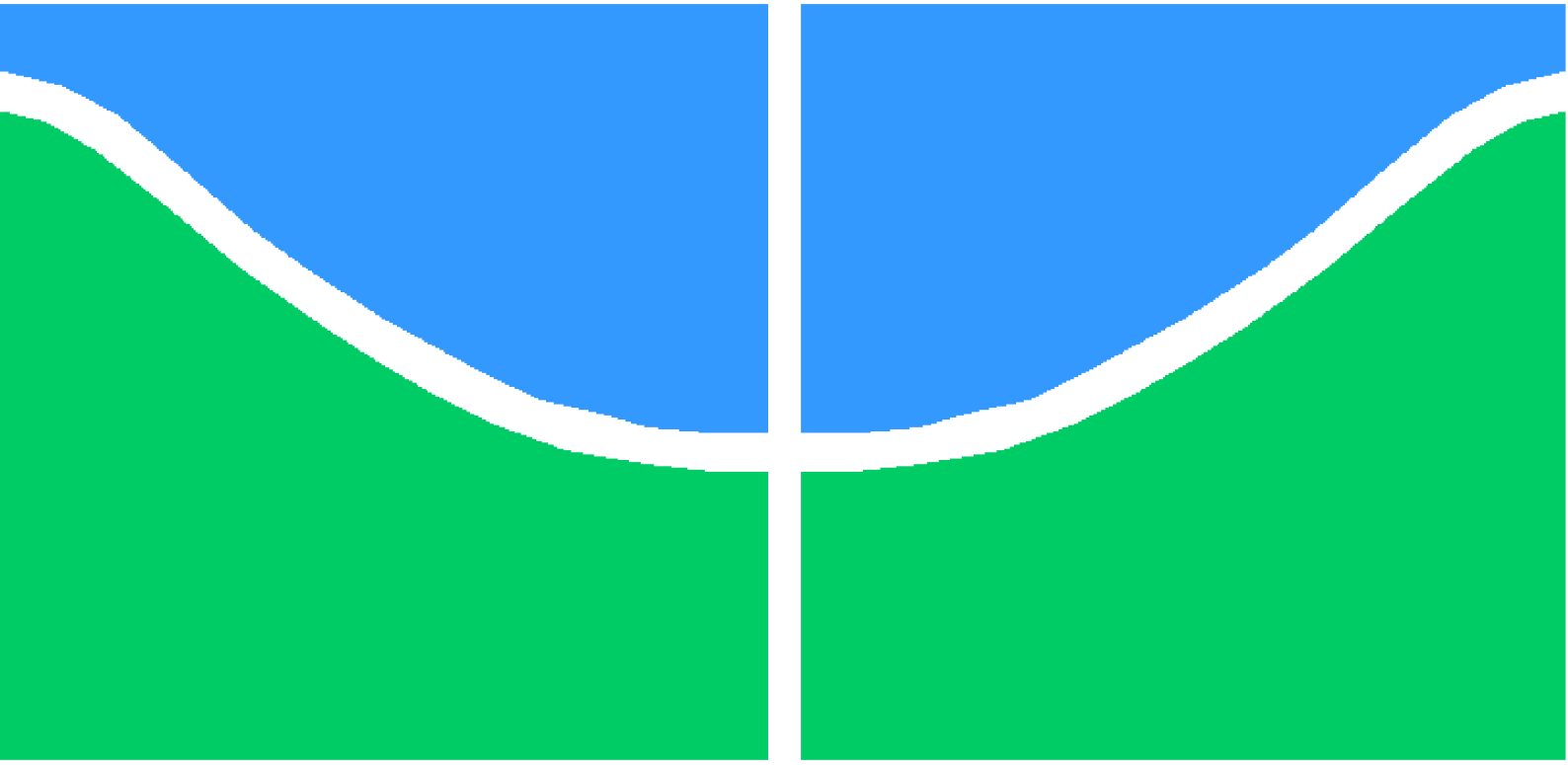

Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Determinando a taxa de autoria dentro de um projeto usando Git

Autor: Laércio Silva de Sousa Júnior
Orientador: Dr. André Luiz Peron Martins Lanna
Coorientadora: MSc. Cristiane Soares Ramos

Brasília, DF
2018

Laércio Silva de Sousa Júnior

Determinando a taxa de autoria dentro de um projeto usando Git

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Dr. André Luiz Peron Martins Lanna

Coorientadora: MSc. Cristiane Soares Ramos

Brasília, DF

2018

Laércio Silva de Sousa Júnior
Determinando a taxa de autoria dentro de um projeto usando Git/ Laércio Silva de Sousa Júnior. – Brasília, DF, 2018-
54 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. André Luiz Peron Martins Lanna

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2018.

1. autoria. 2. git. I. Dr. André Luiz Peron Martins Lanna. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Determinando a taxa de autoria dentro de um projeto usando Git

CDU 02:141:005.6

Laércio Silva de Sousa Júnior

Determinando a taxa de autoria dentro de um projeto usando Git

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 10 de dezembro de 2018

Dr. André Luiz Peron Martins Lanna
Orientador

MSc. Cristiane Soares Ramos
Coorientadora

MSc. Ricardo Ajax Dias Kosloski
Convidado

Brasília, DF
2018

Resumo

Na maior parte das vezes um produto de software moderno não é desenvolvido por apenas uma pessoa mas por uma equipe de desenvolvimento de software. Determinar a quantidade de código que cada desenvolvedor adicionou a um projeto não é uma tarefa difícil bastando apenas somar todas as linhas adicionadas por ele. Entretanto pode ser trabalhosa, dependendo do tamanho do projeto em questão, além do mais saber o quanto de código que esse desenvolvedor produziu ao longo do tempo, pode não ser simples. O objetivo desse trabalho é determinar uma abordagem para avaliar a quantidade de autoria de desenvolvedores, pois tal medida ajuda a analisar questões em várias áreas da Engenharia de Software como, por exemplo, qualidade de software, melhoria da manutenção, análise forense, dentre outros. Em especial, tal medida torna possível determinar uma taxa de autoria para cada desenvolvedor em relação ao projeto inteiro.

Palavras-chaves: Git, Autor, Autoria, Blame, taxa de autoria.

Abstract

Most of the time a modern software product is not developed by just one person but by a software development team. Determining the amount of code that each developer added to a project is not a difficult task, just adding up all the lines added by it. However, it may be that the project is subordinate to the project in question, in addition to what is more important is the project produced over time, it can not be simple. The same number of tasks to evaluate a quantity of authorship of developers, such as the measurement of an average in several areas of Software Engineering, for example, software quality, maintenance improvement, forensic analysis, among others. In particular, you can reserve an author fee for each company in relation to the entire project.

Key-words: Git, Author Authorship, Blame, Quantity of authorship.

Lista de ilustrações

Lista de abreviaturas e siglas

SVN	Apache Subversion
TFVC	Team Foundation Version Control
LOC	Lines of code
SLOC	Source Lines of code
GQM	Goal Question Metric
NPM	Node Package Manager
Node	Node.js
CLI	Command Line Interface

Sumário

1 Introdução

1.1 Contexto

O desenvolvimento de um produto de software moderno na grande maioria das vezes envolve uma equipe de desenvolvedores de software tendo cada membro seu papel bem definido.. A equipe como um todo é autora do projeto, mas apenas os desenvolvedores são autores do código- fonte. Determinar a quantidade de código que um desenvolvedor adicionou a um projeto não é uma tarefa difícil, basta somar todas as linhas adicionadas por ele, mas saber o quanto de código que esse desenvolvedor adicionou ainda permanece no código fonte não é algo trivial de ser feito.

Imagine a seguinte situação: três amigos decidem iniciar um projeto, cada um desenvolve sua parte por conta própria. Em certo momento do projeto uma empresa faz uma oferta para comprar o software o qual os três amigos desenvolveram. Os amigos discutem o valor e concluem que o valor é aceitável pelo projeto. Visto que cada um dos desenvolvedores terá desenvolvido quantidades diferentes de código não seria justo dividir igualitariamente o valor pago pela empresa pelos três. Intuitivamente, o mais justo seria definir a quantidade de autoria do projeto referente a cada desenvolvedor como a relação entre a quantidade de linhas de autoria de um desenvolvedor dividida pela quantidade total de linhas do código. Com base em tal informação consegue-se estabelecer a contribuição de cada desenvolvedor no projeto e, no caso do exemplo mencionado, definir o valor justo a ser recebido por cada desenvolvedor do projeto.

O exemplo apresentado ilustra o fato de que quantificar autoria de uma equipe de desenvolvimento de software dentro de um projeto pode ainda não contar com uma abordagem bem definida, havendo ainda a possibilidade de que o método e técnicas escolhidos para aferir a autoria não reflitam a realidade. A autoria atualmente é quantificada de diversas formas para resolver diversos problemas como análise de qualidade de software, análise forense de software e melhorias da manutenção de software (MENG et al., 2013). A quantidade de *commits* - linhas adicionadas e removidas - podem não representar a autoria de cada um dos desenvolvedores de um sistema em um determinado momento. O trabalho de levantar o quanto trabalho de desenvolvedor ainda permanece no projeto pode ser mecânico e não refletir a quantidade real.

A avaliação da quantidade de autoria pode ser mais detalhada e refinada, com uma abordagem definida para quantificar a contribuição de um desenvolvedor para que tenha mais acurácia e que reflita melhor a realidade do projeto. A investigação relacionada a autoria não deve apenas medir a contribuição de um desenvolvedor no projeto mas também

o quanto do seu trabalho ainda se mantem no projeto. Ou seja, se caso algum componente que o desenvolvedor fez for removido, ele terá seu autoria no projeto diminuída, caso for adicionado, terá sua autoria aumentada, caso seja substituído terá sua autoria reduzida, entre outros casos.

Em se tratando de versionamento de código durante seu desenvolvimento, o uso desse tipo de ferramenta é de extrema relevância para o projeto desenvolvido, haja visto que um versionador mantem o histórico da evolução do software, associa autores a cada trecho de código escrito além de paralelizar o desenvolvimento do projeto(ROCHKIND, 1975). Além disso, esse tipo de ferramenta é capaz de estabelecer uma linha do tempo da evolução do software desenvolvido de modo a rastrear quando e onde foram adicionados cada um dos trechos de código. Existem vários softwares de versionamento a citar Git (COMMUNITY, 2018), Apache Subversion (SVN) (APACHE, 2018), Microsoft Team Foundation Version Control (TFVC) (MICROSOFT, 2018) como os mais conhecidos e utilizados atualmente.

Gerenciar um histórico de desenvolvimento via ferramenta de versionamento é fundamental para definir os autores do código e permitir que pessoas trabalhem simultaneamente fazendo com que o desenvolvimento seja fluído e as integrações de código sejam mais fáceis de serem realizadas. O mercado adotou o Git como padrão de mercado sendo usado pelos desenvolvedores de software na indústria (MAJUMDAR et al., 2017). Essa ferramenta é capaz de manter todas as versões dos arquivos, seus autores, datas e horas em que foram modificados. Este trabalho será baseado no sistema de versionamento o Git pois, além de ser um software de código aberto, ela possui uma das maiores plataformas de código que adota essa ferramenta de versionamento sendo essa plataforma chamada de GitHub.

A partir das características e informações disponibilizadas pelo software de versionamento será proposta uma abordagem juntamente com uma série de métodos e técnicas nas quais terão coleta de métricas que podem representar a autoria deixada por um desenvolvedor ao longo do tempo ou em um instante de tempo. Definindo assim melhor as métricas que relacionam a autoria do código, determinando a autoria do código por meio de uma abordagem definida e concisa.

1.2 Problema Geral

A definição da taxa de autoria é intuitivamente simples e definida como sendo uma relação entre a quantidade de linhas de autoria de um determinado desenvolvedor e a quantidade total de linhas de código do projeto. Contudo as dificuldades em computar tal taxa de autoria aparecem em nível técnico especialmente no modo como as medidas de autoria são afetadas com base nas alterações feitas em arquivo. Em outras palavras

alterações em código podem não refletir adequadamente a autoria de um desenvolvedor.

A alteração mais simples possível em uma linha de código se dá em nível de caractere. Uma simples adição, remoção ou substituição de um caractere por um desenvolvedor faz com que sistemas de versionamento atribua a esse desenvolvedor a autoria de toda linha de código. Obviamente atribuir a autoria a um desenvolvedor por qualquer alteração que ele promove em uma linha de código não reflete adequadamente a relação de autoria daquele e dos demais desenvolvedores. O motivo pelo qual esse tipo de alteração afeta a medição de autoria se dá pelo fato de sistemas de versionamento lidarem com alterações cuja granularidade é expressa por linhas adicionadas ou removidas em cada *commit*. Para que a medição de autoria reflita adequadamente a relação entre os desenvolvedores e o código faz-se necessário uma medição de granularidade mais fina, no caso, caracteres alterados por cada desenvolvedor.

Ainda que as alterações realizadas sejam de granularidade maior, *i.e* alterações realizadas sobre uma ou um conjunto de linhas de código, elas podem também não refletir adequadamente a autoria. Uma remoção de linhas de código seguida por um *commit* e a reinserção das mesmas linhas de código (sem alterações em seu conteúdo) fazem com que a autoria de tais linhas seja daquele desenvolvedor que as reinseriu. Esse tipo de alteração pode não ser comum ou corriqueira durante o desenvolvimento de um software mas mostra claramente uma limitação na definição de autoria das ferramentas de versionamento atuais.

Isso exposto pode-se concluir que atualmente a medição de autoria é frágil e suscetível a pequenas e/ou simples alterações em código. Tal suscetibilidade faz com que a autoria calculada não represente, necessariamente, a verdadeira relação entre desenvolvedores e código-fonte. Em resumo, o problema pode ser definido como segue:

Problema: A granularidade das informações de autoria de código-fonte oferecidas pelos sistemas de versionamento atuais não refletem adequadamente a relação entre os desenvolvedores e código-fonte.

1.2.1 Objetivos

Com base no contexto e problemas mencionados acima, o objetivo geral deste trabalho é "Definir uma abordagem para avaliar a autoria de desenvolvedores em projetos de software".

Desta forma, a partir do objetivo geral, foram definidos os seguintes subobjetivos:

- Identificar quais abordagens, métodos e técnicas que são utilizadas para quantificação da autoria de um desenvolvedor;
- Identificar métricas que podem ajudar na taxa de autoria;

- Elaborar modelo de abordagem, no contexto de análise de autoria;
- Elaborar modelo de métodos, dentro do contexto da abordagem;
- Elaborar modelo de técnicas, dentro do contexto da métodos;
- Elaborar modelo de métricas, nas quais serão os objetivos específicos das métricas;
- Avaliar se os modelos propostos representam a autoria dos desenvolvedores em um projeto.

1.3 Estrutura do trabalho

Além desta introdução, este trabalho está organizado nos seguintes capítulos:

- Capítulo 2 - Revisão de literatura;
  - A revisão de literatura é uma fundamentação teórica na qual o trabalho se baseia, possibilitando construir as ideias para s abordagem do problema apresentado. A avaliação de autoria dos desenvolvedores. Nesse capítulo terá pesquisa em relação a autoria de código, formas de contar as linhas, definição da abordagem, métricas, sistemas de versionamento, ferramentas e linguagem de programação.
- Capítulo 3 - Metodologia;
  - O capítulo de metodologia trata como o trabalho foi realizado, seus métodos e procedimentos. Será descrito como o procedimento de pesquisa foi feito.
- Capítulo 4 - Proposta;
  - Esse capítulo apresenta uma solução ao problema relatado, quantificar a autoria do código fonte, no qual foi apresentado no capítulo de introdução e embasado no capítulo de revisão de literatura. A solução envolve a definição de uma abordagem para resolver o problema, sendo que a abordagem deve ter métodos e técnicas, bem definidos para que de fato seja uma abordagem. Nesse capítulo é definido as métricas e como elas são coletadas.
- Capítulo 5 - Considerações finais.
  - O capítulo de considerações finais possui as próximas etapas do estudo juntamente com os resultados e limitações desse estudo.

2 Revisão de Literatura

A revisão de literatura, é um capítulo importante neste trabalho, pois ele embasa teoricamente o conteúdo abordado, que no caso é uma abordagem para quantificar a autoria de código fonte. A definição de abordagem é a descrição da natureza de uma assunto a ser estudado, muitas vezes é indiscutível a menos que seja em relação a eficácia dos métodos que provem dele (ANTHONY, 1963). A abordagem apesentada nesse trabalho terá métodos para a discussão em relação a sua eficácia, a definição de método é um plano geral para uma procedimento ordenado, no qual não se contradiz, sendo que todo o método é baseado em uma abordagem (ANTHONY, 1963). Incluso no método, existem técnicas que por definição são artifícios ou estratégias específicos para atingir um objetivo imediato (ANTHONY, 1963). Os objetivos imediatos das técnicas desse trabalho serão as métricas que remetem a autoria de código de um projeto no git.

Para que seja possível elaborar métodos, técnicas e os objetivos imediados das técnicas que serão as métricas é necessário uma pesquisa teórica, com os seguintes assuntos:

- Autoria de Código;
  - Problemas nos quais a autoria de código é a solução;
  - Abordagens, métodos e técnicas que são utilizadas para quantificação da autoria;
- Abordagem para coleta de métricas;
- Métrica;
  - Quais métricas são usadas para quantificar autoria;
- Sistemas de Versionamento;
  - Sistemas de versionamento usados atualmente e suas características;
- Tecnologias e Ferramentas;
  - Linguagens de programação;
  - Pacotes e softwares que podem ajudar na quantificação da autoria;

2.1 Autoria de Código

A autoria é a informação de quem escreveu aquele fragmento de texto. A atribuição de autoria é de baixa complexidade quando é escrita por uma única pessoa, sendo

assim, a autoria do trabalho inteiro pertence a ela. A complexidade aumenta assim que mais pessoas são associadas ao trabalho, mais pessoas escrevem o mesmo texto, que no caso o objeto de estudo deste trabalho é o um projeto de software, sendo seu trecho definido como um fragmento de código fonte. Em um desenvolvimento de software ocorrem mudanças contínuas em várias partes do código, podendo mais de uma pessoa ter escrito ou modificado um trecho específico de código.

A autoria em sua essência é a informação de quem escreveu aquele fragmento de código, mas existem outras características a serem vistas no estudo. Como por exemplo se o trecho de código foi escrito por mais de um desenvolvedor, ou até mesmo a representação de autoria ao longo do tempo, o quanto das contribuições de código dos autores ainda existe no projeto e não foram retiradas.

Existem certos cenários onde a propriedade do código se aplica e se tem uma relevância na solução, alguns desses cenários são:

- Qualidade de software
  - A autoria do código simplifica a responsabilização dos gerentes e ajuda a encontrar especialistas para as tarefas que podem ser solicitadas (RAHMAN; DEVANBU, 2011), por exemplo se caso exista um módulo no qual um desenvolvedor tem mais autoria é muito provável que ele seja o mais especialista em incrementa-lo, gerando assim uma maior qualidade. Entretanto altos níveis de autoria em relação ao código fonte, podem levar a insuficiência de qualidade (RAHMAN; DEVANBU, 2011).
- Análise de software forense
  - A determinação de autoria do código fonte, tem implicações sobre a comunidade de segurança em relação a análise forense (ROSENBLUM; ZHU; MILLER, 2011).
- Melhoramento da manutenção
  - Os sistemas de software evoluem, sendo que nem todos os desenvolvedores conhecem todo o sistema. Precisando assim saber qual desenvolvedor tem mais conhecimento sobre qual parte do sistema (GIRBA et al., 2005).
- Rastreamento de *bug*
  - Determinando a quantidade de autoria é possível por exemplo melhorar a manutenção de um software e rastrear sobre o *bug*, por exemplo, quando foi inserido e quais desenvolvedores o inseriram o *bug* (YIN et al., 2011).

As ferramentas usadas hoje em dia podem aproximar o nível de autoria, assumindo que a última pessoa que mudou a linha do código é autora da linha inteira, desconsiderando qualquer mudança intermediária (MENG et al., 2013).

Atualmente existem diversos modos de quantificar a autoria, esses modos usam diferentes métricas para quantificar a autoria de um desenvolvedor ao longo do desenvolvimento do projeto de software. Uma das métricas mais antigas que envolve tamanho é LOC.

A base da maioria das métricas envolve o SLOC*(Source Lines of Code)* ou simplesmente LOC*(Lines of Code)*, uma métrica de quantidade de linhas de código fonte. LOC é uma métrica tradicional e a mais popular métrica de tamanho, surgiu a princípio como indicador custo de software (NGUYEN et al., 2007). O LOC pode ser definido em duas variações de *Physical LOC*, *Logical LOC*.

- *Physical LOC* remete a quantidade de linhas de código físico, ou seja, comentários e linhas em branco contam como *Physical LOC*.

- *Logical LOC* cotam as linhas que contem lógica envolvida, ou seja, as linhas que contem comentários e linhas em branco não são contabilizadas(NGUYEN et al., 2007).

O *Physical LOC* não tem o seu uso tão comum em sistemas de contagem, sendo mais usado o *Logical LOC*, visto que a maioria dos sistemas pelo menos desconsidera linhas em branco. Sendo *Logical LOC* melhor visto para determinar valor e relevância a contagem, pois essa métrica só considera aquilo que tem valor lógico (NGUYEN et al., 2007).

O LOC pode ser associado a cada desenvolvedor do projeto de software. Deste modo, cada desenvolvedor tem uma quantidade de linhas desenvolvidas no projeto. Os problemas começam quando vários desenvolvedores alteram o mesmo arquivo, gerando divergências nas contagem das linhas.

Uma outra forma de contar a autoria do código fonte é por meio da quantidade de *commits* dentro de um projeto. Esta forma de realizar a contagem da autoria, foi usada para determinar os efeitos da qualidade do software. O somatório dos *commits* é feito para cada desenvolvedor, deste modo é determinado os desenvolvedores *major* e *minor*. O *TOP CONTRIBUITOR* é definido por aquele que tem o maior número do somatório de *commits*. (BIRD et al., 2011)

Tendo a informação de quantidade de autoria por componente ou arquivo, é possível definir os autores daquele componente ou porção de arquivos. Determinando assim os responsáveis pela escrita daquele pedaço de código (GIRBA et al., 2005).

2.2 Métrica de Software

O termo Métrica de Software foi definido como "Uma unidade de medida para o produto de software ou processo relacionado a software", foi definido por Hamer e Frewin em 1985(CôTé et al., 1988). Nesse estudo, que se trata de uma métrica de autoria, está relacionada ao produto de software e também ao processo de desenvolvimento do software.

As métricas de software são compreendidas como um conjunto de técnicas associadas a medidas quantitativas e qualitativas em produtos de software e no processo de desenvolvimento, sendo que essas técnicas tentam quantificar a qualidade do produto de software, como manutenibilidade, legibilidade, complexidade, confiabilidade, entre outras técnicas. (LECCISO; MAINETTI; MORASCA, 1986)

As técnicas tentam por meio de parâmetros simples, medirem os softwares ou processo de software. Tendo como objetivo final a capacidade de atribuir um significado preciso e quantitativo das características qualitativas do software, com intuito de medir e/ou prever com precisão(LECCISO; MAINETTI; MORASCA, 1986).

2.2.1 *THE GOAL QUESTION METRIC APPROACH*

Desenvolver um software é um trabalho de engenharia com uma certa complexidade, portanto é necessário ter meios de saber o quão bem ou mal o projeto progride. A medição é um mecanismo que cria uma memória da equipe e auxilia a respostas a uma variedade de questões relacionadas a processo de software ou produto de software. Aumentando assim o suporte aos projetos, ajudando a prever custos, avaliar a qualidade, a densidade de defeitos. De uma forma geral avaliar os pontos fracos e fortes do processo de desenvolvimento e do produto de software. Além disso a medição pode avaliar o progresso do desenvolvimento do software e tomar ações corretivas com base na análise das métricas (BASILI; CALDIERA; ROMBACH, 1999).

A abordagem GQM,foi proposta por Victor R. Basili em 1999, foi um estudo que estabeleceu para uma métrica ser efetiva ela deve possuir 3 pontos:

- Focada em objetivos;
- Aplicado em todo ciclo de vida do projeto;
- Baseado na interpretação e caracterização do entendimento organizacional.

A abordagem objetivo, questão e métrica, GQM em tradução livre, baseia-se na hipótese de que a organização deve medir de uma forma propositiva, definindo primeiro os objetivos($GOAL$) para que os dados coletados sejam objetivos operacionais e não sejam apenas coletados apenas por aparência. Desta forma, a organização deixará de forma clara

e objetiva as suas necessidades de informações que devem ser quantificadas, e analisadas para a possível obtenção dos objetivos (BASILI; CALDIERA; ROMBACH, 1999).

A aplicação do modelo de Basili possui três níveis:

- *GOAL* (objetivo)
  - O objetivo é definido por uma variedade de pontos. Os objetos de medição normalmente são: produtos, processos e recursos. Produtos é tudo aquilo que é produzido pela equipe como: documentos, diagramas, código fonte, especificações. Processos, é relacionado ao processo de desenvolvimento do software, como: suas atividades, papéis e artefatos. Os recursos são pessoas, equipamentos e o próprio espaço de trabalho.
- *QUESTION* (questão)
  - Agrupamento de perguntas com o propósito de caracterizar o objeto de medição, podendo ser o produto, processo ou recurso. Em relação ao problema de qualidade selecionado e determinar sua qualidade.
- *METRIC* (métrica)
  - Métricas são dados quantitativos, que procuram responder cada questão em nível operacional de forma mensurável.

Desta forma o GQM nos permite ter uma visão dos objetivos, das questões e das métricas, para que seja possível resolver os problemas apresentados. Além disso é necessário o estudo em outros assuntos como por exemplo o versionamento.

2.3 Versionamento

Um sistema de controle de versionamento de código fonte, é uma ferramenta de software feita auxiliar os projetos de programação com suas alterações no código fonte. O sistema deve prover facilidades para armazenar, editar e recuperar todas as versões dos módulos por número de versão, de modo com que seja controlado os privilégios de alteração no código fonte (ROCHKIND, 1975).

O versionamento surgiu para resolver os problemas entre as diferenças mudanças que acontecem em um software, em suas diferente versões. Desta forma o sistema de versionamento surgiu para armazenar um histórico de versões, reunindo informações das modificações realizadas no desenvolvimento de software. Um sistema de controle de versionamento de código fonte que possuem as seguintes características: (ROCHKIND, 1975)

- Armazenamento;
  - Um versionador precisa ter a funcionalidade de armazenamento, sem o armazenamento não é possível prover as características de um versionador de código. Todas as versões dos códigos, sem replicações devem ser armazenadas, sendo todas as versões acessíveis, para as pessoas que tem permissão de acessar.
- Identificação;
  - A identificação em um sistema de versionamento deve existir para que seja possível atribuir a responsabilidades dos autores por cada treço de código e arquivo. O sistema previamente configurado, irá reunir informações de identificação do desenvolvedor, além do número de versão, data, hora e uma mensagem identificando aquele envio de código.
- Proteção;
  - Somente aqueles com permissão de alteração poderá realiza-la. Ou seja, os módulos nos quais os programadores não tem acesso não seriam possíveis de serem alterados.
- Documentação.
  - Tempo:
    - Quando foi feita, armazenando a data e hora da modificação;
  - Identificação:
    - Quem realizou a modificação;
  - Localização:
    - Onde a modificação foi feita;
  - Motivo:
    - O motivo pelo qual a modificação foi feita.

2.3.1 Ferramentas

As ferramentas de versionamento de código fonte são da mais diversas, exitem ferramentas open-source e comerciais. Algumas das ferramentas mais usadas são Git (COMMUNITY, 2018), Apache Subversion (SVN) (APACHE, 2018), Microsoft Team Foundation Version Control (TFVC) (MICROSOFT, 2018), que serão melhor descritas nesse sub-seção.

A ferramenta TFVC é uma solução da Microsoft Corporation, comercial de código fechado. Possui todas características fundamentais para definir uma ferramenta de versionamento. Além disso possui uma ferramenta de resolução de conflitos, na qual possui

uma grande usabilidade e facilidade para o usuário. A grande limitação dessa ferramenta é o fato dela ser paga e de código fechado, restringindo o uso da comunidade e o desenvolvimento de componentes para serem integrados a ela.

O SVN por sua vez é uma ferramenta de código aberto, produzida pela The Apache Software Foundation. É um dos mais antigos versionadores de software, que ainda são utilizados atualmente, possui uma série de softwares em seu versionamento. Ele possui a vantagem de ser de código aberto, permitindo assim, o desenvolvimento de componentes pela comunidade, além de ter um alcance maior de usuários. Mas, uma das desvantagens é que para realizar alterações, os *commits* é necessário estar conectado ao repositório remoto, ou seja, não é possível utilizar SVN offline, e logo após subir as alterações.

A ferramenta de versionamento Git, assim como SVN, possui o seu código aberto, mas além disso possui a maior plataforma de repositórios de código aberto, o GitHub. Uma vantagem dele em relação ao SVN é a possibilidade de realizar o *commit* offline e em algum momento enviar as alterações para o repositório remoto. Os novos projetos de código aberto estão presentes no GitHub, por exemplo: Linux, Atom, NGINX entre outros.

Este trabalho irá considerar o Git como a principal ferramenta de controle de versionamento de código fonte, pois é amplamente usado na comunidade de software livre, além de ser amplamente usado por desenvolvedores na indústria de (MAJUMDAR et al., 2017).

A Ferramenta Git já possui um comando no qual consegue representar, de certa forma, a autoria do código fonte dos desenvolvedores, o comando é:

Comando bash

```
git -blame arquivo.code
```

Sendo necessário estar dentro do repositório do projeto a ser analisado, enviando o nome do arquivo como parâmetro, que no caso do comando acima é **arquivo.code**.

Métodos atualmente utilizados, como git -blame, para obter autoria em nível de linhas, possuem perda de informações. Pois uma linha de código pode ser alterada várias vezes por desenvolvedores diferentes, seja para corrigir erros ou evoluir o código. Sendo que essas alterações integram o histórico de uma linha de código(MENG et al., 2013). O git -blame relata apenas a última alteração da linha, ou seja, se caso um desenvolvedor mudar apenas um carácter da linha, ele recebe a autoria daquela linha inteira.

Para resolver essa questão de apenas o último autor ser o autor da linha é necessário criar ponderação para as autorias das linhas, ou seja, fracionar a autoria da linha(MENG et al., 2013). Para fracionar a autoria da linha é necessário ter o histórico de toda a linha, desde o seu surgimento até o momento a ser medido. A autoria da linha pode se descrita

como: A quantidade de caracteres atribuídos ao desenvolvedor divido pela quantidade total de caracteres da linha.

Com o fracionamento da linha a limitação "medição de autoria"é contornada. Pois se caso mais de um desenvolvedor trabalhar na mesma linhas, todos vão receber a autoria e não só apenas o último desenvolvedor que modificou a linha. Entretanto, a complexidade de contagem aumenta muito, por conta de toda lógica envolvida para se contar por caracteres. Uma das razões na qual aumenta a lógica envolvida são as formas de contar os caracteres modificados em uma linha. A modificação pode ser inserida no inicio, no final ou no meio, além disso o desenvolvedor pode apagar parte da linha e adicionar mais caracteres. Ou seja, não existe um único meio de se contar as linhas, se caso forem contadas por caracteres. A contagem de caracteres pode também ser muito onerosa na questão de recursos computacionais, dependendo do tamanho do projeto a ser analisado. Todas linhas de código vão para memória do computador, inclusive suas modificações e todas suas versões a serem comparadas. Se o método de contagem por caracteres for comparado com o método no qual considera o último desenvolvedor que modificou a linha como autor da linha inteira é muito menos complexo, logicamente e computacionalmente.

2.3.2 Linguagem de programação

JavaScript é uma linguagem de programação útil, sendo em encaixada em aplicações da web e várias outras soluções, mas nem sempre fácil de ser desenvolvida, necessitando de técnicas de alto nível em programação para serem desenvolvidas(OYA; KASHIWAKURA, 2016).

O JavaScript iniciou primeiramente para suprir funcionalidades no lado do cliente, o chamado *client-side*, para que os navegadores tivessem mais interatividade sem precisar da participação do servidor, o *server-side*, na lógica de algumas funcionalidades. Essas funcionalidades, *client-side*, são executadas diretamente no navegador do usuário. Com o avanço da tecnologia, outras soluções foram desenvolvidas e surgiu a possibilidade de não só usar o JavaScript no navegador. O JavaScript passou a ser executado no servidor, no cliente, em aplicações independentes e scripts para uso geral. O uso do JavaScript para funcionalidades fora do navegador, como primeiramente foi planejado foi possível por conta do Node.js.

2.3.3 Node.js

Node.js é um ambiente JavaScript baseado no motor JavaScript do Google Chrome, que se chama V8 engine, sendo esse motor focado em performasse, procurando baixo consumo de memória, foi implementado nas linguagens de programação C e C++ (TILKOV; VINOSKI, 2010). Esse ambiente serve para as aplicações em JavaScript serem

executadas.

O node, como é chamado pela comunidade, é amplamente usado em aplicações modernas, inclusive muitas empresas o usam, empresa grandes como Facebook e Google possuem pacotes em node. Os pacotes são funcionalidades prontas e empacotadas para o uso. O uso de bibliotecas e pacotes feitos por terceiros tem agradado bastante os desenvolvedores, visto que esse ecossistema possui benefícios como: qualidade, rapidez, e facilidade de uso. (LERTWITTAYATRAI et al., 2017).

2.3.4 NPM - *Node Package Manager*

Existe uma comunidade no qual os pacotes em node são distribuídos, essa comunidade se chama NPM(*Node Package Manager*). O Gerenciador de pacotes para node, tradução para o NPM, é um grande repositório de pacotes na linguagem JavaScript, o NPM combina um conjunto de ferramentas de código aberto para que os desenvolvedores possam empacotar, descrever, atribuir dependências e registrar o pacote(WITTERN; SUTER; RAJAGOPALAN, 2016). Os pacotes podem ser submetidos a plataforma online, permitindo que outras pessoas o usem. Exitem muitas funcionalidades prontas, para usala basta ler a documentação anexada ao pacote, estar de acordo com a licença do software e sua versão do node ser compatível.

2.3.5 Pacotes NPM

A vantagem de se usar o NPM é grande, por conta de vários algoritmos já estarem implementados e empacotados na plataforma. Assim os programadores não precisam programar novamente algum algoritmo necessário em seu sistema. Basta procurar dentro da plataforma NPM, se caso existir ler a documentação e anexar o pacote ao projeto NPM.

Existem vários pacotes que podem ser usados nesse projeto, no qual irá auxiliar a quantificar as autorias de linha por meio da ferramenta git, esses pacotes de ferramentas são:

- **Pacote:** blamer;
  - **Licença:** MIT;
  - **Descrição:** O pacote blamer é uma ferramenta usada para pegar a informação sobre o autor do código, de acordo com o sistema de versionamento. Sendo que ele suporta SVN e Git;
  - **Utilidade:** É possível buscar dentro de um arquivo os autores dele, a ferramenta funciona como um parse do git blame para objetos JSON;
  - **Link:** https://www.npmjs.com/package/blamer

- **Pacote:** gitlog;
  - **Licença:** BSD
  - **Descrição:** Um parse do Git log para Node.js;
  - **Utilidade:** Permite pegar a lista de commits, contendo as informações: autor, indentificador, data, hora e arquivos dentro do commit;
  - **Link:** https://www.npmjs.com/package/gitlog
- **Pacote:** underscore;
  - **Licença:** MIT;
  - **Descrição:** O pacote underscore é um utilitário para JavaScript que fornece funções sem estender do núcleo do JavaScript, aumentando assim a facilidade do desenvolvimento;
  - **Utilidade:** Contar atributos dentro dos objetos JavaScript, que no caso serão os autores;
  - **Link:** https://www.npmjs.com/package/underscore
- **Pacote:** json-table.
  - **Licença: MIT;**
  - **Descrição:** O pacote json-table disponibiliza um componente que permite uma maneira fácil de criar tabelas em elementos ASCII;
  - **Utilidade:** Criar tabelas com os dados de autoria;
  - **Link:** https://www.npmjs.com/package/json-table

3 Metodologia

Neste capítulo, apresenta-se a metodologia de pesquisa selecionada neste trabalho. Para isso, inicia-se com a classificação da metodologia de pesquisa, como o tipo, os procedimentos de pesquisas, e as técnicas de coletas de dados adotadas.

3.1 Classificação da metodologia de pesquisa

A metodologia de pesquisa adotada neste trabalho foi classificada quanto à natureza, à abordagem, à tipologia, aos procedimentos técnicos e as técnicas de coleta de dados.

Quanto à natureza, esta pesquisa é aplicada, pois, tem como objetivo a geração de conhecimentos dirigidos à solução de problemas específicos (GIL, 2008), definindo um modelo de medição para determinar o quanto um determinado indivíduo é autor de um projeto usando o GIT.

Quanto à abordagem, é classificada como quantitativa, pois os objetivos dos dados coletados são numerais(YIN, 1989), que se isolados não refletem a qualidade sendo que o objetivo principal do estudo é definir a quantidade e taxa de autoria uma informação quantitativa que melhora a acurácia de problemas de teor qualitativo, mas trataremos neste trabalho apenas da quantidade de autoria.

Quanto aos fins, a presente pesquisa caracteriza-se como descritiva, pois estudos de natureza descritiva tem seus objetivos relacionados a descrever algo ou um fenômeno(YIN, 1989). Neste estudo serão detalhadas as características em relação aos dados de autoria do código fonte, além disso são apresentadas métricas e um método de medição com coleta de métricas, algoritmo de medição, regras de coleta, tudo relacionado a autoria de código fonte.

Quanto aos procedimentos técnicos de coleta de dados, os meios de investigação foram: pesquisa bibliográfica e documental.

3.2 Plano metodológico adotado

Para Gil (2008), um processo de pesquisa envolve: planejamento; coleta de dados com a respectiva análise e interpretação dos mesmos e, em seguida, a redação do resultado. E cada uma dessas grandes fases pode ser subdividida em outras mais específicas, dando origem aos mais diversos esquemas. Nesta seção são descritos os passos seguidos na condução desta pesquisa e os instrumentos utilizados em cada passo.

Nesta pesquisa, o processo adotado compreende as fases: (1) Planejamento; (2) Coleta de Dados; (3) Análise e Interpretação dos dados, e (4) Redação dos Resultados. A partir da metodologia de pesquisa adotada e da determinação das fases que a configuram, foi elaborado um plano metodológico e foram definidas as etapas da pesquisa, nas quais são empregados os procedimentos e as técnicas de coleta de dados.

1. **Planejamento do trabalho**: Esta fase tem como objetivo a delimitação do escopo do trabalho por meio da definição da questão de pesquisa, do objetivo geral e dos objetivos específicos.Estas definições são apresentadas na Introdução deste trabalho (Capítulo 1).

2. **Coleta de dados**: Os procedimentos de pesquisa utilizados para coleta de dados foram:

   - Pesquisa bibliográfica: A revisão de literatura foi feita com o objetivo de pesquisar artigos científicos que embasassem as obtenção das métricas de autoria por um sistema de versionamento. Com isso foram pesquisados artigos relacionado a métricas, sistema de versionamento, quantificação de autoria, uso de métrica de autoria e abordagens de medição.

3. Redação dos resultados: A redação dos resultados desta pesquisa é constituída pelo referencial teórico deste trabalho, obtido por meio da pesquisa bibliográfica; a criação e validação do modelo de medição para cálculo da taxa de autoria de projetos no Git; refinamento do modelo de medição a partir do resultados obtidos na validação; redação da Proposta de medição de autoria de projetos no Git. Estes resultados são apresentados em forma de capítulos nesta monografia, incluindo a conclusão e sugestão de trabalhos futuros.

3.3 Procedimentos para definição do modelo de medição de Autoria

A definição da abordagem, envolve o modo como coletar as métricas e o que é de fato necessário à ser coletado. Ou seja, definir um objetivo no qual as métricas que devem ser coletadas devem contribuir o alcance desse objetivo.

Na definição do método de medição constam as etapas nas quais o método deve passar para que a autoria seja coletada. Ademais devem conter os pormenores da técnica apresentada, como: regras de coleta, o que deve ser contabilizado e o que não deve ser contabilizado e suas limitações.

A proposta de solução envolve uma ferramenta de validação, na qual irá comprovar e automatizar o método descrito no parágrafo anterior, explicar as limitações do projeto,

prováveis impedimentos e as características dos projetos nos quais a ferramenta poderá ser usada. A proposta da solução será descrita brevemente no TCC 1, implementado no TCC 2, justamente com as tarefas subsequentes.

Na tarefa de examinar as ferramentas, será feito um estudo nas ferramentas já presentes atualmente, para alcançar possíveis soluções das etapas descritas no método de coleta de autoria. Esse estudo irá auxiliar o desenvolvimento da validação e será em feito principalmente na ferramenta de controle de versão.

O desenvolvimento da ferramenta será feito logo após os estudos das ferramentas, sendo que a ferramenta irá seguir os passos descritos no método de coleta de autoria.

A análise dos resultados será um um estudo dos resultados entregues pela plataforma após a solicitação de contagem de autoria de um projeto, essa análise irá determinar se a ferramenta conseguiu chegar nos números esperados.

A conclusão do estudo irá deter as informações de dificuldades, limitações, conquistas e próximas etapas do estudo.

O uso de uma metodologia contribui na qualidade do estudo realizado, mantendo um método de trabalho juntamente com objetivos e resultados esperados. Fazendo com que o estudo avance de maneira gradual e correta.

4 Proposta

4.1 Objetivo

O objetivo da proposta deste trabalho é validar um método de medição para métrica de taxa autoria de um desenvolvedor, por meio do desenvolvimento e análise de uma ferramenta de linha de comando. A ferramenta usará o software Git e pacotes em NPM para a coleta das informações. Para determinarmos as métricas e o método de medição, usaremos a abordagem GQM.

4.2 GQM

A metodologia GQM (Goal, Question, Metric), descrita por Victor Basili, revisada na seção 2 deste trabalho foi escolhida para definir a abordagem da medição:

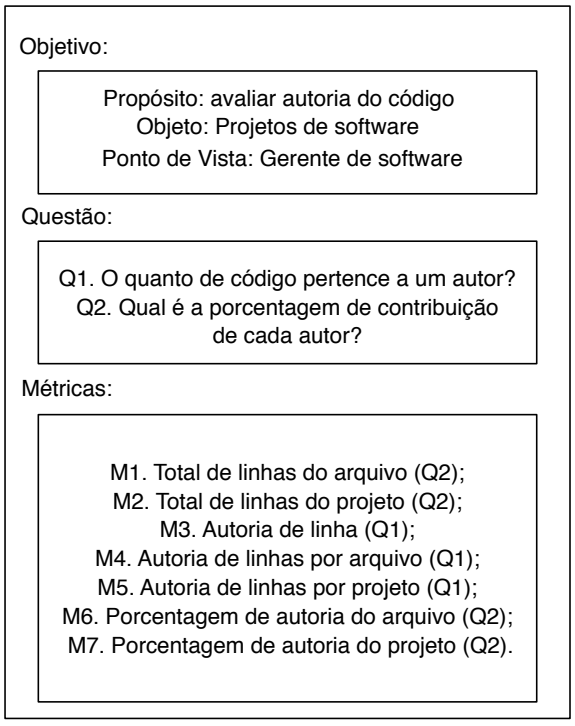

Figura 1 – Objetivo, Questões, Métricas

Deste modo teremos o seguinte objetivo de medição:

Analisar: Projetos de software na ferramenta git

Com o propósito: avaliar

Com respeito a: autoria do código

Do ponto de vista: do gerente de software

No contexto de: Desenvolvimento de software

O propósito é analisar a quantidade de autoria, visto que queremos obter a taxa de autoria dos desenvolvedores do projeto. O objeto de estudo é um projeto de software, que no caso está no repositório de teste versionado no Git, nesse repositório terá todas as informações necessárias para coleta da informação de autoria. Visto que todos os versionadores possuem as características que definem uma ferramenta de controle de versão. Sendo essas características permitem a coleta de autoria do código. Ou seja, com qualquer versionador, por meio de um método, é possível se obter a autoria de um código. Porém, nesse trabalho a tecnologia será usada será para a ferramenta de versionamento Git, por conta da sua comunidade ser grande, o código fonte ser aberto e possuir um grande repositório remoto o GitHub. O ponto de vista que foi escolhido é do Gerente de Software, por conta do papel ser responsável pelos custos, atribuição de trabalho e manutenção dos riscos de um projeto de software.

A Questão é um conjunto de perguntas usado para caracterizar a avaliação dos objetivos a serem alcançados. As perguntas caracterizam o objeto de medição em relação ao problema de qualidade. A Questão a ser abordada é: determinar a taxa de autoria. Para podermos responder essa questão é necessário ter as métricas a serem coletadas. As métricas que podem responder essa questão são: quantidade de autoria por aquivo, quantidade de linhas total do arquivo, quantidade de linhas total do autor, quantidade total de linhas do projeto e finalmente a taxa de autoria do arquivo e do projeto. Sendo a taxa de autoria consequência e uma medida derivada das métricas anteriores.

A quantidade de autoria do arquivo, intuitivamente, é quantidade de linhas pertencente a cada autor do arquivo. A quantidade de linhas total do arquivo, é simplesmente a métrica LOC definida na seção de revisão de literatura desse trabalho, para um arquivo do projeto. A quantidade de linhas total do autor é o somatório de todas as linhas de um autor, de todos os arquivos do projeto. A quantidade de linhas total do projeto é a métrica de LOC de todos os arquivos do projeto.

4.3 Processo de medição

O processo de contagem ocorre por meio do comando Git blame, no qual retorna a última pessoa que modificou aquela linha dentro do projeto. Esse comando é provido pela ferramenta de versionamento Git. A lista de commits é processada de modo com que todos os commits sejam recuperados. O software proposto para realizar o processo de medição irá até cada commit, gerando o git blame de todas as linhas do projeto naquele

instante de tempo no qual o commit foi realizado.

4.3.1 Definindo o algoritmo de medição

O algoritmo de medição é representado pela Figura 2.

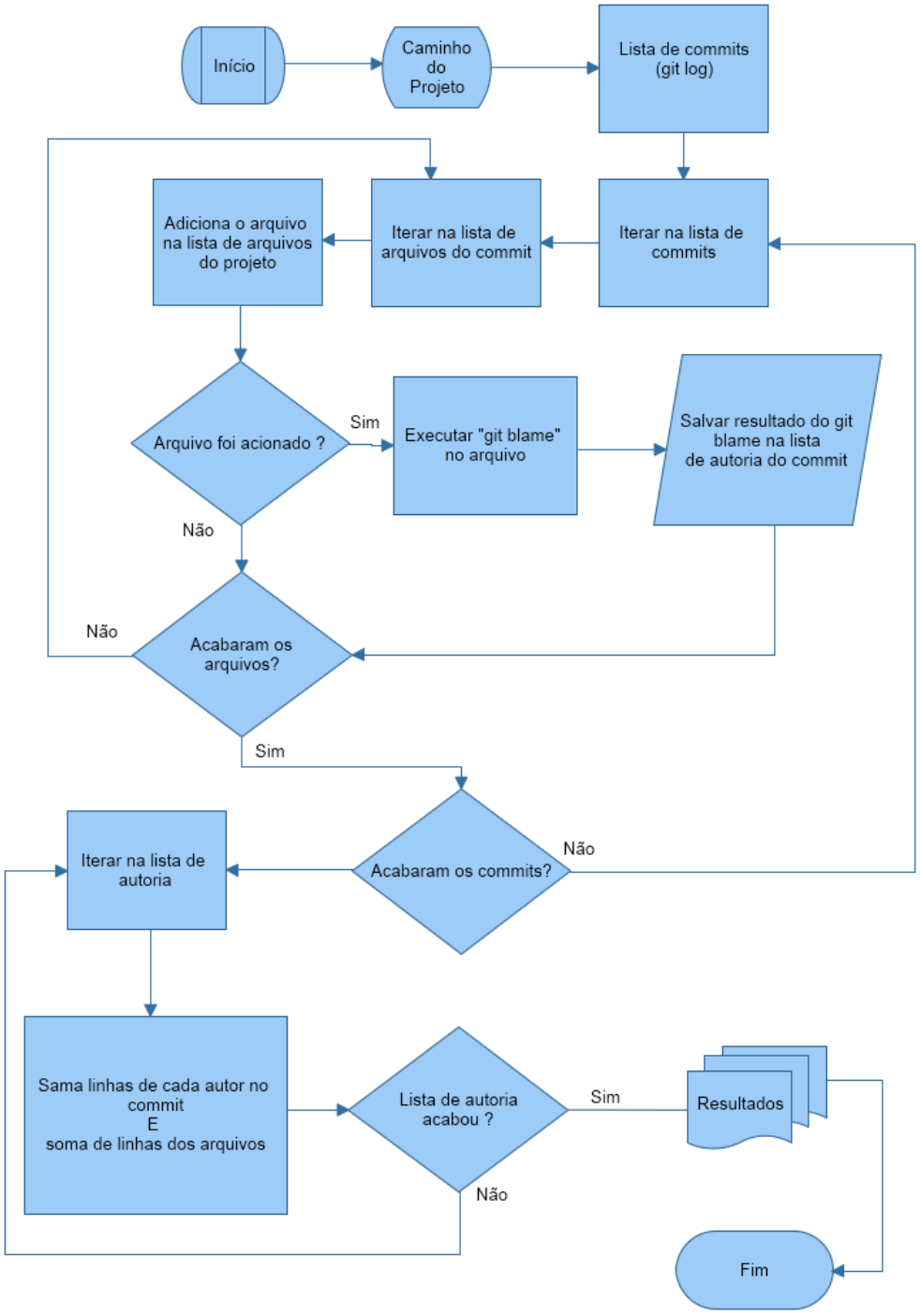

Figura 2 – Diagrama de algoritmo de medição

A descrição das etapas do algoritmo:

- **Caminho do projeto:** O caminho do projeto é o local onde a está localizado o projeto Git no computador, mais conhecido como *path*.
- **Lista de *commits* Git log:** O git log é um comando que trás o histórico de *commits* de um repositório. Esse histórico possui informações de autor, data, hora, mensagem, arquivos, linhas adicionadas e removidas.
- **Itera na lista de *commits*:** Logo após se obter a lista de de *commits*, é possível criar um laço de repetição para percorrer a lista do elemento *commit* mais velho, até os mais recentes.
- **Itera na lista de arquivos do *commit*:** Ao interar na lista de *commits*, se obtém um *commit* por vez. Dentro dos *commits* existem arquivos, que devem ser percorridos para as informações serem armazenadas.
- **Remove o arquivo da lista de arquivos do projeto:** Os arquivos presente no projeto, naquele *commit*, são todos os arquivos dos *commits* anteriores, excluído os arquivos do *commit* atual. Essa lista, é para cada *commit*.
- **Condicional: O arquivo foi adicionado ?** Caso o arquivo seja adicionado em um *commit* é necessário que se execute o git blame nas suas linhas. Caso contrário, se o arquivo for removido, não há nada o que se fazer pois a autoria não existe se o arquivo deixa de existir.
- **Git blame no arquivo:** Ao se executar o comando git blame no *commit* e arquivo específico é possível se obter a autoria de cada linha do arquivo, sendo atribuída a autoria da linha para a última pessoa que a modificou.
- **Lista de autoria das linhas do *commit*:** É uma lista que armazena todas as informações relacionadas a autoria de um *commit*. Salvando os nomes dos arquivos, nome do autor, quantidade de linhas atribuídas para cada autor, quantidade de linhas do arquivo.
- **Lista de autoria das linhas dos *commits*:** É uma lista que armazena todas as autorias de linhas de cada *commit*. Ou seja, é uma lista que contém a lista anterior.
- **Lista de autoria das linhas dos *commits*:** É uma lista que armazena todas as autorias de linhas de cada *commit*. Ou seja, é uma lista que contém a lista anterior.
- **Condicional: Acabaram os arquivos ?** Caso já tenha passado por todos os arquivos do *commit*, o laço irá para o próximo *commit* se caso ele existir.

- **Condicional: Acabaram os *commits* ?** Caso já tenha passado por todos os *commits*, o laço de repetição irá se encerrar.
- **Interar na lista de autoria:** Essa iteração ocorre sobre a lista de autoria de todos os *commits*, ou seja, é uma lista que contém a quantidade de autoria dos autores, quantidade de linhas dos arquivos e sua parcela de autoria dentro dos arquivos. A interação ocorre para o cálculo da parcela de autoria ser em relação ao projeto inteiro.
- **Soma linhas de cada autor no *commit* E soma de linhas dos arquivos:** As somas ocorrem para o cálculo da parcela de autoria de cada autor em relação ao projeto inteiro, gerando assim uma linha do tempo. Essa linha do tempo é uma lista que contem as parcelas dos autores ao longo do tempo.
- **Lista linha do tempo:** A linha do tempo é a lista com os resultado finais, todos os arquivos, todas quantidades de autoria, os tamanhos dos arquivos bem como as informações em relação ao projeto inteiro estarão dentro dessa lista. Sendo que cada informação terá uma data associada.
- **Condicional: Lista de autoria acabou ?** Condicional para Interar sobre a lista de autoria inteira. Caso tenha chegado ao fim, se leva a impressão dos resultados.
- **Resultados** Os resultados serão impressos em tela, no formato de tabela, contendo duas tabelas: uma para autoria dos arquivos e outra para a autoria do projeto como um todo.

4.3.2 Métricas

As métricas definidas para solucionar os problemas propostos por esse estudo são:

- Total de linhas do arquivo;
- Total de linhas do projeto;
- Autoria de linhas por arquivo;
- Autoria de linhas pelo projeto;
- Porcentagem de autoria do arquivo;
- Porcentagem de autoria do projeto.

A partir das métricas foram feitas definições operacionais de métrica, pois é importante garantir que as métricas sejam coletadas de uma forma consistente para que sejam analisadas e esses dados se mantenham consistentes(ROCHA; SANTOS; BARCELLOS, 2012).

Desta maneira para cada métrica foram definidas várias características que compõem a medida.

4.3.2.1 Total de linhas do arquivo

Definição operacional da medida:

- Nome: Total de linhas do arquivo;
- Definição: A quantidade de linhas total do arquivo é o LOC do arquivo;
- Mnemônico: LOFC (Lines of file code);
- Tipo da Medida: Medida Base;
- Entidade da medida: Arquivo;
- Propriedade Medida: Tamanho;
- Unidade de Medida: Linha;
- Tipo de Escala: Escala Absoluta;
- Valores da Escala: Números inteiros não-negativos;
- Intervalo Esperado dos Dados: [0, N];
- Procedimento de Medição: Contagem das linhas dentro de um do arquivo;
- Fórmula de Cálculo de Medida: N, onde N é quantidade de linhas do arquivo;
- Responsável pela Medição: Ferramenta que conta linhas do arquivo;
- Momento da medição: Quando é necessário saber o tamanho do arquivo;
- Periodicidade de Medição: A cada *release* ou a cada *sprint*;
- Procedimento de Análise: Representar em números e gráficos o número de linhas do arquivo, comparando com outras métricas de autoria;
- Momento da Análise de Medição: Quando é necessário comparar métricas do mesmo arquivo;
- Periodicidade da Análise: A cada *release* ou a cada *sprint*;
- Responsável pela Análise: Gerente de desenvolvimento.

4.3.2.2 Total de linhas do projeto

Definição operacional da métrica:

- Nome: Total de linhas do projeto;
- Definição:A quantidade de linhas total do projeto é somatório das linhas de todos os arquivos do projeto;
- Mnemônico: LOC (Lines of code);
- Tipo da Medida: Medida Derivada;
- Entidade da medida: Projeto;
- Propriedade Medida: Tamanho;
- Unidade de Medida: Linha;
- Tipo de Escala: Escala Absoluta;
- Valores da Escala: Números inteiros não-negativos;
- Intervalo Esperado dos Dados: [0, N];
- Procedimento de Medição: Realizar o somatório da métrica total de linhas do arquivo, de todos os aquivos do projeto;
- Fórmula de Cálculo de Medida:

$$Quantidade\ de\ linhas\ total\ do\ projeto = \sum_{LF=1}^{N} LF \tag{4.1}$$

  Onde LF é o total de linhas do arquivo e N é quantidade de arquivos do projeto;
- Responsável pela Medição: Ferramenta que conta linhas do arquivo;
- Momento da medição: Quando é necessário saber o tamanho do projeto;
- Periodicidade de Medição: A cada *release* ou a cada *sprint*;
- Procedimento de Análise: Representar em números e gráficos o número de linhas do projeto, comparando com outras métricas de tamanho;
- Momento da Análise de Medição: Quando é necessário comparar métricas do mesmo projeto;
- Periodicidade Da Análise: A cada *release* ou a cada *sprint*;
- Responsável pela Análise: Gerente de desenvolvimento.

4.3.2.3 Autoria de linhas por arquivo

Definição operacional da métrica:

- Nome: Autoria de linhas por arquivo;
- Definição: A quantidade de autoria de um arquivo, são os somatórios das quantidades de linhas de cada autor do arquivo em questão;
- Mnemônico: ALOFC (Authorship lines of file code);
- Tipo da Medida: Medida derivada;
- Entidade da medida: Arquivo;
- Propriedade Medida: Tamanho da autoria;
- Unidade de Medida: Linhas;
- Tipo de Escala: Escala Absoluta;
- Valores da Escala: Números inteiros não-negativos;
- Intervalo Esperado dos Dados: [0, N];
- Procedimento de Medição: Realizar o somatório da métrica autoria de linha do arquivo, de todos as linhas do arquivo, de todos autores;
- Fórmula de Cálculo de Medida:

$$\textit{Quantidade de autoria de um autor no arquivo} = \sum_{ALOFC=1}^{N} \tag{4.2}$$

  Onde ALOFC é a quantidade de autoria na linha, N é quantidade de linhas do arquivo;
- Responsável pela Medição: Ferramenta que conta autoria do arquivo;
- Momento da medição: Quando é necessário saber o autoria do arquivo;
- Periodicidade de Medição: A cada *release* ou a cada *sprint*;
- Procedimento de Análise: Representar em números e gráficos o número de linhas, que um determinado autor escreveu no arquivo;
- Momento da Análise de Medição: Quando é necessário saber o quanto um autor contribuiu no arquivo;
- Periodicidade Da Análise: A cada *release* ou a cada *sprint*;
- Responsável pela Análise: Gerente de desenvolvimento.

4.3.2.4 Autoria de linhas pelo projeto

Definição operacional da métrica:

- Nome: Autoria de linhas pelo projeto;
- Definição: A quantidade de linhas(ou frações) total do autor é determinado pelo somatório da quantidade de autoria de todos os arquivos do autor específico. Sendo total, se referindo ao projeto inteiro;
- Tipo da Medida: Medida derivada;
- Entidade da medida: Projeto;
- Propriedade Medida: Tamanho da autoria;
- Unidade de Medida: Linhas;
- Tipo de Escala: Escala Absoluta;
- Valores da Escala: Números inteiros não-negativos;
- Intervalo Esperado dos Dados: [0, N];
- Procedimento de Medição: Realizar o somatório da métrica autoria de linha do arquivo, de todos os aquivos do projeto, de todos autores;
- Fórmula de Cálculo de Medida:

$$Quantidade\ de\ linhas\ total\ do\ autor = \sum_{LAF=1}^{N} LAF \tag{4.3}$$

  Onde LAF é a quantidade de linhas do autor do arquivo, N é quantidade total de arquivos do projeto;
- Responsável pela Medição: Ferramenta que conta autoria do projeto;
- Momento da medição: Quando é necessário saber o autoria do projeto;
- Periodicidade de Medição: A cada *release* ou a cada *sprint*;
- Procedimento de Análise: Representar em números e gráficos o número de linhas, que um determinado autor escreveu no projeto;
- Momento da Análise de Medição: Quando é necessário saber o quanto um autor contribuiu no projeto;
- Periodicidade Da Análise: A cada *release* ou a cada *sprint*;
- Responsável pela Análise: Gerente de desenvolvimento.

4.3.2.5 Porcentagem de autoria do arquivo

Definição operacional da métrica:

- Nome: Porcentagem de autoria do arquivo;
- Definição: A Porcentagem de autoria do arquivo, é o resultado da divisão da quantidade de linhas de autoria do desenvolvedor no arquivo pela quantidade total de código do projeto, porcentagem que representa quanto do arquivo pertence ao desenvolvedor;
- Tipo da Medida: Medida derivada;
- Entidade da medida: Arquivo;
- Propriedade Medida: Tamanho da autoria;
- Unidade de Medida: Linhas;
- Tipo de Escala: Razão;
- Valores da Escala: Números reais não-negativos, com precisão de duas casas decimais;
- Intervalo Esperado dos Dados: [0.00, 1];
- Procedimento de Medição: Realizar divisão da quantidade de linhas de autoria do arquivo pelo total de linhas do arquivo;
- Fórmula de Cálculo de Medida:

$$Taxa\ de\ autoria\ do\ arquivo\ = \left(\frac{(LOC\ por\ autor\ do\ arquivo)}{(LOC\ do\ arquivo)} \right) \quad (4.4)$$

- Responsável pela Medição: Ferramenta que define a porcentagem autoria do arquivo;
- Momento da medição: Quando é necessário saber o porcentagem autoria do arquivo;
- Periodicidade de Medição: A cada *release* ou a cada *sprint*;
- Procedimento de Análise: Representar em números e gráficos o número de linhas, ou frações, que um determinado autor escreveu no arquivo;
- Momento da Análise de Medição: Quando é necessário saber o quanto um autor contribuiu no arquivo;
- Periodicidade Da Análise: A cada *release* ou a cada *sprint*;
- Responsável pela Análise: Gerente de desenvolvimento.

4.3.2.6 Porcentagem de autoria do projeto

Definição operacional da métrica:

- Nome: Porcentagem de autoria do projeto;
- Definição: A porcentagem de autoria do projeto, é o resultado da divisão da quantidade de linhas de autoria do desenvolvedor no projeto pela quantidade total de código do projeto, porcentagem que representa quanto do projeto pertence ao desenvolvedor;
- Tipo da Medida: Medida derivada;
- Entidade da medida: Projeto;
- Propriedade Medida: Tamanho da autoria;
- Unidade de Medida: Linhas;
- Tipo de Escala: Razão;
- Valores da Escala: Números reais não-negativos, com precisão de duas casas decimais;
- Intervalo Esperado dos Dados: [0.00, 1];
- Procedimento de Medição: Realizar da quantidade de linhas de autoria do arquivo pelo total de linhas do arquivo;
- Fórmula de Cálculo de Medida:

$$Taxa\ de\ autoria\ do\ projeto\ = \left(\frac{\sum_{LAF=1}^{N} LAF}{\sum_{LF=1}^{N} LF} \right) \tag{4.5}$$

- Responsável pela Medição: Ferramenta que define a porcentagem autoria do projeto;
- Momento da medição: Quando é necessário saber o porcentagem autoria do projeto;
- Periodicidade de Medição: A cada *release* ou a cada *sprint*;
- Procedimento de Análise: Representar em números e gráficos o número de linhas, ou frações, que um determinado autor escreveu no projeto;
- Momento da Análise de Medição: Quando é necessário saber o quanto um autor contribuiu no projeto;
- Periodicidade Da Análise: A cada *release* ou a cada *sprint*;
- Responsável pela Análise: Gerente de desenvolvimento.

A taxa de autoria de código é determinada em um ponto específico ao longo do tempo. Ou seja, durante o projeto a quantidade linhas de um autor pode mudar, consequentemente sua taxa de autoria em relação aos arquivos e ao projeto também podem mudar. Quando se obtém a taxa de autoria a cada *commit*, temos uma linha do tempo da variação da taxa de autoria ao longo do tempo.

4.3.2.7 Regras de contagem de linhas

Um arquivo renomeado não deve perder sua autoria, caso algum desenvolvedor crie um arquivo e outro desenvolvedor faça a renomeação do arquivo, o arquivo não deverá perder suas características de autoria presentes antes da renomeação, ou seja, em hipótese alguma o arquivo renomeado poderá ser considerado deletado e logo em seguida criado outro com o nome diferente.

Por conta da complexidade de contagem por caracteres de linhas, a contagem da autoria das linhas sempre considera o autor como o último desenvolvedor que modificou a linha.

4.4 Aplicação da ferramenta

Esta seção tratará de um cenário específico para os testes, nos quais as modificações serão descritas, versionadas e detalhadas pela ferramenta de avaliação de autoria, seguindo a abordagem determinada por esse trabalho. A ferramenta é uma CLI (*Command Line Interface*), na qual possui interface de linha de commando, será executada após cada *commit*, os *commits* também serão detalhados nessa seção.

O caso de aplicação da ferramenta começa com um arquivo com 3 linhas, seguido de um commit, o commit inicial.

- Nome do arquivo: file1.code

Conteúdo do arquivo:

```
New Line
New Line
New Line
```

- Número do commit: 1;
- Autor do commit: Author1;
- Mensagem do commit: Author1 create file1 and add 3 new lines;
- Linhas adicionadas: 3 linhas no arquivo file1.code;

- Observação: Ocorreram apenas adição de linhas sem deleções de linhas ou arquivos.

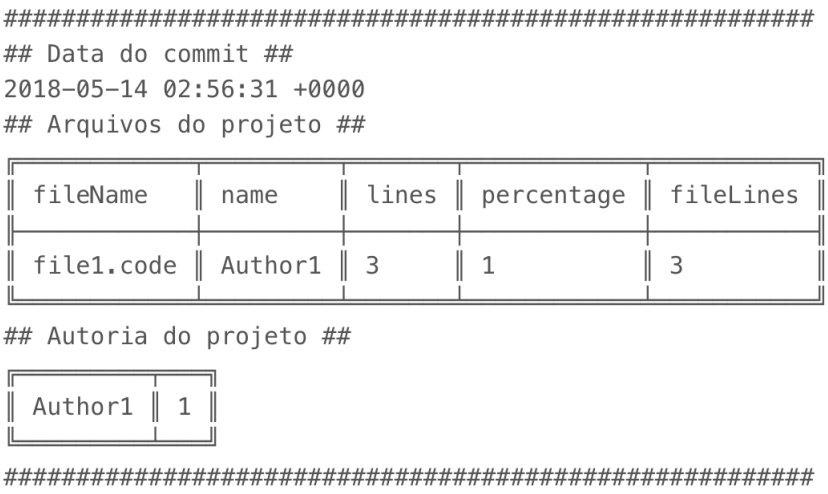

Figura 3 – Resultado *commit* 1

No próximo commit do arquivo, são adicionadas mais 3 linhas ao final do arquivo.

- Nome do arquivo: file1.code

Conteúdo do arquivo:

```
New Line
New Line
New Line
New Line
New Line
New Line
```

- Número do commit: 2;
- Autor do commit: Author1;
- Mensagem do commit: Author 1 modified file1 append 3 new lines;
- Linhas adicionadas: 3 linhas ao final do arquivo file1.code;
- Observação: Ocorreram apenas adição de linhas sem deleções de linhas ou arquivos.

```
...
#########################################################
## Data do commit ##
2018-05-14 02:59:08 +0000
## Arquivos do projeto ##
╔═══════════╦═══════╦═══════╦════════════╦═══════════╗
║ fileName  ║ name  ║ lines ║ percentage ║ fileLines ║
╠═══════════╬═══════╬═══════╬════════════╬═══════════╣
║ file1.code ║ Author1 ║ 6  ║ 1          ║ 6         ║
╚═══════════╩═══════╩═══════╩════════════╩═══════════╝
## Autoria do projeto ##
╔═══════╦═══╗
║ Author1 ║ 1 ║
╚═══════╩═══╝
#########################################################
```

Figura 4 – Resultado *commit* 2

No próximo estado do arquivo, um segundo autor modifica a terceira e quarta linha do arquivo.

- Nome do arquivo: file1.code

Conteúdo do arquivo:

```
New Line
New Line
Modified Line by author 2
Modified Line by author 2
New Line
New Line
```

- Número do commit: 3;
- Autor do commit: Author2;
- Mensagem do commit: Author2 modified file1;
- Linhas adicionadas: 2 linhas no meio do arquivo file1.code;
- Observação: As linhas adicionadas na verdade foram modificadas, as linhas anteriores foram apagadas e adicionando autoria do Author2.

```
...
##########################################################
## Data do commit ##
2018-05-14 03:05:55 +0000
## Arquivos do projeto ##
╔═══════════╦═══════╦═══════╦═══════════════════╦═══════════╗
║ fileName  ║ name  ║ lines ║ percentage        ║ fileLines ║
╠═══════════╬═══════╬═══════╬═══════════════════╬═══════════╣
║ file1.code ║ Author1 ║ 4     ║ 0.6666666666666666 ║ 6         ║
╠═══════════╬═══════╬═══════╬═══════════════════╬═══════════╣
║ file1.code ║ Author2 ║ 2     ║ 0.3333333333333333 ║ 6         ║
╚═══════════╩═══════╩═══════╩═══════════════════╩═══════════╝
## Autoria do projeto ##
╔═══════════╦═══════════════════╗
║ Author1 ║ 0.6666666666666666 ║
╠═══════════╬═══════════════════╣
║ Author2 ║ 0.3333333333333333 ║
╚═══════════╩═══════════════════╝
##########################################################
```

Figura 5 – Resultado *commit* 3

- Nome do arquivo: file.code

Conteúdo do arquivo:

```
New Line
New Line
Modified Line by author 2
Modified Line by author 2
New Line
New Line
```

- Número do commit: 4;
- Autor do commit: Author2;
- Mensagem do commit: Author 2 rename file1.code to file.code;
- Linhas adicionadas: 6 linhas de um arquivo novo(file.code);
- Observação: As linhas adicionadas na verdade foram copiadas do arquivo anterior, o arquivo anterior foi deletado, deste modo a autoria permanece inalterada, seguindo as regras aqui descritas.

```
...
##########################################################
## Arquivos do projeto ##
```

fileName	name	lines	percentage	fileLines
file.code	Author1	4	0.6666666666666666	6
file.code	Author2	2	0.3333333333333333	6

```
## Autoria do projeto ##
```

Author1	0.6666666666666666
Author2	0.3333333333333333

```
##########################################################
```

Figura 6 – Resultado *commit* 4

- Nome do arquivo: file.code

Conteúdo do arquivo:

```
New Line
New Line
Modified Line by author 2
New Line
New Line
```

- Número do commit: 5;
- Autor do commit: Author1;
- Mensagem do commit: Author1 deleted line number 4 of author2;
- Linhas adicionadas: 0 linhas;
- Observação: O Author1 retirou autoria do Author2, quando deletou a linha escrita pelo Author2.

```
...
##########################################################
## Data do commit ##
2018-05-14 03:21:04 +0000
## Arquivos do projeto ##
```

fileName	name	lines	percentage	fileLines
file.code	Author1	3	0.75	4
file.code	Author2	1	0.25	4

```
## Autoria do projeto ##
```

Author1	0.75
Author2	0.25

```
##########################################################
```

Figura 7 – Resultado *commit* 5

- Nome do arquivo: file.code

Conteúdo do arquivo:

```
New Line
New Line
Modified Line by author 2
New Line
New Line
```

- Nome do arquivo: file2.code

Conteúdo do arquivo:

```
New line add by author 2
```

- Número do commit: 6;
- Autor do commit: Author2;
- Mensagem do commit: Author2 Add file2.code;
- Linhas adicionadas: Adição de uma linha no novo arquivo file2.code;
- Observação: O arquivo file.code se manteve inalterado.

```
...
############################################################
## Data do commit ##
2018-11-02 18:21:21 -0300
## Arquivos do projeto ##
```

fileName	name	lines	percentage	fileLines
file.code	Author1	3	0.75	4
file.code	Author2	1	0.25	4
file2.code	Author2	1	1	1

```
## Autoria do projeto ##
```

Author1	0.6
Author2	0.4

```
############################################################
```

Figura 8 – Resultado *commit* 6

5 Considerações Finais

Este trabalho foi de principal importância diante do entendimento da métrica de autoria. Sendo o entendimento da métrica, qual é a origem dela, quais problemas ela soluciona, as formas que são feitas para elas serem coletadas atualmente e suas limitações. A definição de uma abordagem, foi possível a partir desses estudo. Uma abordagem deve conter métodos e técnicas, a abordagem é relacionada ao contexto, uma método é um conjunto de técnicas e as técnicas são pequenas tarefas específicas e simples que possuem um resultado imediato.

Esse trabalho define uma abordagem com os métodos e técnicas para se avaliar a quantidade de autoria dos desenvolvedores de um projeto. Uma ferramenta foi desenvolvida para que validar se a abordagem foi bem construída e retornava o que foi esperado dela.

Os resultados obtidos por meio da ferramenta condizem com a lógica para se avaliar a quantidade de autoria, com os métodos e técnicas descritos nesse trabalho. A ferramenta trouxe resultados coerentes com cada ação de um suposto desenvolvedor durante os testes descritos no capítulo anterior. Os resultados da ferramenta podem ser usados para outros contextos, além dos descritos na introdução deste trabalho. Como por exemplo a avaliação dos professores em disciplinas de programação, desde que exista o controle de versionamento.

Durante a construção deste trabalho, foram avaliados duas ferramentas nas quais faziam um trabalho de avaliar a quantidade de autoria dentro de um código. As ferramentas são "git author"e "git-authors-cli". A ferramenta "git author", é um módulo do git na qual está disponibilizada em um repositório GitHub, essa ferramenta não está empacotada e não é fácil de se executar, por conta de ser necessário realizar a compilação de todo o software Git na máquina. Apesar disso a ferramenta possui um modo de contar diferente do habitual. A ferramenta quantifica a autoria por caracteres, porém apenas um arquivo por vez. Por conta da contagem ser mais complexa o uso de recursos computacionais também aumenta. Já a ferramenta "git-author-cli"é empacotada por meio do gerenciador de pacotes NPM, é mais fácil de ser instalada, porém realiza o git blame em apenas um arquivo por vez e não no projeto inteiro. Em relação a ferramenta "git author"ela é mais simples por contar a autoria apenas pela última pessoa na qual modificou a linha.

A principio o trabalho seria desenvolvido para quantificar a autoria, de forma semelhante a ferramenta "git author", porém dependendo do tamanho do projeto, o uso de recursos computacionais poderia aumentar. Além da complexidade ser muito mais alta em relação a ferramenta desenvolvida, contando a autoria por meio do último desenvolvedor

que editou as linhas.

5.1 Limitações

A abordagem proposta pode ter limitações, como por exemplo, não se sabe qual é o impacto de refatorações no sistema em relação as avaliações de autoria, pois em casos de refatoração a mesma funcionalidade do software é mantida mas a estrutura do seu código é modificada. Ou seja, a refatoração é uma mudança de código na qual não possui evolução do software quanto aos requisitos ou até mesmo a resolução de *BUGs*. A avaliação de autoria é não mede qualidade no software aferido, mas ela pode melhorar a precisão de outras métricas de qualidade. A avaliação de autoria não leva em consideração complexidade do código desenvolvido, nem do tempo gasto para o desenvolvimento;

Existe uma limitação específica, se uma pessoa retirar todo o código do projeto e realizar um *commit*, logo após ela recolocar todo o projeto novamente. Essa pessoa irá deter 100% de porcentagem de autoria do código, retirando o código inteiro e colocando ele novamente, será como se o projeto começasse daquele ponto. Pois no *commit* anterior não existiam linhas, ou seja, o histórico foi apagado.

As autorias das linhas são quantificadas por meio da última pessoa que editou, ou seja, se caso uma pessoa escrever 30 caracteres, entretanto outra pessoa editar aquela linhas, mesmo que seja apenas um caractere, a pessoa anterior terá sua autoria removida. A resolução desse problema não é simples e pode usar os recursos computacionais de forma demasiada.

A co-autoria, normalmente usada em duplas de desenvolvedores, na técnica de pareamento, não é quantificada na ferramenta, deste modo a ferramenta considera o primeiro autor como autor daquela linha e não dos dois ou mais autores.

A ferramenta usa o software de versionamento Git, se caso o projeto de software for versionado em outro sistema de versionamento não é possível utilizar a ferramenta nesse projeto.

As limitações desse trabalho podem ser estudadas e analisadas em trabalhos futuros.

Referências

ANTHONY, E. M. Approach, method, and technique. *ELT Journal*, XVII, n. 2, p. 63–67, 1963. Disponível em: <http://dx.doi.org/10.1093/elt/XVII.2.63>. Citado na página 19.

APACHE. *Apache Subversion*. 2018. Disponível em: <https://subversion.apache.org/>. Citado 2 vezes nas páginas 16 e 24.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The goal question metric approach. In: . [S.l.: s.n.], 1999. Citado 2 vezes nas páginas 22 e 23.

BIRD, C. et al. Don't touch my code!: Examining the effects of ownership on software quality. In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. New York, NY, USA: ACM, 2011. (ESEC/FSE '11), p. 4–14. ISBN 978-1-4503-0443-6. Disponível em: <http://doi.acm.org/10.1145/2025113.2025119>. Citado na página 21.

COMMUNITY, G. *Git*. 2018. Disponível em: <https://git-scm.com/>. Citado 2 vezes nas páginas 16 e 24.

CôTé, V. et al. Software metrics: An overview of recent results. *Journal of Systems and Software*, v. 8, n. 2, p. 121–131, mar. 1988. ISSN 01641212. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/0164121288900052>. Citado na página 22.

GIRBA, T. et al. How developers drive software evolution. In: *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*. [S.l.: s.n.], 2005. p. 113–122. ISSN 1550-4077. Citado 2 vezes nas páginas 20 e 21.

LECCISO, R.; MAINETTI, S.; MORASCA, S. Software metrics: A critical evaluation and an application to pascal. *Microprocessing and Microprogramming*, v. 18, n. 1, 1986. ISSN 0165-6074. Disponível em: <http://www.sciencedirect.com/science/article/pii/0165607486900980>. Citado na página 22.

LERTWITTAYATRAI, N. et al. Extracting Insights from the Topology of the JavaScript Package Ecosystem. In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. Nanjing: IEEE, 2017. p. 298–307. ISBN 978-1-5386-3681-7. Disponível em: <http://ieeexplore.ieee.org/document/8305952/>. Citado na página 27.

MAJUMDAR, R. et al. Source code management using version control system. In: *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. [S.l.: s.n.], 2017. p. 278–281. Citado 2 vezes nas páginas 16 e 25.

MENG, X. et al. Mining software repositories for accurate authorship. In: *2013 IEEE International Conference on Software Maintenance*. [S.l.: s.n.], 2013. p. 250–259. ISSN 1063-6773. Citado 3 vezes nas páginas 15, 21 e 25.

MICROSOFT. *Team Foundation Version Control*. 2018. Disponível em: <https://www.visualstudio.com/pt-br/team-services/tfvc/>. Citado 2 vezes nas páginas 16 e 24.

NGUYEN, V. et al. A sloc counting standard. In: . [S.l.: s.n.], 2007. Citado na página 21.

OYA, M.; KASHIWAKURA, A. JavaScript language extension for non-professional programmers: Sharable own variables. In: *2016 IEEE 5th Global Conference on Consumer Electronics*. Kyoto, Japan: IEEE, 2016. p. 1–2. ISBN 978-1-5090-2333-2. Disponível em: <http://ieeexplore.ieee.org/document/7800390/>. Citado na página 26.

RAHMAN, F.; DEVANBU, P. Ownership, experience and defects: a fine-grained study of authorship. In: . ACM Press, 2011. ISBN 978-1-4503-0445-0. Disponível em: <http://portal.acm.org/citation.cfm?doid=1985793.1985860>. Citado na página 20.

ROCHA, A. R.; SANTOS, G.; BARCELLOS, M. *Medição de Software e Controle Estatístico de Processos*. [S.l.: s.n.], 2012. Citado na página 37.

ROCHKIND, M. J. The source code control system. *IEEE Transactions on Software Engineering*, SE-1, n. 4, p. 364–370, dez. 1975. ISSN 0098-5589. Citado 2 vezes nas páginas 16 e 23.

ROSENBLUM, N.; ZHU, X.; MILLER, B. P. Who Wrote This Code? Identifying the Authors of Program Binaries. In: ATLURI, V.; DIAZ, C. (Ed.). Berlin, Heidelberg: [s.n.], 2011. Citado na página 20.

TILKOV, S.; VINOSKI, S. Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing*, v. 14, n. 6, p. 80–83, nov. 2010. ISSN 1089-7801. Disponível em: <http://ieeexplore.ieee.org/document/5617064/>. Citado na página 26.

WITTERN, E.; SUTER, P.; RAJAGOPALAN, S. A look at the dynamics of the javascript package ecosystem. In: *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. [S.l.: s.n.], 2016. p. 351–361. Citado na página 27.

YIN, R. K. *Case study research: design and methods*. Rev. ed. Newbury Park, [Calif.]: Sage Publications, 1989. (Applied social research methods series, v. 5). ISBN 978-0-8039-3470-2 978-0-8039-3471-9. Citado na página 29.

YIN, Z. et al. How do fixes become bugs? In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. New York, NY, USA: ACM, 2011. (ESEC/FSE '11), p. 26–36. ISBN 978-1-4503-0443-6. Disponível em: <http://doi.acm.org/10.1145/2025113.2025121>. Citado na página 20.