Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Sistemas de Informação

Análise da correlação entre os sentimentos dos desenvolvedores e a qualidade de código-fonte em repositórios Java

Antônio Henrique Passamai Penizollo

JUIZ DE FORA
JANEIRO, 2023

Análise da correlação entre os sentimentos dos desenvolvedores e a qualidade de código-fonte em repositórios Java

Antônio Henrique Passamai Penizollo

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciências da Computação

Bacharelado em Sistemas de Informação

Orientador: Gleiph Ghiotto Lima de Menezes

Coorientador: José Maria Nazar David

JUIZ DE FORA

JANEIRO, 2023

Análise da correlação entre os sentimentos dos desenvolvedores e a qualidade de código-fonte em repositórios Java

Antônio Henrique Passamai Penizollo

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Aprovada por:

Gleiph Ghiotto Lima de Menezes
Doutor em Computação

José Maria Nazar David
Doutor em Engenharia de Sistemas e Computação

Alessandreia Marta de Oliveira Julio
Doutora em Computação

Fabricio Martins Mendonça
Doutor em Ciência da Informação

JUIZ DE FORA

9 DE JANEIRO, 2023

Aos meus amigos e irmãos.

Aos pais, pelo apoio e sustento.

Resumo

Os Sistemas de Controle de Versão (SCV) possuem repositórios de softwares que contém informações históricas de projetos. Através desses repositórios é possível minerá-los para entender como a qualidade de código varia de acordo com o tempo. Com o histórico disponível, é possível calcular a polaridade do sentimento dos desenvolveres por meio das mensagens dos seus *commits*. Este trabalho tem como objetivo investigar a correlação entre polaridade dos sentimentos dos desenvolvedores e a variação das métricas da qualidade de código-fonte da cada *commit*. Para tanto, foi desenvolvida uma solução para capturar as variações das métricas de qualidade de código e as polaridades das mensagens de *commit*. Essa solução analisou três repositórios abertos em Java e os valores encontrados para as correlações não foram significativos.

Palavras-chave: métricas de qualidade; código-fonte; mineração de repositórios de software; análise de sentimento.

Abstract

Version Control Systems (SCV) have software repositories that contain historical project information. Through these repositories, it is possible to mine them to understand how code quality varies over time. With the available history, it is possible to calculate the polarity of the developers' sentiment through the messages of their commits. This work aims to investigate the correlation between the polarity of developers' feelings and the variation of source code quality metrics for each commit. For that, we developed a solution to capture the variations of the code quality metrics and the polarities of the commit messages. This solution analyzed three open repositories in Java, and the values found for the correlations were not significant.

Keywords: quality metrics; source code; mining software repositories; sentiment analysis.

Agradecimentos

A todos os meus parentes, pelo encorajamento e apoio.

Aos professores Gleiph e José Maria pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

"Lembra que o sono é sagrado e alimenta de horizontes o tempo acordado de viver".

Beto Guedes (Amor de Índio)

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

AS	Análise de Sentimento
BS	Backward
DCC	Departamento de Ciência da Computação
FS	Forward Snowballing
MRS	Mineração de Repositórios de *software*
SCV	Sistema de Controle de Versão
UFJF	Universidade Federal de Juiz de Fora

1 Introdução

1.1 Apresentação do tema

Independente da metodologia de desenvolvimento de software utilizada, monitorar a qualidade de código-fonte, manutenibilidade são essenciais para construir softwares que garantam qualidade. Com a ascensão de metodologias ágeis a partir da criação do Manifesto Ágil de Fowler, Highsmith et al. (2001), é desenvolvida a ideia de realizar entregas contínuas, como citado no princípio 3 "entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo".

Com isso, as entregas são feitas gradualmente para os clientes, e desenvolvedores realizam alterações e trabalham em novas funcionalidades, para um processo de melhoria contínua (MARTIN, 2014). Dessa forma, para realizar o controle do código-fonte desenvolvido, se utiliza de sistemas de controle de versão (SCV) e repositórios de código. Como resultado, surgiram diversas plataformas onde é possível que desenvolvedores compartilhem seus códigos, como o GitHub[1] e Bitbucket[2].

Os SCV possuem repositórios de softwares que contêm informações históricas de um projeto. A partir da análise dos dados históricos de projetos, desenvolvedores, gerentes de projetos e pesquisadores podem encontrar e investigar as boas práticas utilizadas e o que se deve evitar no processo de desenvolvimento de software. Além disso, com grande volume de dados brutos nos repositórios, surgiram ferramentas e estudos para explorar esses dados. Este campo de pesquisa se chama mineração de repositórios (WILLIAMS; HOLLINGSWORTH, 2005).

Com a mineração de repositórios, pesquisadores utilizam a área de Análise de Sentimento(AS), também conhecida como Mineração de Opinião, a qual se utiliza do processamento de linguagem natural, análise de texto e linguística computacional para extrair e quantificar a polaridade dos dados (LIU; ZHANG, 2012). Somado a isso, também

[1] ⟨https://github.com/⟩
[2] ⟨https://bitbucket.org/⟩

surgiram pesquisas para investigar e compreender o ciclo de desenvolvimento e como as métricas de código refletem na qualidade do software (SILVA; CIZOTTO; PARAISO, 2020). Entretanto, não foram encontrados trabalhos que relacionem as métricas de qualidade de código com a análise de sentimento dos desenvolvedores.

1.2 Descrição do Problema

Na literatura foram identificadas diversas pesquisas sobre Mineração de Repositórios de *software* (MRS) e Métricas de *Software* (SILVA; CIZOTTO; PARAISO, 2020) e também sobre Análise de Sentimento (AS) em repositórios (SINHA; LAZAR; SHARIF, 2016) (HUQ; SADIQ; SAKIB, 2020). Entretanto, apesar de diversos estudos sobre métricas de qualidade de código e análise de sentimento, não foi encontrado nenhum trabalho que realizasse e gerasse a correlação das métricas de código e as polaridades de sentimento dos desenvolvedores. Portanto, a ausência de estudos e ferramentas que dão um *feedback* para entender o sentimento dos desenvolvedores dificultam gerentes de projeto a tomar decisões sobre a qualidade do código-fonte.

1.3 Motivação e Objetivo

A partir de estudos e ferramentas que gerem *feedbacks* sobre o sentimento do desenvolvedor e a qualidade do código-fonte entregue, gestores podem conseguir entender quais programadores necessitam de um acompanhamento, já que existe uma informação prévia que a qualidade de código está sendo comprometida. Logo, gestores de projetos ou líderes técnicos podem compreender e monitorar sua equipe para criar estratégias voltadas para a qualidade de código.

Portanto, através da união entre a mineração de repositórios, qualidade de software e a análise de sentimento, esta monografia tem o objetivo de investigar o sentimento dos desenvolvedores no processo de desenvolvimento de software para entender se existe uma correlação entre a qualidade do código-fonte, fornecida por métricas de software, com o sentimento do desenvolvedor extraído a partir de mensagens de *commit*.

O objetivo será atingido a partir do mapeamento do estado da arte na literatura

relacionada à análise de sentimento e métricas de qualidade de código-fonte, estudo de algoritmos e ferramentas de análise de sentimento, a escolha de quais métricas de softwares tendem a serem mais coerentes para quantificar qualidade do código-fonte e quais ferramentas podem ser utilizadas para a extração dessas métricas. Sendo assim, foi proposta como questão de pesquisa principal, verificar se existe correlação entre a polaridade dos sentimentos e a variação das métricas de qualidade extraídas de cada versão. Como questões de pesquisas secundárias, foram propostas as seguintes questões:

- (QP1) identificar qual é a distribuição de polaridade do sentimento encontrada nas mensagens de *commit* dos projetos selecionados.
- (QP2) qual é a distribuição da variação das métricas de qualidade de software no histórico dos projetos selecionados.

1.4 Organização

Este trabalho está organizado em cinco capítulos. Neste capítulo foi feita a introdução. No Capítulo 2 é apresentada a fundamentação teórica. No Capítulo 3 é apresentada a abordagem proposta e as decisões arquiteturais do projeto que realiza a mineração das métricas de código-fonte e polaridade dos sentimentos. O Capítulo 4 apresenta as questões de pesquisa, a avaliação dos experimentos, seleção dos projetos utilizados, discussões dos resultados e as ameaças a validade. Por fim, no Capítulo 5, são apresentas contribuições deste Trabalho de Conclusão de Curso e as sugestões de trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo são apresentados os conceitos que apoiam a leitura deste trabalho. A Seção 2.1 define o que são Sistemas de Controle de Versão (SCV). A Seção 2.2 apresenta o campo de pesquisa chamado Mineração de Repositórios. A Seção 2.3 discute o que são métricas de software e a definição das métricas utilizadas neste trabalho. Já na Seção 2.4, é discutido sobre qualidade de código-fonte medidas através das métricas de software. Na Seção 2.5, é realizada uma apresentação das ferramentas de extração de métricas de código-fonte avaliadas para este trabalho. Na Seção 2.6, é feito um detalhamento da ferramenta utilizada, e a Seção 2.7 discute conceitos e ferramentas de análise de sentimento. Por fim, a Seção 2.8 apresenta um estudo sobre trabalhos relacionados, e concluída esta seção com as considerações finais do capítulo na Seção 2.9.

2.1 Sistemas de Controle de Versão (SCV)

O Sistema de Controle de Versão é utilizado para gerenciar as versões de um projeto ao longo do seu ciclo de vida, permitindo que equipes de desenvolvimento possam rastrear as alterações realizadas por cada desenvolvedor (CHACON; STRAUB, 2014). Os SCVs armazenam em um repositório os artefatos de software, que podem ser documentos de projetos ou códigos fonte. O repositório é o local onde o SCV mantém todo o histórico do projeto, registrando qualquer alteração realizada em cada item versionado. Os desenvolvedores não trabalham diretamente nos arquivos do repositório, pois existe uma área de trabalho que contém a cópia dos arquivos do projeto, a qual é monitorada para identificar as mudanças realizadas. Vale ressaltar que cada desenvolvedor possui a sua própria área de trabalho.

A comunicação entre a área de trabalho e o repositório é efetuada através dos comandos de *commit* e *update* nos SCV centralizados. O *commit* é responsável pelo envio das modificações realizadas na área de trabalho para o repositório e o *update* envia as alterações realizadas do repositório para a área de trabalho. O desenvolvimento não

interfere em nada no repositório, já que antes de efetivamente realizar o *commit* todas as alterações ficam na área de trabalho, como ilustrado na Figura 2.1.

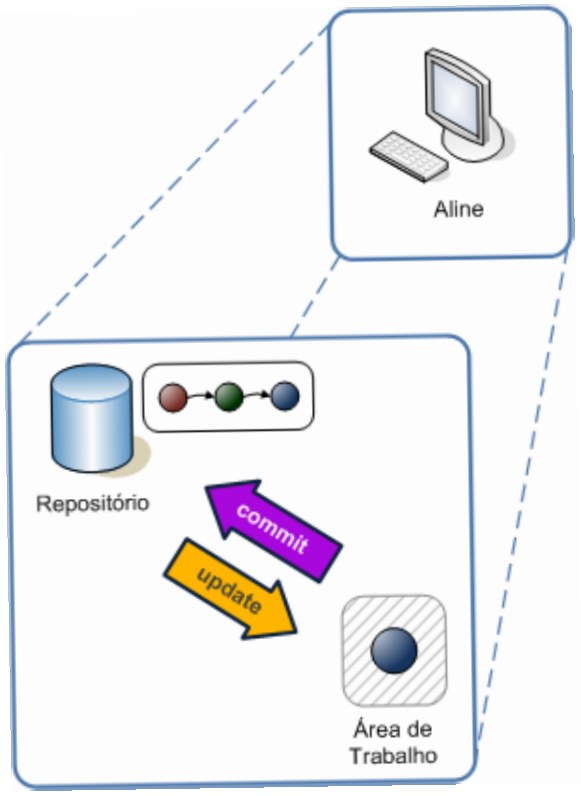

Figura 2.1: Área de trabalho e repositório se comunicando (DIAS, 2016)

Cada *commit* gera uma nova versão do projeto, contendo todas as modificações que o desenvolvedor realizou no *commit*, isto é, alterações que o desenvolvedor decidiu enviar para o repositório. Cada versão possui dados como um identificador único, a data em que foi realizado o *commit* e o autor.

Existem dois tipos de SCV, o centralizado e o distribuído. No primeiro, existe um único repositório central e várias cópias da área de trabalho, enquanto no segundo, cada repositório possui uma área de trabalho acoplada e as operações de *commit* e *pull* acontecem localmente entre eles. Nos SCV distribuídos, um repositório pode se comunicar com qualquer outro, utilizando a operação de *pull* e *push*. O comando *pull* atualiza o repositório local com as alterações feitas no repositório de origem, e o *push* envia as alterações realizadas localmente para o repositório de destino. A Figura 2.2 ilustra os desenvolvedores que recebem e enviam revisões entre eles através de *pull* e *push*.

Cada versão, possui um identificador único. No caso dos SCV distribuídos, esse

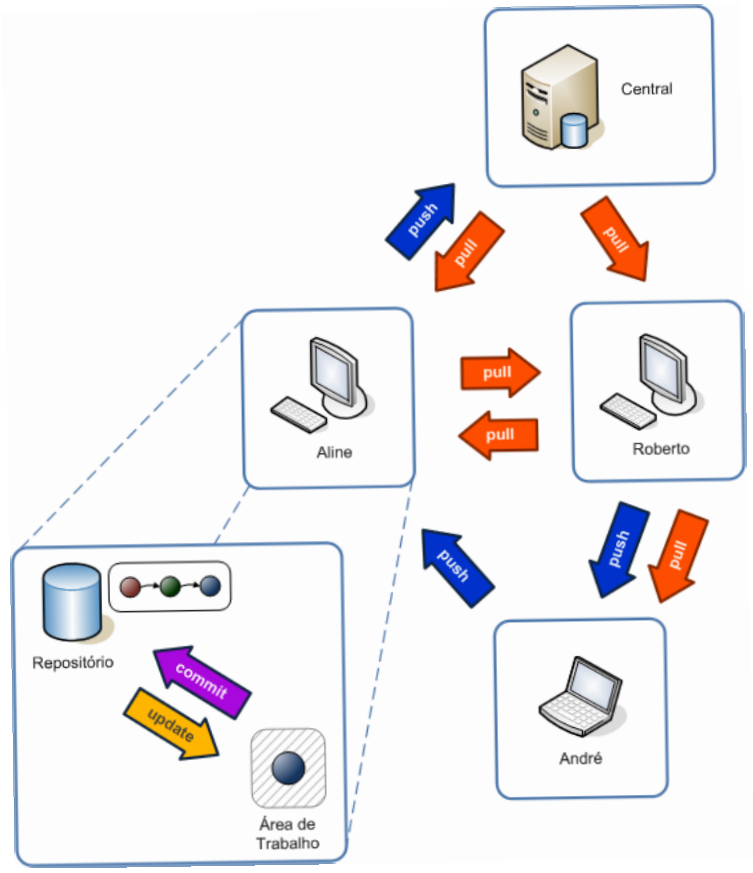

Figura 2.2: Sistema de controle distribuído (DIAS, 2016)

identificador único é representado através do *hash do commit*, que produz um número de 160 bits (40 dígitos na forma hexadecimal). A Figura 2.3 ilustra a execução do comando *git log* no projeto *JMeter*[3] para exemplificar algumas informações de uma versão do projeto. Em destaque está o *hash do commit*. Adicionalmente, é possível identificar a mensagem escrita pelo desenvolvedor, no caso desse *commit* é *Added third party lib properties in build file*, o autor *Michel Stove* e a data em que foi realizado o *commit*. Somado a isso, cada *commit* possui pelo menos um pai(do inglês *parent*). O pai de um *commit* é o *commit* sobre o qual o *commit* é aplicado. *Commits* que não possui nenhum pai são o *commit* raiz do projeto, *commits* com um único pai, são *commits* comuns, e *commits* com mais de um pai, são os *merges*, que serão definidos nesta seção.

Uma vantagem dos SCV é permitir que desenvolvedores trabalhem paralelamente em um mesmo arquivo, e que códigos não sejam sobrescritos (MENS, 2002). A solução para não ocorrerem sobrescritas de código, garantindo assim que não reapareça defeitos ou

[3]⟨https://github.com/apache/jmeter⟩

```
commit 30bdfe423a8a0b948eb1fa3a6817293281548 66c
Author: Michael Stover <mstover1@apache.org>
Date:   Thu May 20 13:38:26 2004 +0000

    Added third party lib properties in build file

    git-svn-id: https://svn.apache.org/repos/asf/jakarta/jmeter/trunk@324567 13f79535-47bb-0310-9956-ffa450edef68

    Former-commit-id: 9eaeecd9ddb909cb487eeba23c3572f461fbb1a5
```

Figura 2.3: *git log* no projeto *JMeter*

perda de funcionalidade, é a combinação das revisões concorrentes em uma única versão resultante, isso é conhecido como *merge*. Porém, caso os dois desenvolvedores tenham trabalhado em um mesmo trecho de código, o *Git* não conseguirá efetuar a combinação. Neste caso, ocorre um conflito de controle de versão, sendo necessário que algum desenvolvedor faça a análise do trecho de código em conflito e realize o *merge* manualmente.

Para exemplificar um caso de mesclagem, antes é necessário definir o conceito de *branches*, também conhecido como ramificações. Na definição do livro Pro Git de (CHACON; STRAUB, 2014), "ramificação significa que você diverge da linha principal de desenvolvimento e continua a trabalhar sem alterar essa linha principal". Logo, quando for necessário criar uma funcionalidade, ou resolver um *bug*, basta criar uma ramificação para encapsular essas mudanças em um ramo. Na Figura 2.4, existe um ramo chamado *1.1.0-release* que representa a *release* de um projeto, enquanto a ramificação *main* representa a versão principal, isto é, mais estável do projeto. Dessa forma, é possível que os desenvolvedores trabalhem e testem a ramificação *1.1.0-release* sem comprometer a versão mais estável da aplicação. Além disso, perceba que o ramo *1.1.0-release* foi criado a partir do ramo *main*.

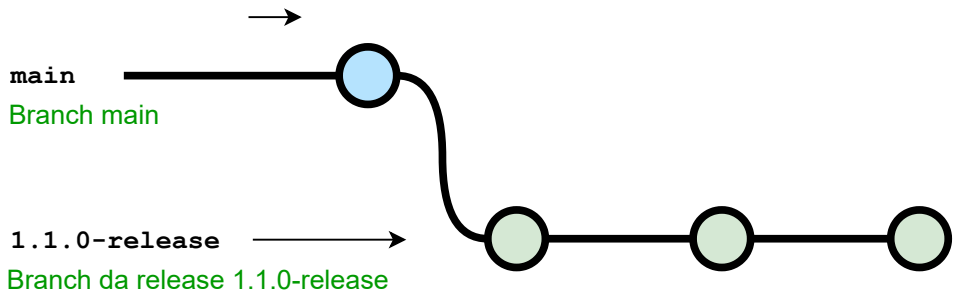

Figura 2.4: Repositório com as ramificações *main* e *1.1.0-release*

Após demonstrar o que são ramificações, a mesclagem é usada para mesclar uma ou mais ramificações na ramificação que você verificou. Ela então avançará a ramificação atual para o resultado da mesclagem (CHACON; STRAUB, 2014). O comando *git merge* permite que se una o que foi desenvolvido na *release 1.1.0* com a ramificação *main*, resultando em um novo *commit* representado pela Figura 2.5.

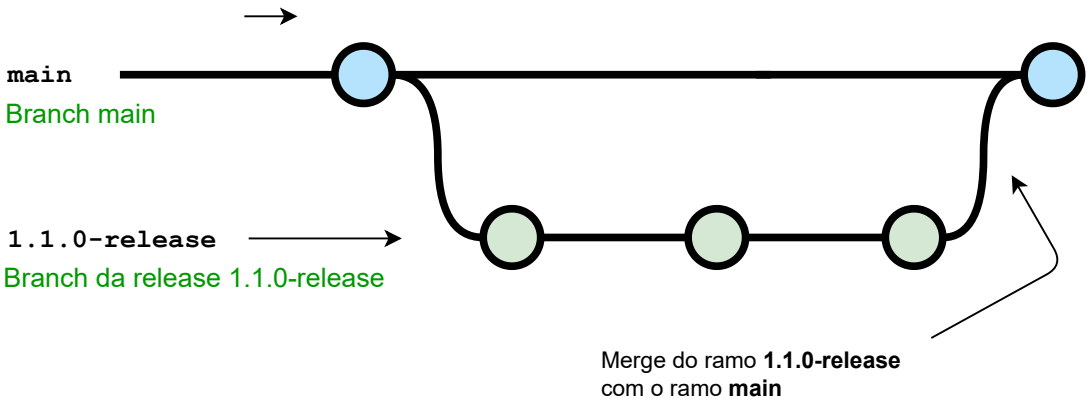

Figura 2.5: Merge entre *main* e *release 1.1.0*

Por fim, a Figura 2.6 representa um conjunto de ramificações para ilustrar a evolução de um software até ser realizado um *commit* para a ramificação principal. Para tanto, foi criada uma ramificação chamada *tarefa-34* a partir do ramo *develop*. Após finalizar a tarefa, foi realizado um *merge* para o ramo *develop*, depois outro *merge* para a ramificação *release* e por fim, um último *merge* para a ramificação *main*.

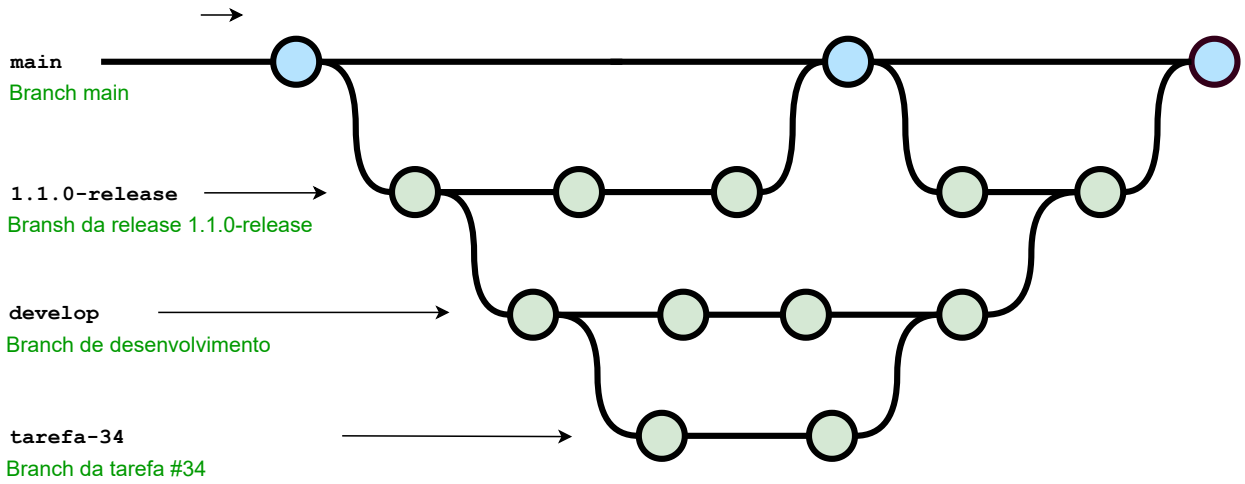

Figura 2.6: Representação de um ciclo de desenvolvimento iniciando na *branch tarefa-34* e finalizando na *branch main*

Após definir o que são SCV, abaixo são listadas algumas das vantagens em se utilizar o SCV no processo de desenvolvimento de software (ATLASSIAN, 2022):

- históricos de alteração: os SCVs possuem um histórico de tudo que foi alterado, excluído ou adicionado. Dessa forma, desenvolvedores conseguem acompanhar, por exemplo, o ciclo de vida de uma dada classe em Java. Além disso, o histórico das alterações é acompanhado pelo nome do autor do *commit*, a data da modificação e a mensagem do *commit* que apresenta o objetivo da alteração;

- ramificação e mesclagem: O trabalho simultâneo pode gerar conflitos, sobrescrever código, reaparecimento de defeitos ou perdas de funcionalidades. Com a criação de ramos são gerados fluxos de trabalhos independentes. Quando o trabalho de um dos ramos foi desenvolvido e testado pode ser realizada uma mesclagem ou *merge*;

- rastreabilidade: A partir do histórico de alterações é possível rastrear cada alteração feita no código-fonte, logo, é possível rastrear *bugs* de maneira mais eficiente.

2.2 Mineração de Repositórios

A mineração de repositórios de software (MSR) analisa a evolução do software através de técnicas de mineração dos dados históricos dos desenvolvedores nos repositórios (HASSAN, 2008).

O campo da MSR está crescendo graças ao advento dos projetos de código aberto e ao crescimento de plataformas como o *GitHub*, facilitando o acesso a repositórios de grandes projetos (HASSAN, 2008). Sendo assim, a abundância de dados em repositórios despertou um grande esforço de pesquisas na área de mineração de repositórios (VOINEA; TELEA, 2006).

A partir da MSR é possível extrair conhecimento com o intuito de compreender quais práticas e metodologias de desenvolvimento de softwares são eficientes. Já que está disponível uma grande fonte de dado do estado atual e todo o histórico de alteração de um código, é possível desenvolver diversas pesquisas e ferramentas para auxiliar no desenvolvimento de software.

Estudos de D'Ambros et al. (2008) cita alguns tópicos abordados no campo de análise de repositórios de software, são eles: esforço do desenvolvedor e análise de redes sociais, nesse tópico, é analisado o esforço dos programadores de uma equipe na manutenção e a evolução do software e como eles se comunicam entre si. Outro tópico estudado é o impacto e propagação da mudança em um software, onde é avaliado o impacto da adição de um novo recurso ou a alteração de um recurso existente.

Existem diversos dados que podem ser explorados a partir de repositórios abertos, como mensagens de *commits* dos desenvolvedores e o código-fonte produzido pelo desenvolvedor. Dessa forma, quando se escolhe utilizar o código-fonte como material de estudo, as métricas de software podem contribuir para encontrar resultados quantitativos que podem indicar boas ou más práticas de desenvolvimento.

2.3 Métricas de Software

Segundo Fenton e Bieman (2019), "Métricas de software são parâmetros para a medição do desempenho de um software. Uma métrica é um padrão de medida do grau em que um sistema ou processo de software é dotado de uma dada propriedade". Sendo assim, utilizar métricas pode ser uma grande vantagem quando necessita quantificar resultados.

De acordo com Henderson-Sellers (1995), as métricas podem ser agrupadas em três grandes grupos. São eles: métricas de processo, métricas de projeto e métricas de produto. As métricas de processos se referem ao processo de desenvolvimento do software, metodologia e tempo de desenvolvimento. Já as métricas de projeto monitoram a situação do projeto. Por fim, as métricas de produto descrevem o código-fonte em qualquer fase de seu desenvolvimento, elas medem o tamanho do código, desempenho, complexidade e capacidade de manutenção (HENDERSON-SELLERS, 1995). As métricas também podem ser classificadas quanto aos critérios de utilização, podendo ser objetivas e subjetivas. As objetivas são geradas a partir de regras definidas, já as subjetivas, podem partir de um valor, porém, necessitam de um julgamento para serem levantadas. Um exemplo de uma métrica subjetiva pode ser a complexidade de código. A métrica pode ser definida no intervalo de 0 a 100, por exemplo, entretanto, ainda assim, é necessário a definição de quanto é considerado baixa complexidade. As métricas objetivas podem

ser calculadas automatizadamente, como a métrica *TLOC*, que calcula o número total de linhas de código, ignorando comentários, espaços em branco e diferenças de formatação.

A partir desta definição e considerando que toda ciência busca por medições quantitativas, há um esforço em quantificar o desenvolvimento de software através de métricas. A partir desta medição, objetiva e quantitativa, é possível aferir qualidade de código. Segundo Rawat, Mittal e Dubey (2012), as métricas de softwares ideais devem ser simples, bem definidas e objetivas e não podem ser de difícil extração.

Métricas de software são úteis para gerar conhecimento e auxiliar na tomada de decisão. Por exemplo, um gerente de projeto com acesso a métricas de software, que mostram que um desenvolvedor está produzindo abaixo da média dos demais desenvolvedores, pode tomar uma decisão para entender o porquê daquele desenvolvedor não está sendo produtivo. Um artigo feito por Oliveira et al. (2017), publicou um estudo utilizando 348 métricas de software para medir a qualidade do código com o propósito de quantificar o esforço dos alunos de um curso de programação em C. A partir deste estudo foram criados perfis de alunos a partir de métricas que mensuravam a dificuldade e competências de cada aluno. A partir do perfil de cada aluno e de uma série de indicadores de aprendizagem, o professor realizou as devidas intervenções para que o aluno continuasse progredindo.

2.4 Qualidade de Código-Fonte

Para Sommerville et al. (2011), "a medição de software preocupa-se com a derivação de um valor numérico ou o perfil para um atributo de um componente de software, sistema ou processo. Comparando esses valores entre si e com os padrões que se aplicam a toda a organização, pode-se tirar conclusões sobre a qualidade do software ou avaliar a eficácia dos métodos, das ferramentas e dos processos de software". Dessa forma, as métricas de código são uma excelente escolha para se medir qualidade de código, já que o resultado de sua análise garante um valor quantitativo e imutável para o cenário e contexto que foi calculada. Isto é, para uma métrica calculada a partir de uma classe *Java* localizada em um *commit*, o valor da métrica será sempre o mesmo.

Independente da metodologia utilizada para desenvolvimento de software, é necessário monitorar a qualidade de código. Um código compilado e que apresenta uma

ótima cobertura de teste só pode apresentar o funcionamento atual e que corresponde com as regras de negócios estabelecidas, porém, não garante manutenibilidade, modularidade e flexibilidade. Neste contexto, são as métricas de software que podem garantir o monitoramento da qualidade de software.

Beck (2007) afirma que desenvolvedores tomam decisões de código enquanto estão programando e essas decisões influenciam na qualidade de código. Além disso, Baldwin e Clark (2006) e Meirelles et al. (2010) mostraram que a motivação para o desenvolvedor se interessar por um projeto de software livre é criticamente afetada pela modularidade de seu código. Portanto, códigos com baixa qualidade e alta complexidade podem afastar desenvolvedores que buscam contribuir em projetos de código aberto.

Na Figura 2.7, é representado o modelo de qualidade de software de Fenton e Bieman (2014). Este modelo classifica a *Operação do Produto* e *Revisão do Produto*, derivadas em itens de qualidade, como usabilidade, confiabilidade, eficiência e reusabilidade, que também são conhecidos como requisitos não-funcionais. Além disso, conforme a Figura 2.7, as métricas podem ser construídas para a avaliação desses itens.

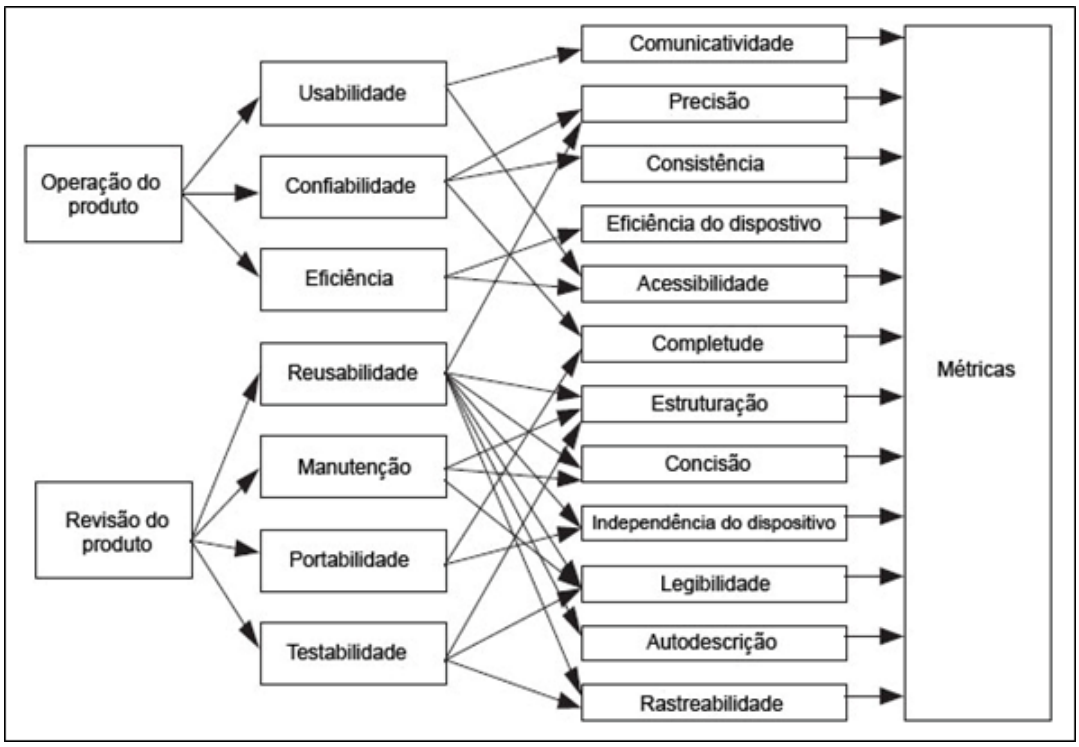

Figura 2.7: Modelo de Qualidade de Software, adaptado de Fenton e Bieman (2014, p.19)

Logo, segundo a Figura 2.7 e estudos de Kitchenham (2010) comprovam que as

métricas podem se relacionar com itens de qualidade de software.

2.5 Ferramentas de Métricas de Código

Nesta seção serão apresentadas algumas das ferramentas avaliadas para extrair as métricas de código dos projetos selecionados. Para este trabalho foram definidos os seguintes requisitos para a ferramenta de extração de métricas:

- **R1**: a ferramenta deve analisar um projeto sem a necessidade de compilá-lo, ou mesmo ser compilável;
- **R2**: a ferramenta deve analisar uma classe isoladamente;
- **R3**: a saída resultante da ferramenta, após extrair as métricas, deve ser um arquivo que possa ser manipulado por outras aplicações. Exemplo, o produto final deve ser um arquivo *.json* ou *.xml*;
- **R4**: a ferramenta deve ser software livre, disponível sem qualquer tipo de restrição.

Na Tabela 2.1, estão representadas as ferramentas avaliadas para realizar a extração das métricas de software, onde as colunas de R1 até R4 representam os requisitos listados acima e a coluna "Qnt Métricas", representa a quantidade de métricas que a ferramenta pode analisar. Nesta tabela o (X) significa que a ferramenta preencheu o requisito e o traço (-) significa que não foi possível verificar.

Ferramenta	**Linguagens**	**R1**	**R2**	**R3**	**R4**	**Qnt Métricas**
Analizo[4]	C, C++, JAVA	X			X	25
Analyst4j[5]	JAVA	-	-	-	-	-
CCCC[6]	C++, Java		X	.html	X	10
CK Java Metrics[7]	Java	X		.csv	X	38
JaSoMe[8]	Java	X	X	.xml	X	70

Tabela 2.1: Lista de ferramentas de extração de métricas de software

Foram avaliadas cinco ferramentas que realizam a coleta de métricas de código. São elas: Analizo, Analyst4j, CCCC, CK e JaSoMe. Após realizar a análise dos requisitos levantados, foi escolhida como ferramenta para este Trabalho de Conclusão de Curso a ferramenta *JaSoMe*, visto que ela preencheu os requisitos R1, R2, R3 e R4, além de possuir uma quantidade relevante de métricas.

2.6 JaSoMe: Java Source Metrics

O JaSoMe é um analisador de código-fonte que extrai métricas internas de qualidade de projetos baseados apenas no código-fonte. Isso distingue o JaSoMe de ferramentas semelhantes por não exigir que o projeto seja compilado primeiro, ou mesmo compilável (HILTON; UFER, 2013).

A maioria dos analisadores trabalha apenas em projetos que compilam com sucesso, o que no caso de projetos Java significa que todas as dependências devem ser satisfeitas, quaisquer bibliotecas externas estão corretamente no caminho de classe, quaisquer utilitários de geração de código devem ser executados corretamente e assim por diante. No entanto, quando tudo o que precisamos são as métricas de qualidade para o código-fonte Java, esse requisito de compilação é desnecessário, pois a maioria das métricas pode ser derivada apenas do código-fonte (HILTON; UFER, 2013).

A execução do *JaSoMe* é relativamente simples. É feita a busca por todos os arquivos com extensão *.java*. Em seguida, é executado o analisador de métricas que retorna o valor de cada métrica para cada pacote, classe e método, por fim, é gerado um arquivo *XML*.

É possível executar a ferramenta apontando para a raiz do projeto como também para um pacote ou classe específica. A Listagem 2.1 apresenta um exemplo de um arquivo *XML* onde foi executada a ferramenta JaSoMe na classe Version localizada no projeto *Junit4*[9]. O XML é um exemplo para ilustrar a estrutura resultante da ferramenta. Vale ressaltar que foi removida a maioria das métricas de classes e métodos do XML, para demonstrar apenas a estrutura final gerada.

Executando somente a classe *Version*, são geradas as métricas de pacote ao qual a classe pertence. Na linha 7 é listado o nome do pacote, neste caso é o *junit.runner*. Da linha 8 até a 10, existe a listagem das métricas do pacote *junit.runner*. Já na linha 13 até a linha 15, é feita a listagem das métricas de classe para a classe que está sendo executada. Caso fosse executada a ferramenta dentro de algum pacote, existiria uma listagem de classes. Na linha 18 até a linha 20 é apresentada uma listagem das métricas de método para o método cuja assinatura é *private Version Version()*. Vale notar que

[9] ⟨https://github.com/junit-team/junit4⟩

na linha 17 também são representados os parâmetros do método, caso exista, juntamente com a linha de início(*lineStart*) e a linha final(*lineEnd*).

No README.md do projeto existe a listagem completa com a descrição e as referências bibliográficas de cada métrica e como elas são calculadas. Além disso, existe uma série de métricas ainda em andamento ou que o autor do projeto pretende calcular.

A ferramenta *JaSoMe* trabalha com diversas métricas, porém, neste documento será dado foco nas métricas listadas na Tabela 2.2. A tabela possui duas colunas, onde a esquerda é o nome da métrica e sua sigla e a direita é a tradução ou uma breve explicação. Além disso, ao final do texto à direita de cada uma, é citado se a métrica é de projeto, pacote, método ou classe. Porém, podem existir métricas que são de projeto, pacote, método e classe, como a TLOC.

Listagem 2.1: XML da classe Version do projeto Junit4 na versão referente ao *hash* b6a0693

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Project sourceDir="D:\projetos-extrator-experimento\junit4\junit\runner\Version.java">
    <Metrics>
        <Metric description="Total Lines of Code" name="TLOC" value="7"/>
    </Metrics>
    <Packages>
        <Package name="junit.runner">
            <Metrics>
                <Metric description="Total Lines of Code" name="TLOC" value="7"/>
            </Metrics>
            <Classes>
                <Class lineEnd="14" lineStart="6" name="Version" sourceFile=".">
                    <Metrics>
                        <Metric description="Class Relative System Complexity" name="ClRCi" value="1,0"/>
                    </Metrics>
                    <Methods>
                        <Method constructor="true" lineEnd="9" lineStart="7" name="private Version Version()">
                            <Metrics>
                                <Metric description="McCabe Cyclomatic Complexity" name="VG" value="1"/>
                            </Metrics>
                        </Method>
                        <Method constructor="false" lineEnd="13" lineStart="11" name="public static String id()">
                            <Metrics>
                                <Metric description="Number of Control Variables" name="NVAR" value="0"/>
                            </Metrics>
                        </Method>
                    </Methods>
                </Class>
            </Classes>
        </Package>
    </Packages>
</Project>
```

Number of Methods (All) (Ma)	Métodos que podem ser invocados em uma classe (herdados, substituídos, definidos). (classe)
Lack of Cohesion Methods (LCOM)	Uma medida para a Coesão de uma classe. Calculado com o método de Henderson-Sellers, com base no número de conjuntos disjuntos formados pela comparação de métodos com os atributos que eles usam. (classe)
Fan-out (Fout)	O número de métodos imediatamente subordinados a um método. (método)
Fan-in (Fin)	O número de métodos que invocam um método. (método)
Total Lines of Code (TLOC)	O número total de linhas de código, ignorando comentários, espaços em branco e diferenças de formatação. (projeto, pacote, classe, método)
Number of Children (NOCh)	Número de classes que estendem diretamente esta classe (class)
Number of Descendants (NOD)	Número total de classes que têm esta classe como ancestral. (classe)
Number of Parents (NOPa)	Número de classes que esta classe estende diretamente (classe)
McCabe Cyclomatic Complexity (VG)	O número de caminhos possíveis exclusivos por meio do código. (método)
Number of Comparisons (NCOMP)	Número de comparações em um método. (método)
Number of Control Variables (NVAR)	Número de variáveis de controle referenciadas em um método. (método)
McClure's Complexity Metric (MCLC)	NCOMP + NVAR. (método)
Class Relative System Complexity (ClRCi)	Complexidade relativa do sistema de classe. (classe)

Tabela 2.2: Lista de métricas utilizadas neste trabalho

2.7 Análise de Sentimento

A Análise de Sentimento (AS), também conhecida como mineração de opinião (KONTOPOULOS et al., 2013), utiliza o processamento de linguagem natural, análise de texto e linguística computacional para determinar a polaridade de uma expressão como positiva ou negativa (WILSON; WIEBE; HOFFMANN, 2005). A AS pode ser aplicada a diversos campos. Alguns campos em que ela é amplamente utilizada são: o reconhecimento do sentimento através da voz de um cliente, avaliações de produtos em *e-commerce*, comentários em mídias sociais, entre diversos outros (SARLAN; NADAM; BASRI, 2014). A AS é uma das áreas que mais vem ganhando espaço para desenvolver pesquisas. Este crescimento é devido ao advento de métodos de aprendizagem de máquina e a maior disponibilidade de *datasets* para treinamento de algoritmos (SARLAN; NADAM; BASRI, 2014).

O objetivo da AS é classificar as sentenças através de uma polaridade representada por um valor (grau) positivo ou negativo. O valor da polaridade depende do método de classificação. Existem métodos que possuem um valor de saída binário ("positivo ou "negativo) e outros como ternário ("positivo", "negativo"ou "neutro"). Quando a sentença não possui polaridade, é classificada como neutra. Para exemplificar, seguem três sentenças e suas polaridades:

- eu amo chocolate - **positiva**;
- hoje é 3 de setembro - **neutra**;
- eu odeio dias chuvosos - **negativa**.

Além da polaridade, existe um termo chamado de "força"do sentimento, também representado como a saída de métodos de AS. Este valor normalmente varia de -1 a 1, e pode ser utilizado como métrica para entender se um sentimento é positivo, negativo ou neutro.

Para extrair a polaridade de fontes textuais, existem duas técnicas: supervisionada e não supervisionada. A técnica supervisionada exige que seja feita a etapa de treinamento de um modelo onde possui amostras já classificadas. Já a técnica não supervisionada, não é necessária uma amostra previamente rotulada e treinamento de um

modelo. Uma das vantagens nas técnicas não supervisionadas é exatamente não depender de amostras previamente rotuladas. Sendo assim, não mantemos a aplicação restrita ao contexto em que ela foi treinada (BENEVENUTO; RIBEIRO; ARAÚJO, 2015).

Em relação às técnicas não supervisionadas, cabe ressaltar a abordagem baseada em dicionário léxico. É chamado de dicionário porque cada palavra possui um valor quantitativo, representado por -1, indicando um sentimento mais negativo e 1, representando um sentimento mais positivo ou qualitativo ("positivo"/"negativo, "feliz"/"triste"). Abordagens léxicas assumem que cada palavra possui a sua polaridade prévia expressa por um valor numérico ou classe, e este valor independe do contexto que a palavra está inserida (TABOADA et al., 2011).

Na Figura 2.8, é ilustrado um funcionamento da abordagem de dicionário léxico. O processo inicia com o *input* de uma frase, em seguida é feito o processamento de linguagem natural e a pesquisa no léxico dos termos que formam a frase. No final do processo, o método infere qual é a polaridade na frase de entrada.

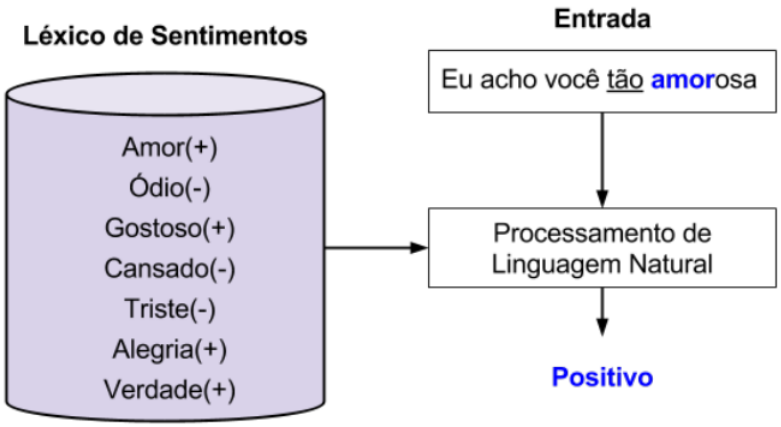

Figura 2.8: Léxico de sentimentos, figura retirada de (BENEVENUTO; RIBEIRO; ARAÚJO, 2015)

2.8 Trabalhos Relacionados

Para encontrar os trabalhos relacionados às métricas de qualidade associadas à análise de sentimento, foram utilizadas algumas técnicas da abordagem de Revisão Sistemática da Literatura realizada no estudo de Mourão et al. (2017). Nesse estudo foi proposta

a estratégia de uma busca hibrida em uma biblioteca digital específica (*Scopus*). Essa estratégia visa preparar uma *string* de busca com base nas questões de pesquisa proposta, realizar a busca em um banco de dados de alguma biblioteca digital, aplicar os critérios de inclusão e exclusão nos estudos recuperados e, em seguida, usando os estudos selecionados como um conjunto de sementes para *Backward* (BS) e *Forward Snowballing* (FS). Neste Trabalho de Conclusão de Curso, foi utilizada a base *Scopus* como biblioteca digital. Após encontrar os estudos, foi realizada a etapa onde é aplicado o processo de seleção dos estudos recuperados para descartar estudos que não possuem relação com a pesquisa proposta. A etapa de BS e FS não foi aplicada neste Trabalho de Conclusão de Curso.

2.8.1 Artigos de Base

Para a Revisão Bibliográfica, foram selecionados alguns artigos de base para construir *string* de busca. Para este propósito, foram selecionados artigos de Mineração de Repositórios de Software relacionados com a qualidade de código, Análise de Sentimento e Métricas de Software para medir a qualidade de código.

Sendo assim, os seguintes artigos foram utilizados como artigos de controle:

- ISLAM, Md Rakibul; ZIBRAN, Minhaz F. Sentiment analysis of software bug related commit messages. Network, v. 740, p. 740, 2018.
- SINHA, Vinayak; LAZAR, Alina; SHARIF, Bonita. Analyzing developer sentiment in commit logs. In: Proceedings of the 13th international conference on mining software repositories. 2016. p. 520-523.
- GUZMAN, Emitza; AZÓCAR, David; LI, Yang. Sentiment analysis of commit comments in GitHub: an empirical study. In: Proceedings of the 11th working conference on mining software repositories. 2014. p. 352-355.

2.8.2 String de Busca

A partir dos artigos de base, foi criada a *string* de busca. Dessa forma, foram utilizados dois parâmetros. São eles: população e saída, onde população, está relacionado com qualidade de *software* medida a partir de métricas de software e o sentimento relacionado,

e a saída representa as técnicas e metodologias utilizadas. Não foi introduzida a etapa de intervenção.

população: (("software quality" OR "software metrics") AND "commit") OR ("software quality" AND "sentiment analysis") OR ("commit" AND "sentiment analysis")

saída: (approach OR method OR methodology OR definition OR experience OR findings OR research OR study OR technique OR knowledge OR tool OR support))

Dessa forma, foi criada a *string* de busca abaixo:

TITLE-ABS-KEY ((("software quality" OR "software metrics") AND "commit") OR ("software quality" AND "sentiment analysis") OR ("commit" AND "sentiment analysis") AND (approach OR method OR methodology OR definition OR experience OR findings OR research OR study OR technique OR knowledge OR tool OR support))

Para essa monografia foi utilizada a base *Scopus*[10] e até a data 27 de setembro de 2022, foram encontrados 136 documentos.

2.8.3 Processo de Seleção

Após executar a *string de busca*, é necessário realizar alguns filtros, isto é, eliminar estudos que não possuem relação e não serão úteis para este trabalho. Dessa forma, para eliminar trabalhos indesejados, o primeiro passo foi realizar a leitura do título e resumo e descartar os artigos que não tinham relação com o trabalho. Com os estudos restantes, foi feita a leitura da Introdução e Conclusão e eliminados novamente os artigos que não tinham relação com o trabalho. Por fim, foi realizada a leitura completa do artigo e, caso ainda tivesse algum estudo fora do domínio de pesquisa, seria descartado.

Os artigos foram classificados utilizando os seguintes critérios de exclusão: artigo que não se relaciona com AS e artigo que não se relaciona com métricas de código-fonte de qualidade. Conforme ilustrado na Figura 2.9, a partir dos 136 documentos, 123 estudos foram descartados, realizando somente a leitura do título e resumo. Dos 13 estudos que restaram do primeiro processo, 3 foram eliminados realizando a leitura da Introdução e Conclusão. Após realizar a leitura dos 10 estudos que restaram, foi concluído que os 10 podem ser úteis para esta monografia. Logo, dos 136 artigos encontrados, 126 foram

[10] ⟨https://www.scopus.com/⟩

excluídos e 10 podem contribuir para este trabalhado.

Segundo o gráfico da Figura 2.9, 10 trabalhos podem ser relevantes para esta monografia, que representam 7.35 porcento. Porém, não foi encontrado nenhum artigo que discute sobre o relacionamento de análise de sentimento com a qualidade de software extraída a partir de uma seleção de métricas de software. Sendo assim, em seguida, serão apresentados estudos de análise de sentimento de desenvolvedores em repositórios de software e trabalhos sobre qualidade de código medida através de uma seleção de métricas de software.

Figura 2.9: Gráfico representando a quantidade de artigos excluídos em uma das etapas.

Sobre análise de sentimento em mensagens de *commits*, foram encontrados em nossa busca os trabalhos: *Analyzing Developer Sentiment in Commit Logs* de Sinha, Lazar e Sharif (2016), *Is developer sentiment related to software bugs: An exploratory study on github commits* de Huq, Sadiq e Sakib (2020). No artigo de Sinha, Lazar e Sharif (2016), os autores investigaram a relação entre os sentimentos dos desenvolvedores a partir de mensagens de *commits* e a linguagem de programação e os dias da semana. Além disso, foi encontrada uma forte correlação entre o número de arquivos alterados e o sentimento expresso pelos *commits* dos quais os arquivos faziam parte. Somado a isso, o

artigo *A Developer Recommendation Method Based on Code Quality* de Silva, Cizotto e Paraiso (2020) oferece uma série de métrica de qualidade que também são calculadas na ferramenta *JaSoMe*, utilizada para extrair métricas nesta monografia. No trabalho *Sentiment polarity and bug introduction* de Romano et al. (2020) foi realizada a investigação da associação entre a polaridade do sentimento dos desenvolvedores e a introdução de *bugs* e foi descoberto que a negatividade dos desenvolvedores está associada à introdução de *bugs*. Logo, para este estudo, o monitoramento contínuo dos desenvolvedores pode evitar novos *bugs*.

2.9 Considerações Finais do Capítulo

Neste capítulo foram apresentados os conceitos de sistemas de controle de versões e suas vantagens no desenvolvimento de software, plataformas de hospedagem de código-fonte como o *GitHub*, conceitos de mineração de repositórios, métricas de software e qualidade de código-fonte. Em seguida, foi apresentada a ferramenta utilizada neste trabalho para realizar a extração das métricas. Por fim, apresentamos os conceitos de análise de sentimento e polaridade e os trabalhos relacionados. O próximo capítulo apresenta a solução proposta para esta pesquisa.

3 Extração de métricas de qualidade e sentimentos de *commits*

Neste capítulo é apresentada a abordagem desenvolvida. Para tanto, ele possui a seguinte divisão: a Seção 3.1 apresenta a visão geral dos dois módulos, como eles se relacionam e a etapa de clone dos projetos. A Seção 3.2 apresenta o módulo Extrator de métricas e as funcionalidades de extrair, calcular e armazenar a variação média de métricas. A Seção 3.3 apresenta o módulo Extrator de sentimento, responsável pela extração de polaridade.

3.1 Visão Geral

A abordagem proposta é dividida em dois módulos: o *Extrator de métrica*, responsável por obter as variações das métricas entre os *commits*, e o *Extrator de sentimento*, que extrai as polaridades dos sentimentos expressos nas mensagens dos *commits*. As métricas de qualidade, apresentadas na Tabela 2.2, são baseadas em classes e métodos, enquanto as polaridades de sentimentos são extraídas utilizando os algoritmos apresentados na Seção 3.3.

A Figura 3.1 é uma representação em alto nível de todo o processo de extração desenvolvido. Em suma, o processo consiste em extrair as métricas, armazená-las e calcular a sua variação média, e calcular a polaridade do sentimento de cada mensagem de *commit*. O primeiro passo do processo de extração é a realização dos clones dos projetos selecionados para este trabalho.

Após a realização do clone do projeto, são iniciados, os dois módulos do projeto (Extrator de métricas e Extrator de sentimento) para executar cada versão dos projetos selecionados. Em cada *commit* processado na etapa do extrator de métricas, ilustrado na Figura 3.1, é realizada a extração das métricas de código da Tabela 2.2 e concorrentemente, cada *commit* processado na etapa extrator de sentimento, é calculada a polaridade da mensagem do *commit*. Ao final da etapa do módulo Extrator de métricas e do módulo

Extrator de sentimento de cada *commit* é realizada a gravação dos resultados no banco de dados *Extrator-DB*. Sendo assim, o fluxo Processo de extração de métricas e polaridade do sentimento da Figura 3.1, é finalizado quando não existir mais *commits* para analisar. Na seção 3.2, será abordada de forma mais detalhada o processo do Extrator de métricas e na seção 3.3 o processo do Extrator de sentimento.

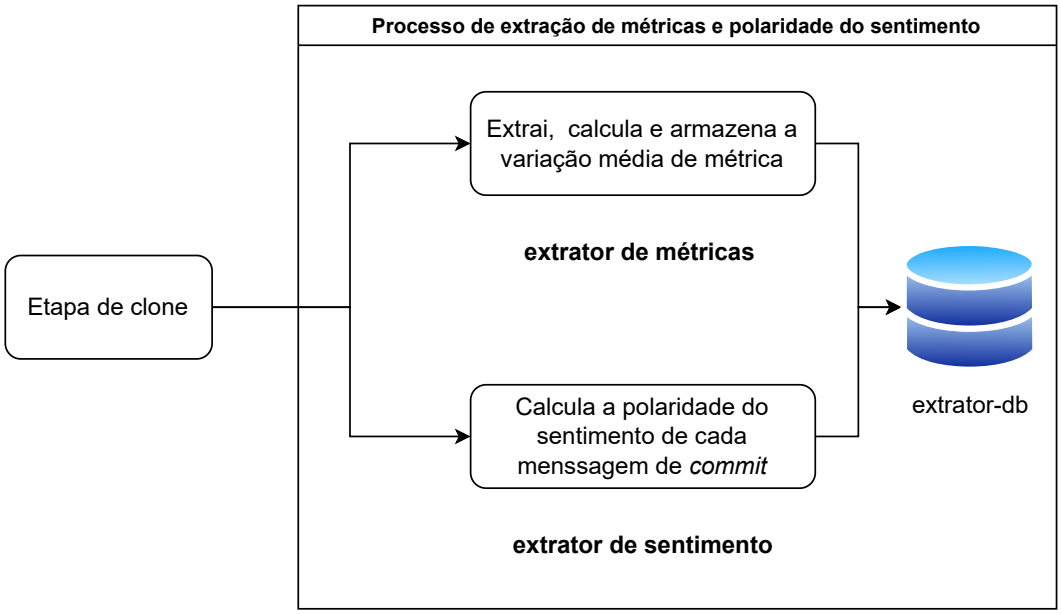

Figura 3.1: Visão geral dos dois módulos do projeto

3.2 O Extrator de métricas

O Extrator de métricas é o módulo responsável por toda a extração e armazenamento de métricas de software de um projeto que utiliza o *SCV Git*. Ele foi implementado utilizando a linguagem *Java* com o *framework Spring Boot* [11], que oferece a geração de um *template* para desenvolvimento e traz diversas vantagens como o gerenciamento de dependências e configurações de bibliotecas. Além disso, foi utilizado o banco de dados *PostgreSQL*[12].

[11] ⟨https://spring.io/projects/spring-boot⟩
[12] ⟨https://www.postgresql.org/⟩

3.2.1 Funcionalidade

O Extrator de métricas foi desenvolvido para extrair e armazenar a variação média das métricas de qualidade de software para as classes e métodos. Este módulo recebe como entrada um repositório de software e o seu histórico é analisado, *commit* a *commit*, para encontrar as variações das métricas.

A visão geral deste módulo é ilustrada pela Figura 3.2 que apresenta desde a obtenção das classes de entrada até o armazenamento no banco de dados. A primeira etapa do módulo é inicializada com a obtenção da versão atual e a versão pai, executando o comando *git checkout [hash do commit]* para cada uma delas.

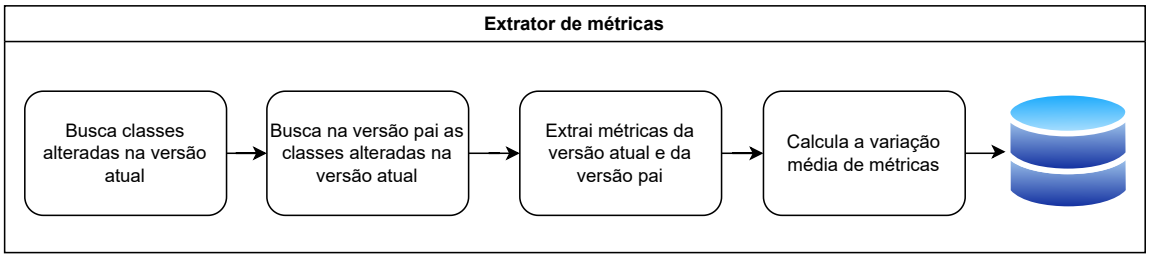

Figura 3.2: Representação do fluxo de extração e cálculo das médias de métricas das classes alteradas com seus métodos.

Em seguida são identificadas as classes e os métodos alterados das duas versões(*commit* atual e *commit* pai) para calcular a variação média das métricas de qualidade deles. Para determinar quais classes foram alteradas em um *commit*, foi utilizado o comando do *Git* apresentado na Listagem 3.1. Esse comando retorna todos os arquivos alterados, inclusive arquivos com extensão diferente de *.java*. Após termos todos os arquivos alterados do *commit*, foi feito um filtro para eliminar todos os arquivos diferentes da extensão *.java*, já que a ferramenta *JaSoMe* só analisa arquivos com essa extensão, conforme citado na Seção 2.6.

Listagem 3.1: Git show para um *hash* específico.

```
git show [hash do commit] --pretty=format:" --name-only
```

Em seguida é feita a extração das métricas de classe e método com o auxílio da ferramenta *JaSoMe*. Para exemplificar, imagine que uma das classes alteradas na versão referente ao *hash af9634a* do projeto *Junit4* foi a *Version.java*. Logo, é feita a extração das

métricas de classe e método de *Version.java* executando o comando *"./jasome [caminho da classe do projeto]"*, o retorno dessa execução foi demonstrado de forma simplificada na Listagem 2.1 no capítulo de Fundamentação Teórica.

A saída da ferramenta *JaSoMe* é dada em um arquivo *XML*, organizado em pacotes, classes e métodos, onde cada elemento possui o seu conjunto de métricas. Portanto, o módulo Extrator de métrica faz o *parser* do documento para identificar o valor das métricas de qualidade da Tabela 2.2. Vale ressaltar que a ferramenta *JaSoMe*[13] possui um número muito maior de métricas, porém, para este Trabalho de Conclusão de Curso, foi selecionada somente as métricas que se enquadravam melhor da definição de qualidade e complexidade de código, como, por exemplo, *VG*, *NCOMP*, *NVAR*, *MCLC* e *CLRCi*.

Para identificar a variação média das classes e métodos é realizada a comparação da versão atual da análise e a versão que a originou (*i.e.,* versão pai). Para encontrar a versão pai de um *commit* foi utilizado o *Git* para buscar todos os *hashes* que são pais do *commit* atual, através do comando: *git log –pretty=%P -n 1 [commit atual]*. Vale ressaltar que algumas versões são originadas da mesclagem de duas ou mais versões, chamadas de versões de *merge*. Os *merges* geralmente apresentam uma mensagem padrão, por exemplo, *"Merge pull request 476 from FelixGV/fix_client_shell_list_printing_logic"* representado pelo *commit 3787b4fc8* no projeto *Voldemort* [14], e devem ser tratados de outra maneira por possuírem mais de um pai. Portando, por questão de tempo, elas foram excluídas desta versão da abordagem. A exclusão dos *commits* com mensagens padronizadas ficará mais claro na explicação da etapa que extração de sentimento.

Considerando as versões que possuem apenas um pai, é inicializada a identificação das variações das métricas para as classes e os métodos. Para tanto, o *JaSoMe* é executado nas classes que tiveram alguma alteração da versão pai para a versão resultante para identificar a variação média das métricas. Vale ressaltar que apenas as classes ou métodos que tiveram variação no valor das métricas serão consideradas.

Para ilustrar, considere o projeto *Junit4*[15] para o cálculo da variação média de métricas de classe para o *commit* de *hash* **02aaa01**. Neste *commit* houve a alteração

[13] ⟨https://github.com/rodhilton/jasome⟩
[14] ⟨https://github.com/voldemort/voldemort⟩
[15] ⟨https://github.com/junit-team/junit4⟩

somente da classe *FailOnTimeoutTest*. Sendo assim, executando a ferramenta *JaSoMe* no *hash* da versão atual para esta classe, o valor da métrica *TLOC* foi de 125. Executando novamente a ferramenta *JaSoMe* na versão anterior, cujo *hash* é **e9a75f4**, foi encontrado para a métrica *TLOC* o valor de 128. Portanto, para a classe *FailOnTimeoutTest*, o valor da métrica *TLOC* é de 125 na versão atual e 128 na versão pai. Neste exemplo está sendo feita a demonstração apenas da métrica TLOC, porém, a lógica pode ser aplicada em qualquer métrica.

Após encontrar o valor da métrica na versão atual e versão pai, o próximo passo é calcular a variação média de classe da versão onde *hash* é igual a **02aaa01**. Para isso, basta encontrar o somatório da métrica *TLOC* de todas as classes da versão atual, subtrair pelo somatório da métrica *TLOC* de todas as classes da versão anterior e dividir o resultado pelo número de classes alteradas na versão atual, conforme a Equação 3.1.

$$\sum_{i=0}^{n} \frac{x_i - x_{i-1}}{n} \tag{3.1}$$

$$\sum \frac{x_i - x_{i-1}}{n} = \frac{(125 - 128)}{1} = -3$$

Dessa forma, o cálculo da variação média da métrica *TLOC* na versão analisada será de -3. Cada uma das métricas segue a mesma abordagem, busca as classes alteradas e calcula a variação média para aquela métrica. Ao final do processamento de um *commit*, é armazenado o resultado da variação média de métricas de todas as classes alteradas do *commit*.

O motivo de considerar somente as classes alteradas é a desvantagem em analisar repetidamente classes que não sofreram nenhuma alteração. Por exemplo, considere um projeto com trezentas classes, onde foi feito um *commit* para corrigir um *bug* em uma classe específica. Considerando todas as classes da versão atual, seria necessário extrair as métricas de todas as trezentas classes, sendo que o *commit* impactou somente uma única classe. Além de realizar um trabalho redundante, o desempenho no momento da extração de métrica cairia drasticamente. Portando, em cada execução, é realizada a comparação das métricas das classes alteradas na versão atual e os valores das mesmas da versão

pai. Neste caso, é possível identificar se o código entregue, aumentou ou diminuiu, por exemplo, o valor da métrica *TLOC*.

Após realizar a listagem das classes alteradas e o cálculo da variação média de métricas de classe, o próximo passo é realizar esse mesmo cálculo com os métodos das classes alteradas. O cálculo para encontrar as variações das métricas de método segue a mesma abordagem de classe, a diferença está em como um método é classificado como alterado. Neste caso, foi considerado que um método teve alteração quando o valor da métrica na versão anterior difere da versão atual. Sendo assim, existem dois cenários. O primeiro ocorre quando a métrica de um método teve uma alteração no valor da versão atual para anterior. O segundo cenário é quando o método atual não foi encontrado na versão pai, implicando que o método foi criado na versão atual e a variação da métrica para este método é o próprio valor da métrica.

Para exemplificar, considere a classe A, que possui dois métodos na versão pai, onde o valor da métrica *TLOC* é de 10 e 10, para os métodos *a* e *b*, respectivamente. Já na versão atual, a classe A, sofreu alterações e o valor da métrica *TLOC* foi de 12 e 10, para os métodos *a* e *b*, respectivamente, e surgiu um novo método *c* com valor de 20. Nesse caso, o método *a* teve uma variação de +2, o *b* de 0 e o método *c* de +20. Portando, segundo a abordagem, os únicos métodos que sofreram alterações foram *a* e *b*. O método *c* pode ter sofrido alterações também, porém, segundo a abordagem proposta, foi desconsiderada.

Portando, após calcular a variação média das métricas de classe e método, demonstrado na Figura 3.2, o fluxo é finalizado com a inserção das variações médias no banco de dados e uma nova versão é iniciada para análise. Esse fluxo é repetido até que todas as versões do projeto sejam analisadas.

3.3 Extrator de sentimento

Extrator de sentimento é o módulo responsável pela extração da polaridade do sentimento de cada mensagem de *commit*. Este módulo foi implementado utilizando a linguagem de programação *Python*, o banco de dados *PostgreSQL*[16] e as ferramentas de análise de

[16] ⟨https://www.postgresql.org/⟩

sentimento *SentiStrenght* e *TextBlob*.

O *SentiStrength*, conforme o próprio site da ferramenta, "estima a força do sentimento positivo e negativo em textos curtos, mesmo para linguagem informal. Tem precisão de nível humano para textos curtos da *Web* social em inglês, exceto textos políticos" (SENTISTRENGTH,). A escolha da ferramenta *SentiStrenght* foi baseada na leitura e nos resultados apresentados pelo trabalho de (BENEVENUTO; RIBEIRO; ARAÚJO, 2015), onde foi feito um estudo de métodos para análise de sentimento e os três métodos mais bem classificados foram o SentiStrenght (THELWALL, 2014), AFINN (NIELSEN, 2011) e Opinion Lexicon (HU; LIU, 2004). Como o tempo de um Trabalho de Conclusão de Curso é limitado, não foram utilizadas as demais ferramentas do estudo. O *SentiStrenght* possui algumas formas de demonstrar o resultado de uma polaridade. Segunda a própria documentação, *binary* gera resultados positivo ou negativo, o *trinary* gera resultados positivo, negativo ou neutro e o *scale* gera resultados escalares no intervalo de -4 a 4 (THELWALL, 2017). Para este trabalho, foi utilizada a metodologia de *scale*, já que o objetivo é representar o sentimento das mensagens de *commits* quantitativamente para realizar a comparação com das métricas de código extraídas pelo Extrator de métricas.

A segunda ferramenta utilizada foi a *TextBlob* (LORIA et al., 2018). A escolha desta ferramenta foi baseada em estudos encontrados que também a utilizaram (CHAUDHRI; SARANYA; DUBEY, 2021), (DIYASA et al., 2021). Os estudos de Diyasa et al. (2021) utilizaram a ferramenta *TextBlob* para mineração de opinião na rede social do microblog Twitter[17]. Somado a isso, a ferramenta *TextBlob* possui uma documentação clara, além de ser uma ferramenta simples de implementação e executar os experimentos. Esses aspectos foram fundamentais para a escolha da ferramenta. Conforme a própria documentação do *TextBlob*, a ferramenta tem como resultado gerar um valor de polaridade e um valor de subjetividade. Porém, para este trabalho será utilizado somente o valor da polaridade. Diferente do *SentiStrenght*, que possui dados discretos, o retorno da polaridade do *TextBlob* é contínuo representado pelo intervalo de -1 a 1.

[17] ⟨https://twitter.com/⟩

3.3.1 Funcionalidade

Para extrair a polaridade do sentimento foi desenvolvida uma rotina para calcular a polaridade através da mensagem de cada *commit*. Após o Extrator de métricas identificar a variação média das métricas de cada versão e gravar no banco de dados com a mensagem do *commit* da versão analisada, o Extrator de sentimento executa a rotina para calcular a polaridade das mensagens utilizando as ferramentas *SentiStrenght* e *TextBlob* e registra no banco de dados para posteriormente realizar as análises.

A Figura 3.3 apresenta a visão geral da análise de sentimento das mensagens de *commit*. O primeiro passo consiste em buscar pelos *commits* que foram analisados pelo Extrator de métricas e que ainda não tiveram a sua polaridade calculada. Para identificar esses *commits*, foi criada uma coluna no banco de dados que realiza este controle. Desta forma, realizando uma simples busca é possível encontrar todos os *commits* pendentes, isto é, que ainda não foram analisados pelo Extrator de sentimento. Após encontrar os *commits*, é iniciada a rotina que utiliza o *SentiStrenght* e o *TextBlob* para calcular as polaridades e atualizar a base de dados com o valor das polaridades e da *flag* responsável por controlar os *commits* que já foram analisados pelo Extrator de sentimento.

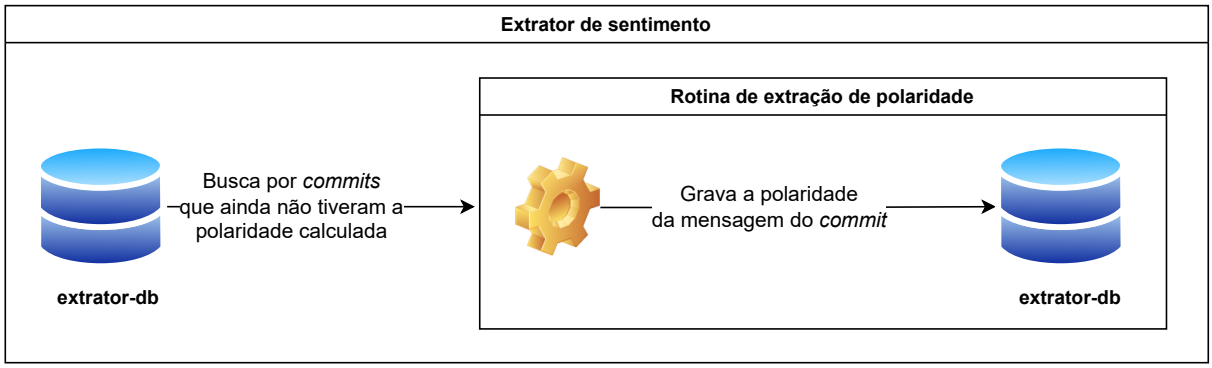

Figura 3.3: Representação do fluxo de extração da polaridade de cada *commit*

3.4 Considerações Finais

Neste capítulo foram apresentadas as abordagens de pesquisas propostas para a extração de métricas de software com o Extrator de métricas e extração de polaridade do sentimento com o Extrator de sentimento, além disso, foram apresentadas as funcionalidades da

ferramenta desenvolvida neste Trabalho de Conclusão de Curso. O capítulo foi dividido na apresentação de dois módulos, sendo a Seção 3.2 destinada para o Extrator de métricas, responsável por extrair, armazenar e calcular a variação média de métricas de classe e método, e a Seção 3.3 para o módulo Extrator de sentimento, responsável pelo cálculo da polaridade do sentimento. Dessa forma, após a apresentação das funcionalidades da ferramenta, será possível compreender os resultados encontrados no próximo capítulo.

4 Um estudo sobre a correlação entre análise de sentimentos e a qualidade do código alterado

Neste capítulo são apresentados os resultados encontrados a partir da execução do módulo Extrator de métricas e Extrator de sentimento em três projetos. Sendo assim, este capítulo está organizado da seguinte maneira. Na Seção 4.1, são apresentadas as três questões de pesquisa deste Trabalho de Conclusão de Curso. Na Seção 4.2, são apresentados os critérios para escolha dos projetos analisados e escolhidos. Na Seção 4.3, são reportados os resultados encontrados a partir da execução do Extrator de métricas e dos algoritmos de análise de sentimento. Por fim, na Seção 4.4, são discutidas as ameaças à validade do experimento e as limitações na etapa de implementação. Finalmente, na Seção 4.5, é realizada as considerações finais deste capítulo.

4.1 Questões de Pesquisa

O objetivo deste trabalho é verificar se existe uma relação entre o sentimento dos desenvolvedores, extraídos a partir de mensagens de *commit*, com a qualidade de software, medida por métricas de código. Sendo assim, foi proposta a seguinte questão de pesquisa (QP) e duas questões de pesquisas secundárias (QP1 e QP2) foram formuladas para responder à questão de pesquisa principal.

QP Principal: Existe correlação entre a polaridade dos sentimentos e a variação das métricas de qualidade extraídas de cada versão?

Para esta questão de pesquisa será calculada, para cada projeto, a correlação da variação média de métricas de qualidade com os valores de polaridade encontrados a partir da execução da ferramenta *SentiStrength* e *TextBlob*. A correlação será calculada seguindo o método de *Pearson* (BENESTY et al., 2009), já que a relação entre as variáveis podem ser lineares.

QP1: Qual é a distribuição de polaridade do sentimento encontrada nas mensagens de *commit* dos projetos selecionados?

Esta questão visa identificar a distribuição da polaridade do sentimento dos desenvolvedores a partir das mensagens de *commit*, já que neste trabalho a única fonte de dado textual que expressa o sentimento do desenvolvedor são os *commits*. Sendo assim, para cada projeto, será gerada a distribuição das polaridades utilizando as ferramentas *TextBlob* e *SentiStrength*.

QP2: Qual é a distribuição da variação das métricas de qualidade de software no histórico dos projetos selecionados?

Esta questão de pesquisa visa identificar o histórico da variação média de cada um das 13 métricas coletadas das versões dos projetos selecionados.

4.2 Seleção de projetos

Para execução deste experimento foram selecionados três projetos que obedecem as seguintes restrições: ser codificado predominantemente na linguagem *Java*, possuindo uma porcentagem maior que 90% de códigos *Java*, ter no mínimo 2 mil *commits* e ter acesso público ao repositório GitHub. A porcentagem de cada linguagem no projeto e o número de *commits* podem ser visualizados na página inicial de cada um dos repositórios no *GitHub*[18]. A restrição por projetos *Java* é justificada pela ferramenta *JaSoMe*, responsável pela extração de métricas, enquanto a quantidade de *commits* é utilizada para remover projetos "brinquedo" ou "tarefas de casa".

Os projetos selecionados são apresentados na Tabela 4.1. Nesta tabela, a coluna *Commits* representa a quantidade de *commits* que o projeto possuía até a data da extração e a coluna *Data último commit* representa a data do último *commit* analisado, este dado é importante para reprodutibilidade do experimento. Por fim, a coluna *Colaboradores* representa o número de desenvolvedores em um projeto.

Durante a seleção de projetos, o objetivo foi buscar aqueles que tratam de diferentes domínios, com quantidade variada de colaboradores e que obedecessem às restrições citadas. Deste modo, foram selecionados os seguintes projetos: *JUnit4* que é um

[18] ⟨https://github.com/⟩

framework para escrever testes repetíveis; o *Vlcj* que fornece uma estrutura *Java* para permitir que uma instância de um *media player VLC* nativo seja incorporada em um aplicativo Java; e o *Voldemort*, responsável pelo armazenamento de dados distribuído na forma chave-valor.

Projetos	***Commits***	***Data último commit***	**Colaboradores**	**pcJava**
JUnit 4	2486	15 de Maio de 2022	151	97.3%
Vlcj	2761	12 de Maio de 2022	3	100%
Voldemort	4262	30 de Agosto de 2017	65	94.4%

Tabela 4.1: Projetos analisados pelo Extrator de métricas e Extrator de sentimento.

4.3 Resultados da Extração de Métricas e Polaridade

Nesta seção serão apresentados os resultados da execução do módulo Extrator de métricas e Extrator de sentimento para cada um dos projetos selecionados (Tabela 4.1). Esses projetos totalizam 9.509 *commits* o que inviabiliza representar o valor de cada variação média das métricas para cada *commit*. Dessa forma, como o objetivo é encontrar a distribuição de polaridade e a distribuição das métricas de qualidade dos códigos dos projetos, foi optado por representar os dados por meio de histogramas, visto que são úteis para resumir grandes conjuntos de dados visualmente.

Durante a execução da ferramenta Extrator de métricas, muitos *commits* foram descartados e nem foram analisados pela ferramenta. Por exemplo, *commit* onde não existiu nenhuma alteração de arquivos com extensão *.java* eram descartados. Além disso, durante a execução da ferramenta *JaSoMe*, foram encontradas classes em *java* que estavam sintaticamente inválidas, nesses arquivos. Como resultado, não foi possível realizar a extração das métricas por retornar algum erro da própria ferramenta, logo, para os casos de erro, o *commit* foi descartado. Sendo assim, ao final do processo foram analisados 2.170 *commits* no projeto *Vlcj*, 1.507 no projeto *Junit4* e 3.154 no projeto *Voldemort* totalizando 6.831 *commits* analisados, e 2.678 *commits* foram descartados.

Processando os *commits* no módulo Extrator de sentimento, responsável pela extração da polaridade nas mensagens de *commit*, é possível responder a primeira questão de pesquisa (QP1) através da ilustração das distribuições das polaridades, representada

por histogramas. Nesses histogramas, o eixo x representa o valor da polaridade e o eixo y representa a quantidade da polaridade presente no projeto. Na subseção 4.3.1 referente ao *Vlcj*, 4.3.2 referente ao *Junit4* e na 4.3.3, referente ao *Voldemort*. O gráfico à esquerda representa a frequência de polaridades da ferramenta *SentiStrength*, onde os valores são discretos, e o gráfico a direita representa a ferramenta *TextBlob*, onde os valores são contínuos e possuem um intervalo no eixo x de 0.25. Para melhor visualização dos dados, na construção dos histogramas foi descartada a ocorrência de polaridades neutras, isto é, polaridade com o valor 0 (zero). Um ponto que deve ser considerado é o fato de existir bem menos polaridades neutras na ferramenta *TextBlob* comparado com a ferramenta *SentiStrength*, isso se deve porque a ferramenta *TextBlob* apresenta resultados contínuos e a ferramenta *SentiStrength* resultados discretos. Após executar os projetos no Extrator de métricas, será possível responder a segunda questão de pesquisa (QP2) com a ilustração da distribuição das variações médias das métricas, representada por histogramas. Neles, o eixo x representa o valor da variação média de uma métrica específica e o eixo y representa a quantidade desta variação presente no projeto.

Por fim, após ilustrar os resultados encontrados para os três projetos analisados pelo módulo Extrator de métricas e Extrator de sentimento, é possível responder à questão de pesquisa principal (QP Principal), com a apresentação dos valores de correlação da variação média de cada métrica com a polaridade encontrada. Dessa forma, as seguintes subseções discutem os resultados de cada um dos projetos selecionados.

4.3.1 (QP1) - Distribuição de Polaridade para o *Vlcj*

No projeto *Vlcj*, a distribuição de polaridade é ilustrada pela Figura 4.1. Analisando o gráfico e os resultados encontrados, tem-se que, na ferramenta *SentiStrength* o número de ocorrências com polaridade neutra representa um total de 1671 *commits* e na ferramenta *TextBlob* um total de 76. Isso se deve porque a ferramenta *TextBlob* gera valores contínuos, e para ser considerado uma polaridade neutra será considerado somente polaridades exatamente iguais a zero.

Dessa forma, no gráfico do *SentiStrength*, a maior quantidade de polaridade encontrada foi -1, 1 e -2, respectivamente, pois são resultados discretos. Já no gráfico da

ferramenta *TextBlob*, como os valores das polaridades são contínuos, é mais viável observar o maior volume, ou seja, o intervalo de polaridade que possui a maior concentração no gráfico. No caso do projeto *Vlcj*, está mais próximo de 0.25, isto é, a maioria dos resultados são de polaridade próximas de neutra. Além disso, na ferramenta *TextBlob* foi encontrado um total de 1322 *commits* com polaridade maior que 0 e 772 *commits* abaixo de 0, sendo assim, pode-se concluir que a maioria dos *commits* são otimistas. Já no *SentiStrength* os resultados mostram um sentimento mais negativo, onde 153 *commits* possuem polaridade maior que zero e 346 menor que zero.

Na análise do *Vlcj* observou-se que 1322 (60.92%) *commits* apresentaram polaridade positiva, 722 (35.58%) dos *commits* apresentaram polaridade negativa e 76 (3.50%) dos *commits* apresentaram polaridade neutra com a abordagem *TextBlob*. Por outro lado, utilizando a abordagem *SentiStrength* é possível observar 153 (7.05%) *commits* obtiveram polaridade positiva, 346 (15.94%) *commits* obtiveram polaridade negativa e 1671 (77.00%) apresentaram polaridade neutra.

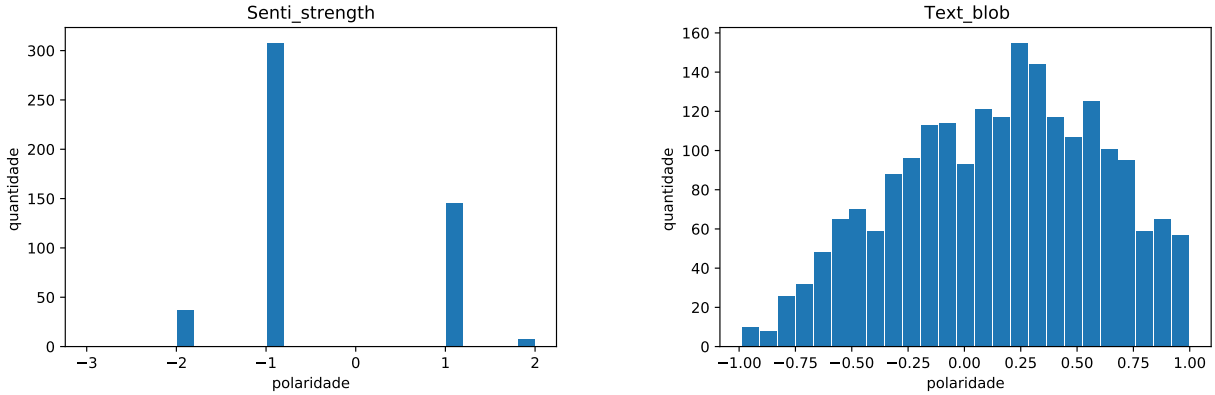

Figura 4.1: Variação de polaridade do projeto *Vlcj*.

4.3.2 (QP1) - Distribuição de Polaridade para o *Junit4*

A distribuição de polaridade no projeto *Junit4* é ilustrada pela Figura 4.2. Analisando o gráfico e os resultados encontrados, tem-se que, na ferramenta *SentiStrength*, o número de ocorrências com polaridade neutra representa um total de 1022 *commits* e na ferramenta *TextBlob* um total de 36.

Dessa forma, no gráfico do *SentiStrength*, a maior quantidade de polaridade en-

contrada foi -1, 1 e -2, onde a ocorrência de -1 representa 251 (16.66 %), 1 representa 127 (8.43 %) ocorrências e -2 um total de 94 (6.24 %). Já no gráfico da ferramenta *TextBlob*, como os valores das polaridades são contínuos, é mais viável observar onde está o maior volume, no caso do projeto *Junit4*, está entre 0 e 0.25, resultando em uma polaridade neutra. Além disso, na ferramenta *TextBlob* foi encontrado um total de 885 *commits* com polaridade maior que 0 e 586 *commits* abaixo de 0.

Na análise do *Junit4* observa-se que 885 (58.73%) *commits* apresentaram polaridade positiva, 586 (38.89%) dos *commits* apresentaram polaridade negativa e 36 (2.39%) dos *commits* apresentaram polaridade neutra com a abordagem *TextBlob*. Por outro lado, utilizando a abordagem *SentiStrength* é possível observar que 137 (9.09%) *commits* obtiveram polaridade positiva, 348 (23.09%) *commits* obtiveram polaridade negativa e 1022 (67.82%) apresentaram polaridade neutra.

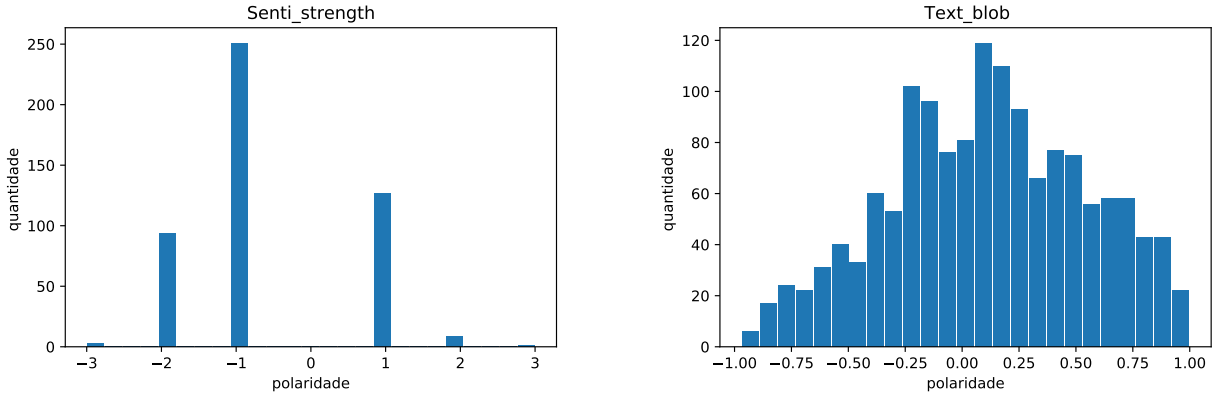

Figura 4.2: Variação de polaridade do projeto *Junit4*

4.3.3 (QP1) - Distribuição de Polaridade para o *Voldemort*

A Figura 4.3 apresenta a distribuição de polaridade no projeto *Voldemort*. Analisando o gráfico e os resultados encontrados, tem-se que, na ferramenta *SentiStrength* o número de ocorrências com polaridade neutra representa um total de 2147 (68.05%) *commits* e na ferramenta *TextBlob* um total de 77 (2.44%).

Sendo assim, no gráfico do *SentiStrength*, a maior quantidade de polaridade encontrada foi -1, 1 e -2, seguindo os mesmos resultados dos demais gráficos de polaridade, onde a ocorrência de -1 representa 577 (18.29%), 1 representa 236 (7.48%) ocorrências e

-2 um total de 158 (5.01%). Já no gráfico da ferramenta *TextBlob*, como os valores das polaridades são contínuos, é mais viável observar onde está o maior volume, no caso do projeto *Voldemort*, está entre 0 e 0.25. Além disso, na ferramenta *TextBlob* foi encontrado um total de 1930 (61.19%) *commits* com polaridade maior que 0 e 1147 (36.37%) *commits* abaixo de 0.

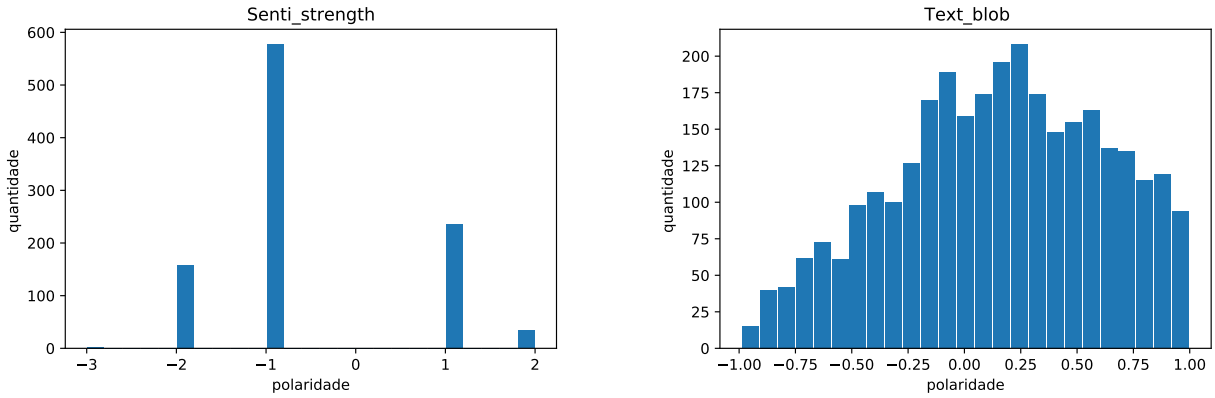

Figura 4.3: Variação de polaridade do projeto *Voldemort*

Portando, na análise do *Voldemort* observa-se que 1930 (61.19%) *commits* apresentaram polaridade positiva, 1147 (36.37%) dos *commits* apresentaram polaridade negativa e 77 (2.44%) dos *commits* apresentaram polaridade neutra com a abordagem *TextBlob*. Por outro lado, utilizando a abordagem *SentiStrength* é possível observar que 270 (8.56%) *commits* obtiveram polaridade positiva, 737 (23.37%) *commits* obtiveram polaridade negativa e 2147 (67,07%) apresentaram polaridade neutra.

4.3.4 QP1 - Análise dos resultados

Após apresentar as distribuições das polaridades dos três projetos selecionados por histogramas e discutido sobre os resultados encontrados, foi gerada a Tabela 4.2 com o intuito de organizar os dados para facilitar a análise final dos resultados encontrados a partir da execução da ferramenta desenvolvida neste Trabalho de Conclusão de Curso.

Analisando a Tabela 4.2, podemos perceber que, para os três projetos selecionados, a ferramenta *TextBlob* trouxe resultados predominantemente positivos, por outro lado, a ferramenta *SentiStrength* trouxe predominantemente resultados neutros, neste caso, teremos duas possibilidades de interpretações: a primeira possibilidade é a ferra-

menta realmente identifica que as mensagens de *commits* são neutras, a segunda é que a ferramenta não conseguiu identificar a polaridade das mensagens apresentadas.

	Commits	Polaridade	*TextBlob*	%	*SentiStrength*	%
Vlcj	2170	pos	1322	60,92	153	7,05
		neutra	76	3,5	1671	77
		neg	772	35,58	346	15,94
Junit4	1507	pos	885	58,73	137	9,09
		neutra	36	2,39	1022	67,82
		neg	586	38,89	348	23,09
Voldemort	3154	pos	1930	61,19	270	8,56
		neutra	77	2,44	2147	68,07
		neg	1147	36,37	737	23,37

Tabela 4.2: Porcentagem das polaridades encontradas nos projetos selecionados.

4.3.5 (QP2) - Distribuição de Métricas para o *Vlcj*

Analisando a Figura 4.4, a variação média da métrica *Fin*, existe uma distribuição onde as maiores variações estão entre 0 e 0.5, e o intervalo de variação está entre -2 a 2. A métrica *Fin*, que representa o número de métodos invocados por um método, quando a variação é negativa, demonstra que houve uma baixa na dependência entre métodos. Isso pode representar uma alta na qualidade. Pode ser observado que a presença do -1 expressivamente, significando que alguns *commits* resultaram em uma diminuição do valor da métrica *Fin*, logo, foi realizada a diminuição do acoplamento. Já na métrica *Fout*, a maior concentração encontra-se entre 0 e 0.7 e a presença de algumas variações próximas de 2 e -1, são as variações que se destacam isoladamente, vale ressaltar que, de 2170 *commits*, 185 (8,53%) são maiores que zero e 21 (0,97%) são menores que zero. Isso demonstra que, número de métodos imediatamente subordinados a um método aumentou na maior parte dos *commits* que diferiam de zero.

Na Figura 4.5, a métrica *Lcom*, possui variações muito próximas de zero, isto é, a medida que o código crescia, houve pouca alteração nesta métrica, por isso, quando representamos o histograma, a impressão é que os valores estão em zero, entretanto, o eixo x variou até 46, isso ocorreu porque um único *commit* cujo *hash* é *(2d6dac5)*, possui valor de variação média de 46, o restante não ultrapassou a variação de 5.5. Como a métrica *Lcom* representa a perda de coesão, isso significa que durante o desenvolvimento

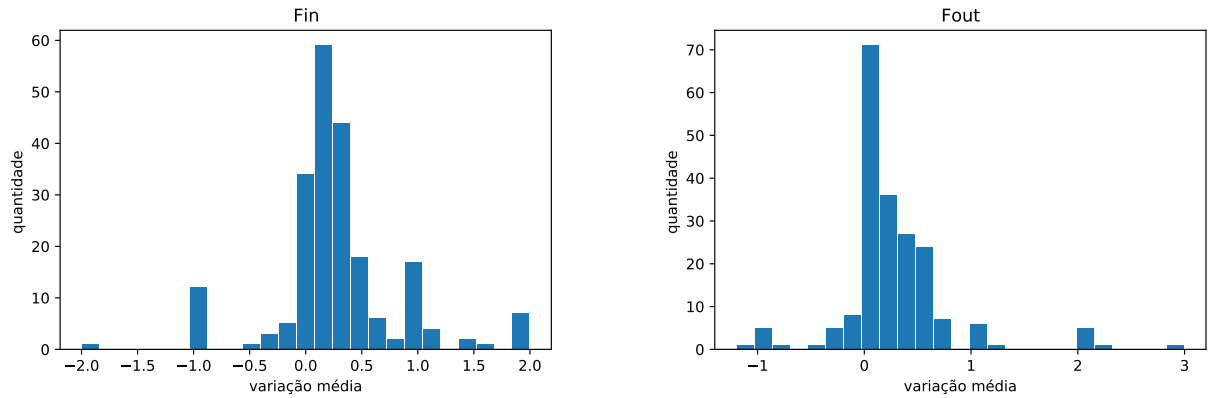

Figura 4.4: Variação média das métricas *Fin* e *Fout* do projeto *Vlcj*.

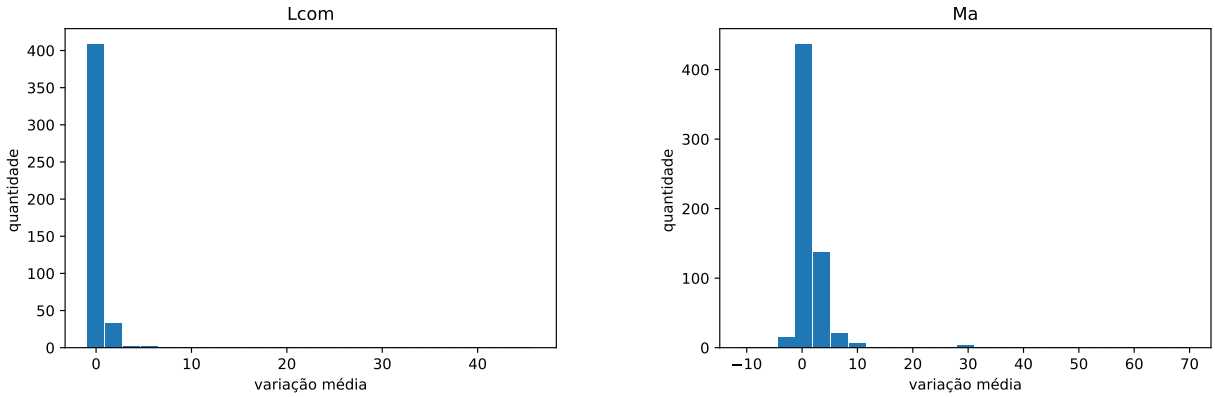

Figura 4.5: Variação média das métricas *Lcom* e *Ma* do projeto *Vlcj*.

do projeto, não houve *commits* que aumentaram significativamente a perda de coesão. Outra métrica analisada é a *Ma*, onde representa o número de métodos que podem ser invocados em uma classe, perceba que, para o projeto *Vlcj*, a concentração, encontram-se próximas de 0 a 10.

Na Figura 4.6, a métrica *Ncomp* representa o número de comparações em um método, portando, a medida que a variação aumenta, a complexidade para entender um método também aumenta, visto que um código pode seguir por vários caminhos diferentes e isso acaba tornando o código mais complexo e de difícil compreensão. A métrica *Nvar* representa o número de variáveis de controle referenciadas em um método, portando, métodos com muitas variáveis de controle pode prejudicar o entendimento de um método, logo, quanto maior o número de variáveis de controle, maior pode ser a complexidade para entender o código. Para o projeto *Vlcj* a variação máxima e mínima de 5, -4.5 e as

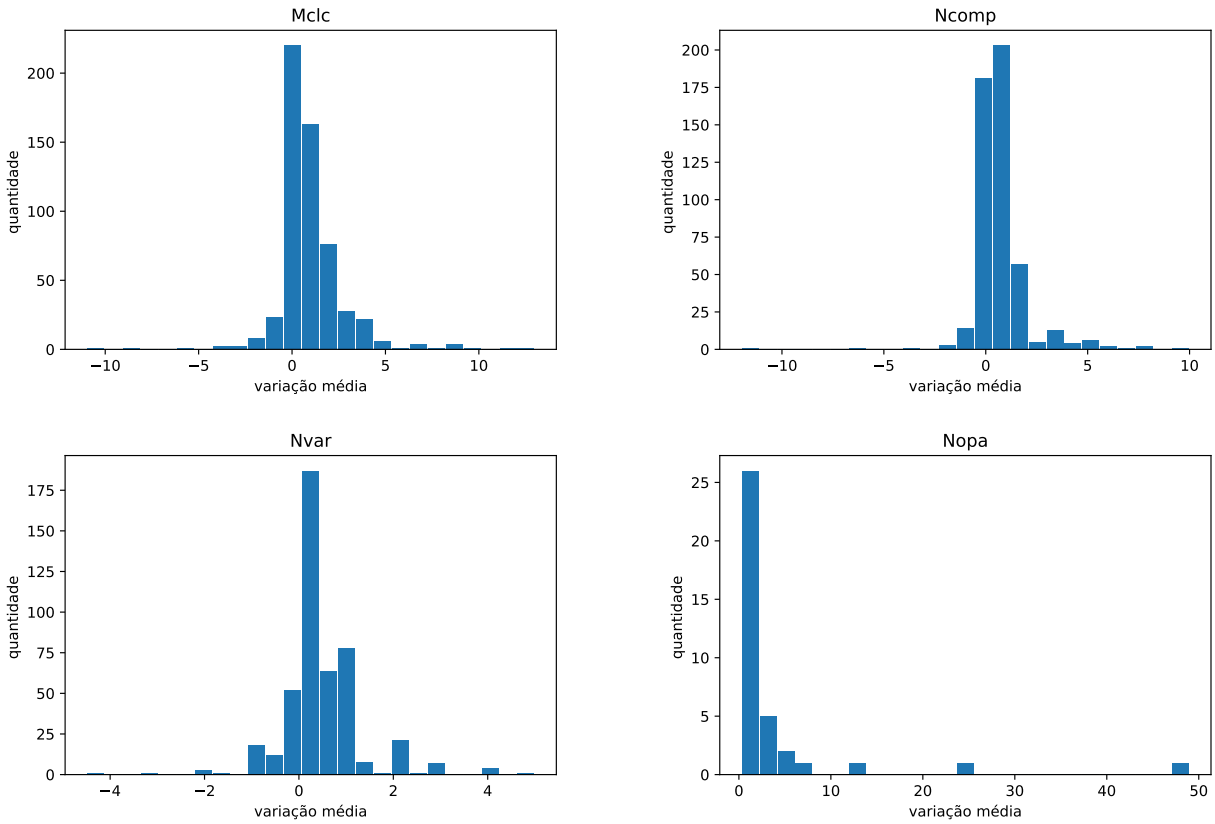

Figura 4.6: Variação média das métricas *Mclc*, *Ncomp*, *Nvar* e *Nopa* do projeto *Vlcj*.

três variações com maiores frequências foram de 1, 0.33 e 0.5.

Portanto, para a métrica *Nvar*, valores de variações negativos favorecem a qualidade de código, já que diminui o número de variáveis de controle. Já a métrica *Mclc* é a soma das métricas *Ncomp* e Nvar (*Ncomp* + *Nvar*) e representa a complexidade de *McClure*, também pode ser interpretada seguindo a mesma lógica da *Ncomp*. Variações positivas representam um aumento de complexidade e variações negativas pode representar uma diminuição da complexidade, podendo resultar em melhor qualidade de código. Os valores mais expressivos das métricas *Ncomp* e *Nvar* também estão sendo na métrica *Mclc*.

O último gráfico da Figura 4.6 representa a métrica *Nopa*, que calcula o número de classes que esta classe estende diretamente, ela possui variações mais isoladas, onde a variação máxima e mínima é de 49 e 0, respectivamente. As três variações médias mais frequentes foram de 1.5, 0.3 e 0.7.

As métricas *Noch* e *Nod* da Figura 4.7 possuem uma maior concentração nos

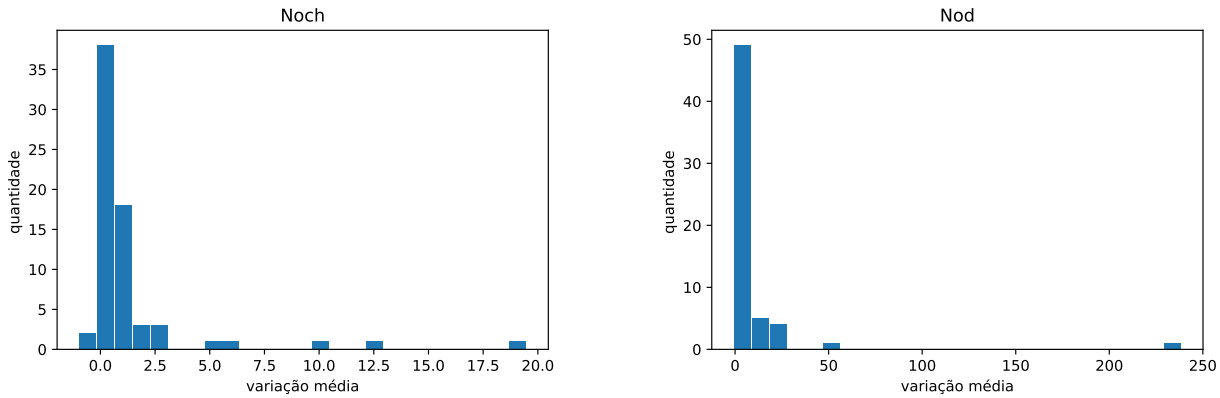

Figura 4.7: Variação média das métricas *Noch* e *Nod* do projeto *Vlcj*.

valores próximos de zero e alguns pontos isolados distantes da variação zero, isso é de se esperar, já que a métrica *Noch* representa o número de classes que estendem diretamente a classe analisada e a métrica *Nod* representa número total de classes que possui essa classe como ancestral. Além disso, para a métrica *Noch*, a maior e menor variação média é de 19.5 e -1, respectivamente, e as três maiores ocorrências são de 1, 0.34 e 0.5, já na métrica *Nod*, a maior e menor variação média é de 238.6 e -1, respectivamente, e as três maiores ocorrências são de 1, 0.2 e 0.3.

As métricas *Tloc* de método e classe da Figura 4.9 possuem uma relação, já que os métodos alterados estão presentes nas classes alteradas das versões. Na *Tloc* de classe e método as variações máximas e mínimas foram de 194 e -73 para classe e 47 e -15 para método. A métrica *Clrci* da Figura 4.8 e a métrica *Tloc* de classe e método possuem uma distribuição mais harmônica. Isto é, existe uma tendência de alta ou baixa no gráfico a medida que aumenta o valor do eixo x, além disso, para a métrica *Clrci*, que representa a complexidade relativa do sistema de classe, é esperado que a maioria dos *commits* estejam concentrados próximos de zero e a medida que o x aumenta, a ocorrência de variações maiores diminua, o mesmo se aplica para as métricas *TLOC*, visto que a maioria dos *commits* possuem pouca adição ou remoção de linhas de códigos.

A métrica *Vg* da Figura 4.8 possui um gráfico com distribuições parecidas com *Nvar*, e ela representa a complexidade ciclomática, isto é, o número de caminhos possíveis exclusivos por meio do código. Logo, quanto maior o seu valor, maior a dificuldade de se entender, modificar e, consequentemente, testar o código-fonte (MCCABE, 1976).

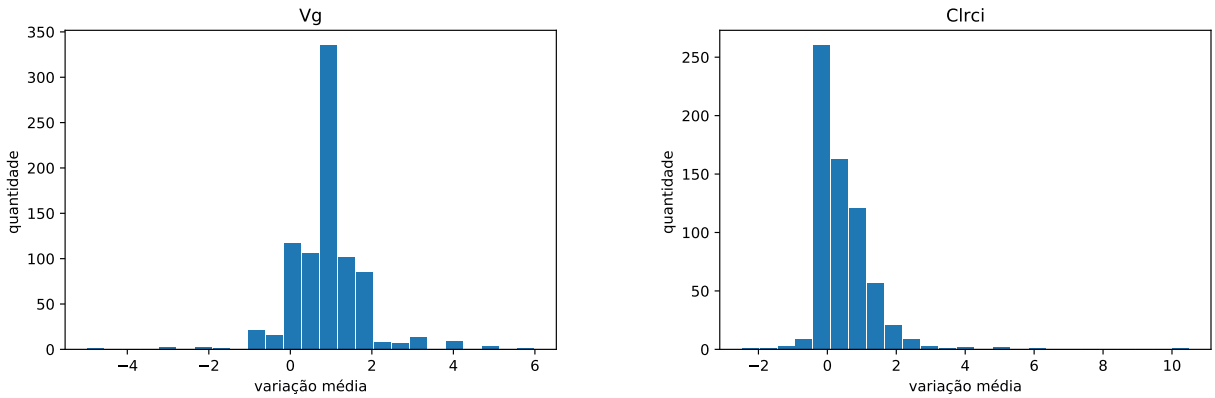

Figura 4.8: Variação média das métricas *Vg* e *Clrci*do projeto *Vlcj*.

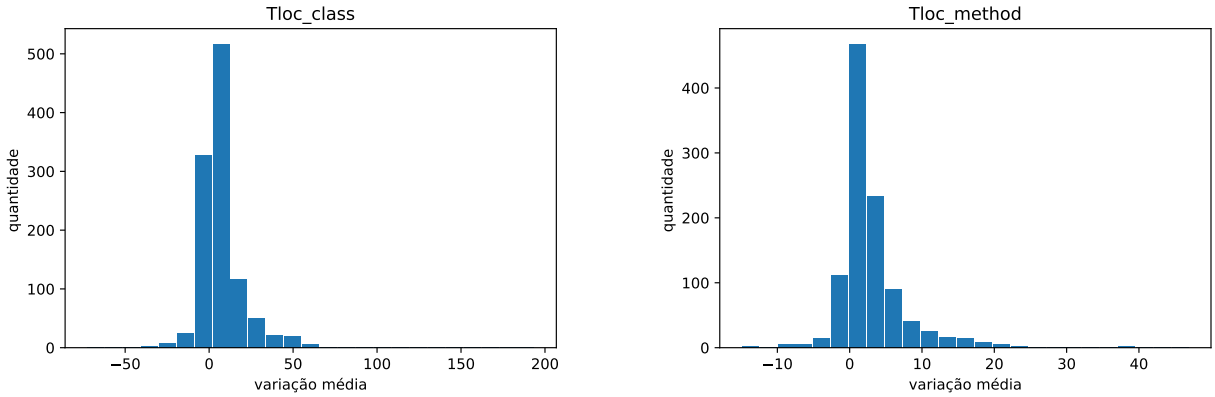

Figura 4.9: Variação média das métricas *Tloc* de classe e *Tloc* de método do projeto *Vlcj*.

Na métrica *Clrci*, suas variações máximas e mínimas são de 10.5 e -2.5, respectivamente. *Clrci* representa a complexidade relativa de classes, a concentração se encontra mais próximo de zero porque a métrica a divide o valor pelo número de métodos da classe. como o melhor resultado em termos de qualidade, e à medida que o eixo x aumenta, são encontradas cada vez menos variações.

4.3.6 (QP2) - Distribuição de Métricas para o *Junit4*

Analisando a primeira sequência de gráficos da Figura 4.10, é possível observar que as métricas *Fin* e *Fout* possuem distribuições parecidas, os dois valores com maiores frequências estão localizados no intervalo de 0 a 0.5.

Na segunda linha da Figura 4.10 são apresentadas as métricas *Lcom* e *Ma*. A *Lcom* mostrou-se ter pouca variação, suas variações máxima e mínima foi de 2 e -1,

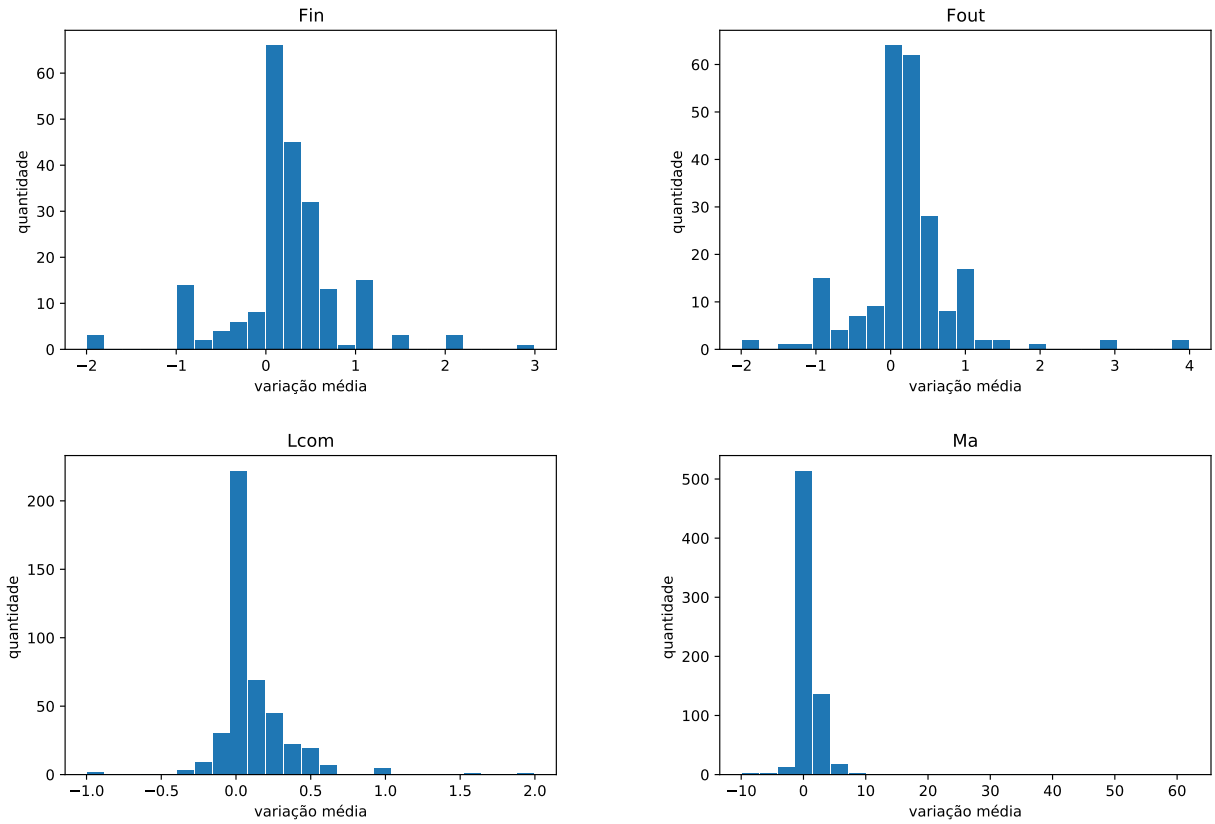

Figura 4.10: Variação média das métricas *Fin*, *Fout*, *Lcom*, *Ma* do projeto *Junit4*.

respectivamente. A métrica *Ma* possui um valor máximo e mínimo de variação de 62 e -10, respectivamente, isso significa que em um dado *commit*, foi inserida ou alterada uma classe e esta possui 62 ou mais métodos que podem ser invocados, o mesmo se aplica ao valor de -10, onde foram removidos ou sofreram uma refatoração de nome 10 métodos que poderiam ser invocados por uma classe. Como o extrator de métricas não identifica casos de refatoração de nome, esses métodos deixaram de existir.

Na primeira linha da Figura 4.11 são apresentadas as métricas *Mclc* e *Ncomp*. Na métrica *Mclc*, sua variação máxima e mínima foi de 7 e -11, respectivamente. Isso representa que, em um dado *commit* um desenvolver reduziu a complexidade em uma escala de -11 e outro desenvolvedor aumentou a complexidade em uma escala de 7. A grande concentração está localizada entre as variações -2.5 e 2.5. Já a métrica *Ncomp*, sua maior variação foi de 4.8 e menor foi de -11, e a sua concentração está localizada entre -2 e 2, isso representa que, na maioria dos *commits* os desenvolvedores produzem códigos aumentando ou diminuindo o número de comparações em um método na escala de -2 e

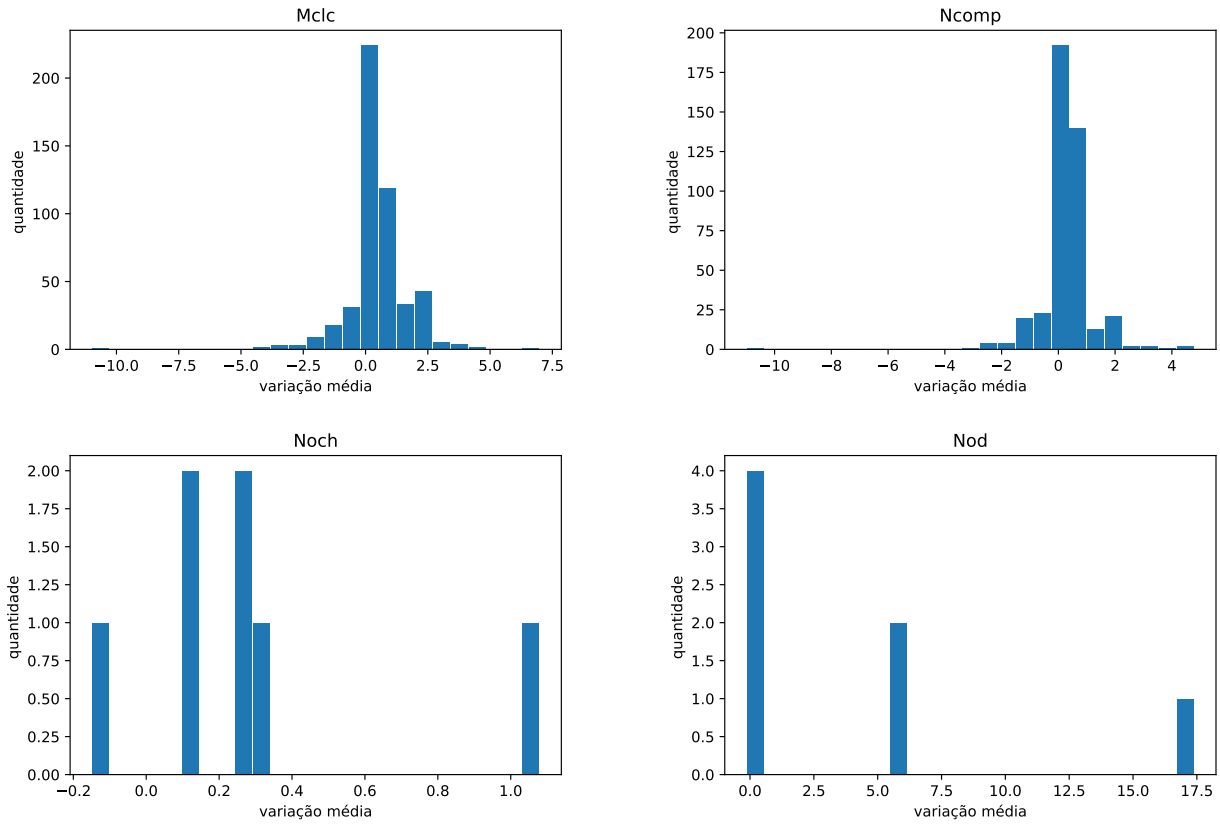

Figura 4.11: Variação média das métricas *Mclc*, *Ncomp*, *Noch* e *Nod* do projeto *Junit4*.

2, porém, existiram casos onde um desenvolvedor reduziu em uma escala de -11, nesses casos, pode ser uma grande refatoração de código.

Na segunda linha da Figura 4.11 são representadas as métricas *Noch* e *Nod*, onde a métrica *Noch* representa o número de classes que estendem diretamente uma classe e a métrica *Nod* o número total de classes com esta classe como ancestral. Como elas representam herança de classes e uma forma de aumentar o acoplamento pode ser exatamente a implementação de herança, já que uma entidade não consegue existir sem a sua entidade base, tem-se que, quanto maior a variação positiva das métricas, maior pode ser a perda de qualidade. Além disso, para a métrica *Noch*, a maior e menor variação média é de 1.1 e -0.15, respectivamente, já na métrica *Nod*, a maior e menor variação média é de 17.41 e -0.15, respectivamente.

Na Figura 4.12, o primeiro gráfico representa a métrica *ClRCi* e o segundo a métrica *Nopa*. É possível perceber a semelhança das distribuições da métrica *Nopa*, com as métricas *Noch* e *Nod* da Figura 4.11, as três métricas possuem resultados mais isolados,

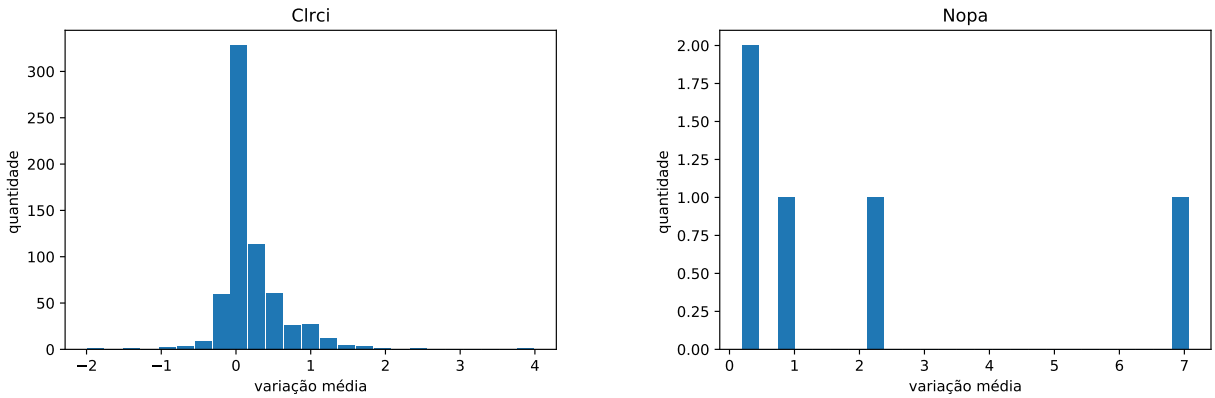

Figura 4.12: Variação média das métricas *Clrci* e *Lcom* do projeto *Junit4*.

não apresentando nenhuma tendência de alta ou baixa. Já a métrica *ClRCi*, para este projeto, analisando graficamente as tendências, se assemelha com a métrica *Lcom* da Figura 4.10, inclusive, os intervalos das variações médias são parecidos. A métrica *ClRCi* variou de -2 a 4 e a métrica *Lcom* de 2 a -1.

Na Figura 4.13, são apresentadas as métricas *Nvar* e *Tloc* de classe na primeira linha. A métrica *Nvar* variou de -2.4 a 3 e suas maiores concentrações são de 44 ocorrências da variação 1, 27 da variação 0.5 e 22 da variação 0.3. A métrica *Tloc* de classe e método, possuem relação já que, a variação média dos métodos também se encontram na variação média de classe, onde a variação mínima e máxima, da métrica *Tloc* de classe é, respectivamente, -50 a 156 e a variação mínima e máximo, da métrica *Tloc* de método é, respectivamente, -15 a 17.1. Por fim, a métrica *Vg*, que representa a Complexidade Ciclomática *McCabe*, a qual representa o número de caminhos possíveis únicos através do código. A métrica possui pontos máximos e mínimos de 4.4 e -2, respectivamente, e as quatro maiores concentrações da variação média estão localizadas próximas de 1, totalizando 317, seguido por 2, totalizando 22 ocorrências, 0,33 totalizando 22 ocorrências e 1.24 com 18 ocorrências.

4.3.7 Distribuição de Métricas para o *Voldemort*

Analisando os gráficos da Figura 4.14, pode-se observar que as métricas *Fin* e *Fout* não possuem distribuições parecidas, conforme descrito no projeto *Junit4* pela Figura 4.10. Os intervalos com maior frequência para a métrica *Fin* estão localizados próximo de -2 e

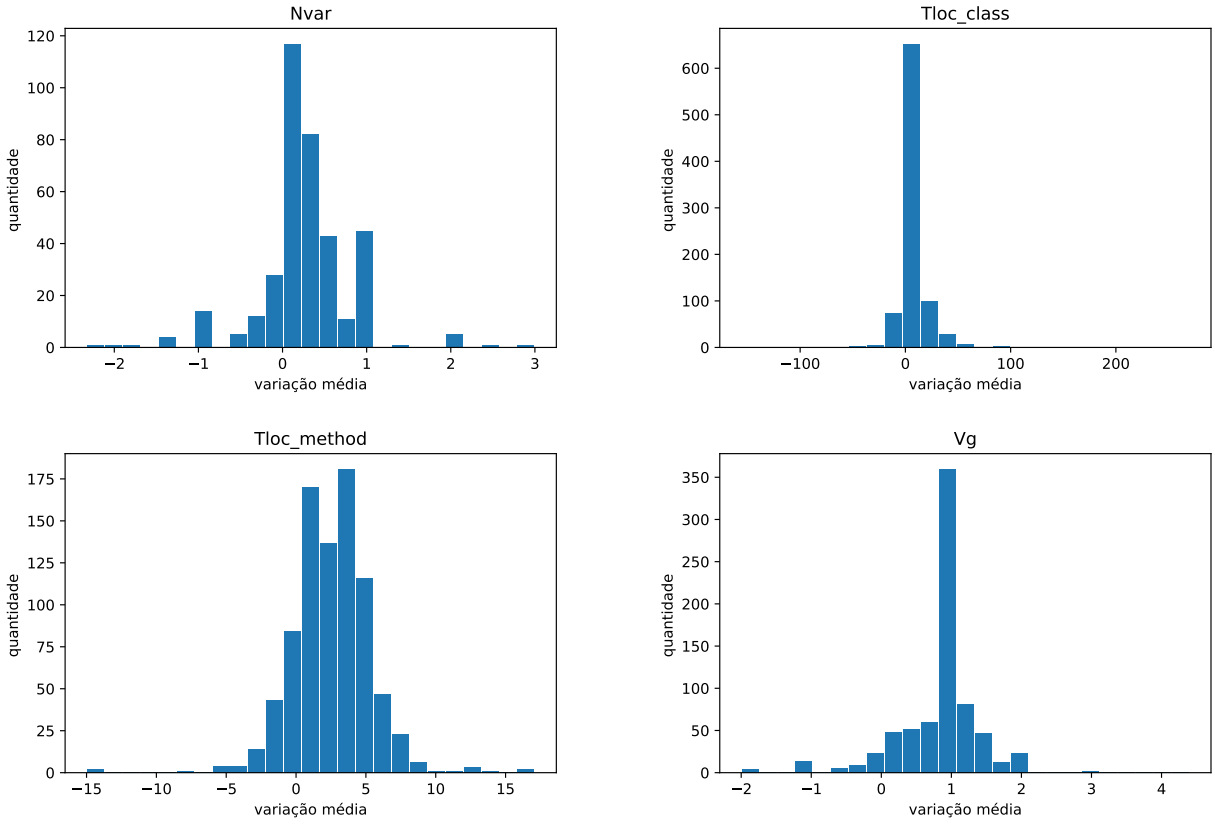

Figura 4.13: Variação média das métricas *Nvar*, *Tloc* de classe, *Tloc* de método e *Vg* do projeto *Junit4*

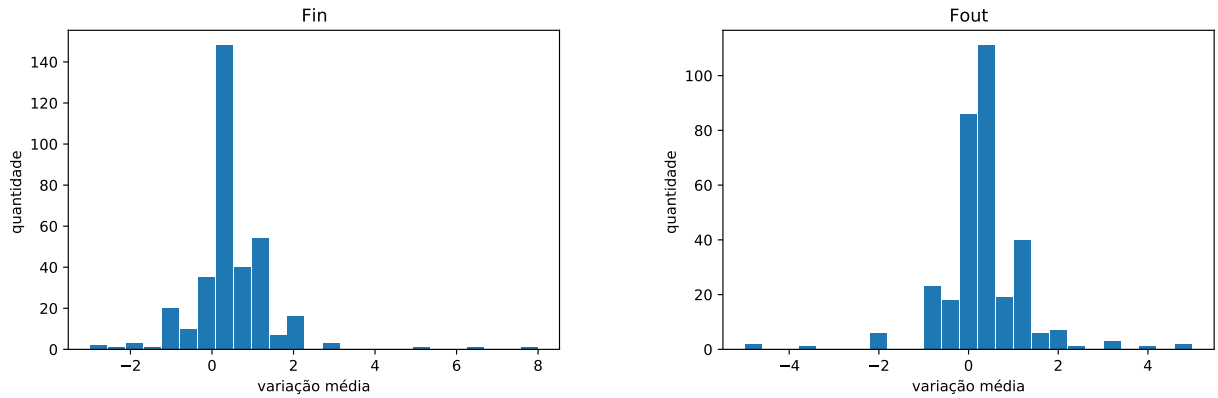

Figura 4.14: Variação média das métricas *Fin* e *Fout* do projeto *Voldemort*

2. Como já mencionado, a métrica *Fin* representa o número de métodos invocados por um método, portanto, é demonstrado que ocorreu uma baixa variação na métrica que calcula a dependência entre métodos, oscilando apenas de -2 a 2. Além disso, existiram algumas variações negativas, principalmente próximas de -1, isso é vantajoso, já que diminui a dependência entre métodos. A métrica *Fout* representa o número de métodos imediatamente subordinados a um método, isto é, o número de funções chamadas por uma dada função, nesses casos, existiram variações de -2 a 4.

Na Figura 4.15, são apresentadas as métricas *Lcom* e *Ma*, a *Lcom* possui pouca variação, exceto por uma única versão que obteve variação de 43, cujo *hash* é *6c955d2*, o restante dos dados foram de 2.2 a -1.1. Perceba que, por conta de uma única variação fora da concentração encontrada pela maioria para esta métrica, o intervalo no eixo x foi extenso. Já a métrica *Ma* possui um valor máximo e mínimo de variação de 45 e -8, respectivamente. Isso significa que em um dado *commit*, foi inserida ou alterada uma classe que possui 45 ou mais métodos que podem ser invocados, o mesmo se aplica ao valor de -8, foram removidos ou 8 métodos sofreram refatoração de nome que poderiam ser invocados por uma classe.

Na Figura 4.16, é possível observar métrica *Mclc* e a métrica *Ncomp*. Na métrica *Mclc*, suas variações foram de no máximo 36 e mínimo -14.4. Fazendo um arredondamento das variações, a maior concentração está localizada nas variações com valores iguais a: 1, -1, 2, -2, 3 e 4. Já a métrica *Ncomp*, sua maior variação foi de 30 e menor foi de -8, porém, as maiores concentrações se encontram nos valores 1, 2, -1, 0 e 3. Além disso, existiram alguns

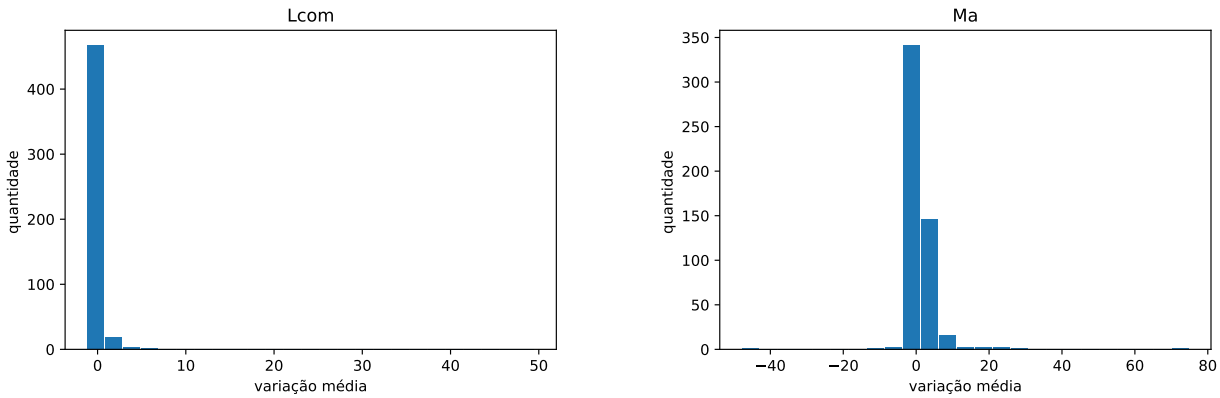

Figura 4.15: Variação média das métricas *Lcom* e *Ma* do projeto *Voldemort*.

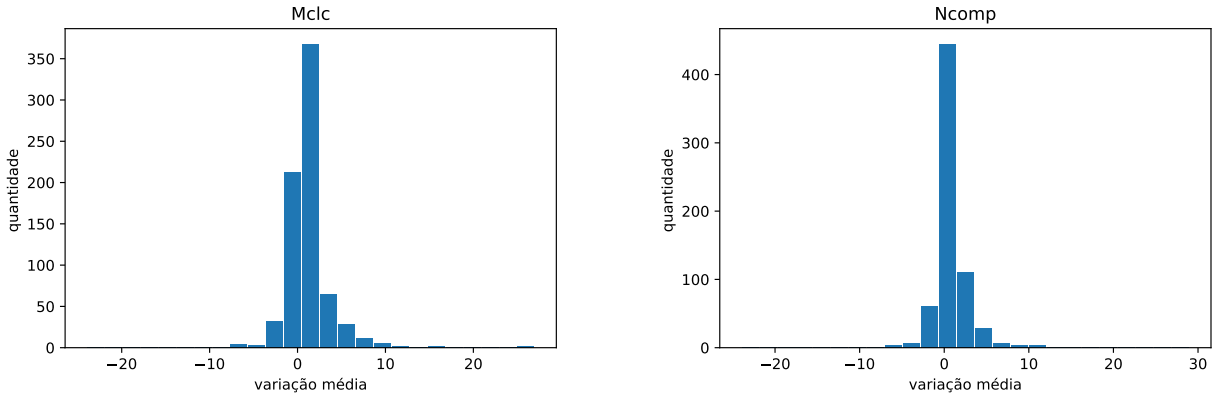

Figura 4.16: Variação média das métricas *Mclc* e *Ncomp* do projeto *Voldemort*.

commits com valores de 30 e -8. Onde o valor 30 pode representar indícios de uma baixa na qualidade de código e o valor -8 pode representar uma melhora na qualidade. A métrica *Mclc* representa a complexidade de *McClure* e também pode ser interpretada seguindo a mesma lógica da *Ncomp*, variações positivas representa um aumento de complexidade e variações negativas, pode representar uma baixa na complexidade, consequentemente, melhora no entendimento do código-fonte.

Na Figura 4.17, são representadas as métricas *Noch* e *Nod*, onde a métrica *Noch* representa o número de classes que estendem diretamente esta classe e a métrica *Nod* representa número total de classes com essa classe como ancestral. Nas duas pode-se observar um comportamento diferente das demais métricas apresentadas até o momento para este projeto, porém, comparando as duas métricas com os demais projetos, elas se parecem semelhantes, já que os pontos de distribuição não estão seguindo nenhuma

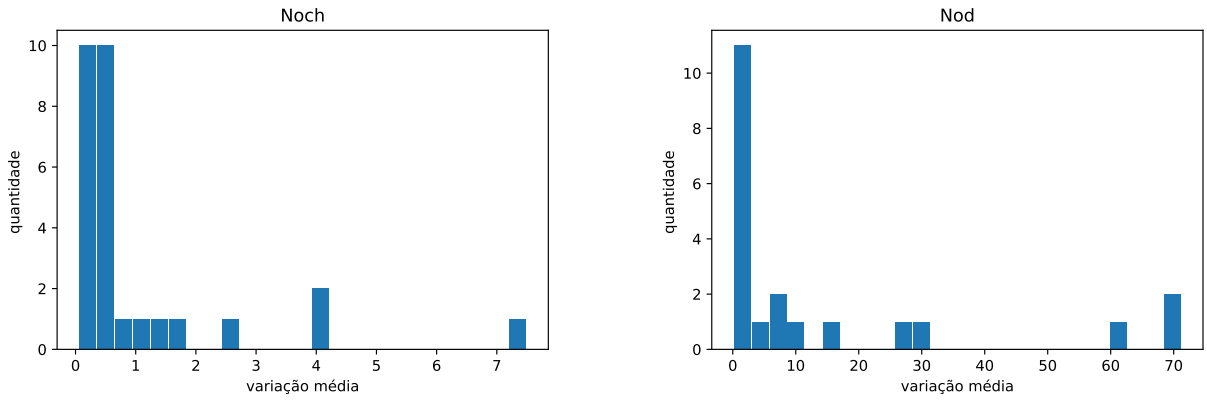

Figura 4.17: Variação média das métricas *Noch* e *Nod* do projeto *Voldemort*.

tendência de alta ou baixa, são pontos mais isolados de variação média. Os valores de maiores e menores variações para as métricas *Noch* são, 7 e 0, respectivamente, cuja variação 7 pode ser encontrada no *hash fbd0f95* e a variação 0 pode ser encontrada no *hash 53325f8*. Já os valores de maiores e menores variações para a métrica *Nod* são de 43.5 e 0, respectivamente, e a variação 43.5 pode ser encontrada no *hash 6083eb1*, já a variação 0, que possui várias, pode ser encontrada, por exemplo, no *hash fe84d88*.

Analisando a primeira sequência de gráficos presentes na Figura 4.18, é possível observar a semelhança das distribuições da métrica *Nopa*, com a métrica *Nod* da Figura 4.17. Isso porque a métrica *Nopa* representa o número de classes que esta classe estende diretamente e a métrica *Nod* representa número total de classes com essa classe como ancestral. Já a métrica *ClRCi*, para este projeto, visualmente ela se concentrou próximo de zero, já que a maioria das variações possui os seguintes valores, 1, 2, 0.5 3, porém, existem algumas exceções como variações de 116 no *hash a37abfb* e no *hash ffcd4ae*.

Na Figura 4.19, são apresentadas as métricas *Nvar* e *Tloc* de classe. A métrica *Nvar* variou de -5 a 8, cujo *hash* de -5 é *b584138* e 8 é *5a00311*, além disso, suas maiores concentrações são de 143 ocorrências para a variação a 1, 47 ocorrências para 0.5 e 45 ocorrências para 2. Por isso a concentração do gráfico se concentra próximo de 0 a 2. A métrica *Tloc* de classe e método, possui relação já que, a variação média dos métodos também se encontram na variação média de classe, onde a variação máximo da métrica *Tloc* de classe foi -80 a 196 e a variação máximo da métrica *Tloc* de método foi de -27 a 71. Por fim, a métrica *Vg* possui pontos máximos e mínimos de 14 e -13.8, respectivamente,

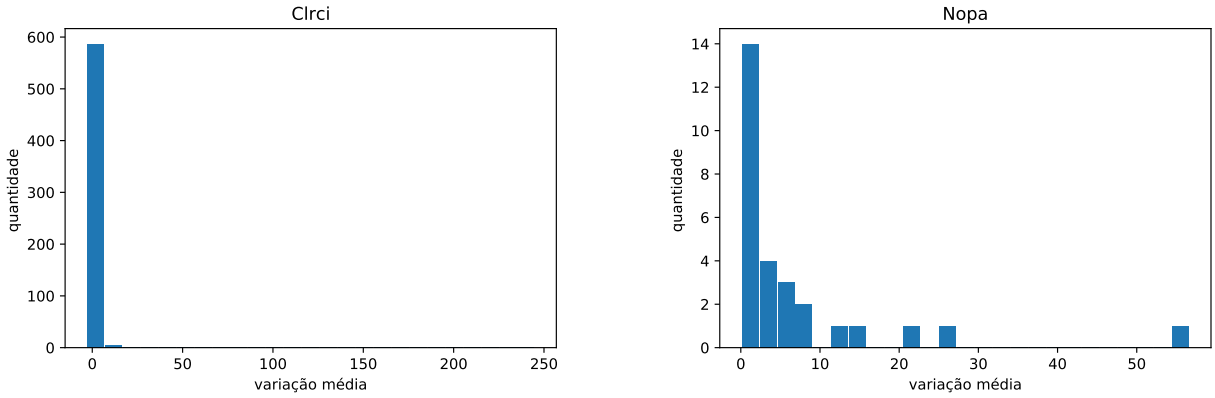

Figura 4.18: Variação média das métricas *Clrci* e *Nopa* do projeto *Voldemort*

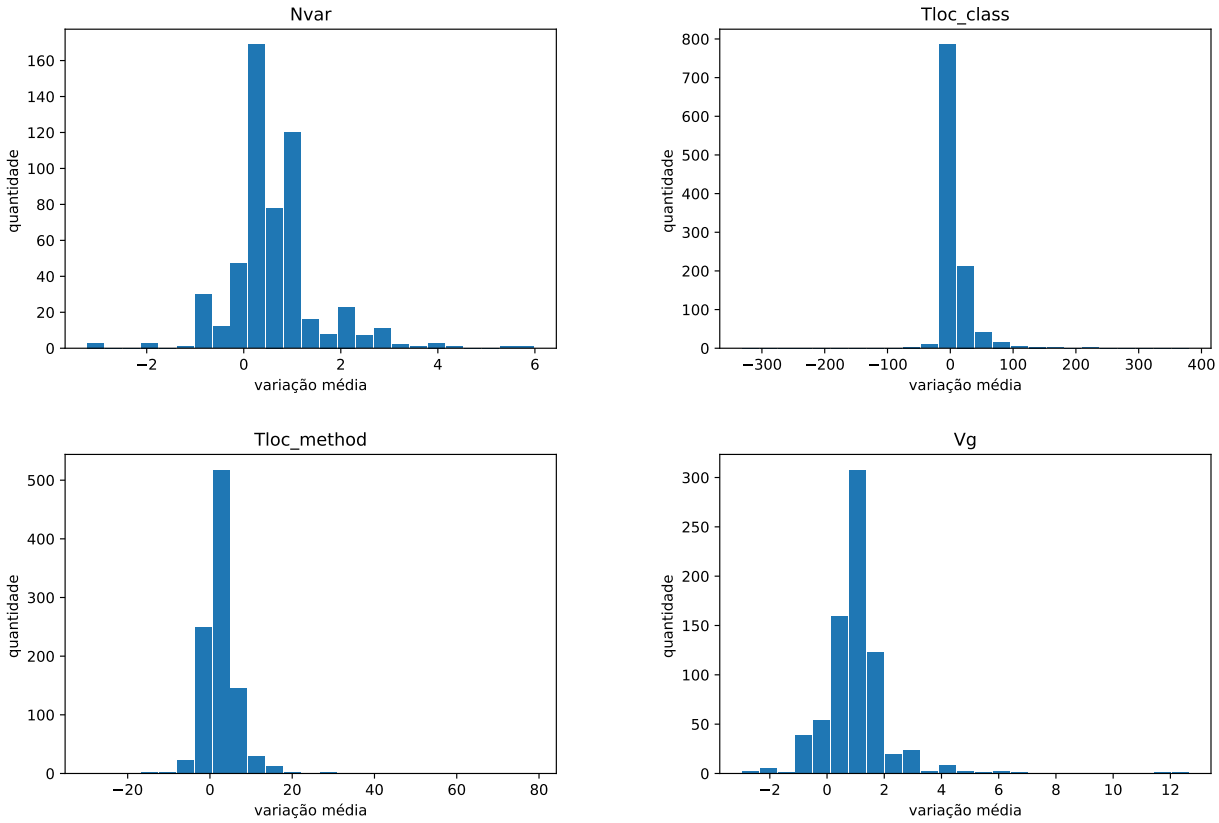

Figura 4.19: Variação média das métricas *Nvar*, *Tloc* de classe, *Tloc* de método e *Vg* do projeto *Voldemort*

e a maior concentração da variação média está localizada próximo de 0 e 2, onde existem 231 ocorrências da variação 1, 95 ocorrências de 2 e 47 ocorrências de 0.3.

QP Principal: Existe correlação entre a polaridade dos sentimentos e a variação das métricas de qualidade extraídas de versão?

Após ilustrar os resultados encontrados para os três projetos analisados pelo módulo Extrator de métricas e Extrator de sentimento, é possível responder à questão de pesquisa principal (QP Principal). Para responder a QP Principal, foi calculada a correlação utilizando o método *Pearson* (BENESTY et al., 2009), da variação média de cada métrica, extraída através da execução do Extrator de métricas, com o valor da polaridade da mensagem de *commit* calculada pelos algoritmos *SentiStrength* e *TextBlob*, extraídas a partir da execução do Extrator de sentimento. O resultado das correlações está na Tabela 4.4, representando as métricas *Ma*, *Lcom*, *Tloc* de classe, *Noch*, *Nod*, *Nopa* e *Fout*, e na Tabela 4.5, representando as métricas *Fin*, *Tloc* de método, *Vg*, *Ncomp*, *Nvar*, *Mclc*, *Clrci*, onde *SS* é a abreviação de *SentiStrength* e *TB* é a abreviação de *TextBlob* e as colunas representam as métricas presentes na Tabela 2.2.

Para auxiliar nas análises, se utiliza a Tabela 4.3 para indicar o grau de correlação.

Tamanho da correlação	**Interpretação**
0,90 a 1,00 (-0,90 a -1,00)	Correlação positiva (negativa) muito alta
0.70 a 0.90 (-0.70 a -0.90)	Alta correlação positiva (negativa)
0,50 a 0,70 (-0,50 a 0,70)	Correlação positiva (negativa) moderada
0,30 a 0,50 (-0,30 a 0,50)	Baixa correlação positiva (negativa)
0,00 a 0,30 (0,00 a -0,30)	Correlação insignificante

Tabela 4.3: Regra para interpretar o coeficiente de correlação (HINKLE; WIERSMA; JURS, 2003)

.

		Ma	**Lcom**	**Tloc de classe**	**Noch**	**Nod**	**Nopa**	**Fout**
Vlcj	SS	0.09	0.01	0.1	0.02	0.01	0.01	0.05
	TB	0.07	0.05	0.08	0.02	0.03	0.01	0.03
Junit4	SS	0.03	-0.01	0.04	0.01	0.01	0.01	0.01
	TB	0.04	0.03	0.07	0.03	0.04	0.03	-0.02
Voldemort	SS	0.05	0.04	0.08	0.01	-0.01	0.02	0.06
	TB	0.02	0.01	0.04	0.02	0.02	0.01	0.03

Tabela 4.4: Correlação entre as polaridades da ferramenta Senti Strength e TextBlob para as métricas Ma, Lcom, Tloc de classe, Noch, Nod, Nopa e Fout.

		Fin	**Tloc de método**	**Vg**	**Ncomp**	**Nvar**	**Mclc**	**Clrci**
Vlcj	SS	0.07	0.08	0.11	0.04	0.04	0.04	0.06
	TB	0.07	0.07	0.14	0.08	0.04	0.07	0.1
Junit4	SS	-0.01	-0.0	-0.03	-0.04	-0.01	-0.04	0.02
	TB	-0.02	0.06	0.06	0.04	0.03	0.04	0.07
Voldemort	SS	0.07	0.08	0.1	0.07	0.06	0.08	0.07
	TB	0.02	0.04	0.06	0.05	0.03	0.05	0.03

Tabela 4.5: Correlação entre as polaridades da ferramenta Senti Strength e TextBlob Fin, Tloc de método, Vg, Ncomp, Nvar, Mclc e Clrci.

Analisando as correlações da Tabela 4.4 e 4.5, para todos os três projetos e todas as métricas, foi encontrada uma correlação insignificante, segundo a faixa de valores de correlações da Tabela 4.3. Isso mostra que, para este conjunto de projetos, não foi encontrada nenhuma correlação entre o sentimento extraído a partir da polaridade das mensagens de *commit* com as métricas de código selecionadas neste trabalho.

Uma possível justificativa em não encontrar nenhuma correlação pode ser que, como os projetos analisados possuem uma certa complexidade para desenvolvedores inexperientes, a maioria dos colaboradores desses projetos devem possuir um certo grau de especialista, logo, levando isso em consideração, tem-se que, desenvolvedores seniores acabam não sendo influenciados por suas emoções.

4.4 Ameaças à Validade

Nesta seção serão discutidas as ameaças à validade que surgiram durante o processo de desenvolvimento dos módulos de extração de métricas e sentimento.

Na etapa de extração das métricas de determinada versão, a ferramenta calcula as métricas de classe e método, e grava os dados em um banco de dados. Caso ocorra uma refatoração no nome de determinado método na próxima versão, neste trabalho, será considerado que surgiu um método novo. Não foi desenvolvida uma abordagem para identificar se houve alteração no nome do método, classe ou pacote.

Para exemplificar, considere que um *commit* de um projeto existia um método chamado *calculaTaxa*, após uma refatoração, um desenvolvedor alterou o nome do método para *calculaTaxaJuros*. Neste caso, estamos considerando que foi criado um método chamado *calculaTaxaJuros* e o método *calculaTaxa* deixou de existir. Sendo assim, como não

existe a rastreabilidade desse método refatorado, as variações médias do método acabam sendo maiores, já que é um novo método criado daquele *commit*.

Além disso, como já explicado no Capítulo 3, para o cálculo da variação média, foi considerado o *hash* do *commit* atual e o *parent hash*, logo, se existisse mais de um pai, seria necessário desenvolver uma abordagem para realizar o cálculo da variação média deles. Somado a isso, neste Trabalho de Conclusão de Curso foram analisados apenas três projetos por limitação de tempo, o ideal seria explorar um conjunto maior de projetos, além disso, todos os projetos foram *Open Source*, visto que analisar projetos privados possui uma burocracia. A escolha das métricas também é considerado uma ameaça à validade, visto que poderiam ter existido métricas com maiores correlações.

Por fim, a última ameaça à validade é que todos os projetos analisados neste Trabalho de Conclusão de Curso foram predominantemente em *Java*, isso porque a ferramenta *JaSoMe*, como já mencionado Capítulo 2, busca, exclusivamente, por todos os arquivos *.java* para realizar o cálculo das métricas de código.

4.5 Considerações Finais

Neste Capítulo foram apresentadas e respondidas às três questões de pesquisa através dos resultados encontrados a partir da execução do módulo Extrator de métricas e Extrator de sentimento. A análise dos resultados foi dividida e apresentada seguindo a ordem das questões de pesquisa, por fim, foi feita as devidas considerações sobre as ameaças à validade.

5 Conclusões

Segundo as buscas realizadas na Seção 2.8, não foram encontrados trabalhos que relacionam diretamente a Análise de Sentimento (AS), extraída a partir da polaridade das mensagens de *commit*, com qualidade de código. Sendo assim, este estudo desenvolve uma abordagem e ferramenta implementada em Java e Python utilizando ferramentas de AS para investigar se existe correlação entre o sentimento dos desenvolvedores com a qualidade do software desenvolvido.

O estudo foi feito com três projetos para responder a questão de pesquisa principal (QP Principal) e suas secundárias (QP1 e QP2). Portanto, a questão QP1 foi respondida a partir da análise das distribuições das polaridades. Os três projetos se mostraram mais otimistas no *TextBlob* e mais negativos no *SentiStrength*. A questão QP2 foi respondida a partir da análise das variações médias das métricas selecionadas e a questão principal foi respondida a partir do cálculo da correlação dos resultados da questão QP1 e QP2.

Após calcular a correlação das polaridades com a variação média das métricas, observou-se que a correlação entre polaridade e qualidade de código para os três projetos selecionados são insignificantes, segundo a Tabela 4.3. Como mencionado na conclusão das correlações apresentadas no Capítulo 4, uma justificativa das correlações entre o sentimento e a qualidade de código serem insignificantes pode ser que desenvolvedores experientes e especialistas podem não ser influenciados pelo estado emocional no trabalho.

Este Trabalho de Conclusão de Curso traz as seguintes contribuições:

- uma ferramenta para extrair a variação média de métricas e polaridade dos sentimentos das versões de sistemas em Java em repositórios *Git*;
- resultados e análises da extração da variação média de métricas de código e das polaridades dos *commits* dos três projetos selecionados e suas correlações.

5.1 Trabalhos Futuros

A partir dos projetos selecionados, ferramentas utilizadas e resultados encontrados, visto que não foi possível encontrar correlação entre os sentimentos extraídos dos *commits* e as métricas de códigos selecionadas, como trabalhos futuros, podemos citar: (i) novos experimentos com um número maior de projetos podem ser conduzidos; (ii) outras ferramentas de extração de polaridades para extrair o sentimento do desenvolvedor podem ser utilizadas; a análise das polaridades poderiam ser expandidas para outros tipos de fonte textual utilizadas por desenvolvedores, como, por exemplo, mensagens em *pull request*. Somado a isso, é importante explorar um conjunto maior de métricas que não foram utilizadas neste Trabalho de Conclusão de Curso. Na própria ferramenta *JaSoMe* existem métricas que não foram utilizadas neste trabalho.

Na Seção 4.4, que se refere à ameaça à validade, é citado que não foi considerado no processamento das versões *commits* com mais de um pai (*parent*). Além disso, pacotes, classes e métodos que sofreram refatoração de nome eram considerados novos pacotes, classes ou métodos, sendo assim, como trabalhos futuros é interessante considerar a construção de uma abordagem para tratar esses casos de refatoração.

Por fim, para complementar a ferramenta desenvolvida neste TCC, pode-se implementar a extração de métricas para outras linguagens além do Java e criar uma ferramenta *front-end* para integrar com o *back-end* já desenvolvido para exibir gráficos, estatísticas e a possibilidade de o usuário selecionar projetos que se deseja analisar.

Bibliografia

ATLASSIAN. *O que é controle de versão?* 2022. Urlhttps://www.atlassian.com/br/git/tutorials/what-is-version-control, addendum=Acessado em 17/08/2022.

BALDWIN, C. Y.; CLARK, K. B. The architecture of participation: Does code architecture mitigate free riding in the open source development model? *Management science*, INFORMS, v. 52, n. 7, p. 1116–1127, 2006.

BECK, K. *Implementation patterns*. [S.l.]: Pearson Education, 2007.

BENESTY, J.; CHEN, J.; HUANG, Y.; COHEN, I. Pearson correlation coefficient. In: *Noise reduction in speech processing*. [S.l.]: Springer, 2009. p. 1–4.

BENEVENUTO, F.; RIBEIRO, F.; ARAÚJO, M. Métodos para análise de sentimentos em mídias sociais. *Sociedade Brasileira de Computação*, 2015.

CHACON, S.; STRAUB, B. *Pro git*. [S.l.]: Springer Nature, 2014.

CHAUDHRI, A. A.; SARANYA, S.; DUBEY, S. Implementation paper on analyzing covid-19 vaccines on twitter dataset using tweepy and text blob. *Annals of the Romanian Society for Cell Biology*, p. 8393–8396, 2021.

DIAS, A. F. Conceitos básicos de controle de versão de software—centralizado e distribuído. *Artigo retirado do site da empresa Pronus Engenharia de Software, publicado*, 2016.

DIYASA, I. G. S. M.; MANDENNI, N. M. I. M.; FACHRURROZI, M. I.; PRADIKA, S. I.; MANAB, K. R. N.; SASMITA, N. R. Twitter sentiment analysis as an evaluation and service base on python textblob. In: IOP PUBLISHING. *IOP Conference Series: Materials Science and Engineering*. [S.l.], 2021. v. 1125, n. 1, p. 012034.

D'AMBROS, M.; GALL, H.; LANZA, M.; PINZGER, M. Analysing software repositories to understand software evolution. In: *Software evolution*. [S.l.]: Springer, 2008. p. 37–67.

FENTON, N.; BIEMAN, J. *Software metrics: a rigorous and practical approach*. [S.l.]: CRC press, 2014. 19 p.

FENTON, N.; BIEMAN, J. *Software metrics: a rigorous and practical approach*. [S.l.]: CRC press, 2019.

FOWLER, M.; HIGHSMITH, J. et al. The agile manifesto. *Software development*, [San Francisco, CA: Miller Freeman, Inc., 1993-, v. 9, n. 8, p. 28–35, 2001.

HASSAN, A. E. The road ahead for mining software repositories. In: IEEE. *2008 Frontiers of Software Maintenance*. [S.l.], 2008. p. 48–57.

HENDERSON-SELLERS, B. *Object-oriented metrics: measures of complexity*. [S.l.]: Prentice-Hall, Inc., 1995.

HILTON, R.; UFER, D. *JaSoMe: Java Source Metrics*. [S.l.]: GitHub, 2013. ⟨https://github.com/rodhilton/jasome⟩.

HINKLE, D. E.; WIERSMA, W.; JURS, S. G. *Applied statistics for the behavioral sciences*. [S.l.]: Houghton Mifflin College Division, 2003. v. 663.

HU, M.; LIU, B. Mining and summarizing customer reviews. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2004. p. 168–177.

HUQ, S. F.; SADIQ, A. Z.; SAKIB, K. Is developer sentiment related to software bugs: An exploratory study on github commits. In: IEEE. *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. [S.l.], 2020. p. 527–531.

KITCHENHAM, B. What's up with software metrics?–a preliminary mapping study. *Journal of systems and software*, Elsevier, v. 83, n. 1, p. 37–51, 2010.

KONTOPOULOS, E.; BERBERIDIS, C.; DERGIADES, T.; BASSILIADES, N. Ontology-based sentiment analysis of twitter posts. *Expert systems with applications*, Elsevier, v. 40, n. 10, p. 4065–4074, 2013.

LIU, B.; ZHANG, L. A survey of opinion mining and sentiment analysis. In: *Mining text data*. [S.l.]: Springer, 2012. p. 415–463.

LORIA, S. et al. textblob documentation. *Release 0.15*, v. 2, n. 8, 2018.

MARTIN, R. C. *Clean Coder*. [S.l.]: mitp Verlags GmbH & Co. KG, 2014.

MCCABE, T. J. A complexity measure. *IEEE Transactions on software Engineering*, IEEE, n. 4, p. 308–320, 1976.

MEIRELLES, P.; JR, C. S.; MIRANDA, J.; KON, F.; TERCEIRO, A.; CHAVEZ, C. A study of the relationships between source code metrics and attractiveness in free software projects. In: IEEE. *2010 Brazilian Symposium on Software Engineering*. [S.l.], 2010. p. 11–20.

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.

MOURÃO, E.; KALINOWSKI, M.; MURTA, L.; MENDES, E.; WOHLIN, C. Investigating the use of a hybrid search strategy for systematic reviews. In: IEEE. *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. [S.l.], 2017. p. 193–198.

NIELSEN, F. Å. A new anew: Evaluation of a word list for sentiment analysis in microblogs. *arXiv preprint arXiv:1103.2903*, 2011.

OLIVEIRA, M. G. de; NEVES, A.; LOPES, M. F. S.; MEDEIROS, H. F.; ANDRADE, M. B.; REBLIN, L. L. Um curso de programação a distância com metodologias ativas e análise de aprendizagem por métricas de software. *RENOTE*, v. 15, n. 1, 2017.

RAWAT, M. S.; MITTAL, A.; DUBEY, S. K. Survey on impact of software metrics on software quality. *IJACSA) International Journal of Advanced Computer Science and Applications*, Citeseer, v. 3, n. 1, 2012.

ROMANO, S.; CAULO, M.; SCANNIELLO, G.; BALDASSARRE, M. T.; CAIVANO, D. Sentiment polarity and bug introduction. In: SPRINGER. *International Conference on Product-Focused Software Process Improvement*. [S.l.], 2020. p. 347–363.

SARLAN, A.; NADAM, C.; BASRI, S. Twitter sentiment analysis. In: IEEE. *Proceedings of the 6th International conference on Information Technology and Multimedia*. [S.l.], 2014. p. 212–216.

SENTISTRENGTH. *Web Site*. ⟨http://sentistrength.wlv.ac.uk/⟩, Último acesso em 2022-09-22.

SILVA, M. C. da; CIZOTTO, A. A. J.; PARAISO, E. C. A developer recommendation method based on code quality. In: IEEE. *2020 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2020. p. 1–8.

SINHA, V.; LAZAR, A.; SHARIF, B. Analyzing developer sentiment in commit logs. In: *Proceedings of the 13th international conference on mining software repositories*. [S.l.: s.n.], 2016. p. 520–523.

SOMMERVILLE, I. et al. Engenharia de software.[sl]. *Pearson Education*, v. 19, 2011.

TABOADA, M.; BROOKE, J.; TOFILOSKI, M.; VOLL, K.; STEDE, M. Lexicon-based methods for sentiment analysis. *Computational linguistics*, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , v. 37, n. 2, p. 267–307, 2011.

THELWALL, M. Heart and soul: Sentiment strength detection in the social web with sentistrength, 2017. *Cyberemotions: Collective emotions in cyberspace*, 2014.

THELWALL, M. The heart and soul of the web? sentiment strength detection in the social web with sentistrength. In: *Cyberemotions*. [S.l.]: Springer, 2017. p. 119–134.

VOINEA, L.; TELEA, A. Mining software repositories with cvsgrab. In: *Proceedings of the 2006 international workshop on Mining software repositories*. [S.l.: s.n.], 2006. p. 167–168.

WILLIAMS, C. C.; HOLLINGSWORTH, J. K. Automatic mining of source code repositories to improve bug finding techniques. *IEEE Transactions on Software Engineering*, IEEE, v. 31, n. 6, p. 466–480, 2005.

WILSON, T.; WIEBE, J.; HOFFMANN, P. Recognizing contextual polarity in phrase-level sentiment analysis. In: *Proceedings of human language technology conference and conference on empirical methods in natural language processing*. [S.l.: s.n.], 2005. p. 347–354.