ANÁLISE DO IMPACTO DE SISTEMAS DE CONTROLE DE VERSÃO CENTRALIZADOS E DISTRIBUÍDOS NO ESFORÇO DE INTEGRAÇÃO DE CÓDIGO-FONTE: UM ESTUDO EMPÍRICO

Vinicius Lima Ville[1]
Kleinner Farias[2]

Resumo: Este estudo explora a importância dos sistemas de controle de versão (VCS) centralizados e distribuídos em projetos de desenvolvimento de software colaborativo, com foco específico no impacto desses sistemas no esforço de resolução de conflitos de integração. Foi realizado a análise de 116.388 conflitos de integração em 50 repositórios de projetos de código aberto disponíveis no GitHub, utilizando métodos estatísticos, tais como o Wilcoxon e o t-test pareado, visando de responder se há vantagem significativa na escolha de um tipo de VCS ou outro, com relação ao esforço na resolução de conflitos de integração. Os resultados obtidos neste estudo indicam que não é possível identificar uma vantagem estatisticamente significativa no esforço de resolução de conflitos de integração ao utilizar o Git (VCS distribuído) em comparação com SVN (VCS centralizado). Este resultado contesta a ideia generalizada de que sistemas distribuídos são inerentemente superiores em praticamente todos os aspectos. A contribuição significativa deste estudo reside na evidência empírica fornecida, oferecendo dados concretos para as decisões de desenvolvedores e equipes de projeto, destacando a necessidade de uma abordagem mais ponderada ao escolher entre sistemas de controle de versão. Além disso, o estudo propõe áreas de pesquisa futura, incluindo a exploração de fatores específicos de desenvolvimento, a implementação de algoritmos mais inteligentes para resolução de conflitos e a integração de ferramentas de inteligência artificial para aprimorar a análise de contexto durante a resolução de conflitos de integração.

Palavras-chave: VCS. Git. SVN. Mesclagem. Integração. Código-aberto

Abstract: This study explores the importance of centralized and distributed version control systems (VCS) in collaborative software development projects, with a specific focus on the impact of these systems on the integration conflict resolution effort. An analysis of 116,388 integration conflicts was carried out in 50 open source project repositories available on GitHub, using statistical methods, such as Wilcoxon and the paired t-test, aiming to answer whether there is a significant advantage in choosing a type of VCS or another, in relation to the effort to resolve integration conflicts. The results obtained in this study indicate that it is not possible to identify a statistically significant advantage in the integration conflict resolution effort when using Git (distributed VCS) compared to SVN (centralized VCS). This result challenges the widespread idea that distributed systems are inherently superior in virtually every aspect. The significant contribution of this study lies in the empirical evidence provided, offering concrete data for the decisions of developers and project teams, highlighting the need for a more considered approach when choosing between version control systems. Additionally, the study proposes areas for future research, including exploring specific development factors, implementing smarter algorithms for conflict resolution, and integrating artificial intelligence tools to enhance context analysis during conflict resolution integration.

[1]Graduando em de Ciência da Computação pela Unisinos. Email: viniville@gmail.com
[2]Mini-currículo do orientador. Email: kleinnerfarias@unisinos.br

Keywords: VCS. Git. SVN. Merge. Integration. Open-source

1 INTRODUÇÃO

Um projeto de software é composto, basicamente, por um conjunto de diretórios e arquivos de código-fonte, onde uma equipe de desenvolvedores trabalha em conjunto e de forma concorrente para conceber o software. Durante o processo de implementação, os desenvolvedores constantemente modificam, criam e removem arquivos e diretórios do projeto. Para tornar mais produtivo o processo de desenvolvimento, é fundamental que os membros da equipe compartilhem as funcionalidades que estão sendo desenvolvidas de forma independente com os demais integrantes da equipe e, para que isto seja possível, é essencial ter um local centralizado onde os artefatos alterados possam ser copiados, evitando erros decorrentes de um controle manual.

Com a crescente necessidade de desenvolver e manter sistemas complexos, é indispensável utilizar sistemas de controle de versão, também conhecidos como *Version Control Systems* (VCS). Ferramentas como Git, Subversion (SVN), Mercurial e Concurrent Versions System (CVS) facilitam o gerenciamento de versões de arquivos de código-fonte e a colaboração entre os desenvolvedores, possibilitando um desenvolvimento mais eficiente e reduzindo erros causados por conflitos no código-fonte, além de simplificar a execução de tarefas indispensáveis e frequentes no processo de desenvolvimento software, como comparar versões anteriores de um trecho de código-fonte após realizar uma alteração. Neste contexto, sistemas de controle de versão desempenham um papel essencial, sendo uma ferramenta que oferece o rastreamento de mudanças, colaboração em equipe, controle de versões, testes e implantação seguros, além de backup e recuperação de dados. Essas funcionalidades possibilitam o acompanhamento da evolução de um software além de contribuir para a eficiência, qualidade e segurança do processo de desenvolvimento.

No contexto de desenvolvimento colaborativo de software, os conflitos de integração são um problema comum, e ocorrem quando dois ou mais desenvolvedores fazem alterações no mesmo arquivo, ou trecho de código-fonte e tentam mesclá-las ao repositório principal. Os conflitos de integração podem surgir devido a diferenças nas alterações, como adição, remoção ou modificação de linhas de código, o que resulta em um estado ambíguo em relação à qual versão deve ser considerada correta. Isso pode levar a erros, perda de código ou resultados inesperados. Resolver estes conflitos requer comunicação efetiva entre os desenvolvedores envolvidos, além do uso de ferramentas adequadas fornecidas pelos sistemas de controle de versão. A colaboração ativa, a revisão de código-fonte e o estabelecimento de práticas de integração e atualizações regulares podem ajudar a mitigar os conflitos, garantindo que as alterações sejam integradas de forma correta.

É de conhecimento generalizado que o Sistema de Controle de Versão (VCS) distribuído, por exemplo, o Git (GIT, 2023), fornece um suporte superior para rotinas de desenvolvimento concorrente, fornecendo um fluxo de trabalho não linear, repositório completo com histórico

completo e recursos completos de rastreamento de versão. Assim, a expectativa é que a forma distribuída de edição e gerenciamento de código-fonte pode: (1) produzir código-fonte integrado de saída ($C_{integrado}$) mais próximo do código-fonte desejado de saída ($C_{desejado}$), resolvendo automaticamente as alterações conflitantes diretas e acomodando-as adequadamente em um código-fonte base (C_{base}); (2) oferecer suporte a mecanismos de integração mais eficazes, reduzindo a quantidade de trechos de código indevidamente inserido no (ou removido do) código-fonte integrado; (3) aliviar o esforço de integração por resolver adequadamente as mudanças conflitantes diretas e, consequentemente, diminuindo o número de intervenções manuais necessárias para produzir o código-fonte desejado de saída ($C_{desejado}$).

No entanto, existem poucos estudos empíricos produzidos até agora para confirmar (ou não) este conhecimento generalizado. Consequentemente, os desenvolvedores acabam adotando sistemas de controle de versão sem qualquer suporte de conhecimento empírico e opiniões de especialistas sobre eles não são necessariamente verdadeiras, muitas vezes levando a conclusões divergentes. Se um VCS específico fornece aos desenvolvedores recursos completos de rastreamento de versão, no entanto, tem um impacto prejudicial no esforço de integração, é bastante discutível se traz grande contribuição para suporte ao trabalho simultâneo - onde os conflitos de integração são problemas sempre presentes. Tendo o conhecimento empírico acessível, os desenvolvedores podem escolher racionalmente um determinado VCS mais adequado às suas necessidades.

Este trabalho, portanto, apresenta um estudo empírico exploratório que visa investigar o impacto dos VCS centralizados e distribuídos no esforço e eficácia de integração do código-fonte. O estudo procura: (1) entender o esforço que os desenvolvedores investem para mesclar códigos-fonte com partes de códigos conflitantes; (2) investigar o impacto de VCS centralizados e distribuídos no esforço de integração de código-fonte; (3) medir a eficácia de tais sistemas para resolver proativamente conflitos diretos de códigos-fonte. Com este foco, foi realizado um experimento orientado para a tecnologia, reproduzindo cenários de conflitos de integração ocorridos em 50 repositórios de projetos de código aberto disponíveis no site GitHub (GITHUB, 2023). Como critério de escolha, buscaram-se repositórios de projetos que tenham no mínimo 5.000 estrelas, tenham no mínimo 100 contribuidores, tenham recebido alguma atualização de código-fonte ainda em 2023 e tenham como linguagem de programação principal uma das cinco primeiras classificadas no TIOBE Index (TIOBE, 2023) do mês de junho de 2023. Foram descartados repositórios de tutoriais e educacionais, assim como os que não tivessem como idioma principal o inglês. Foram analisados mais de 625 mil *commits*, realizados por mais de 41 mil desenvolvedores, abrangendo aproximadamente 16.1 milhões de linhas de código-fonte.

Com base no histórico de *commits* dos repositórios, foi reproduzido os cenários de integração utilizando os VCS centralizado e distribuído mais populares, SVN e Git, respectivamente, e então realizou-se a comparação das duas versões de código-fonte integrado de saída, produzidas pelo VCS distribuído (Git) e pelo VCS centralizado (SVN).

As seções a seguir deste estudo foram organizadas da seguinte forma. Na Seção 2 é abor-

dado os principais conceitos citados neste estudo, com objetivo de fornecer embasamento e compreensão sobre sistemas de controle de versão, políticas de gerenciamento associadas e os desafios relacionados a conflitos de integração em código-fonte. Na Seção 3 são apresentados os trabalhos anteriores que se relacionam com o contexto deste estudo. Na Seção 4 é apresentado as principais decisões relacionadas ao desenho experimental adotado neste estudo. A Seção 5 examina os dados coletados no estudo e apresenta os resultados obtidos a partir destes dados. Por fim, na Seção 6 é apresentado as conclusões deste estudo empírico, além da oferecer uma direção para futuras pesquisas relacionadas ao tema deste estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção aborda os principais conceitos citados neste estudo, com objetivo de fornecer embasamento e compreensão sobre sistemas de controle de versão, as políticas de gerenciamento associadas e os desafios relacionados a conflitos de integração em código-fonte. Na Seção 2.1 é apresentado o conceito geral de sistemas de controle de versão e suas aplicações. A seguir, na Seção 2.2 é abordado as principais características dos sistemas de controle de versão centralizados, sua aplicação, além de suas vantagens e desvantagens. Na Seção 2.3 é explorado as características, vantagens, desvantagens e implicações dos sistemas de controle de versão distribuídos. Na Seção 2.4 é explicado as características das duas abordagens de políticas de gerenciamento de controle de versão e suas limitações, vantagens e desvantagens. Além disso, é explicado como elas moldam o fluxo de trabalho colaborativo e influenciam a eficiência do desenvolvimento de software. Na Seção 2.5 é abordado os tipos de conflitos de integração, além de explicar em que situações podem ocorrer estes conflitos. Na Seção 2.6 é abordado as duas principais técnicas de integração de código-fonte, demonstrando suas características, vantagens e desvantagens.

2.1 Sistemas de Controle de Versão (VCS)

O propósito de um sistema de controle de versão (VCS) é gerenciar e rastrear as alterações feitas no código-fonte de um projeto de software ao longo do tempo. Ele permite que várias pessoas trabalhem em um mesmo projeto simultaneamente, controlando e registrando as modificações feitas por cada colaborador. O VCS mantém um histórico completo de todas as versões anteriores do código-fonte, facilitando o rastreamento de alterações, a reversão a versões anteriores estáveis e a análise do progresso do projeto. Além disso, o VCS promove a colaboração eficiente, permitindo que os desenvolvedores compartilhem código-fonte, revisem alterações, resolvam conflitos e integrem suas contribuições. Ele também ajuda no rastreamento de bugs, associando problemas específicos a alterações no código-fonte. Em resumo, um VCS pode ser classificado como centralizado ou distribuído e é essencial para o desenvolvimento de software, fornecendo controle, gerenciamento e histórico das alterações no código-fonte. (ZOLKIFLI;

NGAH; DERAMAN, 2018; OTTE, 2009).

2.2 Sistemas de Controle de Versão Centralizados (CVCS)

Nos sistemas de controle de versão de centralizado, como, por exemplo, o Subversion (SVN), há um único repositório central onde todas as alterações são armazenadas. Este repositório é basicamente um servidor, o qual é o responsável por armazenar os arquivos, além de gerenciar usuários e permissões e fornecer um protocolo para que os clientes possam se comunicar. Os desenvolvedores, por meio de uma aplicação cliente, podem obter uma cópia desses arquivos do repositório central para trabalhar em suas próprias máquinas locais para realizar alterações, mas precisam se conectar ao repositório central para realizar operações de controle de versão como, criar ramos, comparar revisões, gravar alterações e ver histórico de alterações. Entre as principais desvantagens neste tipo de VCS, pode-se destacar a dependência de um servidor central e a complexidade da criação de ramos de desenvolvimento. (ZOLKIFLI; NGAH; DERAMAN, 2018; OTTE, 2009).

Este tipo de sistema de controle de versão oferece algumas vantagens notáveis. Primeiramente, sua estrutura simplificada e a natureza centralizada do repositório promovem uma compreensão intuitiva do fluxo de trabalho, facilitando a adoção deste tipo de VCS por equipes e desenvolvedores menos familiarizados. Além disso, ele geralmente proporciona um controle preciso de acesso, permitindo que administradores configurem permissões específicas para usuários ou grupos, promovendo a segurança do código-fonte. Também oferece auditoria e histórico detalhado das alterações no código-fonte, que simplificam a rastreabilidade sobre quem fez quais alterações e quando. Adicionalmente, a resolução de conflitos controlada, baseada no bloqueio de arquivos, minimiza conflitos diretos, tornando o processo de integração mais gerenciável.

No entanto, o modelo centralizado apresenta algumas desvantagens. A principal delas é a limitação na capacidade de desenvolvedores trabalharem simultaneamente, quando adotado a estratégia pessimista de bloqueio de arquivos para alteração, gerando possíveis gargalos e atrasos em equipes grandes. A dependência de um servidor central também é uma desvantagem, já que a inacessibilidade do servidor impede que os desenvolvedores realizem operações básicas, tornando o desenvolvimento vulnerável a interrupções. Além disso, a criação e gerenciamento de *branches* podem ser mais complexos, especialmente em comparação com sistemas distribuídos. Essas limitações podem afetar a escalabilidade e flexibilidade, especialmente em projetos que exigem colaboração intensiva e desenvolvimento simultâneo em diversas funcionalidades.

2.3 Sistemas de Controle de Versão Distribuídos (DVCS)

Em um sistema de controle de versão distribuído, como, por exemplo, o Git, não existe a necessidade de um servidor que contenha o repositório central. Todo desenvolvedor possuí em

seu computador local o repositório completo, com todo histórico de versões. No repositório local o desenvolvedor pode trabalhar totalmente offline, e realizar qualquer operação de controle de versão, como, criar *branches*, comparar revisões, gravar alterações e ver histórico de alterações. As funcionalidades disponíveis neste tipo de sistema são semelhantes aos sistemas centralizados, como, por exemplo, *commits*, criação de *branches* e *tags*, porém as operações acontecem no repositório local do desenvolvedor. Existem operações específicas para enviar as operações realizadas localmente para um repositório remoto ou que esteja em outro diretório. Os sistemas distribuídos são mais flexíveis, permitem um trabalho mais independente e possuem recursos avançados de integração e rastreamento de alterações. (ZOLKIFLI; NGAH; DERAMAN, 2018; OTTE, 2009).

Um dos maiores benefícios deste tipo de sistema de controle de versão é a capacidade de operar offline, permitindo que os desenvolvedores realizem *commits*, revisões e até mesmo operações de ramificação sem a necessidade de uma conexão constante com um servidor central. Isso não apenas aumenta a flexibilidade, mas também melhora a produtividade, especialmente em ambientes descentralizados ou quando a conectividade é intermitente. Além disso, a facilidade e rapidez com que ramos (*branches*) podem ser criados, integrados e descartados facilitam a experimentação e o desenvolvimento paralelo, promovendo uma abordagem mais ágil e adaptável ao ciclo de vida do projeto. A redundância de dados entre repositórios locais e o repositório central oferece uma camada adicional de segurança, tornando os dados menos propensos a perdas acidentais.

Por outro lado, os sistemas de controle de versão distribuídos também apresentam desafios. A complexidade do modelo distribuído pode tornar mais desafiadoras a compreensão do fluxo de trabalho e das operações para iniciantes. Além disso, a gestão de conflitos, embora mais eficaz em muitos casos, pode ser mais complexa em comparação com sistemas centralizados, especialmente quando ocorrem conflitos em áreas sensíveis do código-fonte. O modelo distribuído também pode aumentar a demanda por armazenamento, pois cada cópia local do repositório contém a história completa do projeto.

2.4 Políticas de Gerenciamento de Controle de Versão

Existem duas abordagens de políticas de gerenciamento de controle de versão: pessimista e otimista. Na abordagem pessimista, os desenvolvedores bloqueiam e reservam explicitamente um arquivo ou recurso antes de fazer qualquer modificação nele. Essa abordagem é baseada na premissa de que apenas uma pessoa pode trabalhar em um arquivo por vez, evitando assim conflitos diretos entre alterações concorrentes. Quando um desenvolvedor deseja modificar um arquivo, ele primeiro obtém uma permissão exclusiva para editar o arquivo e durante esse período, outros desenvolvedores não podem modificar o mesmo arquivo. Essa abordagem pode ser eficiente em equipes com poucos desenvolvedores, e é comumente usada em sistemas centralizados de controle de versão (CVCS), como o Concurrent Versions System (CVS), Subversion

(SVN) e o Team Foundation Version Control (TFVC), porém torna-se inadequada para equipes maiores, pois com o bloqueio dos arquivos o trabalho concorrente não é possível, necessitando que um desenvolvedor aguarde a conclusão do trabalho do colega.

A abordagem otimista é baseada na premissa de que conflitos podem ocorrer, mas podem ser resolvidos posteriormente, se necessário. No controle de versão otimista, os desenvolvedores podem trabalhar em cópias locais dos arquivos e realizar alterações sem impedimentos. Quando desejam enviar suas alterações para o repositório central, eles precisam sincronizar e mesclar (ou combinar) suas modificações com quaisquer alterações que outros desenvolvedores tenham feito no mesmo arquivo. Essa forma de gerenciamento se mostra mais adequada para equipes maiores, pois permite que os desenvolvedores trabalhem de forma concorrente e simultânea, inclusive alterando o mesmo arquivo. (MISTRíK et al., 2010).

2.5 Conflitos de Integração

Conflitos de integração ocorrem quando duas ou mais alterações conflitantes são feitas em um mesmo arquivo, ou conjunto de arquivos durante o processo de integração em um sistema de controle de versão. Esses conflitos surgem quando o sistema de controle de versão não consegue determinar automaticamente como combinar as alterações de forma coerente e podem ocorrer em diferentes situações, como:

- **Modificações conflitantes:** Quando duas ou mais pessoas fazem alterações em partes do mesmo arquivo que se sobrepõem ou afetam as mesmas linhas ou trechos do código-fonte.
- **Exclusões conflitantes:** Se um desenvolvedor exclui um arquivo ou uma linha de código-fonte enquanto outro desenvolvedor faz modificações no mesmo arquivo ou na mesma linha.
- **Movimentação ou renomeação conflitante:** Se um desenvolvedor move ou renomeia um arquivo ou diretório enquanto outro desenvolvedor está modificando o mesmo arquivo ou diretório.

Quando uma das situações acima ocorre, o sistema de controle de versão geralmente sinaliza esse conflito e solicita que o desenvolvedor resolva manualmente. Para a resolução dos conflitos de integração, os desenvolvedores podem usar ferramentas e recursos fornecidos pelo sistema de controle de versão, como editores de código-fonte integrados ou externos, para analisar as diferenças, selecionar as versões corretas das alterações e ajustar o código-fonte conforme necessário.

A ocorrência de conflitos de integração acontece de forma comum em projetos colaborativos que utilizam a abordagem otimista de gerenciamento de controle de versão, pois muitos desenvolvedores trabalham simultaneamente no mesmo projeto, alterando com frequência os mesmos arquivos de código-fonte. (VALE et al., 2021; CARBONERA et al., 2023; GRAEFF; FARIAS; CARBONERA, 2023; DALCIN et al., 2023).

2.6 Técnicas de Integração

As técnicas de integração de código-fonte desempenham um papel fundamental na gestão eficiente do desenvolvimento colaborativo de software, permitindo que múltiplos desenvolvedores contribuam simultaneamente para um projeto sem comprometer a integridade do código-fonte. Duas abordagens proeminentes nesse contexto são a Two-way Merge, que compara e combina duas versões de um arquivo, e a Three-way Merge, que envolve a comparação de três versões do mesmo arquivo. Essas técnicas são cruciais para reconciliar alterações concorrentes, evitar conflitos e manter a consistência do código-fonte em sistemas de controle de versão. Neste contexto, explorar essas técnicas oferece percepções valiosas sobre as estratégias que facilitam a colaboração eficaz em projetos de software.

- **Two-way Merge (Integração de duas vias):** Nesta técnica, apenas duas versões são consideradas durante o processo de integração. É realizado a comparação entre a versão modificada do arquivo (branch de desenvolvimento) com a versão original do arquivo (branch de origem ou base), sem depender de uma versão ancestral comum do qual ambas as versões se originaram. Deste modo, quando há um conflito durante a integração, normalmente é necessário que o desenvolvedor resolva manualmente o conflito, escolhendo uma versão a ser mantida ou combinando as alterações de forma adequada. Essa abordagem é mais comum em sistemas de controle de versão centralizados, como o Concurrent Versions System (CVS) e o Subversion (SVN).

- **Three-way Merge (Integração de três vias):** Nesta técnica, três versões diferentes do mesmo arquivo são consideradas durante o processo de integração. É realizado a comparação da versão modificada do arquivo (branch de desenvolvimento) com a versão original do arquivo (branch de origem ou base) e também com a versão ancestral comum dos dois arquivos. Essa abordagem é amplamente utilizada em sistemas de controle de versão distribuídos, como o Git, que suportam fluxos de trabalho mais complexos e envolvem várias ramificações.

Resumidamente, a principal diferença entre as duas técnicas é a quantidade de versões consideradas durante o processo de integração. Three-way Merge é considerado mais avançado e geralmente capaz de lidar com uma variedade maior de cenários de integração e conflitos. Consequentemente, quase todas as ferramentas de integração atualmente disponíveis fazem uso da integração de Three-way Merge. (MENS, 2002).

3 TRABALHOS RELACIONADOS

Esta seção busca apresentar a contextualização do presente estudo empírico, fornecendo uma visão abrangente das pesquisas anteriores que exploram temas semelhantes. Busca-se destacar as contribuições significativas de estudos anteriores, identifica lacunas no conhecimento

existente e posiciona este trabalho no contexto mais amplo da área de pesquisa. Ao examinar as investigações prévias, busca-se não apenas compreender o estado atual do tema, mas também identificar oportunidades para avançar o conhecimento e desenvolver abordagens inovadoras.

A Tabela 1 apresenta os critérios utilizados na pesquisa dos trabalhos relacionados.

Tabela 1 – Critérios de pesquisa de trabalhos relacionados

Termos pesquisados	merging merge effort version control system
Publicações	Periódicos, conferências, revistas científicas, dissertações de mestrado e teses de doutorado
Período de análise	A partir de 2010
Banco de dados científicos	Arvix, Google Scholar, IEEExplore, ACM e Springer

3.1 Aprensentação dos Trabalhos Relacionados

(MEHDI; URSO; CHAROY, 2014) propõem uma metodologia para medir o esforço necessário para utilizar o resultado de uma ferramenta de integração de código-fonte no desenvolvimento de software em larga escala. Os pesquisadores desenvolveram uma ferramenta para calcular automaticamente essa medida, analisando histórico de desenvolvimento de código aberto disponíveis publicamente. Foi avaliado a qualidade dos algoritmos de integração comparando os resultados automatizados com os resultados de integração aprovados pelos desenvolvedores. Por meio da análise de seis repositórios de código aberto, que abrangem mais de 2,5 milhões de linhas de código, é apresentado resultados significativos de comparação entre algoritmos de integração, mostrando como esses resultados podem ser empregados para aprimorar ainda mais esses algoritmos.

(BRINDESCU et al., 2014) apresenta um estudo empírico aprofundado e em grande escala que analisa a influência do Sistemas Distribuídos de Controle de Versão (DVCS) nas mudanças de software. Participaram da pesquisa 820 pessoas, sendo 85% desenvolvedores da indústria, com objetivo esclarecer a prática do uso do DVCS. Além disso, para obter mais informações sobre como o DVCS afeta as alterações de código-fonte, foram analisados 409 milhões de linhas de alterações de código de 358.300 *commits*, feitos por 5.890 desenvolvedores em 132 repositórios, contendo um total de 73 milhões de linhas de código. As descobertas deste estudo indicam que o uso de DVCS leva os desenvolvedores realizarem *commits* menores e mais frequente em comparação com os desenvolvedores que utilizam Sistemas Centralizados de Controle de Versão (CVCS). O estudo também explora a influência do tamanho da equipe e da idade do projeto no uso do VCS e nas práticas de *commit*.

(SANTOS; MURTA, 2012) propõem a extração de métricas destinadas a avaliar a complexidade envolvida na fusão de diferentes ramos de desenvolvimento, permitindo a identificação dos ramos mais críticos e o acompanhamento da evolução dessas métricas desde a criação do ramo. Este estudo revela que determinadas métricas apresentaram um desempenho superior na

estimativa da complexidade da junção de ramos, destacando-se a métrica da "Quantidade de Conflitos Físicos", que demonstrou uma correlação de até 99% com o esforço real necessário para a junção.

(OLIVEIRA et al., 2023) investiga como as refatorações de código-fonte influenciam o esforço envolvido na fusão de código-fonte durante o desenvolvimento colaborativo de software. Foi analisado a relação entre a ocorrência e a quantidade de refatorações nas ramificações em um *commit* de integração e sua influência nas chances e na intensidade do esforço de integração. Os experimentos abrangeram 28 projetos populares de código aberto, revelando que as refatorações aumentam tanto as chances quanto a intensidade do esforço de integração. Isso sugere a necessidade de considerar mudanças comportamentais na aplicação de refatorações e melhorias nas ferramentas de suporte à integração de código-fonte, juntamente com recomendações para refatorações.

(BRINDESCU et al., 2020) propõem um estudo empírico abordando os tipos, a frequência e as implicações dos conflitos de integração, sendo o impacto avaliado em termos de *commits* de correção de *bugs* relacionados a esses conflitos. A pesquisa incluiu a análise de 143 projetos de código aberto, revelando que quase 1 em cada 5 integração resulta em conflitos. Em 75,23% desses cenários, os desenvolvedores tiveram que analisar a lógica do programa para a sua resolução. Além disso, foi constatado que o código-fonte associado a conflitos de integração tem o dobro de probabilidade de conter um *bug*. Por fim, quando a intervenção manual é necessária para tratar dos conflitos de integração, a probabilidade de o código apresentar um *bug* aumenta em 26 vezes.

3.2 Análise Comparativa dos Trabalhos Relacionados

Foram definidos cinco Critérios de Comparação (CC) para realizar a análise de similaridades e diferenças entre o estudo proposto e os trabalhos selecionados. Esta comparação é importante para auxiliar na identificação de oportunidades de pesquisa utilizando critérios objetivos. Os critérios são descritos a seguir:

- **Estudo empírico (CC1):** é uma pesquisa baseada em evidências concretas e observáveis coletadas por meio de métodos experimentais, observacionais ou analíticos, visando a análise prática de fenômenos na realidade.

- **Conflitos de integração (CC2):** o estudo aborda e analisa ou explora os conflitos de integração de código-fonte.

- **Comparação entre sistemas de controle de versão centralizados e distribuídos (CC3):** o estudo aborda, sob algum aspecto, uma comparação entre os sistemas de controle de versão centralizados e distribuídos.

- **Análise do esforço para resolução de conflitos de integração (CC4):** o estudo analisa, sob algum aspecto, o esforço para resolução de conflitos de integração.

- **Testes estatísticos (CC5):** o estudo aplica testes estatísticos para fazer inferências ou generalizações sobre os dados coletados.

Oportunidades de pesquisa. A Tabela 2 apresenta a comparação dos estudos selecionados, com relação aos critérios de comparação estipulados. Com base na tabela de comparação é possível inferir que: (1) nenhum dos trabalhos relacionados contempla todos os critérios de comparação, ou seja, não existe nos estudos relacionados outro que tenha a mesma proposta apresentada neste estudo; (2) todos trabalhos selecionados são estudos empíricos; (3) o CC2 é atendido total ou parcialmente em 4 dos cinco trabalhos relacionados, mostrando a relevância do tema Conflitos de Integração e como ele é abordado sob aspectos diferentes; (4) o CC3 é atendido em somente 1 trabalho, evidenciando uma possível lacuna de estudos sobre o tema; (5) o CC4 é atendido total ou parcialmente em 4 dos 5 estudos, evidenciando a relevância do tema; (6) o CC6 é atendido total ou parcialmente em todos os trabalhos, evidenciando a importância da análise estatística para obtenção de resultados relevantes e confiáveis.

Portanto, com base na análise desta tabela comparativa, pode-se concluir que, de forma geral, os estudos relacionados se conectam diretamente com o tema deste estudo. Também é possível inferir a necessidade de estudos que explorem a comparação entre VCS centralizado e distribuído no processo de desenvolvimento de software.

Tabela 2 – Análise comparativa dos Trabalhos Relacionados selecionados

Trabalho Relacionado	**Critério de Comparação**				
	CC1	**CC2**	**CC3**	**CC4**	**CC5**
Trabalho Proposto	●	●	●	●	●
(MEHDI; URSO; CHAROY, 2014)	●	●	○	●	●
(BRINDESCU et al., 2014)	●	○	●	○	◐
(SANTOS; MURTA, 2012)	●	◐	○	●	◐
(OLIVEIRA et al., 2023)	●	◐	○	●	◐
(BRINDESCU et al., 2020)	●	●	○	◐	●

Legenda: ● Atende Completamente ◐ Atende Parcialmente ○ Não Atende

4 METODOLOGIA DO ESTUDO

Nesta seção é apresentado as principais decisões relacionadas ao desenho experimental de nossa investigação empírica. Inicia-se em Objetivos e Questões da Pesquisa (Seção 4.1), onde é estabelecido o propósito do estudo, assim como as questões da pesquisa. A seguir é apresentado a Formulação de Hipóteses (Seção 4.2), onde será proposto as hipóteses que serão utilizadas

como base para aplicação dos métodos estatísticos. Em Seleção de Variáveis e Método de Quantificação (Seção 4.3), é definido a variável independente e as variáveis dependentes que fornecerá os parâmetros principais utilizados para realização deste estudo. A seguir, em Projetos Alvo (Seção 4.4), é descrito os critérios e motivações para escolha dos projetos analisados neste estudo, além de relacionar os projetos selecionados com seus dados básicos. Em Processo Experimental (Seção 4.5) busca-se descrever resumidamente as etapas realizadas ao decorrer deste estudo. Por último, em Procedimento de Análise (Seção 4.6), é descrito os procedimentos executados para analisar os projetos selecionados e obter os resultados esperados do estudo.

4.1 Objetivos e Questões da Pesquisa

Nesta seção é estabelecido o propósito deste estudo, seguindo o modelo GQM (WOHLIN et al., 2012), da seguinte maneira: investigar os efeitos dos sistemas de controle de versão em relação ao esforço e à eficácia, sob a perspectiva dos desenvolvedores, no contexto da evolução do código-fonte. Especificamente, o foco reside na avaliação do impacto dos sistemas de controle de versão (distribuídos em comparação a centralizados) no esforço dos desenvolvedores e na eficácia da integração do código-fonte na saída. Portanto, contra-se em duas Questões de Pesquisa (QP):

- **QP 1:** Qual é o esforço relativo de mesclar códigos-fonte usando um sistema de controle de versão distribuído em relação ao centralizado?
- **QP 2:** O sistema de controle de versão distribuído é mais eficaz que o centralizado para mesclar código-fonte conflitante?

4.2 Formulação de Hipóteses

Hipótese 1. Sugere-se a hipótese de que, embora o sistema de controle de versão distribuído ofereça um processo de trabalho não linear e uma abordagem mais sistemática para mesclar o código-fonte, sua aplicação em projetos de software realistas não demonstra eficácia satisfatória na redução do esforço demandado pelos desenvolvedores para realizar a integração de dois conjuntos de código-fonte. É possível que os desenvolvedores tenham que dedicar um esforço consideravelmente maior para lidar com conflitos de integração mais complexos. Com base nessa alegação, é estabelecido as hipóteses nula e alternativa da seguinte maneira:

Hipótese Nula 1, H_{1-0}**:** o sistema de controle de versão distribuído requer menos (ou igual) esforço do que o centralizado para produzir o código-fonte integrado de saída desejado.

H_{1-0}**:** Esforço(C_A,C_B)$_{Distribudo} \leq$ Esforço(C_A,C_B)$_{Centralizado}$

Hipótese Alternativa 1, H_{1-1}**:** o sistema de controle de versão distribuído requer mais esforço do que o centralizado para produzir o código-fonte integrado de saída desejado.

H_{1-1}**:** Esforço(C_A,C_B)$_{Distribudo} >$ Esforço(C_A,C_B)$_{Centralizado}$

Ao examinar essa primeira hipótese, é possível determinar se o tipo de sistema de controle de versão desempenha um papel fundamental na redução do esforço exigido pelos desenvolvedores para gerar o código-fonte desejado, fornecendo evidências empíricas sobre como essas abordagens reconciliam trechos conflitantes do código-fonte. Ao compreender o esforço necessário para mesclar o código-fonte, os desenvolvedores podem fazer uma escolha embasada em dados empíricos ao invés de confiar apenas em intuição ou opiniões especializadas.

Hipótese 2. Conforme descrito anteriormente, os desenvolvedores enfrentam a tarefa de resolver conflitos e, quando esses conflitos são numerosos, tendem a investir mais esforço na integração dos trechos de código em conflito. É importante observar que, se o esforço exigido for significativamente alto, pode-se questionar a escolha de um VCS específico. Uma das principais vantagens de utilizar sistemas de controle de versão distribuídos é a capacidade de evitar e lidar de forma mais adequada com conflitos complexos durante a integração. Isso ocorre, em parte, devido ao fluxo de trabalho não linear desses sistemas, que tende a aproximar o código-fonte local do código-fonte desejado após a integração, bem como devido à eficácia do mecanismo de integração incorporado. Além disso, supõe-se também que permitir que os desenvolvedores trabalhem com maior liberdade em suas versões locais e em mais de ramificações aumente a probabilidade de mesclar os conjuntos de código-fonte.

No entanto, não está claro se, de fato, promover um maior paralelismo auxilia os desenvolvedores a produzir código-fonte integrado mais próximo do código-fonte desejado. Por exemplo, os desenvolvedores podem enfrentar maiores dificuldades ao mesclar código-fonte proveniente de diferentes ramificações, resultando em uma menor capacidade do sistema de controle de versão distribuído em mesclar corretamente mais de conjuntos de código-fonte quando comparado a um sistema centralizado. A hipótese é que o sistema de controle de versão centralizado é mais eficaz na produção de código-fonte integrado que se aproxima do código-fonte desejado em comparação a um sistema distribuído.

Levando isso em consideração, a segunda hipótese investiga se o sistema de controle de versão distribuído de fato auxilia os desenvolvedores a melhorar a eficácia da integração de código-fonte. Deste modo, é formulado as hipóteses nula e alternativa da seguinte forma:

Hipótese Nula 2, H_{2-0}**:** o sistema de controle de versão centralizado é significativamente menos eficaz do que o distribuído para mesclar o código-fonte.

H_{2-0}**:** Eficácia$(C_{integrado})_{Centralizado} \leq$ Eficácia$(C_{integrado})_{Distribudo}$

Hipótese Alternativa 2, H_{2-1}**:** o sistema de controle de versão centralizado é significativamente mais eficaz do que o distribuído para mesclar o código-fonte.

H_{2-1}**:** Eficácia$(C_{integrado})_{Centralizado} >$ Eficácia$(C_{integrado})_{Distribudo}$

4.3 Seleção de variáveis e Método de Quantificação

Neste estudo, a variável independente é a escolha da técnica de controle de versão, a qual é uma variável nominal com dois valores possíveis: Distribuído e Centralizado. Essas variáveis representam os tratamentos que são investigados e seu impacto nas variáveis dependentes a seguir.

Effort. Refere-se ao número de operações necessárias para transformar o código-fonte integrado de saída no código-fonte desejado. O esforço é quantificado usando L($C_{integrado}$)$_{i,v}$, que representa o conjunto de linhas presentes em um determinado código-fonte integrado de saída *i*, produzido por uma técnica *v*, e L($C_{integrado}$)$_f$, que representa o conjunto de linhas presentes no código-fonte desejado de *i*. Esta variável é examinada na primeira hipótese.

Precision. É a relação entre o número de linhas de código no código-fonte desejado de saída ($C_{desejado}$) e o número total de linhas no código-fonte integrado de saída ($C_{integrado}$). A Equação 1 é usada para calcular essa métrica.

$$Precision_{i,v} = \frac{|L(C_{integrado})_{i,v} \cap L(C_{desejado})_i|}{|L(C_{integrado})_{i,v}|} \tag{1}$$

Recall. É a relação entre o número de linhas de código encontradas no código-fonte de saída desejada e o número total de linhas de código no código-fonte integrado de saída. A Equação 2 calcula essa métrica.

$$Recall_{i,v} = \frac{|L(C_{integrado})_{i,v} \cap L(C_{desejado})_i|}{|L(C_{desejado})_i|} \tag{2}$$

Accuracy. É a razão entre o número de linhas de código encontradas no código-fonte de saída desejada e o número total de linhas de código encontradas no código-fonte de saída desejada mais aquelas encontradas apenas no código-fonte integrado de saída. A Equação 3 calcula essa métrica.

$$Accuracy_{i,v} = \frac{|L(C_{desejado})_i|}{|L(C_{desejado})_i| + |L(C_{desejado})_i - L(C_{integrado})_{i,v}| + |L(C_{integrado})_{i,v} - L(C_{desejado})_i|} \tag{3}$$

F-measure. É combinado as medidas de *precision* e *recall* em uma média harmônica, chamada F-measure (ou F-score), para calcular a eficácia da variável. A Equação 4 calcula esse indicador.

$$F - measure_{i,v} = 2 * \frac{precision_{i,v} * recall_{i,v}}{precision_{i,v} + recall_{i,v}} \tag{4}$$

4.4 Projetos Alvo

O GitHub é uma plataforma em nuvem, criada para hospedagem de repositórios de código-fonte e desenvolvimento colaborativo entre desenvolvedores de software, que utiliza o Git como sistema de controle de versão. O GitHub permite que os desenvolvedores e empresas armazenem seus repositórios de código-fonte e oferece recursos para facilitar o trabalho colaborativo entre estes desenvolvedores, além de fornecer ferramentas como as de rastreamento de erros, revisão de código-fonte e integração contínua. Atualmente o GitHub é utilizado por mais de 100 milhões de desenvolvedores, mais de 4 milhões de organizações, além de hospedar mais de 330 milhões de repositórios de projetos. (GITHUB, 2023).

Inicialmente busca-se descobrir as linguagens de programação mais populares no momento da realização deste estudo. Para obter esta informação é utilizado como referência o TIOBE Index, que é uma classificação criada pela organização TIOBE, que mensalmente utiliza uma abordagem estatística para classificar as linguagens de programação com base em sua popularidade relativa no mundo da programação. Foram selecionadas as 5 linguagens de programação mais bem colocadas na classificação do mês de junho de 2023: Python, C, C++, Java, e Javascript/TypeScript. (TIOBE, 2023).

Foi utilizado o recurso de pesquisada avançada do GitHub para selecionar os 50 projetos utilizados neste experimento, que estão relacionados na Tabela 5 (Apêndice A). Na busca destes repositórios foram utilizados como principais critérios que o repositório tivesse ao menos 5 mil estrelas, no mínimo 100 contribuidores (desenvolvedores), tenha recebido alguma atualização de código-fonte ainda em 2023 e tenha como principal linguagem de programação alguma das selecionadas anteriormente. Além disso, foram descartados repositórios que não estivessem no idioma inglês e também repositórios de aprendizado, que não constituem projetos de software com aplicação real. Também foi necessário descartar, por questão de limitação técnica, repositórios muito grandes que tinham mais de 500MB ou que possuíam mais de 70 mil *commits* no ramo principal.

4.5 Processo Experimental

A Figura 1 apresenta o processo experimental descrevendo as três fases principais realizadas para coleta e analise dos dados de interesse deste estudo, detalhando cada etapa realizada em cada uma das fases. O objetivo desta seção é facilitar a compreensão da abordagem adotada, proporcionando uma visão clara sobre como o experimento foi conduzido, garantindo a replicabilidade e validade dos resultados obtidos.

- **Fase 1:** Seleção dos Projetos e Desenvolvimento da Ferramenta

  Primeira etapa. Realizado a busca e download dos 50 projetos selecionados, conforme os critérios descritos na subseção 4.4.

Segunda etapa. Desenvolver a ferramenta [3] responsável por ler cada conflito de integração do projeto original e replica-lo em um projeto cópia, utilizando Git e SVN.

- **Fase 2:** Execução da Ferramenta, Coleta e Agrupamento dos Resultados

  Primeira etapa. Consiste em executar a ferramenta criada para cada um dos repositórios, com o objetivo de recriar os cenários de conflito de integração ocorridos no projeto original. A ferramenta recebe 3 parâmetros como entrada:

  - **Parâmetro 1:** Diretório onde pode encontrar os projetos Git que serão utilizados como base para replicação dos casos de conflito.
  - **Parâmetro 2:** Tipo de replicação que será realizado, podendo ser "svn"ou "git".
  - **Parâmetro 3:** Diretório de saída, onde será criado o projeto espelho com a replicação dos casos de conflito de integração, além do arquivo csv contendo o total de linhas, linhas adicionadas e linhas removidas, para cada conflito reproduzido.

  A aplicação inicia buscando o histórico de *commits* na branch HEAD da aplicação, ordenando de forma crescente por data de efetivação da alteração. É percorrido os *commits* do repositório original seguindo a ordenação, ou seja, do mais antigo para o mais atual. Para cada alteração, verifica se é um *commit* de integração, analisando se possui mais de um *commit* pai, e lida com eles criando ramos temporários para cada *commit* pai, realizando a integração correspondente no repositório espelho. Após a integração ser realizada, uma comparação com a versão final do repositório original é realizada, fornecendo as informações necessárias de quantas linhas precisam ser adicionas e removidas para se chegar no código-fonte desejado. O número de linhas adicionadas e removidas é basicamente o que é considerado como esforço de integração. Ao final, um arquivo csv é gerado para cada projeto processado, contendo o nome do arquivo de código-fonte, total de linhas, linhas adicionadas e linhas removidas. Cada projeto processado terá um arquivo csv de saída para cada sistema de controle de versão (SVN e Git).

  Segunda etapa. Responsável por agrupar, para cada projeto, os dois arquivos csv (Git e SVN) em um único arquivo, já pareando cada registro de conflito integração existente nos 2 arquivos [4]. São considerados apenas os conflitos que tenham ocorrido em ambos os sistemas de controle de versão.

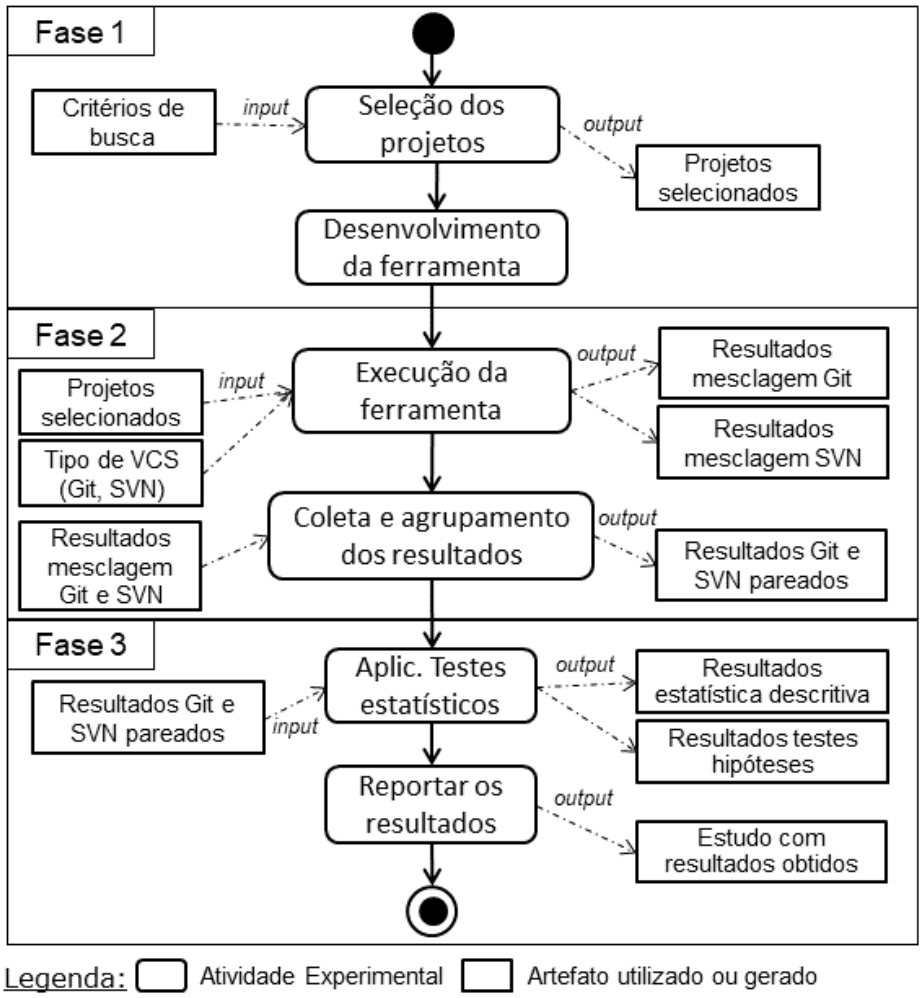
- **Fase 3:** Aplicação dos Testes Estatísticos e Reportar os Resultados

  Primeira etapa. É aplicado os testes estatísticos necessários para compreensão dos dados resultantes e validação das hipóteses formuladas, conforme descrito na subseção 4.6.

  Segunda etapa. É consolidado os resultados obtidos da análise dos testes estatísticos neste estudo.

[3] Ferramenta disponível em: https://github.com/viniville/tcc-app-replicador
[4] Ferramenta disponível em: https://github.com/viniville/tcc-app-leitura-agrup-arq-resultados

Figura 1 – Processo experimental



Fonte: Desenvolvido pelo autor

4.6 Procedimento de Análise

Os 50 projetos selecionados, que estão relacionados na Tabela 5, constituem um cenário consistente para testar as hipóteses previamente formuladas. Desta maneira, foi realizado a estatística descritiva destes projetos com o propósito de examinar sua distribuição normal e inferência estatística, permitindo testar as hipóteses mencionadas anteriormente. O nível de significância dos testes de hipóteses adotados foi de 0,05. Foram analisados os dados coletados para testar as hipóteses de cada projeto alvo individualmente, bem como de todos os projetos juntos. Para testar H1, foi aplicado o Wilcoxon e o t-test pareado. Também foi aplicado estes métodos para testar a segunda hipótese (e suas sub-hipóteses).

Estatística descritiva. É a análise e interpretação de um conjunto de dados, fornecendo ferramentas essenciais para resumir, organizar e compreender informações complexas de maneira mais simples. Seu propósito principal é extrair significado e percepções de conjuntos de

dados, nos ajudando a compreender e descrever, além de oferecer uma visão clara das características principais, tendências e variabilidades presentes. A estatística descritiva utiliza medidas de tendência central, como a média e a mediana, juntamente com medidas de dispersão, como o desvio padrão, para proporcionar uma compreensão abrangente da distribuição dos dados. Além disso, a apresentação gráfica, por meio de histogramas, gráficos de barras e *boxplots*, acrescenta uma dimensão visual que facilita a comunicação e a interpretação dos resultados. (CONNER; JOHNSON, 2017).

Teste dos Postos Sinalizados de Wilcoxon. É uma análise estatística não paramétrica usada para avaliar se há diferenças significativas entre dois conjuntos de dados relacionados. É um teste útil quando os dados não atendem aos pressupostos necessários para testes paramétricos, particularmente quando os dados não seguem uma distribuição normal. Consiste em comparar as disparidades entre pares de observação, classifica essas diferenças e utiliza os *ranks* para calcular uma estatística de teste. O propósito deste teste é determinar se as discrepâncias médias entre os pares de dados são diferentes de zero, sendo comumente empregado em situações onde as variações individuais têm relevância. A hipótese nula do teste é que não há diferença entre as duas amostras, enquanto a hipótese alternativa é que há diferença. (TAHERI; HESAMIAN, 2013).

Teste t pareado. É uma técnica estatística amplamente utilizada para comparar duas condições ou medidas emparelhadas em uma amostra. Este teste é particularmente aplicável quando cada sujeito ou elemento da amostra é submetido a duas medições, como em experimentos onde os pares de observações são naturalmente emparelhados. A principal característica do teste t pareado é sua sensibilidade à diferença nas médias das observações pareadas, calculando a média das diferenças e testando se essa média é significativamente diferente de zero. Essa abordagem considera a dependência entre as observações, sendo particularmente útil quando os dados exibem uma distribuição normal.

5 RESULTADOS DO ESTUDO

Esta seção apresenta as principais decisões subjacentes ao desenho experimental desta investigação empírica. Esta seção examina os dados coletados com relação aos efeitos dos sistemas de controle de versão no esforço que os desenvolvedores investem para resolver conflitos de integração. Primeiramente é como se dá a distribuição dos dados coletados, para depois testar as hipóteses formuladas.

5.1 H1: Esforço e Sistema de Controle de Versão

Estatísticas descritivas. Esta seção analisa os dados coletados com relação ao impacto do sistema de controle de versão no esforço que os desenvolvedores investem para resolver conflitos de integração. Para isso, é calculado estatísticas descritivas para compreender a distribuição

dos dados, incluindo suas principais tendências e dispersão. A Tabela 3 apresenta as estatísticas descritivas do esforço de resolução de conflitos. O esforço geral (primeira linha da tabela) é calculado com base em 116.388 casos de integração, que corresponde a soma dos casos de integração de todos os projetos analisados.

Dos 50 projetos analisados, 37 deles (74%) apresentaram maior esforço de integração para resolver conflitos (ou seja, número de operações para transformar o código-fonte integrado no código-fonte desejado) quando utilizado o sistema de controle de versão centralizado (SVN). Análise geral dos projetos (Tabela 3, linha 1), permite extrair algumas informações importantes: (1) a análise da mediana sugere que a utilização do VCS centralizado tem uma tendência ligeiramente maior de requerer mais operações, e os quartis são consistentes entre os dois tipos de VCS, (2) a análise da média também indica que o VCS centralizado pode exigir mais operações para produzir o código-fonte desejado, (3) tanto o VCS centralizado quanto no distribuído tem um desvio padrão semelhante, indicando uma dispersão comparável dos dados ao redor da média, porém estes valores são muito elevados, indicando que existem dados que se desviam significativamente do restante do conjunto de dados.

Esses resultados sugerem que, apesar da média relativamente baixa de 86,8 e 76,6, a presença de um desvio padrão significativo indica uma dispersão considerável dos dados. A mediana, sendo menor que a média, pode indicar uma possível assimetria à direita na distribuição. A análise em conjunto dos dados estatísticos confirma a distribuição assimétrica e altamente dispersa dos dados, onde a média pode não ser uma medida representativa dos dados devido à influência de valores extremos.

Tabela 3: Estatística descritiva do esforço de resolução de conflitos.

Projeto	SCV	N	Min	1Q	Med.	Mean	3Q	Max	St.D.
Geral	Cent	116.388	0	3,00	9,00	86,80	33,00	441.721	2191,8988
	Distr	116.388	0	3,00	8,00	76,60	29,00	400.149	2139,2452
Ant-design	Cent	7.747	0	4,00	10,00	317,30	30,00	441.721	8154,7273
	Distr	7.747	1	3,00	9,00	338,00	29,00	400.149	8094,5720
Axios	Cent	120	1	4,00	16,00	35,08	39,75	293	53,1263
	Distr	120	1	3,00	6,00	20,56	18,00	296	40,4194
Bitcoin	Cent	13.148	1	3,00	9,00	41,96	29,00	3.797	130,4670
	Distr	13.148	1	3,00	9,00	31,99	26,00	3.404	96,7429
Bootstrap	Cent	4.667	1	2,00	8,00	110,90	33,00	18.567	582,8999
	Distr	4.667	1	2,00	6,00	103,50	29,00	9.857	508,7940
Caffe	Cent	238	1	4,00	8,00	25,31	21,75	425	49,3851
	Distr	238	1	3,00	6,00	22,61	21,00	425	47,0226
Code-server	Cent	6.480	0	4,00	12,00	65,99	42,00	19.225	447,4595
	Distr	6.480	0	4,00	12,00	61,66	42,00	15.766	335,6403
Core	Cent	987	0	2,00	6,00	24,25	21,50	971	63,2755
	Distr	987	0	2,00	5,00	21,72	17,00	1.950	80,2242
Django	Cent	275	1	6,00	13,00	35,08	32,00	610	66,1476
	Distr	275	1	4,00	8,00	18,55	17,50	215	30,7870
Dubbo	Cent	1.967	0	4,00	11,00	31,44	30,00	1.385	68,9180
	Distr	1.967	1	4,00	9,00	41,82	29,00	2.286	144,3642
Electron	Cent	2.443	1	3,00	7,00	20,88	19,00	1.107	50,1079
	Distr	2.443	1	3,00	7,00	19,68	17,00	1.196	58,9788

Continua...

Projeto	SCV	N	Min	1Q	Med.	Mean	3Q	Max	St.D.
Emscripten	Cent	1.712	1	5,00	17,00	95,04	62,00	15.103	480,3206
	Distr	1.712	1	4,00	10,00	51,09	36,00	3.522	178,6094
Faceswap	Cent	39	2	8,00	23,00	51,00	54,00	499	101,3093
	Distr	39	2	5,00	8,00	22,41	25,50	99	26,7500
Flask	Cent	624	1	3,00	8,00	44,97	36,00	1.451	121,0513
	Distr	624	1	3,00	8,00	52,31	38,00	1.459	147,9481
Googletest	Cent	904	1	4,00	8,00	36,51	24,25	3.234	141,2769
	Distr	904	1	3,00	6,00	27,14	18,00	3.430	139,3715
Hashcat	Cent	2.003	1	2,00	8,00	52,23	21,00	2.968	204,0160
	Distr	2.003	1	1,00	4,00	31,28	19,00	1.915	136,8045
Hibernate-orm	Cent	1.064	1	6,00	20,00	108,89	70,25	3.520	309,6747
	Distr	1.064	1	6,00	24,50	136,40	94,00	3.530	361,2837
Jq	Cent	48	1	4,00	12,00	51,02	41,50	428	104,5958
	Distr	48	1	2,00	7,50	23,96	21,00	342	61,3514
Jquery	Cent	247	1	7,00	25,00	66,25	67,50	698	106,1431
	Distr	247	1	5,00	14,00	35,22	37,50	666	63,7719
Json	Cent	829	1	6,00	25,00	326,50	116,00	22.507	1212,5489
	Distr	829	1	6,00	24,00	230,50	117,00	7.409	727,4909
Junit5	Cent	76	1	4,00	11,00	31,33	35,50	173	42,8400
	Distr	76	1	4,00	11,00	32,91	30,50	413	59,2349
Keras	Cent	866	1	4,00	14,00	75,49	40,75	6.418	362,8243
	Distr	866	1	3,00	9,00	49,02	32,00	2.031	161,0542
Libuv	Cent	290	1	5,00	18,50	105,50	82,00	1.747	243,4071
	Distr	290	1	5,00	24,00	119,30	112,50	1.753	249,0175
Lombok	Cent	384	1	5,00	19,00	43,48	47,00	529	71,3201
	Distr	384	1	4,00	10,00	24,14	23,25	331	44,3104
Magisk	Cent	26	1	6,50	25,00	57,35	53,75	456	94,9619
	Distr	26	1	5,25	16,50	42,35	48,75	212	55,9821
Mockito	Cent	262	0	3,00	6,00	20,66	20,00	330	40,4177
	Distr	262	1	4,00	8,00	28,20	8,00	4.593	283,6243
Nest	Cent	15.194	0	3,00	6,00	158,90	31,00	62.292	1538,4769
	Distr	15.194	0	2,00	6,00	100,80	20,00	34.689	1129,3622
Netdata	Cent	260	1	1,00	3,00	20,50	17,00	372	43,7594
	Distr	260	1	2,00	4,00	24,49	16,00	413	54,8177
Netty	Cent	58	1	3,00	12,00	55,33	32,50	699	141,2181
	Distr	58	1	2,25	8,00	39,24	21,00	707	123,0258
Obs-studio	Cent	482	1	4,00	16,00	54,30	44,00	1.688	133,1683
	Distr	482	1	3,00	8,00	26,16	23,00	846	65,1346
Stb	Cent	304	1	7,75	31,50	104,18	115,25	2.230	217,3577
	Distr	304	1	4,00	9,00	49,11	37,00	1.364	131,7685
Pandas	Cent	1.931	1	4,00	14,00	57,78	40,50	7.036	264,7749
	Distr	1.931	1	3,00	9,00	33,50	28,00	1.593	106,9234
Protobuf	Cent	7.376	0	3,00	9,00	66,29	32,00	13.167	357,9333
	Distr	7.376	0	3,00	9,00	63,65	30,00	11.411	313,0890
React	Cent	723	1	2,00	8,00	55,91	27,00	9.634	470,2434
	Distr	723	1	2,00	8,00	65,22	22,00	9.742	599,8838
Redis	Cent	1.390	1	6,00	24,00	108,40	76,00	4.937	331,5377
	Distr	1.390	1	3,00	11,00	65,21	39,00	3.735	224,2479
Redux	Cent	174	1	3,00	8,50	35,01	16,00	3.279	250,6172
	Distr	174	1	2,00	6,50	30,23	14,75	3.183	240,9535
Requests	Cent	478	1	4,00	9,00	30,94	24,75	2.106	113,4223
	Distr	478	1	3,00	8,00	19,86	18,75	459	39,8580
Retrofit	Cent	78	1	2,00	5,00	25,36	11,50	380	64,9031
	Distr	78	1	2,00	4,00	19,76	13,00	380	50,3678
RxJava	Cent	444	0	10,00	30,00	129,61	92,25	4.909	431,4591
	Distr	444	1	7,75	22,00	105,73	60,00	12.019	674,5587
Scrapy	Cent	2.920	1	3,00	9,00	30,98	26,00	2.122	78,3791
	Distr	2.920	1	3,00	8,00	29,48	26,00	2.130	74,2866

Continua...

Projeto	SCV	N	Min	1Q	Med.	Mean	3Q	Max	St.D.
Scrcpy	Cent	47	1	11,50	38,00	107,90	83,00	2.132	312,1522
	Distr	47	1	3,50	8,00	21,00	19,00	154	34,8456
Socket.io	Cent	134	1	5,25	10,00	42,06	22,00	1.733	161,4138
	Distr	134	1	3,00	9,50	32,51	20,00	1.829	160,6593
Spring-boot	Cent	29.457	0	3,00	8,00	41,85	28,00	2.430	129,5919
	Distr	29.457	0	2,00	8,00	43,80	29,00	2.416	135,7343
Spring-framework	Cent	1.954	1	6,00	18,00	65,96	57,00	4.611	235,0213
	Distr	1.954	1	5,00	15,00	63,35	53,00	4.174	204,7628
Terminal	Cent	1.364	1	4,00	11,00	46,27	40,25	1.811	114,4920
	Distr	1.364	0	4,00	10,00	38,55	34,00	1.834	105,4734
Tesseract	Cent	165	1	3,00	8,00	69,74	24,00	1.829	241,0508
	Distr	165	1	3,00	7,00	21,61	16,00	776	69,6055
Thefuck	Cent	95	1	1,00	4,00	11,16	12,50	107	18,2560
	Distr	95	1	1,00	4,00	9.084,00	11,50	67	12,4154
Vue	Cent	27	2	2,00	2,00	55,48	9,00	1.218	233,0316
	Distr	27	2	2,00	2,00	52,15	7,00	1.218	233,2044
X64dbg	Cent	967	0	2,00	11,00	66,12	57,00	4.390	192,0799
	Distr	967	0	6,00	39,00	149,20	139,00	4.406	341,9670
Youtube-dl	Cent	718	1	2,00	8,00	69,85	41,75	5.062	303,0424
	Distr	718	0	2,00	6,00	45,18	23,75	4.316	222,1788
Zstd	Cent	2.202	1	4,00	12,00	57,53	42,00	2.710	154,9085
	Distr	2.202	1	4,00	13,00	58,76	44,00	3.017	154,6457

Legenda:
SCV: Sistema de Controle de Versão (Cent = Centralizado, Distr = Distribuído)
N: Conflitos analisados / Min.: Mínimo / 1Q: 1º Quartil / Med.: Mediana / Mean: Média / 3Q: 3º Quartil
Max: Máximo / St.D.: Desvio padrão

Testando hipóteses. Para verificar se de fato os resultados do esforço são estatisticamente significativos, foram realizados os testes estatísticos de Wilcoxon e t-test pareado. A hipótese é que o VCS distribuído (Git) tende a exigir um esforço maior do que seu equivalente centralizado (SVN). Para indicar uma verdadeira significância, considerou-se o nível de confiança de 0,05 (p-value 0,05). A Tabela 4 mostra os p-value para a comparação pareada. Dos 50 projetos analisados, 6 projetos (sinalizados em negrito) apresentaram p-value abaixo de 0,05, indicando que nestes casos, há evidência estatística para sugerir que o esforço médio do Git é estatisticamente maior do que o esforço médio do SVN. Ao analisar o resultado geral (primeira linha da tabela), é possível verificar que ambos os testes não encontraram evidências convincentes para afirmar que há uma diferença significativa entre as duas amostras pareadas. Os p-value elevados (acima de 0,05) indicam que não há suporte estatístico para a rejeição da hipótese nula (H_{1-0}) de que as médias ou medianas das duas amostras não são diferentes. *A principal conclusão é que, na análise geral dos projetos, não é possível afirmar que o VCS centralizado oferece um esforço geral significativamente menor para resolver os conflitos do que usando VCS distribuído.*

Tabela 4: Resultados de Wilcoxon signed-rank test e o Teste t pareado para esforço de resolução de conflitos.

	Wilcoxon		Paired t-test		
Projeto	**p-value**	**Wilcoxon**	**t**	**df**	**p-value**
Geral	1,000	-2,000	-2,080	116.387	0,981
Ant-design	1,000	-2,500	0,299	7.746	0,382
Axios	1,000	-23,000	-3,307	119	0,999
Bitcoin	0,956	-1,000	-9,018	13.147	1,000
Bootstrap	0,940	-1,500	-1,041	4.666	0,851
Caffe	0,833	-5,000	-0,949	237	0,828
Code-server	**1,03E-32**	**6,000**	-1,464	6.479	0,928
Core	1,000	-4,000	-1,115	986	0,867
Django	1,000	-12,500	-4,237	274	1,000
Dubbo	0,996	-3,500	**3,316**	**1.966**	**4,64E-04**
Electron	0,946	-1,500	-1,045	2.442	0,852
Emscripten	1,000	-17,500	-3,750	1.711	1,000
Faceswap	0,976	-168,000	-1,694	38	0,951
Flask	0,387	-1,500	**1,985**	**623**	**0,024**
Googletest	1,000	-8,000	-1,517	903	0,935
Hashcat	1,000	-7,000	-4,521	2.002	1,000
Hibernate-orm	**0,015**	**1,000**	**2,437**	**1.063**	**0,007**
Jq	0,967	-35,500	-1,612	47	0,943
Jquery	1,000	-30,000	-4,336	246	1,000
Json	0,749	-5,500	-2,509	828	0,994
Junit5	0,457	-8,000	0,270	75	0,394
Keras	1,000	-9,000	-2,204	865	0,986
Libuv	**0,015**	**1,500**	**1,861**	**289**	**0,032**
Lombok	1,000	-19,000	-6,045	383	1,000
Magisk	0,753	-18,000	-0,940	25	0,822
Mockito	0,992	-7,000	0,426	261	0,335
Nest	1,000	-4,000	-4,617	15.193	1,000
Netdata	0,139	-1,000	1,496	259	0,068
Netty	0,947	-17,500	-1,591	57	0,941
Obs-studio	1,000	-18,500	-5,177	481	1,000
Stb	1,000	-43,500	-5,033	303	1,000
Pandas	1,000	-10,000	-4,125	1.930	1,000
Protobuf	0,897	-1,000	-1,128	7.375	0,870
React	0,890	-3,500	0,659	722	0,255
Redis	1,000	-27,000	-6,235	1.389	1,000
Redux	0,992	-5,500	-1,483	173	0,930
Requests	0,992	-5,000	-2,246	477	0,987
Retrofit	0,625	-3,000	-1,119	77	0,867
RxJava	1,000	-30,000	-0,687	443	0,754
Scrapy	0,914	-2,000	-1,273	2.919	0,899
Scrcpy	1,000	-78,500	-1,884	46	0,967
Socket.io	0,971	-6,000	-1,888	133	0,969
Spring-boot	**8,43E-63**	**1,500**	**8,659**	**29.456**	**2,51E-18**
Spring-framework	1,000	-6,500	-0,965	1.953	0,833
Terminal	1,000	-5,500	-2,383	1.363	0,991
Tesseract	0,999	-20,000	-2,578	164	0,995
Thefuck	0,697	-3,000	-1,300	94	0,902
Vue	0,827	-59,000	-1,003	26	0,837
X64dbg	**6,51E-23**	**36,500**	**7,998**	**966**	**1,80E-15**
Youtube-dl	1,000	-8,000	-2,706	717	0,997
Zstd	**1,71E-04**	**2,000**	**0,590**	**2.201**	**0,278**

5.2 H2: Eficácia e Sistema de Controle de Versão

Esta seção examina os dados coletados com relação ao impacto do sistema de controle de versão na eficácia da integração realizada. Foi calculado o número de linhas de código adicionadas e removidas para transformar o código-fonte integrado no código-fonte desejado. A Tabela 6 (Apêndice A) mostra as operações realizadas em cada projeto alvo. Utilizando o VCS centralizado, no total, foram realizadas 10.103.336 operações para produzir o código-fonte desejado, sendo 5.647.266 adições e 4.456.070 remoções. Por outro lado, foram necessárias 8.910.840 operações para esse fim utilizando o VCS distribuído, sendo 4.795.184 adições e 4.115.656 remoções. Isso representa uma diferença de 1.192.496 operações a menos para realizar no VCS distribuído, indicando que o uso do VCS centralizado causa um aumento de aproximadamente 13,38% no número de operações necessárias. Essas medidas de operações são a base para calcular os indicadores de *precision* (Equação 1), *accuracy* (Equação 3), *recall* (Equação 2) e *F-measure* (Equação 4). Estes indicadores estão no centro da avaliação de eficácia.

Testando hipóteses. Para avaliar se essas diferenças são estatisticamente significativas, foram realizados os testes estatísticos de Wilcoxon e t-test pareado às medidas coletadas. A hipótese é que o VCS centralizado (SVN) tende a ser mais eficaz que o distribuído (Git). Conforme descrito anteriormente, foi considerado o nível de significância sendo 0,05 (p-value 0,05). A Tabela 7 (Apêndice A) apresenta os resultados das medidas de *accuracy* e *precision*. A Tabela 8 (Apêndice A) apresenta os resultados para os dados de *recall* e *f-measure*.

Teste de hipóteses para Accuracy. A análise do resultado dos testes de hipótese para a *accuracy* geral (Tabela 7 - Linha 1 - Apêndice A), mostra p-values elevados (ambos iguais a 1,00), o que sugere que não há evidência estatística para rejeitar a hipótese nula (H_{2-0}) de que o VCS centralizado SVN é significativamente menos eficaz do que o VCS distribuído Git para mesclar o código-fonte. Em 4 projetos (sinalizados em negrito na tabela) dos 50 analisados, os resultados apresentaram p-values inferiores a 0,05 em ambos os testes, indicando que nestes casos há evidência estatística para rejeitar a hipótese nula (H_{2-0}), de que o VCS centralizado é significativamente menos eficaz que o distribuído na integração do código-fonte, favorecendo a hipótese alternativa (H_{2-1}), de que o VCS centralizado é significativamente mais eficaz do que o distribuído para integração de código-fonte. *Portanto, na análise geral, não é possível rejeitar a hipótese nula (H_{2-0}) de que a eficácia do VCS SVN é significativamente menor ou igual à eficácia do VCS distribuído Git para integrar o código-fonte.*

Teste de hipóteses para Precision. A análise dos resultados dos testes em relação a *precision* geral (Tabela 7 - Linha 1 - Apêndice A), demonstra inconsistência entre os resultados. O t-test pareado sugere evidência estatística significativa contra a hipótese nula (H_{2-0}), enquanto o teste de Wilcoxon não fornece essa evidência. Neste caso podem existir algumas possíveis causas para essa inconsistência, o t-test pareado assume que há uma distribuição normal dos dados e pode ser sensível a valores extremos, enquanto o teste de Wilcoxon é não paramétrico e mais robusto em relação a distribuições não normais e valores atípicos. A divergência nos resultados

pode indicar a presença de assimetria nos dados ou a influência de *outliers* (dados que se diferenciam drasticamente de todos os outros). Em 5 projetos (sinalizados em negrito na tabela) dos 50 analisados, é possível observar p-values inferiores a 0,05 em ambos os testes, indicando que nestes casos que há evidência estatística para rejeitar a hipótese nula (H_{2-0}), de que o VCS centralizado é significativamente menos eficaz que o distribuído na integração do código-fonte, favorecendo a hipótese alternativa (H_{2-1}), de que o VCS centralizado é significativamente mais eficaz do que o distribuído para integração de código-fonte. *Portanto, na análise geral, não é possível rejeitar a hipótese nula (H_{2-0}) de que a eficácia do VCS SVN é significativamente menor ou igual à eficácia do VCS distribuído Git para integrar o código-fonte.*

Teste de hipóteses para Recall. A análise dos resultados dos testes em relação a *recall* geral (Tabela 8 - Linha 1 - Apêndice A) indica p-values elevados (ambos iguais a 1,00) tanto no teste de Wilcoxon quanto no t-test pareado. Este resultado sugere que não há evidência estatística para rejeitar a hipótese nula (H_{2-0}) de que o VCS centralizado SVN é significativamente menos eficaz do que o VCS distribuído Git para mesclar o código-fonte. Apenas 1 projeto (sinalizado em negrito na tabela) dos 50 analisados, foi possível observar p-value inferior a 0,05 em ambos os testes, indicando que neste caso há evidência estatística para rejeitar a hipótese nula (H_{2-0}), de que o VCS centralizado é significativamente menos eficaz que o distribuído na integração do código-fonte, favorecendo a hipótese alternativa (H_{2-1}), de que o VCS centralizado é significativamente mais eficaz do que o distribuído para integração de código-fonte. *Em resumo, com base nos resultados obtidos, de forma geral não é possível rejeitar a hipótese nula (H_{2-0}) de que a eficácia do VCS SVN é significativamente menor ou igual à eficácia do VCS distribuído Git para integrar o código-fonte.*

Teste de hipóteses para F-measure. A análise dos resultados dos testes em relação a *F-measure* geral (Tabela 8 - Linha 1 - Apêndice A) indica p-values elevados (ambos iguais a 1,00), sugerindo que não há evidência estatística para rejeitar a hipótese nula (H_{2-0}) de que o VCS centralizado SVN é significativamente menos eficaz do que o VCS distribuído Git para mesclar o código-fonte. Em 3 projetos (sinalizados em negrito na tabela) dos 50 analisados, foi possível observar p-values inferiores a 0,05 em ambos os testes, indicando que nestes casos, há evidência estatística para rejeitar a hipótese nula (H_{2-0}), de que o VCS centralizado é significativamente menos eficaz que o distribuído na integração do código-fonte, favorecendo a hipótese alternativa (H_{2-1}), de que o VCS centralizado é significativamente mais eficaz do que o distribuído para integração de código-fonte. *Resumidamente, com base nos resultados obtidos, de forma geral não é possível rejeitar a hipótese nula (H_{2-0}) de que a eficácia do VCS SVN é significativamente menor ou igual à eficácia do VCS distribuído Git para integrar o código-fonte.*

5.3 Discussão Adicional

Com base nos resultado obtidos, esta seção discute implicações para a indústria e academia, bem como introduz algumas limitações do estudo.

5.3.1 Implicações para a Industria

Ausência de uma vantagem estatisticamente significativa entre Git e SVN. O estudo em questão traz implicações significativas para a indústria de desenvolvimento de software, especialmente no que diz respeito à escolha de sistemas de controle de versão, ao considerar os resultados que indicam a ausência de uma vantagem estatisticamente significativa entre Git e SVN na resolução de conflitos de integração, as empresas podem reavaliar os critérios utilizados na escolha dessas ferramentas.

Importância de uma abordagem holística na escolha de sistemas de controle de versão. Em geral, as empresas costumam analisar vários aspectos ao escolher um sistema de controle de versão. A eficiência na resolução de conflitos de integração é um fator importante, mas é apenas um deles. Outros fatores, como a facilidade de colaboração, desempenho, suporte a ramificações, integração com outras ferramentas de desenvolvimento, robustez do histórico de versões, popularidade e tamanho da comunidade, complexidade, backup e recuperação e a natureza do projeto, podem ser cruciais nessa análise. O resultado deste estudo reforça a importância de uma abordagem holística na escolha de sistemas de controle de versão, considerando não apenas um aspecto específico, mas a experiência geral de desenvolvimento.

Considerar a escolha do sistema de controle de versão com base em outros critérios mais relevantes. Ao indicar a ausência de uma clara vantagem em escolher um sistema de controle de versão distribuído, como o Git, sobre um centralizado, como o SVN, do ponto de vista da resolução de conflitos de integração, este estudo desafia a percepção comum de que sistemas distribuídos, devido à sua natureza descentralizada, oferecem vantagens significativas em todos os aspectos. As empresas agora podem reconsiderar a escolha do sistema de controle de versão com base em outros critérios mais relevantes para suas necessidades específicas, sem se apegar à ideia de que a descentralização é inerentemente superior nesse contexto.

Escolha do VCS considerando as características únicas de projetos e equipes. Neste cenário, a escolha personalizada, considerando as características únicas de projetos e equipes, parece ser o caminho mais adequado a ser considerado pela indústria. Isso significa que a decisão sobre o sistema de controle de versão pode ser mais flexível, dependendo das nuances específicas de cada contexto de desenvolvimento.

Enfoque na eficiência global do desenvolvimento de software. Considerando o ciclo de vida completo do desenvolvimento, desde a colaboração inicial até a entrega final, o enfoque na eficiência global do desenvolvimento de software parece ser o caminho natural e mais adequado a ser seguido. A escolha do sistema de controle de versão pode, assim, fazer parte de uma estratégia mais ampla para otimizar o fluxo de trabalho de desenvolvimento.

Importância contínua de pesquisas e avaliações contextuais. Os resultados deste estudo ressaltam a importância contínua de pesquisas e avaliações contextuais na indústria de desenvolvimento de software. As ferramentas e práticas estão em constante evolução, e as empresas devem estar abertas a reavaliar suas escolhas com base em evidências e resultados empíricos.

A pesquisa contínua pode fornecer percepções valiosas para informar as decisões de desenvolvimento de software no cenário em constante mudança da indústria.

5.3.2 Implicações para a Academia

Necessidade de novas pesquisas experimentais com abordagem mais abrangente. Os resultados deste estudo sugerem a necessidade de novas pesquisas experimentais que explorem questões mais profundas relacionadas ao processo de desenvolvimento de software. Assim como em (BRINDESCU et al., 2014), onde as descobertas indicam que o uso de VCS distribuídos leva os desenvolvedores realizarem *commits* menores e mais frequentes em comparação com os desenvolvedores que utilizam VCS centralizados, futuros estudos podem analisar como fatores como a arquitetura de projetos, o tipo de processo de desenvolvimento adotado, a formação das equipes e a complexidade das funcionalidades impactam a dinâmica de ocorrência de conflitos de integração. Uma abordagem mais abrangente que considere esses aspectos pode fornecer percepções valiosas sobre a eficácia de sistemas de controle de versão em diferentes contextos.

Propor novos estudos e implementações de algoritmos que considerem semântica e contexto do software. Este estudo indica a oportunidade de propor novos estudos e implementações de algoritmos que identifiquem a semântica e contexto do software, considerando alterações e refatorações no código-fonte. Além disso, a capacidade de sugerir mesclagens e refatorações de maneira mais inteligente, eficaz e com rastreabilidade mais consistente seria um avanço significativo no desenvolvimento de sistemas de controle de versão e no processo de desenvolvimento de software em geral.

Ferramentas de controle versão adaptadas ao contexto dos projetos. Os resultados destacam a importância de desenvolver ferramentas de controle de versões mais adaptativas, capazes de se ajustar dinamicamente aos contextos específicos de cada projeto e às preferências das equipes, oferecendo uma solução mais eficaz para a gestão de conflitos de integração. Isso implica um movimento em direção a sistemas mais flexíveis e personalizáveis que se alinham às nuances individuais de diferentes ambientes de desenvolvimento.

Colaboração entre especialistas e abordagens interdisciplinares. A busca por soluções avançadas para a resolução de conflitos de integração pode envolver a integração de abordagens interdisciplinares. Colaborações entre especialistas em controle de versão, desenvolvedores de algoritmos, cientistas de dados e especialistas em IA podem levar a inovações mais substanciais. A combinação de conhecimentos em diferentes disciplinas pode resultar em abordagens mais abrangentes e eficientes para lidar com os desafios apresentados pelo estudo.

Utilização de inteligência artificial e aprendizado de máquina. Outra importante contribuição nesta área pode vir da incorporação de ferramentas de inteligência artificial (IA) e aprendizado de máquina (AM) para análise de contexto durante a resolução de conflitos de integração. A aplicação de IA e AM pode contribuir para uma compreensão mais aprofundada

da aplicação e do código-fonte, permitindo propostas mais assertivas de refatorações e identificação de possíveis bugs, além de fornecer uma rastreabilidade maior ao histórico de alterações do código-fonte. Essa abordagem poderia revolucionar a maneira como as equipes lidam com conflitos de integração, tornando o processo mais automatizado, adaptável e capaz de aprender com padrões recorrentes.

Incentivo à inovação contínua e fomento à criatividade. Fica evidente, ao analisar os resultados deste estudo, a necessidade de incentivar a inovação contínua em alternativas mais eficazes que possam contribuir na resolução de conflitos de integração, seja por meio de competições, hackathons ou programas de pesquisa que estimulem a comunidade acadêmica e profissional a buscar soluções inovadoras. O estudo indica que há espaço para melhorias significativas, e a promoção de um ambiente que fomente a criatividade pode acelerar o desenvolvimento de abordagens mais eficientes.

Novos estudos com número ampliado de projetos analisados. Este estudo analisou 50 repositórios de projetos de código aberto, e o aumento e facilidade de acesso ao código-fonte desses projetos, disponíveis em repositórios virtuais como o GitHub, favorecem o desenvolvimento de novos estudos que ampliem o número de projetos analisados, assim como a diversidade de tipos e linguagens de programação e contexto de aplicação destes projetos. Projetos de diferentes tamanhos, finalidades e comunidades de desenvolvimento podem apresentar desafios distintos na gestão de conflitos de integração.

Importância de considerar a flexibilidade e a adaptabilidade de acordo com o contexto. O estudo evidencia os desafios na busca por soluções generalizadas para a resolução de conflitos de integração. A diversidade de projetos, equipes e processos de desenvolvimento implica que não existe uma abordagem única que se aplique a todos os casos, destacando a importância de considerar a flexibilidade e a adaptabilidade das soluções propostas, reconhecendo que diferentes contextos exigem abordagens distintas. Neste cenário, como já comentado anteriormente, algoritmos e ferramentas que sejam sensíveis ao contexto do projeto podem ter um papel fundamental, oferecendo melhores resultados e facilitando o trabalho de desenvolvimento de software.

Implicações que podem orientar futuras pesquisas. Ao proporcionar uma análise detalhada do impacto dos sistemas de controle de versão na resolução de conflitos de integração, este estudo contribui levantando implicações que podem orientar futuras pesquisas, práticas de desenvolvimento e inovações tecnológicas, promovendo uma abordagem mais informada e eficaz na gestão de versões de software em ambientes colaborativos.

5.3.3 Limitações e Ameaças à Validade do Estudo

Amostra restrita de projetos analisados. Este estudo foi baseado em uma amostra restrita de projetos analisados, que se limitou a 50 repositórios de código aberto. Embora esses projetos possam oferecer percepções relevantes, a generalização dos resultados para todos os projetos

de desenvolvimento de software pode ser comprometida. Uma abordagem mais abrangente envolvendo um número maior e mais diversificado de projetos poderia oferecer uma visão mais completa das nuances envolvidas na resolução de conflitos de integração em diferentes contextos e tipos de aplicação.

Falta projetos privados de empresas na amostra. Outra limitação significativa é a falta de projetos privados de empresas na análise, restringindo-se apenas a repositórios públicos disponíveis no GitHub. Projetos privados muitas vezes têm dinâmicas de desenvolvimento, processos de colaboração e exigências de gerenciamento diferentes em comparação com projetos de código aberto mantidos pela comunidade. A ausência dessa perspectiva pode limitar a aplicabilidade dos resultados em ambientes de desenvolvimento de software empresariais privados.

Análise do contexto e da realidade específica de cada projeto. O estudo reconhece a importância da análise do contexto e da realidade específica de cada projeto, tanto em termos de processos utilizado pelas equipes de desenvolvimento quanto na forma de colaboração entre os desenvolvedores. Deste modo, a falta de consideração destes aspectos no estudo pode representar uma vulnerabilidade para a validade dos resultados. A diversidade de projetos e as nuances específicas de cada contexto de desenvolvimento podem não ter sido totalmente capturadas, impactando a capacidade do estudo de fornecer conclusões aplicáveis a uma gama mais ampla de cenários.

Variação das práticas de desenvolvimento adotadas. Outra limitação inerente está na variação das práticas de desenvolvimento adotadas pelos projetos analisados. Diferenças nos processos, na experiência dos desenvolvedores e na cultura de colaboração podem influenciar significativamente os resultados. A falta de uma análise mais detalhada dessas variáveis pode obscurecer a compreensão dos fatores específicos que contribuem para a eficácia ou falta dela na resolução e na própria ocorrência de conflitos de integração.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalhou apresentou um estudo empírico para analisar o impacto dos sistemas de controle de versão centralizados e distribuídos em projetos de desenvolvimento de software colaborativo, com foco específico no impacto destes VCS no esforço e eficácia de resolução de conflitos de integração. Para isso, foi realizado a identificação, replicação e análise dos casos de conflito de integração de código-fonte em 50 projetos disponíveis no GitHub.

Com base nos 50 projetos selecionados, foram utilizados métodos estatísticos como o Teste dos Postos Sinalizados de Wilcoxon e o Teste t pareado, para analisar 116.388 conflitos de integração.

Os resultados deste estudo, obtidos por meio das análises estatísticas descritas anteriormente, indicaram que não é possível identificar uma vantagem estatisticamente significativa no esforço de resolução de conflitos de integração ao utilizar um VCS distribuído, como o Git, em comparação com um VCS centralizado, como o SVN. Esta constatação desafia a percep-

ção comum na indústria de que sistemas distribuídos oferecem uma melhoria substancial nesse aspecto. A contribuição central deste estudo reside na evidência empírica que proporciona, fornecendo elementos para consideração nas decisões de desenvolvedores e equipes de projeto com dados concretos sobre a eficácia relativa desses sistemas em cenários de conflito de integração.

Para futuras pesquisas, destacam-se várias direções promissoras no campo de controle de versão e esforço na resolução de conflitos de integração. Mostra-se necessário a realização de novos estudos experimentais que explorem questões específicas, como a influência da arquitetura de projetos e o tipo de processo de desenvolvimento na dinâmica de conflitos. Estudos que explorem a proposição e implementação de algoritmos mais sofisticados, capazes de considerar a semântica e o contexto do software, pode ser uma contribuição significativa para melhorar a eficácia dos sistemas existentes. Além disso, estudos que explorem a adaptação e o uso de inteligência artificial e aprendizado de máquina para aprimorar a resolução de conflitos são propostas como áreas promissoras. Ampliar o escopo de projetos analisados, considerando diferentes tamanhos, finalidades e contextos, é recomendado para uma compreensão mais abrangente do tema.

Referências

BRINDESCU, C. et al. An empirical investigation into merge conflicts and their effect on software quality. **Empirical Software Engineering**, Springer, v. 25, n. 1, p. 562–590, 2020.

BRINDESCU, C. et al. How do centralized and distributed version control systems impact software changes? In: **Proceedings of the 36th international conference on Software Engineering**. [S.l.: s.n.], 2014. p. 322–333.

CARBONERA, C. et al. Software merge: A two-decade systematic mapping study. In: **Proceedings of the XXXVII Brazilian Symposium on Software Engineering**. [S.l.: s.n.], 2023. p. 99–108.

CONNER, B.; JOHNSON, E. Descriptive statistics. **American Nurse Today**, v. 12, n. 11, p. 52–55, 2017.

DALCIN, G. et al. Recommendation of uml model conflicts: Unveiling the biometric lens for conflict resolution. In: **XXXVII Brazilian Symposium on Software Engineering**. [S.l.: s.n.], 2023. p. 83–88.

GIT. **Git**. 2023. Disponível em: <https://git-scm.com/about>. Acesso em: 15 jun. 2023.

GITHUB. **GitHub**. 2023. Disponível em: <https://github.com>.

GRAEFF, C. A.; FARIAS, K.; CARBONERA, C. E. On the prediction of software merge conflicts: A systematic review and meta-analysis. In: **Proceedings of the XIX Brazilian Symposium on Information Systems**. [S.l.: s.n.], 2023. p. 404–411.

MEHDI, A.-N.; URSO, P.; CHAROY, F. Evaluating software merge quality. In: **Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering**. [S.l.: s.n.], 2014. p. 1–10.

MENS, T. A state-of-the-art survey on software merging. **IEEE transactions on software engineering**, IEEE, v. 28, n. 5, p. 449–462, 2002.

MISTRíK, I. et al. **Collaborative Software Engineering**. [S.l.]: Springer, 2010. 185 p.

OLIVEIRA, A. et al. Do code refactorings influence the merge effort? **arXiv preprint arXiv:2305.06129**, 2023.

OTTE, S. Version control systems. **Computer systems and telematics**, Citeseer, p. 11–13, 2009.

SANTOS, R. de S.; MURTA, L. G. P. Evaluating the branch merging effort in version control systems. In: IEEE. **2012 26th Brazilian Symposium on Software Engineering**. [S.l.], 2012. p. 151–160.

TAHERI, S.; HESAMIAN, G. A generalization of the wilcoxon signed-rank test and its applications. **Statistical Papers**, Springer, v. 54, p. 457–470, 2013.

TIOBE. **TIOBE Index**. 2023. Disponível em: <https://www.tiobe.com/tiobe-index/>. Acesso em: 15 jun. 2023.

VALE, G. et al. Challenges of resolving merge conflicts: A mining and survey study. **IEEE Transactions on Software Engineering**, IEEE, v. 48, n. 12, p. 4964–4985, 2021.

WOHLIN, C. et al. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012.

ZOLKIFLI, N. N.; NGAH, A.; DERAMAN, A. Version control system: A review. **Procedia Computer Science**, Elsevier, v. 135, p. 408–415, 2018.

APÊNDICE A – TABELAS DE DADOS

Tabela 5: Descrição dos projetos analisados.

Projeto	Repositório	LP	LC	Commits	Contrib.
Ant-design	https://github.com/ant-design/ant-design.git	TypeScript	177.738	25.377	2.291
Axios	https://github.com/axios/axios.git	JavaScript	61.374	1.450	451
Bitcoin	https://github.com/bitcoin/bitcoin.git	C++	681.033	38.515	1.286
Bootstrap	https://github.com/twbs/bootstrap.git	JavaScript	137.013	22.421	1.673
Caffe	https://github.com/BVLC/caffe.git	C++	81.373	4.156	342
Code-server	https://github.com/coder/code-server.git	TypeScript	15.530	3.565	245
Core	https://github.com/home-assistant/core.git	Python	1.770.226	66.112	3.908
Django	https://github.com/django/django.git	Python	624.389	31.734	3.022
Dubbo	https://github.com/apache/dubbo.git	Java	282.446	7.147	640
Electron	https://github.com/electron/electron.git	C++	188.574	27.355	1.478
Emscripten	https://github.com/emscripten-core/emscripten.git	C	1.494.550	26.016	903
Faceswap	https://github.com/deepfakes/faceswap.git	Python	53.528	1.726	111
Flask	https://github.com/pallets/flask.git	Python	18.853	5.096	854
Googletest	https://github.com/google/googletest.git	C++	68.115	4.312	504
Hashcat	https://github.com/hashcat/hashcat.git	C	1.091.958	9.518	196
Hibernate-orm	https://github.com/hibernate/hibernate-orm.git	Java	1.186.203	16.155	699
Jq	https://github.com/jqlang/jq.git	C	48.565	1.597	185
Jquery	https://github.com/jquery/jquery.git	JavaScript	40.575	6.614	341
Json	https://github.com/nlohmann/json.git	C++	127.395	4.541	298
Junit5	https://github.com/junit-team/junit5.git	Java	109.836	7.779	267
Keras	https://github.com/keras-team/keras.git	Python	121.476	9.856	1.322
Libuv	https://github.com/libuv/libuv.git	C	77.917	5.237	552
Lombok	https://github.com/projectlombok/lombok.git	Java	99.542	3.540	166
Magisk	https://github.com/topjohnwu/Magisk.git	C++	53.014	6.480	348
Mockito	https://github.com/mockito/mockito.git	Java	73.227	5.929	346
Nest	https://github.com/nestjs/nest.git	TypeScript	695.985	13.461	492
Netdata	https://github.com/netdata/netdata.git	C	540.181	16.238	660
Netty	https://github.com/netty/netty.git	Java	348.475	11.276	822
Obs-studio	https://github.com/obsproject/obs-studio.git	C	507.039	13.395	651
Pandas	https://github.com/pandas-dev/pandas.git	Python	527.679	32.555	3.460
Protobuf	https://github.com/protocolbuffers/protobuf.git	C++	715.146	15.600	1.245
React	https://github.com/facebook/react.git	JavaScript	411.939	15.821	1.836
Redis	https://github.com/redis/redis.git	C	313.386	11.844	840
Redux	https://github.com/reduxjs/redux.git	TypeScript	360.630	3.793	1.016
Requests	https://github.com/psf/requests.git	Python	11.147	6.156	763
Retrofit	https://github.com/square/retrofit.git	Java	30.511	2.096	215
Rxjava	https://github.com/ReactiveX/RxJava.git	Java	324.153	6.052	356
Scrapy	https://github.com/scrapy/scrapy.git	Python	58.621	9.803	691
Scrcpy	https://github.com/Genymobile/scrcpy.git	C	27.041	2.218	139
Socket.io	https://github.com/socketio/socket.io.git	TypeScript	197.775	1.978	228
Spring-boot	https://github.com/spring-projects/spring-boot.git	Java	462.115	42.984	1.305
Spring-framework	https://github.com/spring-projects/spring-framework.git	Java	839.220	26.693	933
Stb	https://github.com/nothings/stb.git	C	82.387	2.152	228
Terminal	https://github.com/microsoft/terminal.git	C++	320.903	3.637	404
Tesseract	https://github.com/tesseract-ocr/tesseract.git	C++	165.007	6.252	219
Thefuck	https://github.com/nvbn/thefuck.git	Python	11.263	1.650	204
Vue	https://github.com/vuejs/vue.git	TypeScript	73.539	3.547	397
X64dbg	https://github.com/x64dbg/x64dbg.git	C++	156.764	5.173	168
Youtube-dl	https://github.com/ytdl-org/youtube-dl.git	Python	138.765	18.780	1.043
Zstd	https://github.com/facebook/zstd.git	C	118.020	10.313	360
Total			16.122.141	625.695	41.103

Continua...

Projeto	Repositório	LP	LC	Commits	Contrib.

Legenda:
LP: Linguagem de programação principal
LC: Número de linhas de código
Commits: Número de commits realizados
Contrib: Número de contribuidores do projeto

Tabela 6: Descrição de operações necessárias para transformar o código-fonte integrado no código-fonte desejado.

	Centralizado (SVN)		Distribuído (Git)	
Projeto	**LC adicionadas**	**LC removidas**	**LC adicionadas**	**LC removidas**
Ant-design	1.295.465	1.162.451	1.360.377	1.257.895
Axios	3.047	1.162	1.691	776
Bitcoin	325.485	226.252	241.903	178.722
Bootstrap	250.707	266.759	234.253	248.979
Caffe	4.113	1.910	3.421	1.960
Code-server	224.247	203.374	210.698	188.862
Core	13.756	10.182	10.144	11.294
Django	6.454	3.192	4.076	1.024
Dubbo	37.077	24.767	45.507	36.747
Electron	31.963	19.051	26.767	21.299
Emscripten	93.331	69.377	53.354	34.117
Faceswap	1.201	788	516	358
Flask	18.211	9.851	20.365	12.275
Googletest	17.543	15.458	12.653	11.877
Hashcat	47.603	57.014	31.306	31.356
Hibernate-orm	64.444	51.417	76.109	69.002
Jq	1.537	912	722	428
Jquery	9.053	7.311	5.331	3.369
Json	164.100	106.566	102.810	88.260
Junit5	1.233	1.148	1.639	862
Keras	37.476	27.902	24.214	18.234
Libuv	22.420	8.180	25.295	9.300
Lombok	12.137	4.559	5.920	3.349
Magisk	551	940	654	447
Mockito	3.761	1.652	3.945	3.443
Nest	1.381.189	1.033.883	839.373	692.138
Netdata	3.782	1.548	4.060	2.308
Netty	1.142	2.067	628	1.648
Obs-studio	18.854	7.320	8.291	4.319
Pandas	76.326	35.247	42.795	21.892
Protobuf	271.725	217.201	248.858	220.611
React	26.234	14.190	30.510	16.646
Redis	108.132	42.528	58.978	31.662
Redux	1.729	4.363	1.144	4.116
Requests	9.442	5.349	6.057	3.438
Retrofit	924	1.054	842	699
RxJava	39.685	17.863	28.109	18.834
Scrapy	55.736	34.714	44.568	41.525
Scrcpy	3.921	1.149	728	259
Socket.io	4.377	1.259	3.166	1.190
Spring-boot	685.590	547.225	711.841	578.458
Spring-framework	62.313	66.567	56.182	67.598
Stb	21.304	10.368	9.681	5.247
Terminal	40.159	22.957	25.919	26.662

Continua...

	Centralizado (SVN)		Distribuído (Git)	
Projeto	**LC adicionadas**	**LC removidas**	**LC adicionadas**	**LC removidas**
Tesseract	5.550	5.957	1.621	1.945
Thefuck	765	295	490	373
Vue	495	1.003	429	979
X64dbg	38.873	25.061	76.739	67.547
Youtube-dl	28.563	21.587	17.976	14.464
Zstd	73.541	53.140	72.529	56.863
Total	5.647.266	4.456.070	4.795.184	4.115.656

Legenda:

LC adicionadas: Número de linhas de código adicionadas

LC removidas: Número de linhas de código removidas

Tabela 7: Resultados de Wilcoxon signed-rank test e o Teste t pareado para Accuracy e Precision.

	Accuracy: accuracy(SVN) > accuracy(Git)					**Precision: precision(SVN) > precision(Git)**				
	Wilcoxon		**Paired t-test**			**Wilcoxon**		**Paired t-test**		
Projeto	**p-value**	**Wilcoxon**	**t**	**df**	**p-value**	**p-value**	**Wilcoxon**	**t**	**df**	**p-value**
Geral	1,000	-0,005	-12,487	116.387	1,000	1,000	-6,97E-04	**4,679**	**116.387**	**1,45E-06**
Ant-design	1,000	-0,016	-5,821	7.746	1,000	1,000	-0,005	-0,243	7.746	0,596
Axios	1,000	-0,095	-4,126	119	1,000	0,923	-0,030	-0,640	119	0,738
Bitcoin	0,618	-0,001	-2,895	13.147	0,998	**3,03E-07**	**4,55E-04**	0,693	13.147	0,244
Bootstrap	0,999	-0,005	-2,448	4.666	0,993	0,917	-0,002	-0,662	4.666	0,746
Caffe	0,872	-0,017	-1,158	237	0,876	0,318	-0,002	0,134	237	0,447
Code-server	**3,32E-31**	**0,006**	**1,721**	**6.479**	**0,043**	**2,94E-36**	**0,007**	0,983	6.479	0,163
Core	0,999	-0,016	-2,328	986	0,990	0,682	-0,004	-0,245	986	0,597
Django	1,000	-0,017	-3,848	274	1,000	1,000	-0,010	-5,416	274	1,000
Dubbo	0,999	-0,013	**1,956**	**1.966**	**0,025**	0,933	-0,004	5,774	1.966	4,49E-09
Electron	0,997	-0,008	-3,072	2.442	0,999	**7,80E-04**	**0,001**	0,968	2.442	0,167
Emscripten	1,000	-0,009	-7,393	1.711	1,000	1,000	-0,002	-4,017	1.711	1,000
Faceswap	0,958	-0,200	-1,026	38	0,844	0,963	-0,193	-0,428	38	0,664
Flask	0,588	-0,006	0,745	623	0,228	0,140	-7,20E-04	0,979	623	0,164
Googletest	1,000	-0,007	-1,923	903	0,973	0,968	-0,002	-0,220	903	0,587
Hashcat	1,000	-0,021	-9,801	2.002	1,000	1,000	-0,007	-7,135	2.002	1,000
Hibernate-orm	0,277	-0,003	-1,061	1.063	0,856	0,066	-1,91E-04	-2,174	1.063	0,985
Jq	0,969	-0,057	-1,678	47	0,950	0,979	-0,037	-1,716	47	0,954
Jquery	1,000	-0,038	-4,274	246	1,000	1,000	-0,015	-3,616	246	1,000
Json	0,800	-0,002	-1,633	828	0,949	0,083	-8,85E-05	0,363	828	0,358
Junit5	0,713	-0,079	-1,061	75	0,854	0,741	-0,073	-1,195	75	0,882
Keras	1,000	-0,016	-4,035	865	1,000	1,000	-0,004	-2,521	865	0,994
Libuv	**0,029**	**0,001**	**1,895**	**289**	**0,030**	**0,004**	**0,002**	0,644	289	0,260
Lombok	1,000	-0,042	-7,157	383	1,000	1,000	-0,016	-4,306	383	1,000
Magisk	0,852	-0,121	-1,492	25	0,926	0,908	-0,086	-1,672	25	0,947
Mockito	0,838	-0,032	-1,691	261	0,954	0,219	-0,006	-0,440	261	0,670
Nest	1,000	-0,009	-20,035	15.193	1,000	1,000	-0,004	-16,792	15.193	1,000
Netdata	0,059	0,000	0,385	259	0,350	**0,003**	**0,002**	0,560	259	0,288
Netty	0,931	-0,080	-1,740	57	0,956	0,861	-0,030	-1,492	57	0,929
Obs-studio	1,000	-0,011	-4,556	481	1,000	1,000	-0,002	-2,646	481	0,996
Stb	1,000	-0,016	-5,381	303	1,000	1,000	-0,004	-2,604	303	0,995
Pandas	1,000	-0,008	-8,492	1.930	1,000	1,000	-0,002	-3,403	1.930	1,000
Protobuf	0,969	-0,003	-0,001	7.375	0,501	**0,024**	**9,42E-05**	**2,400**	**7.375**	**0,008**
React	0,774	-0,007	-0,733	722	0,768	0,977	-0,005	-0,623	722	0,733
Redis	1,000	-0,017	-9,549	1.389	1,000	1,000	-0,003	-5,277	1.389	1,000
Redux	0,943	-0,029	-1,426	173	0,922	0,959	-0,021	-1,597	173	0,944
Requests	0,972	-0,008	-1,568	477	0,941	0,838	-0,003	-0,230	477	0,591

Continua...

	Accuracy: accuracy(SVN) > accuracy(Git)					Precision: precision(SVN) > precision(Git)				
	Wilcoxon		Paired t-test			Wilcoxon		Paired t-test		
Projeto	**p-value**	**Wilcoxon**	**t**	**df**	**p-value**	**p-value**	**Wilcoxon**	**t**	**df**	**p-value**
Retrofit	0,325	-0,008	-0,311	77	0,622	0,504	-0,006	-1,528	77	0,935
RxJava	0,987	-0,009	-0,513	443	0,696	0,909	-0,003	1,612	443	0,054
Scrapy	0,987	-0,009	-2,176	2.919	0,985	**0,012**	**7,10E-04**	1,598	2.919	0,055
Scrcpy	1,000	-0,131	-3,918	46	1,000	1,000	-0,037	-2,441	46	0,991
Socket.io	0,997	-0,018	-3,201	133	0,999	0,162	-9,88E-04	-1,172	133	0,878
Spring-boot	**1,76E-08**	**0,003**	**2,812**	**29.456**	**0,002**	**2,74E-26**	**0,004**	**4,736**	**29.456**	**1,09E-06**
Spring-framework	1,000	-0,021	-5,319	1.953	1,000	1,000	-0,007	-3,107	1.953	0,999
Terminal	1,000	-0,017	-3,070	1.363	0,999	**5,96E-12**	**0,009**	**2,522**	**1.363**	**0,006**
Tesseract	0,998	-0,035	-3,135	164	0,999	0,845	-0,012	-2,649	164	0,996
Thefuck	0,661	-0,016	-1,210	94	0,885	0,129	-0,002	0,816	94	0,208
Vue	0,750	-0,225	-0,872	26	0,804	0,827	-0,118	-1,134	26	0,866
X64dbg	**1,41E-36**	**0,225**	**16,405**	**966**	**7,68E-54**	**5,19E-46**	**0,453**	**18,645**	**966**	**8,21E-67**
Youtube-dl	1,000	-0,013	-4,345	717	1,000	1,000	-0,003	0,796	717	0,213
Zstd	**0,003**	**0,002**	1,499	2.201	0,067	**2,89E-07**	**0,002**	**2,365**	**2.201**	**0,009**

Tabela 8: Resultados de Wilcoxon signed-rank test e o Teste t pareado para Recall e F-Measure.

	Recall: recall(SVN) > recall(Git)					F-Measure: f-measure(SVN) > f-measure(Git)				
	Wilcoxon		Paired t-test			Wilcoxon		Paired t-test		
Projeto	**p-value**	**Wilcoxon**	**t**	**df**	**p-value**	**p-value**	**Wilcoxon**	**t**	**df**	**p-value**
Geral	1,000	-0,004	-8,839	116.387	1,000	1,000	-0,002	-2,835	116.387	0,998
Ant-design	1,000	-0,015	-5,273	7.746	1,000	1,000	-0,010	-3,078	7.746	0,999
Axios	1,000	-0,085	-3,336	119	0,999	1,000	-0,059	-2,299	119	0,988
Bitcoin	1,000	-0,001	-3,838	13.147	1,000	0,407	-2,12E-04	-1,918	13.147	0,972
Bootstrap	0,980	-0,003	-0,605	4.666	0,727	0,953	-0,002	-0,472	4.666	0,682
Caffe	0,983	-0,019	-1,763	237	0,960	0,871	-0,009	-1,195	237	0,883
Code-server	**2,14E-30**	**0,007**	0,781	6.479	0,218	**0,000**	**0,007**	1,089	6.479	0,138
Core	1,000	-0,019	-4,028	986	1,000	0,998	-0,008	-1,983	986	0,976
Django	1,000	-0,010	-1,827	274	0,966	1,000	-0,009	-3,004	274	0,999
Dubbo	1,000	-0,011	**3,998**	**1.966**	**3,32E-05**	0,998	-0,007	**4,701**	**1.966**	**1,39E-06**
Electron	1,000	-0,012	-5,062	2.442	1,000	0,995	-0,004	-2,456	2.442	0,993
Emscripten	1,000	-0,009	-7,869	1.711	1,000	1,000	-0,005	-6,419	1.711	1,000
Faceswap	0,877	-0,196	-0,426	38	0,664	0,945	-0,161	-0,431	38	0,665
Flask	0,743	-0,006	-0,153	623	0,561	0,532	-0,003	0,521	623	0,301
Googletest	1,000	-0,006	-2,669	903	0,996	1,000	-0,004	-1,616	903	0,947
Hashcat	1,000	-0,014	-9,677	2.002	1,000	1,000	-0,010	-8,817	2.002	1,000
Hibernate-orm	0,694	-0,005	-2,392	1.063	0,992	0,295	-0,002	-2,339	1.063	0,990
Jq	0,962	-0,023	-0,928	47	0,821	0,966	-0,033	-1,435	47	0,921
Jquery	1,000	-0,025	-3,041	246	0,999	1,000	-0,022	-3,329	246	0,999
Json	1,000	-0,004	-2,920	828	0,998	0,790	-0,001	-1,289	828	0,901
Junit5	0,795	-0,073	-1,079	75	0,858	0,743	-0,061	-1,216	75	0,886
Keras	1,000	-0,013	-3,816	865	1,000	1,000	-0,009	-3,303	865	1,000
Libuv	0,096	-0,002	1,566	289	0,059	**0,043**	**2,24E-04**	1,547	289	0,062
Lombok	1,000	-0,036	-6,259	383	1,000	1,000	-0,024	-5,598	383	1,000
Magisk	0,420	-0,056	-0,690	25	0,752	0,823	-0,106	-1,389	25	0,911
Mockito	0,990	-0,042	-3,144	261	0,999	0,880	-0,024	-2,276	261	0,988
Nest	1,000	-0,005	-16,821	15.193	1,000	1,000	-0,005	-17,249	15.193	1,000
Netdata	0,501	-0,006	-0,688	259	0,754	0,073	-3,04E-04	-0,210	259	0,583
Netty	0,926	-0,050	-1,313	57	0,903	0,927	-0,051	-1,467	57	0,926
Obs-studio	1,000	-0,009	-4,310	481	1,000	1,000	-0,006	-3,994	481	1,000
Stb	1,000	-0,012	-4,441	303	1,000	1,000	-0,009	-3,967	303	1,000

Continua...

	Recall: recall(SVN) > recall(Git)					F-Measure: f-measure(SVN) > f-measure(Git)				
	Wilcoxon		Paired t-test			Wilcoxon		Paired t-test		
Projeto	**p-value**	**Wilcoxon**	**t**	**df**	**p-value**	**p-value**	**Wilcoxon**	**t**	**df**	**p-value**
Pandas	1,000	-0,008	-8,829	1.930	1,000	1,000	-0,004	-7,405	1.930	1,000
Protobuf	0,999	-0,003	0,614	7.375	0,270	0,903	-0,001	1,407	7.375	0,080
React	0,821	-0,006	-0,708	722	0,760	0,758	-0,004	-0,667	722	0,747
Redis	1,000	-0,015	-8,586	1.389	1,000	1,000	-0,009	-7,624	1.389	1,000
Redux	0,986	-0,037	-2,358	173	0,990	0,986	-0,025	-2,275	173	0,988
Requests	1,000	-0,011	-3,058	477	0,999	0,978	-0,004	-1,657	477	0,951
Retrofit	0,496	-0,006	-0,300	77	0,617	0,319	-0,004	-0,856	77	0,803
RxJava	0,999	-0,008	-1,348	443	0,911	0,984	-0,005	-0,570	443	0,716
Scrapy	1,000	-0,015	-7,009	2.919	1,000	0,995	-0,006	-3,207	2.919	0,999
Scrcpy	1,000	-0,122	-3,294	46	0,999	1,000	-0,091	-3,113	46	0,998
Socket.io	1,000	-0,023	-3,367	133	1,000	0,996	-0,010	-2,773	133	0,997
Spring-boot	**2,59E-08**	**0,002**	0,981	29.456	0,163	**1,23E-14**	**0,003**	2**,849**	**29.456**	**0,002**
Spring-framework	1,000	-0,018	-6,704	1.953	1,000	1,000	-0,013	-5,224	1.953	1,000
Terminal	1,000	-0,033	-6,814	1.363	1,000	1,000	-0,011	-2,685	1.363	0,996
Tesseract	1,000	-0,026	-2,965	164	0,998	0,998	-0,019	-2,933	164	0,998
Thefuck	0,943	-0,033	-2,042	94	0,978	0,676	-0,011	-1,389	94	0,916
Vue	0,657	-0,237	-0,829	26	0,793	0,750	-0,188	-0,946	26	0,824
X64dbg	**3,55E-32**	**0,402**	**17,299**	**966**	**6,49E-59**	**7,95E-38**	**0,406**	**18,053**	**966**	**2,67E-63**
Youtube-dl	1,000	-0,012	-3,068	717	0,999	1,000	-0,006	-1,346	717	0,911
Zstd	0,126	-4,83E-04	1,092	2.201	0,138	**9,08E-04**	**0,001**	**1,802**	**2.201**	**0,036**