Branch-based or Trunk-based Development?

What do the experts say about it?

Autor: Vitor Cardim Menezes

Orientador: Paulo Henrique Monteiro Borba

Co-Orientadora: Paola Rodrigues de Godoy Accioly

Centro de Informática – Universidade Federal de Pernambuco (UFPE)

vcm3@cin.ufpe.br

Abstract. *This article is part of a course conclusion thesis. It aims to analyze the perception and knowledge about Branch-based development and Trunk-based Development, common workflow strategies. From the perspective of developers/engineers in the area, examining the favorable or unfavorable factors of these flows impacts their realities and projects. The problem is which flow to choose to work with, due to the limited existence of works that indicate flows based on the experience of specialists, the aim is to help with this choice.*

The methodology adopted for data collection was interviews, and the analysis was coding with coding, following a qualitative approach.

Resumo. *Este artigo é parte de um trabalho de conclusão de curso. Tem como objetivo analisar a percepção e o conhecimento sobre o desenvolvimento Branch-based e Trunk-based Development, estratégias comuns de fluxo de trabalho. Na perspectiva de desenvolvedores/emgenheiros da área, examinando os fatores favoráveis ou não favoráveis desses fluxos, impactos em suas realidades e projetos. A problemática é qual fluxo escolher para trabalhar, pela pouca existência de trabalhos que fazem a indicação dos fluxos com base na experiência de especialistas, o intuito é ajudar nessa escolha.*

A metodologia adotada para a coleta de dados foi a entrevista, e a análise codificação com codificação, seguindo uma abordagem qualitativa.

1. Introdução

Técnicas e ferramentas de versionamento de software são essenciais para a adoção do DevOps e garantia da competitividade das empresas de desenvolvimento de software [1]. DevOps é um processo de desenvolvimento de software que inclui uma mudança de cultura organizacional, focado em acelerar a entrega de um software de melhor qualidade [18]. Dentre essas ferramentas, o Git se destaca como um sistema de controle de versão cada vez mais popular por ser de código aberto e permitir a criação de diferentes branches de desenvolvimento de forma rápida e fácil. Isso permite que os desenvolvedores trabalhem em seus projetos e tarefas de forma independente e sem muita necessidade de comunicação.

Blogs, fóruns e redes sociais discutem os melhores fluxos de trabalho para trabalhar com o Git. As abordagens geralmente se enquadram em duas categorias: fluxos de trabalho baseados em branches, onde os desenvolvedores trabalham em funcionalidades e correções de bugs isoladamente, depois reintegrando suas contribuições ao repositório principal, e desenvolvimento baseado em tronco (trunk), que minimiza o número de ramificações, permitindo que todos os desenvolvedores integrem suas contribuições diretamente no repositório principal. Cada uma destas duas abordagens têm vantagens e desvantagens, mas, apesar da controvérsia, apenas alguns estudos científicos investigam mais profundamente esta questão [2,3]

Para entender melhor essas vantagens e desvantagens, este artigo investiga as percepções e o conhecimento de engenheiros e desenvolvedores sobre estratégias comuns de fluxo de trabalho de desenvolvimento baseados em branch e tronco em ferramentas de controle de versão. Através de entrevistas com os profissionais, este estudo visa identificar os impactos favoráveis e não favoráveis de cada abordagem nos projetos reais dos participantes. A metodologia adotada é qualitativa, e os dados foram coletados nas entrevistas e transformados em códigos para obtenção dos fatores, os quais são as determinantes de favorecimento ou não, sendo de uma forma de unificar pontos semelhantes a partir das entrevistas.

Os resultados revelaram uma predominância do uso do Git Flow (modelo de desenvolvimento baseado em branches) sobre o desenvolvimento em tronco entre as pessoas entrevistadas. Fatores são aspectos que influenciam favoravelmente ou não um dos fluxos de gerenciamento de código, indo desde a Rigidez dos Processos até a Complexidade da Gestão do Código. A conclusão é que ambos métodos de controle de fluxo são válidos e eficazes, porém é possível otimizar sua utilização de acordo com os fatores existentes nos contextos de cada empresa, podendo esses fatores serem culturais ou técnicos. Por exemplo, Branch-based Development é mais amigável a equipes com menor senioridade, por causa dos seus focos e etapas, mas mesmo assim pode ser complicado a gestão da mesma.

O restante do artigo está organizado da seguinte forma:

- Na seção 2 é exposto o conhecimento básico necessário para entendimento do artigo.
- Na seção 3 é descrita a metodologia seguida neste trabalho.

- A seção 4 é relativa aos resultados das perguntas de pesquisa expostas na anterior.
- Na seção 5 é onde expõe-se as ameaças à validade deste artigo e como a mitigamos.
- Na seção 6 são expostas as conclusões que inferimos.
- Na seção 7 temos os próximos passos que tomaremos nos próximos trabalhos.
- Na seção 8 tem-se as referências bibliográficas e links importantes que são citados no artigo.

2. Background

Nesta seção será fornecida uma breve descrição de pontos importantes para entendimento do artigo, como: Controle de Versão de Código [12], Git [13,14], Branch-based Development [10,15] e Trunk-based Development [16].

2.1. Controle de Versão de Código

O Controle de Versão de Código, também conhecido como Gerenciamento de Versão ou Controle de Versão de Software, é uma prática crucial no desenvolvimento de software que envolve o gerenciamento de alterações de software. A parte que nos importa aqui é relativa à "Ramificação e Mesclagem" (Branching e Merging), permitindo que desenvolvedores criem branches separadas do código principal para desenvolver novas funcionalidades ou corrigir bugs, fazendo o merge quando finalizado o trabalho.

2.2. Git

Git é um sistema de controle de versão, amplamente utilizado no desenvolvimento de software para rastrear e gerenciar mudanças no código-fonte. As características do Git que são relativas ao nosso trabalho são:

1. **Distribuído**: Cada desenvolvedor trabalha com uma cópia local do repositório de código;
2. **Branches e Merges Eficientes**: Git permite um workflow de branches e merges eficiente;
3. **Flexibilidade de Workflow**: Git suporta vários fluxos de trabalho de desenvolvimento, permitindo que equipes escolham a abordagem que melhor se adapta às suas necessidades;

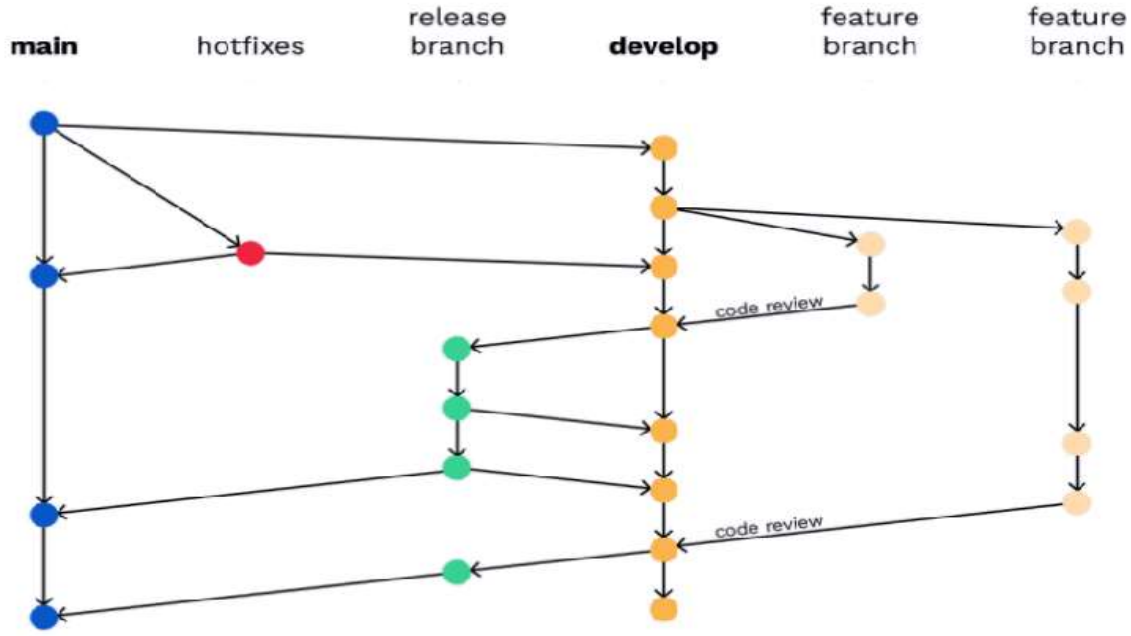

Figura 1. Branch-based Development, Fluxo Git Flow [19]

2.3. Branch-based Development

Ao usar o Git, a equipe pode definir um fluxo de trabalho para versionar o código. Dentre as possíveis estratégias, temos o Branch-based Development, uma estratégia de controle de versão que utiliza diferentes ramificações.

Git Flow é um modelo complexo normalmente preferido por equipes de grandes [20], baseado no Branch-based Development, definindo uma estrutura rigorosa de branches para diferentes finalidades, garantindo clareza e eficiência no desenvolvimento de software. Os principais branches, conforme mostrado na Figura 1, e detalhados abaixo são:

1. **Master**(**main**): A master contém o histórico do projeto;
2. **Develop**: A develop serve como uma união para funcionalidades, onde todas fazem o merge para o próximo release.
3. **Feature Branches**: Criadas a partir do develop, são usadas para desenvolver novas funcionalidades e posteriormente mergeadas na develop. Normalmente, essas branches têm um escopo limitado a uma funcionalidade específica.
4. **Release Branches**: Criadas a partir da develop, são usadas para preparar uma versão produtiva. Quando tudo está pronto, ela é mergeada na master e develop.
5. **Hotfix Branches**: Criadas a partir da master, são usadas para corrigir bugs na versão produtiva. Após a correção, ela é mergeada na master e develop (ou na release atual, se houver).

Também podemos ter outros tipos de branches, dependendo do projeto e como fazem suas gestões, como uma branch para o ambiente de qualidade(qas) ou de pré-produção(uat ou staging).

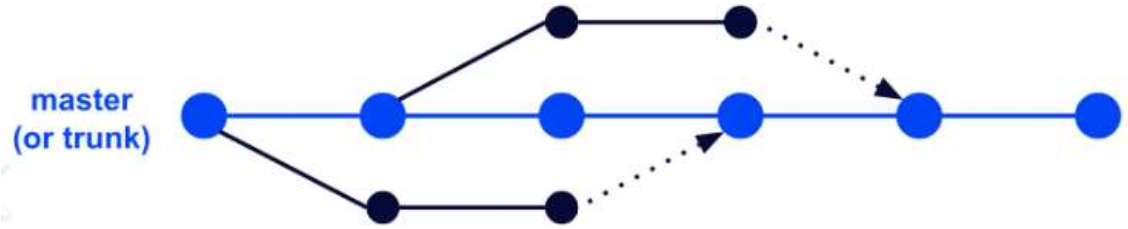

Figura 2. Fluxo Trunk-based Development [9]

2.4. Trunk-based Development

Trunk-based Development é uma estratégia de controle de versão em que todos os desenvolvedores colaboram em um único branch, chamado master, em vez de trabalhar em branches separadas para funcionalidades, melhorias ou correções, conforme mostrado na Figura 2. Essa abordagem promove a integração contínua e a entrega contínua (CI/CD), pois as alterações são frequentemente integradas na master, idealmente várias vezes ao dia.

2.5. Trabalhos Relacionados

Esta seção expõe os trabalhos que influenciaram a linha de pesquisa.

Adams e McIntosh [1] afirmam que o processo de release é responsável por transformar o código desenvolvido pelos programadores em uma versão que possa ser utilizada pelos usuários finais. Esse processo inclui várias etapas, como integração de alterações no código, automatização da construção do software e distribuição. Com a adoção da entrega contínua, essas atualizações chegam aos usuários com muito mais rapidez. O artigo discute a importância de pesquisadores estudarem esse processo para compreender melhor suas práticas e impacto, especialmente porque ainda existe uma lacuna significativa na validação dessas práticas e na compreensão de como elas afetam a qualidade do software, representando uma área rica para pesquisas futuras.

Rios et al. [2] sugerem vários fluxos de trabalho Git, promovendo diferentes métodos para desenvolvimento colaborativo e gerenciamento de qualidade de código. Esses fluxos de trabalho oferecem às equipes opções e compensações entre a proteção do código e a quantidade e tipo de esforço necessários. No entanto, esta diversidade implica que a escolha correta de um fluxo de trabalho para uma equipa exige que alguns dos seus membros estejam familiarizados com vários fluxos de trabalho diferentes. Os estudos têm sinergia, por fazerem a análise dos fluxos de trabalho por diferentes visões, o nosso para ajudar na escolha entre desenvolvimento em Branch-based ou em Trunk-based Development, e o dele fornece um framework para configurar um fluxo de trabalho de acordo com um modelo de features.

De Boer [3] teve como objetivo observar a migração do Desenvolvimento baseado em Trunk-based Development para o Desenvolvimento por Merge-request, analisando principalmente as opiniões dos desenvolvedores. Embora a migração tenha como objetivo reduzir erros no branch master e melhorar a qualidade do código, a transição completa da empresa para o novo modelo ainda não permite afirmar se os

objetivos foram alcançados. Os desenvolvedores tenderam a concordar que o modelo de merge-request traz benefícios para a qualidade do código produzido por eles. Trabalhos futuros poderiam analisar o número de erros antes e depois da migração e usar mais métricas para analisar quantitativamente revisões de código, explorando comparações entre modelos e suas interdependências.

Accioly et al. [11] tenta prever os conflitos de merge, de acordo com os tipos de mudanças feitas no código. Por exemplo, se dois desenvolvedores editam de forma concomitante o corpo do mesmo método. Os estudos se complementam, por conflitos serem algo impactante na estratégia de versionamento, por cada tipo de workflow tentar mitigá-los à sua maneira.

Dias et al. [17] trabalha na linha de análise do código e conflitos de merges, demonstram que branches mais longas podem influenciar nos conflitos. Os estudos são complementares ao nosso, por serem questões entranhadas na escolha da estratégia de versionamento.

Levando em conta os aspectos mencionados, o objetivo deste artigo é apresentar provas ligadas às técnicas de fluxo trabalho fundamentadas no Trunk-based Development e Branch-based Development, examinando os elementos que seriam de interesse para especialistas, desenvolvedores e engenheiros de software, para assim contribuir com nossa investigação, fornecendo resultados significativos e formulando conclusões.

3. Metodologia

Esta seção demonstra os principais pontos que nortearam a pesquisa e os passos para o resultado final.

3.1. Objetivo da Pesquisa

O objetivo deste trabalho é entender como os desenvolvedores utilizam os dois tipos de fluxo de trabalho na prática, como eles percebem as vantagens e desvantagens de cada tipo e se entendem quais fatores (técnicos, organizacionais ou sociais) favorecem ou não a adoção de cada fluxo de trabalho. Com base na sua realidade, esta pesquisa irá gerar recomendações para pessoas que não sabem qual fluxo adotar.

3.2. Perguntas de Pesquisa

Para atingir o nosso objetivo, procuramos responder às seguintes perguntas de pesquisa, visando colher esclarecimentos sobre a utilização de estratégias de controle de versão:

1. PP1 - Quais Fluxos de Trabalho os Desenvolvedores Usam?
2. PP2 - Quais Fatores Influenciam a Adoção das Estratégias Para Versionamento de Código Usada Pelos Desenvolvedores?
3. PP3 - Quais Fatores Favorecem o Uso de Branch-based Development?
4. PP4 - Quais Fatores Não Favorecem o Uso de Branch-based Development?
5. PP5 - Quais Fatores Favorecem o Uso de Trunk-based Development?
6. PP6 - Quais Fatores Não Favorecem o Uso de Trunk-based Development?

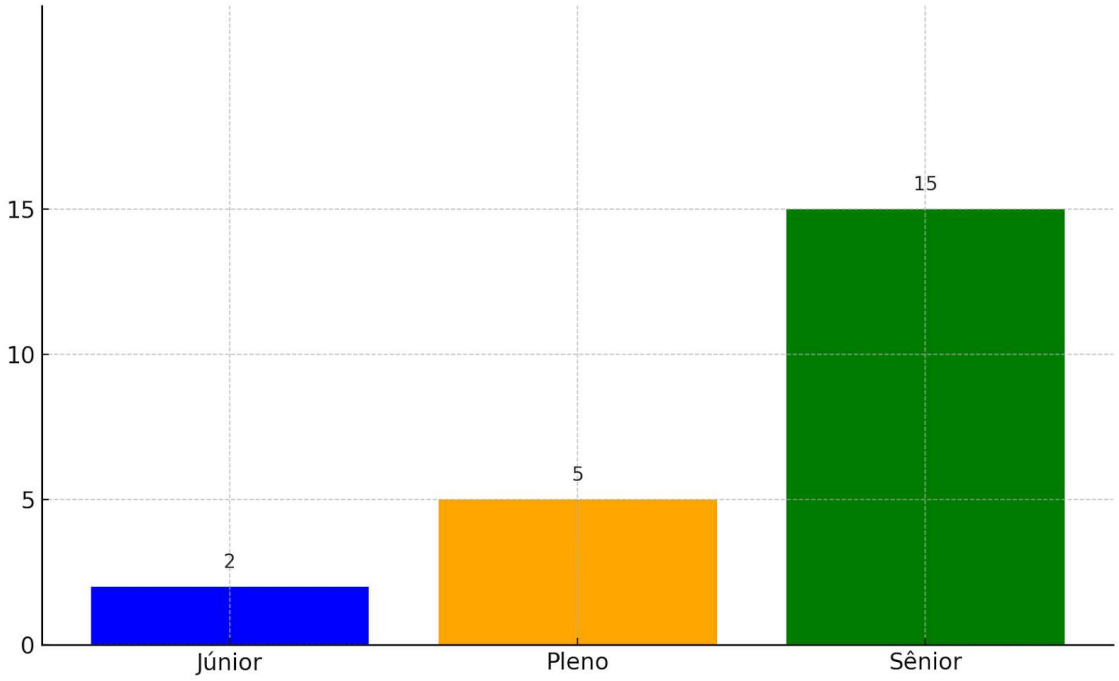

Figura 3. Senioridade dos Entrevistados

3.3. Seleção dos Participantes

Selecionamos os entrevistados por amostra de conveniência e rede de contatos, que construí na universidade e empresas em que trabalhei. Como mostra a Figura 3, preferimos indivíduos com níveis de senioridade mais elevados, começando os convites por eles, tendo 1 recusa, sendo representados pela cor verde, e amarelo os plenos. Porém, não descartamos os juniores, representados pelo azul.

Tabela 1. Entrevistados

ID	Gênero	Desenvolvedores e Engenheiros no Time	Cargo	Startup
E1	Masculino	5	Sênior	Não
E2	Masculino	10	Sênior	Não
E3	Masculino	10	Sênior	Não
E4	Masculino	12	Sênior	Sim
E5	Masculino	22	Pleno	Não
E6	Masculino	3	Pleno	Não
E7	Masculino	4	Pleno	Não
E8	Masculino	15	Sênior	Sim
E9	Masculino	12	Pleno	Não
E10	Masculino	4	Sênior	Não
E11	Masculino	3	Sênior	Sim
E12	Masculino	2	Junior	Sim
E13	Masculino	2	Sênior	Sim

E14	Masculino	5	Junior	Não
E15	Masculino	20	Sênior	Não
E16	Masculino	3	Sênior	Não
E17	Masculino	2	Sênior	Não
E18	Masculino	4	Sênior	Não
E19	Masculino	6	Sênior	Não
E20	Masculino	4	Sênior	Não
E21	Masculino	6	Sênior	Não
E22	Feminino	2	Pleno	Não

Inicialmente foram selecionados 37 nomes possíveis, pela facilidade de acesso, mas escolhemos apenas aqueles exclusivamente engenheiros/desenvolvedores que ainda trabalham diretamente com código e não apenas gerenciando equipes. A razão é que o gerenciamento de código é realizado e afeta mais desenvolvedores do que gerentes, porque quem cuida disso todos os dias são eles. Originalmente, pretendíamos atingir 20 entrevistados, mas entrevistamos 22, porque 2 foram descartados, o motivo é explicito na parte da análise das entrevistas, e queríamos alcançar o número de 20 entrevistas válidas, pois no início do processo tinha-se pensado nesse número decorrente ao tempo disponível para as entrevistas, análises e a entrega final.

Foi garantido aos participantes que tudo seria anônimo, tanto questão pessoal quanto profissional, exemplo do seu nome e onde trabalha, apenas verbalmente. Os entrevistados aceitaram serem gravados para que posteriormente a sua entrevista pudesse ser analisada. Sendo as mesmas feitas remotamente.

3.4. Processo da Entrevista

O roteiro da entrevista iniciou com 11 questões, sendo 7 para saber mais sobre o projeto e os entrevistados e 4 sobre os fatores. A primeira entrevista foi uma entrevista de validação para obtenção de inputs, sendo acrescentada outra pergunta, totalizando 12. Na terceira entrevista foi acrescentada mais pergunta, fechando o total de 13. A contribuição dos participantes motivou o acréscimo das questões 11 e 12; eles acharam que essa poderia ser uma excelente métrica para o processo.

As perguntas 6, 7, 8 e 9 foram feitas baseadas no fluxo de trabalho de cada um, se o entrevistado trabalhava com Trunk-based, foram perguntadas apenas as 8 e 9, caso trabalhasse com Branch-based 6 e 7, e caso ambos fluxos fossem utilizados as 6, 7, 8 e 9.

As perguntas feitas aos entrevistados:

1. Qual seu cargo você atua?
2. Em quantos projetos você trabalha?
3. Quantos desenvolvedores existem no seu projeto?
4. Descreva o passo-a-passo de uma alteração de código do instante em que é implementada até o momento em que é enviada ao repositório remoto, e então colocada em produção no seu projeto.

5. Nos projetos passados, você utilizou um fluxo diferente, sendo Git e profissionalmente? Os fluxos que trabalhou, saberia os nomear?
6. Quais fatores que favorecem o uso de Branch-based?
7. Quais fatores que não favorecem o uso de Branch-based?
8. Quais fatores que favorecem o uso de Trunk-based?
9. Quais fatores que não favorecem o uso de Trunk-based?
10. Você mudaria o modelo atual do seu projeto? Indagar o porquê, sendo positivo ou não.
11. Você classifica a empresa que trabalha como Startup? - Sugestão (E1, Sênior)
12. Você tem problema recorrente com conflitos? - Sugestão (E4, Sênior)
13. Tem alguma sugestão de pergunta que deveria ter sido feita?

Para os como realizar a entrevista, foram utilizados os preceitos expostos por Vasilescu [4], explicando a razão da entrevista entrevista, que era para o trabalho de conclusão de curso e possivelmente a escrita de um artigo, também os deixando mais calmos com uma conversa informal. Seguindo por uma entrevista semi-estruturada, sendo todas respostas abertas e evoluindo no decorrer das entrevistas, como por exemplo as sugestões dos E1 e E4, levando em consideração a experiência de cada entrevistado para fazer as perguntas sobre os fatores, pois o mesmo precisava estar trabalhando com os tipos de estratégias que foram objeto deste trabalho.

Tomou-se um grande cuidado para deixar os entrevistados a vontade, para não se sentirem avaliados ou julgados, explicitando que não existe certo ou errado, aquilo era a opinião pessoal de cada um, também não incorrendo em indução ou condução da entrevista, deixando o participante totalmente livre para responder como quisesse e no tempo que quisesse.

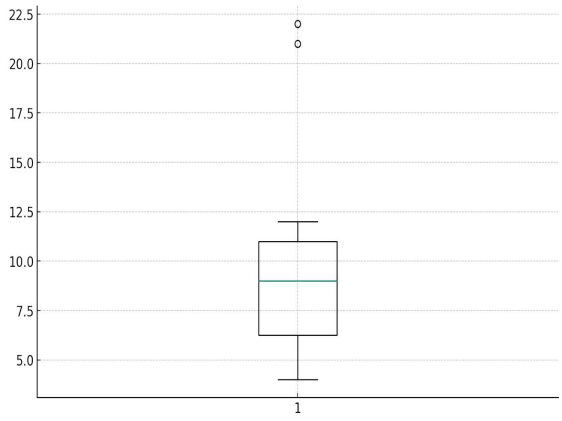

Figura 4. Tempo de Entrevista dos Entrevistados em Minutos

Todas entrevistas foram feitas pela ferramenta Zencastr [7], pois o mesmo consegue garantir que mesmo que a conexão caia ou internet esteja com problemas, o áudio seja preservado e mantenha a qualidade, e a transcrição foi feita pelo Turboscribe [8], utilizando o algoritmo que visa maior precisão, onde verificou-se apenas pequenas inconsistências em palavras, que raramente ocorriam, mas a compreensão não ficando comprometida. Podemos ver a distribuição do tempo de entrevista, usando um box plot na Figura 4, mostrando as variações nos tempos de entrevista, tendo poucos outliers,

sendo muito discrepantes dos quartis e até mesmo do seu máximo, ficando o segundo quartil, sendo chamado também de mediana entre 7,5 e 10 minutos.

3.4. Análise da Entrevista(codificação)

Na parte de análise, seguiu-se Saldana [6], e também a aula de Vasilescu [5] para codificar e analisar os dados, sendo escolhida a abordagem de codificação aberta.

Realizamos entrevistas e analisamos os dados que coletamos. Utilizamos a primeira entrevista como piloto para criar os códigos iniciais e as demais entrevistas como complementares para completar o processo de codificação, no caso o primeiro candidato também fez parte da análise total. Empregamos uma abordagem flexível e ajustamos os códigos conforme necessário. Isso permitiu identificar dez códigos para análise sem categorizá-los.

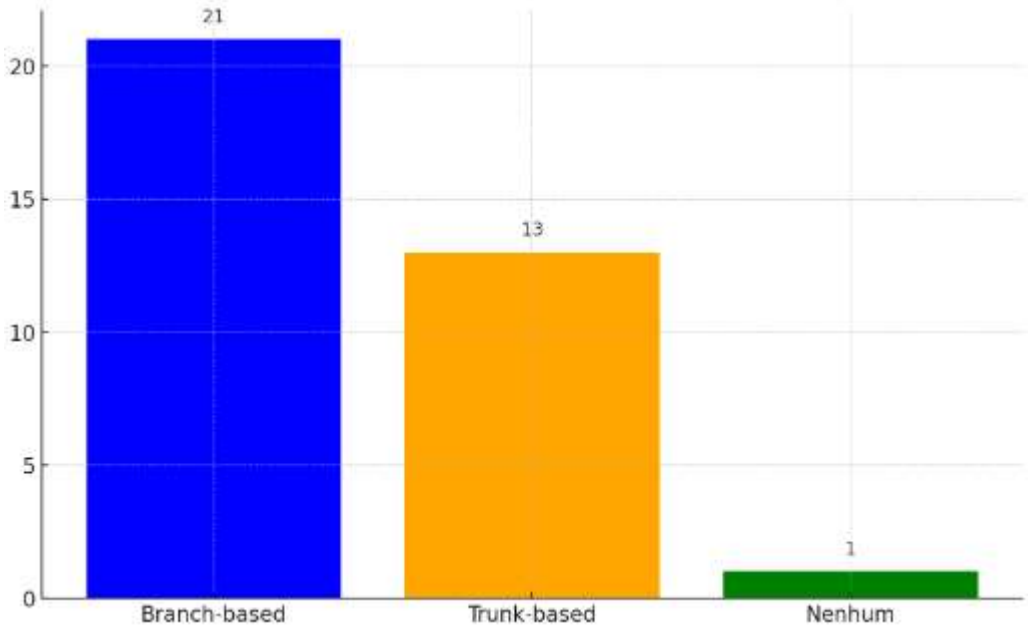

Figura 5. Conhecimento Sobre Estratégia de Versionamento

Dentre os 22 entrevistados, foi necessário excluir 2 candidatos, pois um não sabia sobre versionamento de código ou com que tipo de fluxo trabalhava, conseguiu explicar parcialmente, além de não saber o nome do fluxo. Identificamos o fluxo com base no processo descrito, mas quando questionado sobre o nome do fluxo ele disse que não sabia, e então não vimos validade na entrevista do participante, sendo esta representada por Nenhum na Figura 5, podendo o candidato estar em Branch-based e Trunk-based ao mesmo tempo, pois é a aferição de conhecimento, sendo 13 conhecendo ambos, 8 exclusivamente Branch-based e 1 Nenhum. . outro participante foi excluído por inferir e achar que trabalhava com os dois fluxos, mas só trabalhava com um, sem conhecer seus conceitos, o que também me impediu de ver a validade da análise na codificação.

Após a primeira rodada de análise, meu Orientador e Co-orientadora me ajudaram a validar o processo e discutir seus resultados, sendo sempre o ponto focal para aprovar como o processo e codificação estavam sendo feitas.

4. Resultados

Esta seção apresenta os resultados da pesquisa organizados por perguntas de pesquisa, com acréscimo de um resumo dos resultados que aparecem no tópico “Quais São os Prós e Contras de Cada Fluxo?”.

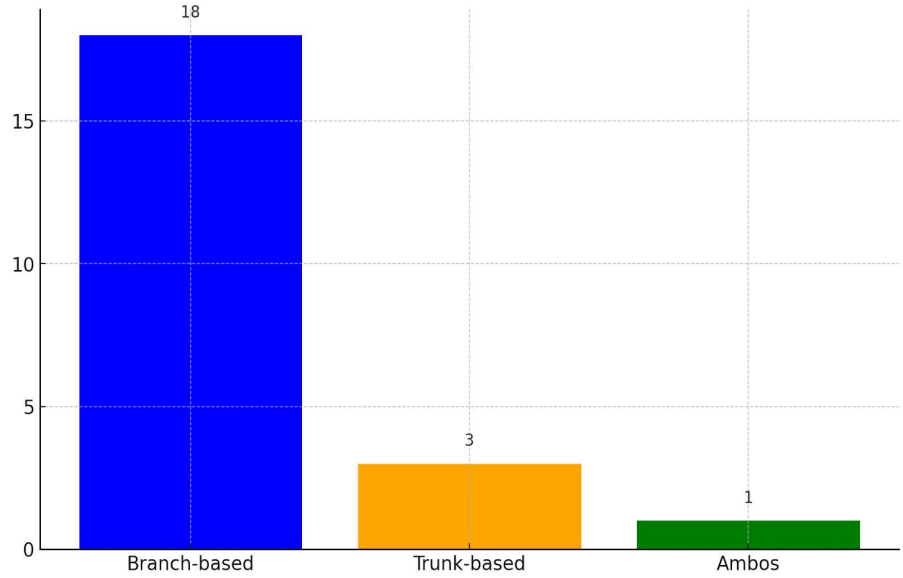

Figura 6. Fluxo de Trabalho

4.1. PP1 - Quais Fluxos de Trabalho os Desenvolvedores Usam?

A Figura 6 mostra que o Branch-based Development, especialmente o Git Flow, é predominante ao Trunk-based Development, pois pode-se ver 18 contra 3. Também é incomum que os projetos usem os 2 fluxos simultaneamente, onde ocorreu pelo monolito utilizar Branch-based e os microsseviços Trunk-based, representando apenas um. Isso mostra que Branch-based Development, também conhecido como Git Flow, é mais comum e usado no nosso espaço amostral.

Figura 7. Fatores Codificados

4.2. PP2 - Quais Fatores Influenciam a Adoção das Estratégias Para Versionamento de Código Usada Pelos Desenvolvedores?

A Figura 7 ilustra os fatores extraídos do processo de codificação. A figura mostra a frequência de citação dos fatores, sendo que os verdes com maior incidência, por exemplo um dos entrevistados tenha comentado sobre o fator 10 vezes, o mesmo só foi quantificado uma vez, e os vermelhos menor, por exemplo, Setor Restrito que foi pouco citado.

Em relação aos fatores, descrevemos brevemente cada um deles, com exemplos, para facilitar a compreensão do que significam. Eles foram extraídos do que os participantes relataram, sendo os mesmos neste tópico, neutros, pois não avaliaram a questão de favorecer ou não algum tipo de estratégia de versionamento.

Processos Rígidos: O fator leva em consideração a gestão do gerenciamento, se existe algum tipo de liberdade ou é algo que é obrigatório e padrão, como exemplo temos o relato de um dos entrevistados que diz: “.... Eu acho que para a gente, aqui onde a gente trabalha, o pior é lidar com a burocracia ....” (E18, Sênior).

Ambientes Separados: O fator leva em consideração a separação de ambientes, como desenvolvimento, qualidade, pré-produção e outros, antes do produtivo, podendo cada um desses ter uma branch ou ser feito a partir de um commit específico, como exemplo temos a fala de um dos entrevistados que diz: “....retrocedendo para um modelo mais tradicional de ambientes separados.” (E1, Sênior), no caso, o mesmo estava trabalhando numa estrutura de Trunk-based Development e passou a trabalhar com Branch-based Development, Git Flow.

Complexidade de Gerenciamento de Código: O fator leva em consideração a complexidade de como gerenciar o versionamento, o quão custoso e complicado pode o ser, como exemplo temos a fala de um dos entrevistados que diz: “...a complexidade é menor…” (E2, Sênior)

Testes e Subida Rápida para Produção: O fator leva em consideração o quão rápido consegue-se ter o código feito até sua subida efetiva em produção, que não seja relativo a incidentes ou correção de bugs, temos como exemplo a fala de um dos entrevistados que diz: "...a vantagem é porque a gente tem menos processos na hora de subir para a produção, porque é só um pipe…" (E10, Sênior), no caso, o mesmo estava trabalhando numa estrutura de Trunk-based Development e passou a trabalhar com Branch-based Development, Git Flow.

Tempo Rápido de Resposta de Incidentes: O fator leva em consideração o quão rápida é a subida de uma implementação de um incidente produtivo, temos como exemplo a fala de um dos entrevistados que diz: "Era muito mais rápida, menos de meio dia a correção já estava em produção." (E1, Sênior).

Equipes composta por sua maioria de desenvolvedores com menor experiência: O fator leva em consideração equipes onde tem incidência de equipes onde o entrevistado mostrou e explicitou o aspecto de senioridade, para falar que a mesma falta, temos como exemplo a fala de um dos entrevistados que diz: "...porque em todos os projetos temos equipes multidisciplinares e com vários níveis de conhecimento, certo? Então, basicamente, nos projetos atuais, a gente tem muito desenvolvedor, diria que não é sênior…" (E15, Sênior).

Foco em Qualidade: O fator leva em consideração a expressa preocupação com qualidade, mesmo ela sendo algo que deveria ser algo implícito a qualquer tipo de implementação, fluxo e afins, pois qualidade é um dos requisitos primordiais para qualquer desenvolvimento, temos como exemplo a fala de um dos entrevistados que diz: "...eu acho que é uma coisa massa. Até por questão de melhoria do código mesmo…" (E9, Sênior).

Autonomia: O fator leva em consideração a liberdade de trabalho em relação ao desacoplamento do que os outros da equipe estão fazendo, sendo um dos preceitos básicos do Git e até esperado de aparecer, temos como exemplo a fala de um dos entrevistados que diz: "E aí você consegue uma certa independência para trabalhar… " (E3, Sênior).

Foco em Velocidade de Entrega: O fator leva em consideração entregas rápidas, onde sejam feitas entregas com menores porções de código, como exemplo temos a fala de um dos entrevistados que diz: "Nosso projeto ele é bastante rápido, então tem features que são pequenas e elas sobem com bastante frequência…" (E13, Sênior).

Setor Restrito: O fator leva em consideração o tipo de setor que o entrevistado trabalha, como exemplo o setor financeiro que tem suas particularidades, temos como exemplo o entrevistado que diz: "... uma empresa do ramo financeiro…" (E5, Sênior).

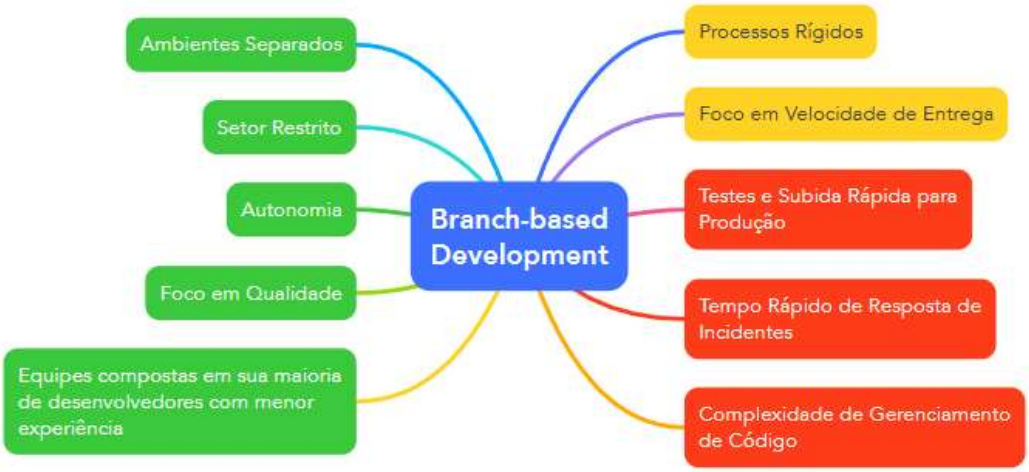

Figura 8. Fatores Branch-based Development

4.3. PP3 - Quais Fatores Favorecem o Uso de Branch-based Development?

Levando em consideração os fatores expostos na Figura 8, onde o impacto favorável de cada fator, sendo os verde, no Branch-based Development, que também podemos exemplificar como Git Flow, e não apenas sua incidência como na Figura 7. Esta seção vai expor os aprendizados resultantes de cada um dos fatores benéficos.

Ambientes Separados: Concluímos que passa uma confiança aos entrevistados, deixando-os mais à vontade no aspecto em si de trabalho, por conseguirem ter um maior controle do que vai subir a produção, por conseguirem ter mais de um grau de controle de subida.

Setor Restrito: Concluímos que se adequou muito bem em ambientes que têm certas particularidades decorrentes do seu tipo de negócio, pois todos os negócios que tinham particularidades faziam o uso desse tipo de versionamento.

Autonomia: Concluímos que é altamente presente, sendo benéfica, seja tanto pela questão das branches de feature, baseadas na develop, por sempre estar atualizada, deixando independente o trabalho da pessoa a qual está implementando.

Foco em Qualidade: Concluímos que o mesmo deveria ser uma preocupação já implícita e levada em consideração independentemente do fluxo escolhido, existiu uma grande preocupação nesse aspecto, principalmente pela questão da unicidade do mesmo com o fator dos ambientes separados, que para os mesmos ajuda no aspecto de garantir uma melhor qualidade de subida a produção.

Equipes composta por sua maioria de desenvolvedores com menor experiência: Concluímos que quando se tem equipes com discrepância de senioridade alta ou a equipe não é muito sênior, foi muito evidente a benesse do fluxo, em concomitância com o aspecto do Foco em Qualidade e Ambientes Separados, sendo os 3 grandes indicativos para trabalhar com equipes que não são compostas por sua maioria sêniores.

4.4. PP4 - Quais Fatores Não Favorecem o Uso de Branch-based Development?

Considerando os fatores apresentados na Figura 9, onde os em vermelho não favorecem e os que estão em amarelo não impactam.

Testes e Subida Rápida para Produção: Concluímos pelos expostos que utilizando esse tipo de fluxo, o mesmo não era aderente, decorrente dos processos de gestão de branches, tendo muitas vezes um deploy oneroso e lento.

Tempo de Rápido de Resposta de Incidentes: Concluímos, em concomitância com o exposto no fator anterior, é somado ao desenvolvedor normalmente está trabalhando na branch feature, que é um espelho da develop, que na maior parte do tempo vai estar com commits a frente da master, demanda maior esforço para a analisar o problema e entregar a correção do problema.

Complexidade de Gerenciamento de Código: Concluímos que pela preocupação na gestão das branches, sendo elas as tradicionais e algumas vezes algumas a mais pela divisão de ambientes, que quanto mais branches existe, mais complexo fica a sua gestão e garantia das mesmas estarem atualizadas como deveriam.

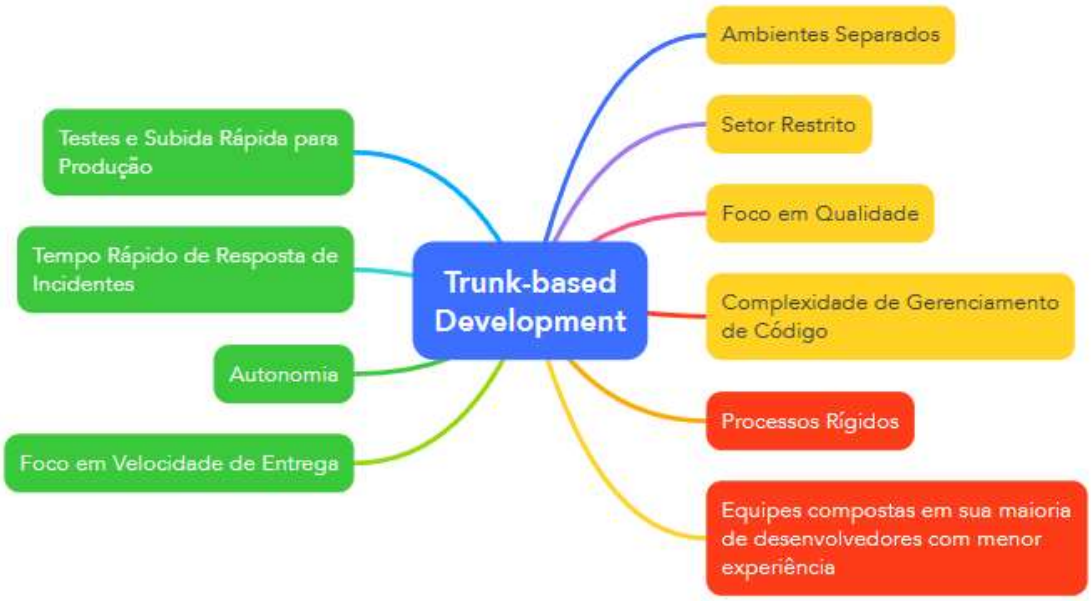

Figura 9. Fatores Trunk-based Development

4.5. PP5 - Quais Fatores Favorecem o Uso de Trunk-based Development?

Considerando os fatores mostrados na Figura 9, demonstrando aqueles em verde, como favoráveis ao Trunk-based Development, explicaremos o que aprendemos sobre eles neste fluxo de versionamento.

Testes e Subida Rápida para Produção: Concluímos que pela simplicidade na gestão das branches, é algo que é extremamente aderente, principalmente pelo o exposto pelos entrevistados.

Tempo de Rápido de Resposta de Incidentes: Concluímos em concomitância com o acima, nosso aprendizado na análise das entrevistas, que os desenvolvedores

estão trabalhando em uma branch baseada na master, sendo espelho de produção, demandando menor tempo de análise do problema.

Foco em Velocidade de Entrega: Concluímos que foi bem expressa a questão de se trabalhar sempre com pequenas porções de código e fazer subidas mais frequentes, que o mesmo é totalmente aderente, sendo um dos fatores principais nesse tipo de fluxo, sendo os 2 dois acima também influenciados por este.

Autonomia: Concluímos que a mesma é altamente presente, sendo benéfica, tanto pela questão de você não ter problemas de interdependência em relação à branches, por sempre ter um espelho da master sempre.

4.6. PP6 - Quais Fatores Não Favorecem o Uso de Trunk-based Development?

Considerando os fatores apresentados na Figura 9, onde foi levado em consideração o impacto de não favorecimento, representados pelos os que estão em vermelho.

Processos Rígidos: Concluímos que o mesmo é um grande impacto a celeridade que o Trunk-based Development traz, pois vem uma ideia onde o rito é mais importante do que a entrega em si.

Equipes composta por sua maioria de desenvolvedores com menor experiência: Concluímos que pelo demonstrado pelos os entrevistados, existe uma questão bem clara do favorecimento ao tipo de fluxo com desenvolvedores com maior grau de senioridade, pois todo código vai direto para branch main, sempre existe a questão de problemas e riscos de conflitos de merge onde os menos experiente podem fazer erroneamente.

4.7. Quais Prós e Contras de Cada Fluxo?

Fazendo uma unificação do exposto nas PP3, PP4, PP5 e PP6, exporemos o que aprendemos sobre elas, levando em consideração as análises de todos os entrevistados e as conclusões que tiramos sobre os cenários.

O Branch-based Development oferece isolamento para trabalhar em funcionalidades, correções, e lançamentos em paralelo, aumentando a colaboração e a organização, mas pode se tornar complexo e desafiador ao gerenciar múltiplos branches e atrasar a integração e feedback das mudanças.

O Trunk-based Development promove a integração contínua e entrega contínua, facilitando o feedback rápido e reduzindo a complexidade do gerenciamento de versões, mas exige uma cultura de desenvolvimento disciplinada e pode aumentar o risco de instabilidade no branch principal.

Ambos os fluxos têm seus méritos e são totalmente condizentes com o fator de autonomia, por ambos darem liberdade para o desenvolvedor trabalhar de modo independente dos outros, e a escolha entre eles depende das necessidades do projeto, da cultura da equipe, e de práticas de desenvolvimento eficazes para gerenciar os desafios associados.

5. Ameaças à Validade

Nesta seção discutiremos que possíveis aspectos da metodologia escolhida podem representar ameaças à validade deste trabalho.

O primeiro ponto é a preferência em termos de implementação, pois pode-se dizer que não o é, mas normalmente existe, resultante de experiências passadas. Neste ponto, fomos muito sucintos e analisamos friamente todos os aspectos de cada fluxo, esquecendo literalmente qualquer tipo de experiência e nos atendo apenas ao que foi relatado pelos entrevistados.

O segundo ponto, a formação das perguntas, como pode ser visto na Figura 4, todas as questões relacionadas aos fluxos são neutras, não há nenhuma especificando algo benéfico ou não para o fluxo específico, e as questões que falam especificamente sobre o fluxo são o mesmo, apenas mudando seu nome.

O terceiro ponto, a escolha dos participantes, também pode ser pensado como um dos pontos que poderia minimizar a validade do estudo, pois por ter viés qualitativo, seria necessário ter insumos bons e válidos, principalmente pela questão que falamos de especialistas, os quais foram escolhidos por uma questão de networking, minimizando problemas na qualidade do material recolhido e privilegiando pessoas com mais experiência profissional como podemos ver na Figura 3.

6. Conclusão

A conclusão que chega-se, é que o Branch-based Development é mais conhecido e mais utilizado entre os entrevistados, sendo ele o mais comumente disseminado, porém não invalida as benesses trazidas pelo Trunked-based Development para os que fazem o uso da mesma.

Ambos fluxos são condizentes com os preceitos de distribuição e autonomia gerada pelo git, dando liberdade para os desenvolvedores trabalharem de modo mais ágil.

O que pode-se concluir sobre o Trunk-based Development tem melhor aderência em projetos que necessitem de celeridade, onde a equipe tenha um nível maior de senioridade, sendo essa senioridade relativa à experiência de trabalho e cargo, não idade.

E o que pode-se concluir sobre o Branch-based Development é mais amigável a equipes com menor senioridade, por causa dos seus focos e etapas, mas mesmo assim pode ser complicado a gestão da mesma.

Levando em consideração que ambos são aderentes nos aspectos base tanto de versionamento de código quanto do git, que foram demonstrados na seção de background, além do fator de autonomia que leva em consideração a distribuição do git e também liberdade de trabalhar de modo ágil e desacoplado do time.

7. Trabalhos Futuros

Apesar de conseguir chegar a conclusões sobre o Trunk-based Development, a quantidade de pessoas as quais foram entrevistadas as quais trabalham com o mesmo foi

pouca, sendo necessária novas entrevistas, para que assim consiga-se dar continuidade em novas análises e posteriormente, utilizando o Grounded Theory, que é uma técnica para criar teorias, criar uma relativo ao uso dos tipos de versionamento baseado nos fatores.

Logo depois da concepção da teoria, tentar validar a mesma com alguma(s) empresa(s) parceira(s), para ver como é o comportamento e chegar-se a uma conclusão final ou modificar a teoria, até conseguir validar uma.

8. Referências

[1] Adams, B., & McIntosh, S. (2016). Modern Release Engineering in a Nutshell: Why Researchers Should Care. SANER, IEEE. http://doi.org/10.1109/SANER.2016.108

[2] Ríos J, Embury S, & Eraslan, S. (2022). A unifying framework for the systematic analysis of Git workflows. Information and Software Technology. https://doi.org/10.1016/j.infsof.2021.106811

[3] De Boer, T. (2021). From Trunk-Based to Merge Requests: A Field Study at Adyen. Dissertação de Mestrado.

From Trunk-Based to Merge Requests: A Field Study at Adyen | TU Delft Repositories

[4] Methods L05 - Interviews [CMU 17803 Empirical Methods - Fall 2022]. Disponível em: <https://bit.ly/48JNbQO>. Acesso em: 23 mar. 2024.

[5] Methods L07 - Qualitative Analysis [CMU 17803 Empirical Methods - Fall 2022]. Disponível em: <https://bit.ly/49Y56UI>. Acesso em: 23 mar. 2024.

[6] Johnny Saldana. 2015. The Coding Manual for Qualitative Researchers. SAGE Publications Ltd.

[7] Zencastr. Disponível em: <https://zencastr.com/>. Acesso em: 23 mar. 2024.

[8] TurboScribe: Transcribe Audio and Video to Text or Subtitles in Seconds. Disponível em: <https://turboscribe.ai/>. Acesso em: 23 mar. 2024.

[9] Git Flow vs Trunk Based: uma comparação de estratégias | Criar Programas | Tudo sobre desenvolvimento de softwares. Disponível em: <https://criarprogramas.com/2023/07/05/git-flow-vs-trunk-based-uma-comparacao-de-estrategias/>. Acesso em: 23 mar. 2024.

[10] What is the best Git branch strategy? | Git Best Practices. Disponível em: <https://www.gitkraken.com/learn/git/best-practices/git-branch-strategy>. Acesso em: 23 mar. 2024.

[11] Accioly P, Borba P, Silva L, Cavalcanti G (2018) Analyzing conflict predictors in open-source java projects. In: Proceedings of the international conference on mining software repositories (MSR). ACM, pp 576–586 Analyzing conflict predictors in open-source Java projects | Proceedings of the 15th International Conference on Mining Software Repositories (acm.org)

[12]ATLASSIAN. What is version control. Disponível em: <https://www.atlassian.com/git/tutorials/what-is-version-control>. Acesso em: 23 mar. 2024.

[13]ATLASSIAN. O que é Git | Atlassian Git Tutorial. Disponível em: <https://www.atlassian.com/br/git/tutorials/what-is-git>. Acesso em: 23 mar. 2024.

[14]ATLASSIAN. Por que usar o Git | Atlassian Git Tutorial. Disponível em: <https://www.atlassian.com/br/git/tutorials/why-git>. Acesso em: 23 mar. 2024.

[15]ATLASSIAN. Gitflow Workflow | Atlassian Git Tutorial. Disponível em: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>. Acesso em: 23 mar. 2024.

[16]ATLASSIAN. Trunk-based Development. Disponível em: <https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development>. Acesso em: 23 mar. 2024.

[17] Dias K, Borba P, Barreto M (2020) Understanding predictive factors for merge conflicts. Information and Software Technology 121 (2020), 106256. https://doi.org/10.1016/j.infsof.2020.106256

[18] O que é DevOps? | IBM. Disponível em: <https://www.ibm.com/br-pt/topics/devops>. Acesso em: 23 mar. 2024.

[19] Disponível em: <https://res.cloudinary.com/practicaldev/image/fetch/s--LoUyfWu3--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https://dev-to-uploads.s3.amazonaws.com/uploads/articles/fs3q763bq555zh9u7faq.png>. Acesso em: 23 mar. 2024.

[20] Shakikhanli, U., & Bilicki, V. (2024). Optimizing Branching Strategies in Mono-and Multi-Repository Environments: A Comprehensive Analysis. Computer Assisted Methods in Engineering and Science. http://dx.doi.org/10.24423/cames.2024.1372