflow. It consists of guidelines for creating five different branches according to the maturity level of the code and the type of task that is implemented [3]. The main idea is that developers work on new features and bug fixes in isolated branches. After implementation, they migrate changes from the most isolated branch until they reach the main branch, which the team shares, as the code is verified and validated. Although GitFlow is the most popular example, each company can define its workflow with other branch structures, such as mirroring system architecture, team structure, or both [13].

On the other hand, in trunk-based workflows, the team commits their changes to the only main branch, the trunk, multiple times a day, without extensive branch creation, ensuring that the codebase is always available on demand [6].

In this range of options, choosing a workflow that maximizes team productivity while promoting software quality becomes a non-trivial task. Despite the discussion in online forums, social networks, and blogs, few scientific studies explore this issue in depth: some focused on branching structures and the effects of excessive branching practices [2, 4, 9, 13], others extracted Git workflows characteristics from public repositories [3], and studies even compared both types of workflow [5, 8], but they focused on a specific development context: companies switching from one workflow to another.

In this work, we conducted both online surveys and semi-structured interviews to understand how developers work collaboratively using different Git workflows in their daily activities and assess their perception of which factors encourage or hinder the usage of one workflow over the other. In total, we conducted 22 interviews and had 54 responses to our survey.

Providing this kind of evidence is necessary because it offers an overall perspective of what developers are using, best practices, potential pitfalls, and the identification of specific contexts where one workflow might perform better than the other. This knowledge can aid companies, particularly those in their early stages, in adopting reported best practices or deciding which workflow to use based on their development context, contributing to future research, such as developing new Git workflow guidelines or automation tools.

Based on our sample, our results indicate that branch-based workflows are more widely known and used in practice than trunk-based ones among Brazilian developers. Also, while trunk-based workflows favor fast-paced projects with experienced developers and smaller teams, branch-based development suits less experienced and larger teams better, despite posing management challenges due to all the steps developers need to follow to deliver their code.

Moreover, due to ethical research concerns involving human subjects, we submitted this study to Plataforma Brasil, a unified national database of research records involving human subjects, which was approved (approval no. 6.740.037).

2 Study Setup

The goal of this work is to learn from software developers which workflows are used the most in practice, considering branch-based and trunk-based categories, how they perceive the advantages and disadvantages of each type, and whether they understand which factors (technical, organizational, or social) favor or hinder the adoption of each workflow. Based on their reality, this research will generate recommendations for individuals unsure of which workflow to adopt.

To achieve this goal, we conducted an online survey and semi-structured interviews to answer the following research questions.

- RQ1. How do developers work with Git Workflows?
- RQ2. What factors favor or hinder using a branch-based or a trunk-based workflow?

We adopted a mixed methods approach that combined both survey and interviews to better understand developers' perspectives. The survey provided an initial quantitative overview that helped identify general trends and patterns, while the interviews allowed deeper qualitative insights, capturing contextual details and nuances that the survey alone could not provide.

To address **RQ1**, both through the survey and the semi-structured interviews, we used the following question:

*Regarding the project you are currently working on (if you work on more than one currently, please choose one), describe the **step-by-step** process of a code change from the moment you implement it until it is pushed to the remote repository and subsequently deployed into production.*

Our objective with this question is to compute the frequency of usage for each category. Besides that, we would like to delve into the step-by-step process of each workflow to identify potential differences among the various possible formats within each category, particularly in branch-based workflows, which can be highly customizable.

Regarding **RQ2**, we did not want to bias the responses of the participants with our research hypotheses regarding factors that favor or hinder a specific workflow. Therefore, we asked the questions in the most generic way possible, inquiring about which factors favor or hinder branch-based and trunk-based workflows without mentioning any specific factors.

In addition, we have crafted some questions that would help better characterize our sample, such as how many years of experience the respondent has in software development activities, the size of their team, and their current role. These data are also helpful in establishing potential correlations, such as whether there is a correlation between the chosen workflow and the participant's experience, among others.

Lastly, we also ask whether respondents want to change their current workflow and why. This question would provide another opportunity to understand the participants' perceptions of the factors they consider relevant for decision-making.

After preparing the initial version of the interview script and the questionnaire for the survey, we piloted both studies with the first participant each to test whether the instruments were suitable.

In the case of the interviews, the participant suggested two new questions, which we added to the remaining interviews. The first question regarded whether the participant classified the company they worked for as a startup, and the second asked if the participant encountered recurring issues with merge conflicts.

Despite these questions mentioning specific factors, we believe that they do not bias responses to the generic questions about factors cited earlier, as we ask them before we mention them. In addition, such questions help provide better context about the interviewee's context.

It is important to mention that the survey and interviews occurred independently from this initial jointly conceived version onward. For this reason, some questions underwent slight alterations after their pilot round, resulting in minor differences. Therefore, in Section 3, there are some discussions in which we will describe graphs with different metrics for both studies.

For example, in the survey, we asked the respondents how many years of experience they had with software development activities. In contrast, in the interviews, we asked about the interviewee's current position, ranging from junior, mid-level, to senior.

2.1 Interviews and Survey Application

For the interviews, we initially selected 37 participants from the authors' contact network, prioritizing engineers/developers with a higher seniority level and who still work directly with code rather than managing teams. The rationale behind this is that code management is performed and affects more developers than managers, as they are the ones dealing with it daily. After this selection process, we interviewed 22 individuals. All instruments used for both survey application and interviewing, as well as interview transcriptions in Portuguese, are available online in our artifact package [7].

Regarding the survey, we shared our online questionnaire on social networks like LinkedIn, Instagram, WhatsApp, and Twitter. In addition to that, we promoted our questionnaire through email lists, with eventual reminders to increase the chances of obtaining more responses. In total, we obtained 54 responses, but we excluded four answers, as participants claimed that they did not use Git or a different VCS since they occupied managerial roles.

2.2 Data Analysis

To answer **RQ1**, we needed to establish if the participant described a branch or trunk-based workflow. During the interview, this process was straightforward since the interviewer could ask for more details. However, some of the answers in our survey were difficult to analyze, as they were more succinct. To make this analysis more systematic and increase our confidence in the collected data, we obeyed the following rules to classify a workflow as branch or trunk-based:

(1) If the participant explicitly cites the type of workflow in his description, we would classify it in its respective category;
(2) We categorize the answer into the branch-based group if a workflow has multiple remote branches where developers collaborate, like develop and feature branches, and as trunk-based otherwise;
(3) If the description were confusing, we would discuss it in pairs to check if we could reach a consensus;
(4) If we could not reach a consensus, we would classify the workflow in the "unidentified" category.

To answer **RQ2** using the interviews' transcriptions as input, we followed the Saldana [12] and Vasilescu [14] approach to encode and analyze the data, opting for an open coding technique. We used the first interview as a pilot to create initial codes, and the subsequent interviews as supplementary to complete the coding process. We employed a flexible approach and adjusted the codes as needed, as we describe in the following section.

To analyze the survey responses, we adopted the card sorting approach inspired by Zimmermann [17] to identify common themes in the open questions. First, we extracted main topics from each response regarding factors that favor or hinder workflows and created cards for them. When closely related topics appeared, like "ease of use" and "lower complexity", which convey the same idea, we grouped them, repeating the process until we addressed every answer. Lastly, we named the groups with the factor that best represented their cards.

3 Results and Discussion

In this Section, we describe the demographic characteristics of our sample and summarize the responses provided for **RQ1** and **RQ2**.

3.1 Demographics

Overall, we interviewed 22 people and received 54 responses to our survey, all from Brazilian developers. However, we cannot specify whether the participants worked for companies only in Brazil or abroad. Moreover, we excluded four responses from our survey, as participants claimed that they did not use Git or a different VCS since they occupied managerial roles.

3.1.1 Years of Experience.

Among the questions used to characterize our sample, in the survey, we asked about the number of years of experience of the participants. In contrast, we asked about the interviewee's current position in the interviews. The histogram in Figure 1 presents data from years of experience, and a discussion on the latter can be seen in Subsection 3.1.3.

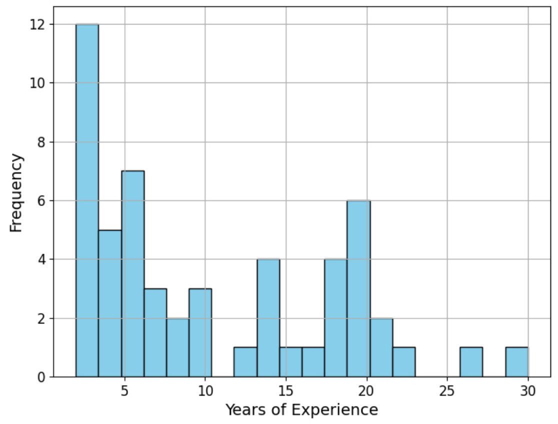

Figure 1: Survey Developers' Years of Experience

In the survey, the Mean (*M*) years of experience is 10.56, the Median (*Mdn*) is 7.5 and the standard deviation (*SD*) is 7.61.

In Figure 1, we can divide our sample into two groups using *Mdn* as the pivot, as this value indicates the exact point at which 50% of the respondents fall below and 50% above. The first group comprises developers with fewer years of experience (ranging between 2 and 7.5), while the other includes more experienced developers (ranging from 7.5 to almost 30 years).

The first group shows more concentrated values, whereas the other is sparser, with our sample having both young and experienced developers. Additionally, a *SD* of 7.61 suggests a moderate variability in experience levels, which contributes to pushing this value upward, as we can see from the sparse data beyond 10.56.

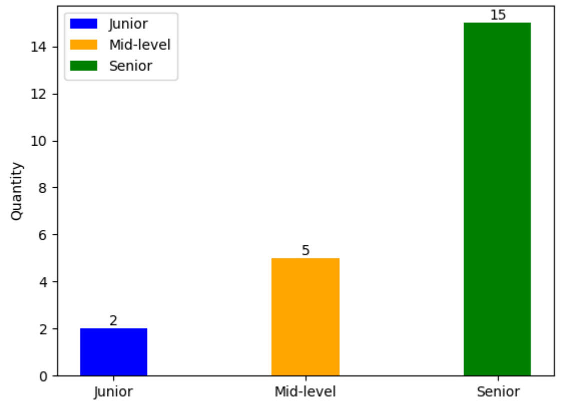

Figure 2: Interviewees' Seniority Level

Figure 2, on the other hand, shows the distribution of the current position of the respondents, ranging from the junior to senior level. As explained in Section 2, we prioritized participants with a higher level of seniority. This is why the bar chart indicates that 75% of our participants consider themselves senior engineers/developers. However, we also interviewed junior and mid-level developers.

3.1.2 *Team Size.*

We asked both interviewees and survey respondents to report the size of the development team with which they were currently working. Figures 3 and 4 show the histograms of the team sizes in the survey and interviews, respectively.

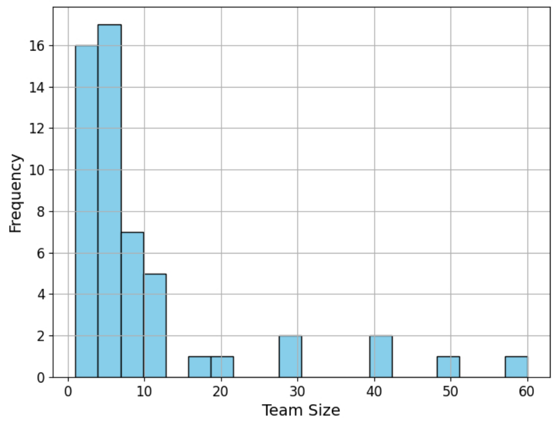

Figure 3: Survey Results: Developers' Team Size Distribution

For the survey, the metrics calculated for the team size were as follows: our *Mdn* is a team size of 5 developers, our *M* is 12, and our *SD* is 21.35. The standard deviation is high because we had an outlier with one participant indicating that their team had 137 people, which is a much higher number compared to the others. Upon reviewing this individual's responses, we found that their current position is Principal Member of Technical Staff. Therefore, they must have included all the employees below them in the company hierarchy rather than a development team.

In Figure 3, a large majority of the answers had a team size number distributed around our *M*, with a maximum of 20 developers, with a few outliers at 30, 40, 49, 60, and 137, inflating our *M* and *SD* by considerable margin. Note that the biggest outlier is not represented in Figure 3, as plotting would worsen the readability of the graph. Additionally, the most recurring value in this set was 5, appearing a total of 17 times.

Furthermore, Figure 4 shows the histogram of the distribution of team sizes among our interviewees. For this set, *M* is 7.09, *Mdn* is 4.5, and *SD* is 5.72. The most recurring values for this set are 2 and 4, which appear 4 times each.

Therefore, considering both parameters, we can conclude that participants worked in teams of different sizes; however, most of the participants in our sample (80%) worked in teams of up to 10 people.

3.1.3 *Job Title.*

Along with the data provided in 3.1.1, understanding developers' job titles can offer valuable findings into the diverse skill sets

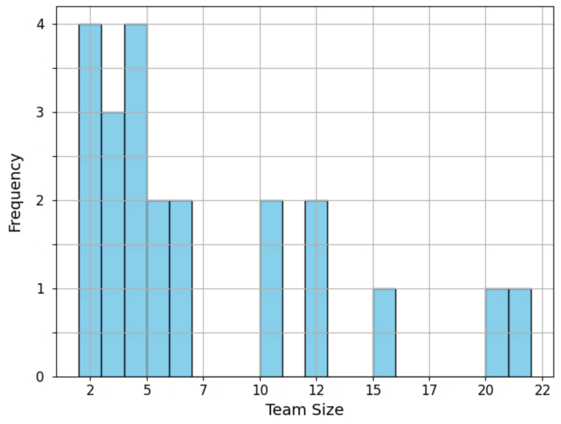

Figure 4: Interviews Results: Developers' Team Size Distribution

present, presenting the context for the roles, responsibilities, and areas of specialization, allowing data interpretation. In the interviews, all participants were developers/software engineers. However, to gain further insight into the distribution of job titles among survey respondents, Figure 5 shows a word cloud visualization that highlights the most prevalent titles among survey participants.

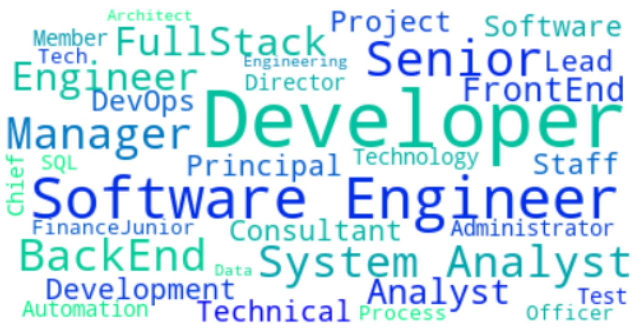

Figure 5: Survey Respondents' Job Titles Word Cloud

The most common job titles were Developer and Software Engineer, which appear 18 and 10 times, respectively, with Backend, Full-Stack, System Analyst, Manager, and Senior being more frequently mentioned after the first two. The Front-End job title appeared less prominent than the others. In addition, job titles involving topics such as SQL, Testing, DevOps, Automation, and Data were cited once.

Developer and Senior were big words in our word cloud because there are a lot of job titles that use this term in common with others, like Frontend Developer, Backend Developer, Senior Developer, and Senior Analyst. But in Figure 5, we can see that our sample is diverse, mostly composed of developers from various scenarios.

3.2 RQ1. How Do Developers Work with Git Workflows?

To address this research question in the survey, we first compare which workflows developers currently use in their projects by analyzing their frequency based on their detailed description of a code change migrating from their local repository until it reaches the main branch. To do so, we classified the workflow into branch or trunk-based following the method described in Section 2.2.

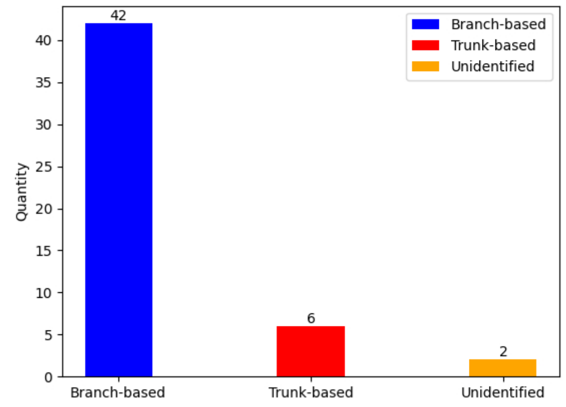

Figure 6: Survey Results: Branch-based vs. Trunk-based

In Figure 6, branch-based models, including GitFlow and GitHub Flow, cited three and two times, respectively, are significantly more popular than trunk-based models in our sample. Out of the total 42 responses (84%) for branch-based, only 6 responses (12%) were for trunk-based. Furthermore, participants cited the Stacked Diffs workflow twice, a model that involves applying a series of changes on top of each other.

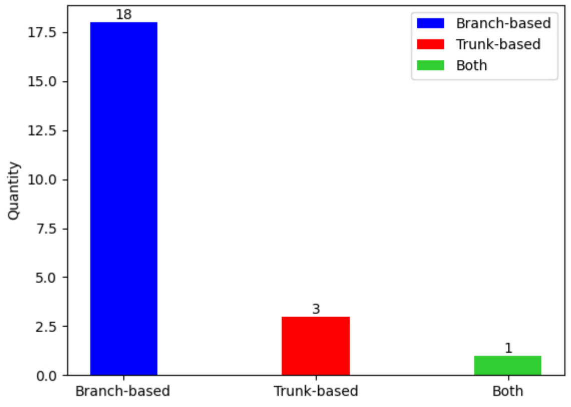

Figure 7: Interviews Results: Branch-based vs. Trunk-based

In the interviews, Figure 7 shows the same trend as in the survey data. That is, branch-based is much more popular than trunk-based. Additionally, one participant claimed to use both workflows simultaneously, choosing between them based on task complexity: for simpler tasks such as maintenance or updates, they preferred trunk-based development, whereas for more complex or ongoing system development, they followed the GitFlow model.

The survey data was then used to conduct two more investigations, one of which was to see the impact of different demographics, such as experience and team size, on the decision-making process of employing a workflow, plotting box plots comparing how the graphs vary if we selected only those who answered a specific model.

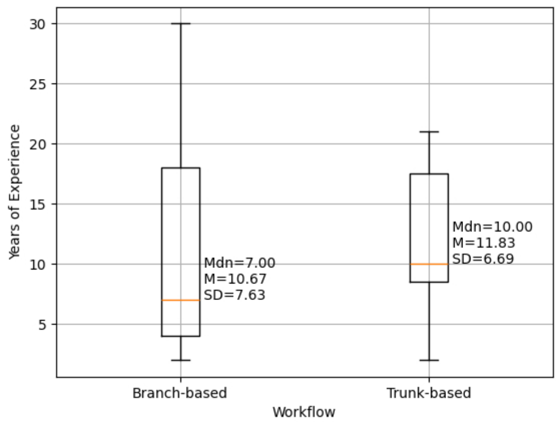

Figure 8: Survey Results: Developers' Years of Experience across Workflows

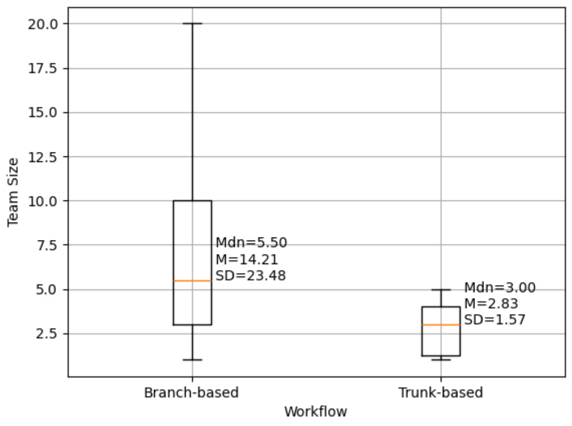

Figure 9: Survey Results: Team Size across Workflows

Figure 8 shows the box plot that compares the experience of developers for each workflow. In the trunk-based boxplot, all answers range between 2 and 22, while the branch-based range from 2 to 30; this more considerable variation explains why the branch-based *SD* is greater than the trunk-based *SD*. On the other hand, the trunk-based model has a higher *Mdn* and *M* than the branch-based model (10 and 11.83 compared to 7 and 10.67), indicating that the developers who answered that use trunk-based development are slightly more experienced than those who use the other model.

To address the second investigation, parallel to how we approach years of experience per workflow box plots in Figure 8, we analyzed how the team size varies if we selected only those who responded who employ a specific model.

Figure 9 shows that the boxplot for branch-based models has metrics close to the ones presented in Subsection 3.1.1, this is due to branch-based users being a large portion of our answers, as stated in Figure 6.

Starting our analysis on trunk-based models, we can see in Figure 9 that it ranges from a single person to at most 5. In contrast, the branch-based box plot ranges from one developer to a maximum of 20. These data show a trend in our sample for trunk-based being preferred by smaller teams.

Note that in branch-based, the *SD* is significantly higher than *Mdn* and *M*, due to the outliers shown in Figure 3, whereas in trunk-based, the *SD* has a low value because, in our sample, all teams with more than 20 developers reported employing a branch-based model.

To ensure that the high variance observed in Figure 9 was not driven solely by a few cases, we performed an additional analysis that removed statistical outliers based on the Interquartile Range (IQR) method. The resulting distribution remained consistent with the original plot, reinforcing our previous findings.

In particular, the trunk-based model continued to exhibit low dispersion and a clustered team size distribution, while the branch-based workflow retained a broader range of team sizes. In our sample, even when controlling for outliers, trunk-based workflows are typically adopted by smaller, more homogeneous teams, whereas branch-based models are employed by teams with more varied sizes.

Additionally, we conducted a preliminary analysis of statistical tests to evaluate these findings. Due to the small trunk-based group (n = 6) and imbalanced sample sizes, distributional assumptions could not be tested. To address this, we used the Mann-Whitney U test, a non-parametric method.

Results showed a significant difference in team size, but not in years of programming experience. A power analysis indicated that the small trunk-based group considerably limits the strength of these statistical test findings (power = 32.9%). Given the limited number of observations in the trunk-based model, we avoid drawing conclusions based on these results, as statistical tests for group differences are unreliable with such small samples.

Furthermore, in the survey, 76.5% of participants indicated they would not change their current project model if they could, while 3.9% would change only if specific factors, such as an increase in the development team, occurred. Additionally, 5.9% expressed a desire for change, and 13.7% did not respond.

We did not conduct this comparative analysis on the data from our interviews because only three interviewees used trunk-based. However, we also asked them if they would like to change their current workflow and why. Seven participants answered yes, but they would only change some steps in their process, like adding CI/CD routines or customizing branches, instead of switching between branch-based and trunk-based workflows.

Taking into account our results, we answer **RQ1** as follows:

Regarding our sample, branch-based models seem far more popular and disseminated than trunk-based models. However, despite GitFlow being the most popular model, participants in both the survey and interviews rarely mention it explicitly. Thus, models customized by each team are more prevalent in practice.

Because all of our respondents are Brazilians, we cannot generalize our results so much. However, it is important to note that we do not know if our participants worked only for Brazilian companies or abroad.

Moreover, our data suggest that teams composed of more experienced developers and smaller development teams may favor trunk-based development, but further investigation is needed to assert this relationship since our sample is imbalanced.

Finally, most participants would not like to change from trunk-based to branch-based and vice versa, although some would like to change parts of their workflow.

3.3 RQ2. What Factors Favor or Hinder Using a Branch-based or a Trunk-based Workflow?

In this Section, we describe the factors that our participants pointed out in both the survey and the interviews as relevant to the choice of a workflow type, organized in descending order of frequency, with each group title including the number of times the factor was cited by respondents. Specifically, we asked which factors favor or disfavor using a specific workflow.

3.3.1 Survey Results.

To address RQ2 in the survey, we used online tools like Figma and the card sorting method [17] to analyze its data. We extracted the central theme from our responses and created a sticker for each on the cardboard.

In cases where related themes appeared, for example, terms like "ease" and "lower complexity" conveying the same idea, we created a group with both stickers, repeating the process until every topic was addressed and in a group. Finally, we named each group with the category that best represented the stickers.

Please note that not every sticker is part of a group. This may be because some stickers represent multiple themes at once, or there simply were not enough stickers to form a group, where at least two stickers representing the same factor were required for grouping. However, the absence of a group does not imply that these topics are any less important.

We will cover the factors that developers answered that encourage or deter the employment of each model, with direct citations from the survey participants (SP) that explain their reasons. The card sorting sessions are available in our Git repository [7].

Regarding branch-based development, the most common themes that promote this model usage were named as follows: Code Organization and Self-Containment, but other groups like Task Division, Maintenance, Conflict and Error Detection, and Flexibility, with other minor groups being present too.

(1) **Code Organization [6 mentions]**: Respondents highlighted that the model has an organized commit flow, such as greater control over commits, promotes organized code practices, and reflects an organized repository structure.
"One of the main factors is the organization of the code." (SP09)
(2) **Self-Containment [4 mentions]**: In this group, the answers emphasized that this model ensures parallel development, has flexible feature development, and praised the self-containment of each feature on its branch.
"I like how I can share my tasks in a modular way by features." (SP19)
(3) **Task Division [3 mentions]**: This theme stressed that this model is appropriate when being worked on by different teams.
"The separation between Development and Quality Assurance teams is important." (SP10)
(4) **Maintenance [3 mentions]**: The respondents in this group emphasized that this model has facilitated rollbacks, release code maintenance, faster feature branch maintenance, and faster deployment to production
"It makes maintenance of code that has already been released viable without altering the flow of the current and future releases." (SP20)
(5) **Conflict and Error Detection [2 mentions]**: The participants stated that the branch-based model helps avoid code conflicts and has quick error detection.
"This model is good at avoiding merge conflicts." (SP12)
"This model helps merge control and avoid conflicts…" (SP54)
(6) **Flexibility [2 mentions]**: This group stressed the versatility of this model and the freedom that developers have in their branches.
"Developers can collaborate in whatever way they prefer in their branches." (SP18)

In addition to these topics, some stickers ended up without a group because they covered many themes at the same time, like one that stated that for applications, such as mobile apps, that undergo store releases, Quality Assurance (QA) reviews, and specific fixes for release, it is more advantageous to have a branch-based model to separate what is still in development, what is in each release, and what and from which release will be fixed.

Moreover, the following groups discourage branch-based workflow usage: Complexity, Small Teams, Branching Issues and Inexperienced Teams.

(1) **Complexity [4 mentions]**: In this group, the responses emphasized that this model has a higher release complexity, a higher code alteration complexity, more documentation, and steps to release features, leading to a longer development cycle.
"Using this model is more labor intensive." (SP20)
(2) **Small Teams [4 mentions]**: Statements like "With fewer team members, the development will take longer due to having more steps and stages to follow".
(3) **Branching Issues [4 mentions]**: Respondents highlighted that branch-based models have extensive product segmentation, that having too many branches with long lifespans can make them outdated and prone to conflicts, and that often a considerable amount of time is spent analyzing each branch.
"If branches have a very long lifespan, they become outdated and cause conflicts." (SP18)
(4) **Inexperienced teams [2 mentions]**: In this group, two statements were cited that express that branch-based models need focused and experienced teams.
"It requires the focus and maturity of the team to maintain organization." (SP06)

One sticker left without a group states that a factor that hinders this model from being employed is that maintenance of epic features can take a long time, often requiring rebases or merges with updates.

In trunk-based development, the main groups that emerged from the survey answers that promote this model were named as follows: Easiness and Practicality, Agility, Quick Versioning, Team and Project Size, and Back end and Server, besides other minor ones explained below.

(1) **Easiness and Practicality [5 mentions]**: This group underlined the minor complexity, simplicity, and ease of use of the model.
"It is easier for the team to align on the workflows to follow when there are fewer branches and steps in a remote work context." (SP08)
(2) **Agility [5 mentions]**: The respondents in this group emphasized the model's speed, agility, and faster development process, stating that the trunk-based workflow works best with applications that need faster feedback cycles.
"Good for applications that are small or require quick feedback." (SP10)
(3) **Quick Versioning [5 mentions]**: Themes like having a less strict workflow were covered in this group, mentioning that, generally, trunk-based models have lower complexity, with more minor changes, such as text or layout not having to pass through many tests.
"It is easier for the team to align on the workflows to follow when there are fewer branches and steps in a remote work context." (SP08)
(4) **Team and Project Size [5 mentions]**: Most of the answers in this group were related to this model being good in small projects or when it has a small team.
"This model is good when only a few people are maintaining the project." (SP01)
(5) **Management [2 mentions]**: In this case, the answers highlighted that this model has easy maintenance and a good application monitoring structure.
"There is a good application monitoring structure." (SP10)
(6) **Failures [2 mentions]**: This group responded that trunk-based models have fewer risk factors and rarer merge failures, although the respondents did not specify such risk factors.
"There are fewer merge failures." (SP06)
(7) **Backend and Server [2 mentions]**: The answers in this group stated that backend code works better with this model due to the less strict versioning flow and that they typically have few or only one main branch when working on server code.

"Backend code functions better with this approach due to the less strict versioning flow." (SP08)

There were only two stickers without a group, one stating that this model promotes CI/CD practices, and the other stating that it favors focused and experienced teams.

However, the following groups were formed that discourage trunk-based workflows: Team and Project Size, Maintenance, Management, and some minor groups like CI/CD and Security.

(1) **Team and Project Size [7 mentions]**: This group stated that trunk-based development could have inferior performance when the project has a lot of contributors, features, tasks, and products, and with that, the team must focus on achieving success.
"If there are too many people committing changes simultaneously, it becomes difficult to keep the code updated." (SP08)
(2) **Maintenance [5 mentions]**: Respondents in this group cited that the model has low flexibility, interference between commits and pushes frequently, and there is no easy way to do feature toggles, meaning that in case they have to remove some feature, this can have an impact on others that are not related.
"It is easy to encounter interference between commits/pushes." (SP18)
"Absence of application monitoring and feature toggles." (SP10)
(3) **Management [2 mentions]**: The answers highlighted that this model does not deal with application monitoring and quality assurance well.
"Quality control becomes challenging." (SP18)
(4) **CI/CD [2 mentions]**: The themes of this group stated that respondents do not recommend this workflow with slow CI/CD pipelines and that teams with no experience in this practice may encounter difficulties.
"This model execution is difficult in teams with little experience in CD." (SP10)
(5) **Security [2 mentions]**: In this one, developers stated that they do not like this model's low security of production code.
"This model lacks testing security." (SP19)

In addition, two stickers were not assigned because the person who responded indicated a lack of experience with trunk-based development. However, he thinks this model could often result in frequent conflicts and challenges in executing rollbacks within a production environment.

3.3.2 *Interviews Results.*

In this Section, we briefly describe ten factors that interview participants (IP) state as relevant when choosing a branch-based or a trunk-based workflow, organized from most to least cited. As mentioned previously, we used open coding to analyze the transcription of our interviews.

(1) **Separate Environments [17 mentions]**: This factor considers the separation of environments, such as development, quality assurance, pre-production, and others, before the production environment, each potentially having a branch or being based on a specific commit. For example, one of the interviewees mentioned: "...reverting to a more traditional model of separated environments." (IP1), in this case, they worked in a trunk-based development structure and transitioned to branch-based, specifically GitFlow.
(2) **Code Management Complexity [12 mentions]**: This factor considers the complexity of managing version control and how costly and complicated it can be when using branch-based models. In general, the more branches there are, the more complex it becomes to manage and ensure they are updated as they should be. For example, one of the interviewees mentioned: "...the complexity is lower (when using trunk-based workflow)..." (IP2).
(3) **Testing and Rapid Deployment to Production [11 mentions]**: This factor considers how quickly the development team can code and effectively deploy code changes unrelated to incidents or bug fixes to production. We have an example from one of the interviewees who said: "...the advantage is because we have fewer processes when deploying to production because it is just one pipeline..." (IP10); in this case, he was working in a trunk-based Development structure, transitioned to GitFlow, and after the migration, noticed the trunk-based model's rapid deployment to production as a factor. Therefore, branch-based workflows suffer in this regard due to the branch management processes, often resulting in costly and slow deployments.
(4) **Rigid Processes [10 mentions]**: This factor takes into account management control. Whether there is any degree of freedom to decide which workflow to use, or if a process is mandatory and standard. For example, we have the quote of one of the interviewees who said: "... For us, where we work, the worst thing is dealing with bureaucracy ..." (IP18). Therefore, in environments where the process is more important than the delivery itself, branch-based workflows tend to be more popular at the expense of trunk-based ones.
(5) **Teams composed mostly of less experienced developers [8 mentions]**: This factor considers teams in which the interviewee explicitly mentioned the aspect of lower seniority among team members. For example, one of the interviewees said: "...because in all projects, we have multidisciplinary teams with various levels of knowledge, right? So basically, in current projects, we have many developers who are not senior..." (IP15). Participants claimed that less experienced developers might be hindered from using a trunk-based workflow, whereas branch-based workflows provide them with greater security.
(6) **Focus on Quality [7 mentions]**: This factor considers the explicit concern for code quality, even though it should be implicit in any implementation, workflow, and so on, as quality is one of the primary requirements for any development. For example, one of the interviewees said: "... It is a cool thing (the branch-based workflow). Even for code improvement reasons..." (IP9). Therefore, participants frequently related branch-based workflows to code quality.
(7) **Autonomy [7 mentions]**: This factor considers the freedom to work independently from what others in the team are doing, being one of the basic precepts of Git and even expected to appear favoring branch-based workflows. For example, one of the interviewees said: "And then you get a certain independence to work..." (IP3).

(8) **Fast Incident Response Time [4 mentions]**: This factor considers how quickly it is to deploy a fix to a production incident. For example, one of the interviewees said: "It (the trunk-based workflow) was much faster; in less than half a day, the fix was already in production." (IP1). Participants reported this factor as disfavoring the use of branch-based workflows.
(9) **Focus on Delivery Speed [4 mentions]**: This factor considers fast deliveries, where smaller portions of code are delivered, favoring trunk-based workflows. For example, one of the interviewees said, "Our project is quite fast, so there are features that are small, and they go up (into production) quite frequently..." (IP13).
(10) **Restricted Sector [4 mentions]**: This factor considers the type of sector in which the interviewee works, such as the financial sector, which has its particularities, like a greater focus on security, favoring branch-based workflows. For example, one interviewee mentioned: "...a company in the financial sector..." (IP5).

3.3.3 RQ2 Results Discussion.

The survey results align with the interview findings, reinforcing the evidence gathered in this study. Furthermore, the trend shown in **RQ1** that smaller teams and developers with higher levels of experience are characteristics that favor the use of trunk-based workflows is complemented and reinforced by the responses in **RQ2**.

While branch-based development can be flexible and offer self-containment in the form of feature branches, leave the code organized, and promote task division and code quality, there are still challenges, whether due to its complexity or because it overly segments the final product into numerous branches, making the integration and deployment process to be often costly, and delaying releases and feedback cycles.

On the other hand, trunk-based development promotes agile development and CI/CD practices with simple and quick versioning. However, it requires a focused development environment, which can be challenging for larger teams or teams that need to work with the same repository, and those without experienced developers due to the maintenance difficulty.

It is interesting to note developers' perceptions of conflicts occurring in both workflows. Some participants indicated that branch-based workflows are prone to conflicts because long-lived branches may diverge more from the main branch, causing more severe conflicts. In contrast, some participants pointed out that the trunk-based workflow is prone to conflicts, especially when team developers have a lower level of seniority.

Nevertheless, both workflows have their merits and are entirely consistent with the autonomy factor, as both provide freedom for the developer to work independently from others. The choice between them depends on the project needs, team culture, and practical development practices to manage associated challenges.

To choose which type of workflow to use, the developer can use all the themes and factors listed in this study as a checklist and choose which has the most characteristics in common with their specific context.

For example, a startup context, where the product is simple, the development team is lean, and there is a need for quick deliveries and customer feedback, would benefit from a trunk-based workflow.

In contrast, the context of a large company, with multiple teams working on the same repository, where there is high developer turnover with different levels of experience, would benefit from the self-containment of a branch-based workflow in which each branch represents a separate environment.

4 Threats to Validity

In this Section, we will discuss the possible factors that might represent threats to this work's validity, which we organize following the classification by Wohlin et al. [16].

Construct Validity. This study may be threatened by subjectivity in classifying workflows from open-ended responses, as well as by differences between survey and interview instruments. Another potential threat is our own bias towards one workflow or another, stemming from previous experiences. To mitigate this issue, we carefully designed our survey and interview questions to remain neutral and avoid references to specific factors that could influence participants' responses. Furthermore, our analysis relied exclusively on participants' statements, avoiding external interpretations.

Internal Validity. Potential confounding factors, such as differences in development methodologies or company policies, were not controlled. Additionally, there is a risk of receiving multiple responses from individuals within the same company or team, which could skew results. To mitigate this, the survey was distributed via general mailing lists and social media, rather than targeting specific companies or environments where employees might be encouraged to participate.

External Validity. Our sample is composed entirely of Brazilian developers, even though some of them work for foreign companies, which may limit the generalizability of our findings to other contexts. Furthermore, even though we used general mailing lists, participation was voluntary, and our invitation was unsolicited, which might have disturbed their daily activities. Unfortunately, there are no dedicated portals or guidelines for gathering volunteers for software engineering research. To mitigate this, we designed a short and cordial invitation email that also acted as an informed consent form, including ethical and privacy considerations, and made participation entirely opt-in [1, 15].

Conclusion Validity. Our sample imbalance, especially the low number of trunk-based workflow users, limits the statistical strength of comparisons.

5 Related Works

The challenge of maximizing team productivity while promoting software quality has led to significant research on collaborative software development workflows.

Studies delved into developers' practices and their effects on productivity and code quality. Shihab et al. [13] analyzed data from two releases of major Microsoft projects, finding that team organization directly influences quality and that branching affects post-release failures. They suggest that aligning branch structure with team organization slightly reduces adverse effects on software quality compared to architecture-based structures. While their research focuses on a single corporate environment, our work amplifies this approach by including various companies, methodologies, and workflows.

Bird and Zimmermann [2] conducted a survey on branch usage in a large Microsoft project, identifying common issues. They performed a what-if analysis to evaluate alternative branch structures based on isolation and liveness. By removing low-benefit, high-cost branches, they found potential savings of 8.9 days in delays with only 0.04 additional conflicts on average. A significant issue highlighted was the lengthy delays in changes moving between teams, often due to excessive branches. In our work, we identified extensive product segmentation as a factor hindering the usage of branch-based models.

Costa et al. [4] surveyed 109 software developers to understand their use of branches, integration challenges, and strategies to minimize issues. They discovered that developers mainly use branches for new features or bug fixes. According to the participants, developers can freely create new branches, but conflicts often arise due to prolonged isolation of branches and poor communication. In our work, some interviewees mentioned long-lived branches as a factor that discourages using a branch-based workflow.

Phillips et al. [10] recruited 140 version control users to participate in an online survey to collect information on branching and merging practices in software development projects. The findings claim that in their sample, the types of branches created influence developers' branching satisfaction, with feature, release, and experimental branches having the most impact. Furthermore, staged (branch-based) topologies are increasingly more common in their sample when filtering the data by project size. Their data also shows that branch-per-release or features were the most popular among respondents. In our work, we also observed this pattern.

Studies compared workflows in a specific development context. De Boer [5] conducted a study in a large software company transitioning from trunk-based to branch-based development with merge requests. Through interviews and surveys with developers before and after the migration, the study identified that trunk-based development favored delivery speed. In contrast, the branch-based model improved perceived code quality, despite some integration challenges. Our study expands on this perspective by comparing multiple companies with different workflows and identifying additional influencing factors, such as Separate Environments, which the literature has not previously addressed.

Neely & Stolt [8] present an experience report on a company transition from time-boxed Scrum sprints to continuous delivery with Kanban, highlighting technical and organizational challenges. Their experience emphasizes factors that align with our findings about trunk-based workflows, such as agility, faster feedback cycles, and reduced complexity in versioning. Similarly, our works indicate the need for test automation and monitoring structures to mitigate risks. Challenges related to maintenance, coordination among teams, and the importance of organizational commitment also appear in our studies, but similarly to [5], their work focused on a single organizational context.

Cortés Ríos et al. [3] extracted characteristics such as branching strategies from workflows in public Git repositories through literature analysis and README file mining. Their framework categorized these workflows, revealing that claimed similarities often masked significant differences. However, they did not evaluate the practical advantages and disadvantages of the various workflows. Future research could explore whether our respondents' workflow descriptions align with this framework.

Rayana et al. [11] introduced GitWaterFlow, a flexible branching model that supports long-term maintenance of older product versions while enabling quick deployment of critical bug fixes. Unlike GitFlow, it uses atomic multi-branch merging, allowing developers to integrate multiple fixes or updates. Their team's experience using the GitWaterFlow model for nearly a year suggests that the model has led to faster deliveries and a general reduction in engineering work due to the new deployment method, successfully meeting their goals in the given setting, with room for fine-tuning and improvement of processes and tools.

6 Conclusions

This work aimed to understand how developers work with Git, considering branch-based and trunk-based workflows, and which factors favor or hinder using a specific workflow. To this end, we conducted survey research with 54 developers and semi-structured interviews with 22 individuals.

Our results indicate that developers still use branch-based workflows more frequently than trunk-based ones. Moreover, there is a preference for trunk-based models among more experienced programmers and small development teams. However, further investigation is required to confirm this relationship due to the significant imbalance in our sample (only 12% employing trunk-based models).

Also, the factors developers cited that promote or hinder the usage of branch-based workflows is that while it can be flexible, offer self-containment in the form of feature branches, leave the code organized, and promote task division, there are still challenges, whether due to its complexity or because it overly segments the final product into numerous branches.

On the other hand, trunk-based development promotes agile development. Allowing CI/CD practices and having simple and quick versioning, but demanding a focused development environment in exchange, can be challenging for teams without experienced developers because of the maintenance difficulty.

Finally, both workflows have their advantages, and choosing the most suitable one depends on various circumstances, such as the project size, necessities, the seniority of the developer team, and the development guidelines adopted by the company.

ARTIFACT AVAILABILITY

All data collected through our survey and interviews are available in this repository [7].

ACKNOWLEDGMENTS

We thank the participants of our survey and interviews. For partially supporting this work, we would like to thank INES (National Software Engineering Institute) and the Brazilian research funding agencies CNPq, FACEPE, and CAPES.

REFERENCES

[1] Sebastian Baltes and Stephan Diehl. 2016. Worse Than Spam: Issues In Sampling Software Developers. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Ciudad Real, Spain) *(ESEM '16)*. Association for Computing Machinery, New York, NY, USA, Article 52, 6 pages. https://doi.org/10.1145/2961111.2962628

[2] Christian Bird and Thomas Zimmermann. 2012. Assessing the Value of Branches with What-If Analysis. In *Assessing the Value of Branches with What-If Analysis* (Cary, North Carolina) *(FSE '12)*. Association for Computing Machinery, New York, NY, USA, Article 45, 11 pages. https://doi.org/10.1145/2393596.2393648

[3] Julio César CORTÉS RÍOS, Suzanne M. Embury, and Sukru Eraslan. 2022. A unifying framework for the systematic analysis of Git workflows. *Information and Software Technology* 145 (2022), 106811. https://doi.org/10.1016/j.infsof.2021.106811

[4] Catarina Costa, José Menezes, Bruno Trindade, and Rodrigo Santos. 2021. Factors That Affect Merge Conflicts: A Software Developers' Perspective. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering* (Joinville, Brazil) *(SBES '21)*. Association for Computing Machinery, New York, NY, USA, 233–242. https://doi.org/10.1145/3474624.3474641

[5] Toon DE BOER. 2021. *From Trunk-Based to Merge Requests: A Field Study at Adyen*. Master's thesis. Delft University of Technology, Delft, Netherlands. https://resolver.tudelft.nl/uuid:8852d755-5ece-4c3e-98ca-70a0782a31cc

[6] Niels Jørgensen. 2001. Putting it all in the trunk: incremental software development in the FreeBSD open source project. *Information Systems Journal* 11, 4 (2001), 321–336. https://doi.org/10.1046/j.1365-2575.2001.00113.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1046/j.1365-2575.2001.00113.x

[7] Pedro Lopes, Paola Accioly, Paulo Borba, and Vitor Menezes. 2025. Choosing the Right Git Workflow: A Comparative Analysis of Trunk-based vs. Branch-based Approaches. Zenodo. https://doi.org/10.5281/zenodo.17054411

[8] Steve Neely and Steve Stolt. 2013. Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). In *Proceedings of the 2013 Agile Conference (AGILE '13)*. IEEE Computer Society, USA, 121–128. https://doi.org/10.1109/AGILE.2013.17

[9] Stephen Phillips. 2020. Working through the pandemic: Accelerating the transition to remote working. *Business Information Review* 37, 3 (2020), 129–134. https://doi.org/10.1177/0266382120953087 arXiv:https://doi.org/10.1177/0266382120953087

[10] Shaun Phillips, Jonathan Sillito, and Rob Walker. 2011. Branching and merging: an investigation into current version control practices. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering* (Waikiki, Honolulu, HI, USA) *(CHASE '11)*. Association for Computing Machinery, New York, NY, USA, 9–15. https://doi.org/10.1145/1984642.1984645

[11] Rayene Ben Rayana, Sylvain Killian, Nicolas Trangez, and Arnaud Calmettes. 2016. GitWaterFlow: a successful branching model and tooling, for achieving continuous delivery with multiple version branches. In *Proceedings of the 4th International Workshop on Release Engineering* (Seattle, WA, USA) *(RELENG 2016)*. Association for Computing Machinery, New York, NY, USA, 17–20. https://doi.org/10.1145/2993274.2993277

[12] J. Saldaña. 2009. *The Coding Manual for Qualitative Researchers*. Sage, USA. https://books.google.com.br/books?id=OE7LngEACAAJ

[13] Emad Shihab, Christian Bird, and Thomas Zimmermann. 2012. The effect of branching strategies on software quality. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (Lund, Sweden) *(ESEM '12)*. Association for Computing Machinery, New York, NY, USA, 301–310. https://doi.org/10.1145/2372251.2372305

[14] Bogdan Vasilescu. 2022. *Methods L07 - Qualitative Analysis [CMU 17803 Empirical Methods - Fall 2022]*. Carnegie Mellon University. https://www.youtube.com/watch?v=Wfi1s66Ig34

[15] Stefan Wagner, Daniel Mendez, Michael Felderer, Daniel Graziotin, and Marcos Kalinowski. 2020. *Challenges in Survey Research*. Springer International Publishing, Cham, 93–125. https://doi.org/10.1007/978-3-030-32489-6_4

[16] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Analysis and Interpretation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 123–151. https://doi.org/10.1007/978-3-642-29044-2_10

[17] Thomas Zimmermann. 2016. Card-sorting: From text to themes. In *Perspectives on Data Science for Software Engineering*. Morgan Kaufmann, Boston, 137–141. https://doi.org/10.1016/B978-0-12-804206-9.00027-1