Associação Propagadora Esdeva
Centro Universitário Academia – UniAcademia
Curso de Sistemas de Informação
Trabalho de Conclusão de Curso

GitLogAnalysis: Uma ferramenta para acompanhar a evolução de Software via Git

Gabriel Rigolon Silva[1]
Centro Universitário Academia, Juiz de Fora, MG

Tassio Ferenzini Martins Sirqueira[2]
Centro Universitário Academia, Juiz de Fora, MG

Linha de Pesquisa: Engenharia de Software

RESUMO

[Contexto] A Evolução e a Manutenção de Software são áreas da Engenharia de Software que estudam e avaliam os processos de melhorias que um *software* pode sofrer durante seu ciclo de vida. Esse tema tem se tornado cada vez mais recorrente dentro das empresas de desenvolvimento, visto que manutenções geram alto custo financeiro. **[Objetivo]** Pensando em facilitar o acompanhamento das versões de software, foi desenvolvido o GitLogAnalysis, uma ferramenta que analisa e compara as métricas de software que tenha usado o *Git* como o versionador de código fonte. As informações geradas podem trazer uma melhor visão do projeto com base nos dados de suas releases. **[Metodologia]** A ferramenta extrai os dados presentes no arquivo de log do *Git* (*git log*) e exibe na tela um comparativo entre duas versões. Para realizar a análise, é necessário escolher os períodos a serem analisados. **[Resultado]** Através da utilização da ferramenta, percebeu-se que há indícios de que a visão de cada uma das releases tornou-se mais clara, e também da comparação de duas releases, visto que a ferramenta apresenta informações quantitativas acerca de *commits*, autores e linhas adicionadas e removidas. Dessa forma, a ferramenta auxilia de forma visual na análise da evolução projeto com base nessas métricas que cada *release* possui. **[Considerações]** Os resultados obtidos sugerem que a ferramenta GitLogAnalysis pode apoiar os usuários nas atividades de análise do ciclo de vida de um software. Entretanto, avaliações adicionais são necessárias.

Palavras-chave: Evolução de Software. Ciclo de Vida. Git

ABSTRACT

[Context] Software Evolution and Maintenance are areas of Software Engineering that study and evaluate the improvement processes that a software can undergo during its life cycle. This theme has become increasingly recurrent within development companies, since maintenance generates a high financial cost for them. **[Objective]** With the aim

[1] Discente do Curso de Sistemas de Informação do Centro Universitário Academia – UniAcademia. E-mail: gabrielrigolon@gmail.com.
[2] Docente do Curso de Sistemas de Informação do Centro Universitário Academia. Orientador.

of facilitating the monitoring of software versions, GitLogAnalysis was developed, a tool that analyzes and compares the metrics of software that has used Git as the source code versioner. The information generated can bring a better view of the project based on the data from its releases. **[Methodology]** The tool extracts the data present in the Git log file (git log) and displays a comparison between two versions on the screen. To perform the analysis, it is necessary to choose the periods to be analyzed. **[Result]** Through the use of the tool, it was noticed that there are indications that the vision of each one of the releases became clearer, and also the comparison of two releases, since the tool presents quantitative information about commits, authors and lines added and removed. In this way, the tool visually assists in the analysis of project evolution based on these metrics that each release has. **[Considerations]** The results obtained suggest that the GitLogAnalysis tool can support users in software lifecycle analysis activities. However, additional evaluations are necessary.

Keywords: Software Evolution. Life cycle. Git

1 INTRODUÇÃO

No final da década de sessenta, o termo “manutenção de software” foi definido como qualquer atividade de desenvolvimento realizada sobre o software após a sua entrega e implantação inicial. Esta ideia é empregada no modelo de desenvolvimento em cascata onde, seguindo sua ordem de acontecimentos, a manutenção encerra aquele ciclo de desenvolvimento (Sokol, 2012).

À medida em que o tempo foi passando, e a Engenharia de Software foi avançando com os estudos, percebeu-se que este modelo provocava uma defasagem nos softwares desenvolvidos à sua base. Este modelo não compreendia mudanças que poderiam ocorrer mesmo depois da fase de levantamento de requisitos. Sendo assim, os softwares já eram entregues carecendo de ajustes e manutenção.

Foi pensando nestes problemas, que [Lehman, 1997] propôs as *leis da evolução de software*, buscando entender e descrever melhor o processo de Evolução de Software:

• Mudança contínua: Um programa da classe E muda continuamente, caso contrário, se torna menos útil gradativamente.

• Complexidade crescente: Com o processo de mudança contínua, a complexidade do software cresce, a menos que sejam dirigidos esforços para reduzir ou manter essa complexidade.

• Auto regulação: A evolução de software é um processo auto regulado pelo feedback das mudanças feitas no sistema e as reações a essas mudanças no domínio em que ele é utilizado.

• Conservação de estabilidade organizacional: A taxa com que as modificações são feitas sobre o software ao longo de seu ciclo de vida é estatisticamente invariante.

• Conservação de familiaridade: A evolução de um programa é limitada pela familiaridade que seus desenvolvedores tem com o sistema.

• Crescimento contínuo: O conteúdo funcional de um programa da classe E precisa ser continuamente aumentado para manter a satisfação do usuário por toda a vida.

• Qualidade em declínio: A qualidade dos programas da classe E parecerá estar em declínio, a menos que sejam rigorosamente mantidos e adaptados às mudanças do ambiente operacional.

• Sistema de *feedback*: Os processos de evolução dos programas da classe E constituem sistemas de *feedback* multi-nível, *multi-loop* e multi-agente e devem ser tratados como tal para alcançar uma melhoria significativa.

Pensando na segurança e na melhoria do processo de desenvolvimento de software, uma medida adotada foi a utilização de sistemas de controle de versão. Estes sistemas permitem que alterações feitas nos arquivos dos softwares sejam revertidas para um estado anterior, possibilitando que seja feita a comparação entre as versões, permitindo assim, criar-se um histórico de mudanças (Sokol, 2012).

Atualmente, existem três tipos de Sistemas de controle de versão: Sistemas Locais de Controle de Versão; Sistemas Centralizados de Controle de Versão; Sistemas Distribuídos de Controle de Versão.

O objetivo deste trabalho é apresentar uma ferramenta que pode auxiliar na análise da evolução de um software de forma facilitada e intuitiva, com os dados extraídos do repositório onde o projeto está armazenado, independente da linguagem em que foi desenvolvido.

Este artigo está organizado da seguinte forma. A seção 2 apresenta os trabalhos relacionados a esta proposta. A seção 3 detalha a ferramenta desenvolvida. Uma diretriz de utilização é apresentada na seção 4. A seção 5 aborda as considerações sobre o trabalho, as ameaças à validade e os trabalhos futuros.

2 REFERENCIAL TEÓRICO

Segundo Chacon (2014), controle de versão é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo para que você possa lembrar versões específicas mais tarde. Neste trabalho, foi criada uma ferramenta para auxiliar a interpretação e análise de dados deixados pelas mudanças feitas nos arquivos presentes nos repositórios.

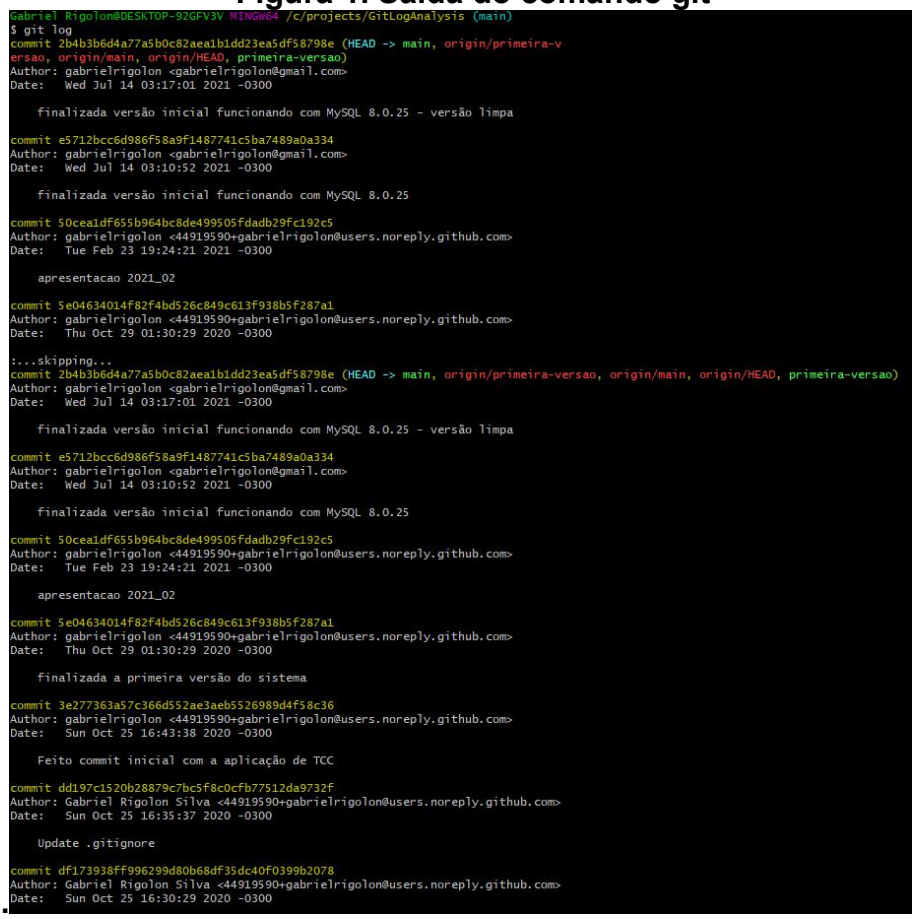
No mercado, há diversos sistemas de controle de versão disponíveis para uso, tanto pessoal, quanto empresarial. Neste trabalho, o sistema de controle de versão

usado foi o *Git*[3]. A principal diferença entre o *Git* e qualquer outro *Version Control System* (VCS), como *Subversion*[4] e similares, é a maneira como o *Git* trata seus dados. A forma como o *Git* registra as mudanças feitas nos arquivos é que o difere dos demais VCSs. o *Git* trata seus dados mais como um conjunto de imagens de um sistema de arquivos em miniatura. Toda vez que você fizer um *commit*, ou salvar o estado de seu projeto no *Git*, ele basicamente tira uma foto de todos os seus arquivos e armazena uma referência para esse conjunto de arquivos (CHACON, 2014).

Para que seja possível analisarmos os dados históricos de determinado projeto que usa o *Git* como VCS, a melhor opção é rodar o comando *git log*. Este comando, quando executado sem argumentos, retornará uma lista dos *commits* feitos em ordem cronológica inversa, ou seja, começará pelo *commit* mais recente.

Conforme Figura 1, um exemplo de saída comando *git log*, executado dentro do repositório pelo *bash* do *git*:

Figura 1. Saída do comando git log.



Fonte: Elaboração Própria.

[3] Git. Disponível em < https://git-scm.com/>. Acesso em: 08 de março de 2022
[4] SVN. Disponível em < https://tortoisesvn.net/>. Acesso em: 28 de junho de 2022

Os trabalhos desenvolvidos na literatura técnica de Manutenção e Evolução de Software apresentam ferramentas e tecnologias muito poderosas para apoiar no acompanhamento do processo de desenvolvimento e de sustentação. Grande parte destas ferramentas são baseadas na análise do código fonte, como as que serão apresentadas a seguir.

Em (Anslow et al., 2013) é apresentada a *SourceVis*, uma ferramenta que fornece visões sobre a evolução de software com base na análise de código fonte. É uma ferramenta desenhada para visualização horizontal, como menus que proporcionam ver a versão do sistema, opções, gráficos. Entre as informações que podem ser visualizadas estão métricas de código fonte como número de acoplamentos, linhas de código e número de classes, dentre outras (Tavares et al., 2015).

A solução proposta neste trabalho deve ser capaz de fornecer visualizações sobre a evolução de software com base na análise dos dados obtidos através do histórico do *Git* no repositório do software. Diferentemente de *SourceVis*, que fornece a visão no nível de código fonte, métricas e visualizações sobre a relação entre entidades de código como classes e métodos, o trabalho desenvolvido apresentará uma visão sobre dados históricos do repositório.

Outra ferramenta já desenvolvida é *ArchView* (Pinzger, 2005). É uma ferramenta de visualização para a análise da evolução de um software, que extrai informações do sistema de controle de versão (CVS) e do sistema de rastreamento de *bugs* (*Bugzilla*[5]) e calcula diversas métricas para cada versão do software. Processados os dados, o *ArchView* mostra ao usuário a visualização de diversas métricas de módulos ou arquivos, de diferentes versões do projeto, exibidas em diagramas de kiviat (Sokol, 2012). Este trabalho tem foco em apresentar o acoplamento dos diferentes módulos do software submetido à sua análise com base no histórico de alterações do repositório.

Ainda outra ferramenta, o *GiveMe Metrics* (Tavares et al., 2015) é um *framework* conceitual cujo objetivo é guiar usuários na tarefa de extrair dados históricos sobre a evolução do *software,* e este *framework* tem como objetivo reportar dados estes dados que foram obtidos. Dados estes que podem ser extraídos de três categorias de repositórios: Repositórios de Código Fonte; Repositórios de Defeitos de *Software*; Repositórios de Dados sobre o Processo de Desenvolvimento de *Software*.

[5] Bugzilla. Disponível em < https://git-scm.com/>. Acesso em: 28 de junho de 2022

Tabela 1. Comparativo das ferramentas.

Ferramenta	Disponível Online	Restrição de Linguagem de Programação	Análise de repositórios
GitLog Analysis	Sim	Não	Sim
ArchView	Não	Não	Sim
SourceVis	Não	Sim	Não
GiveMe Metrics	Não	Sim	Sim

Fonte: Elaboração Própria

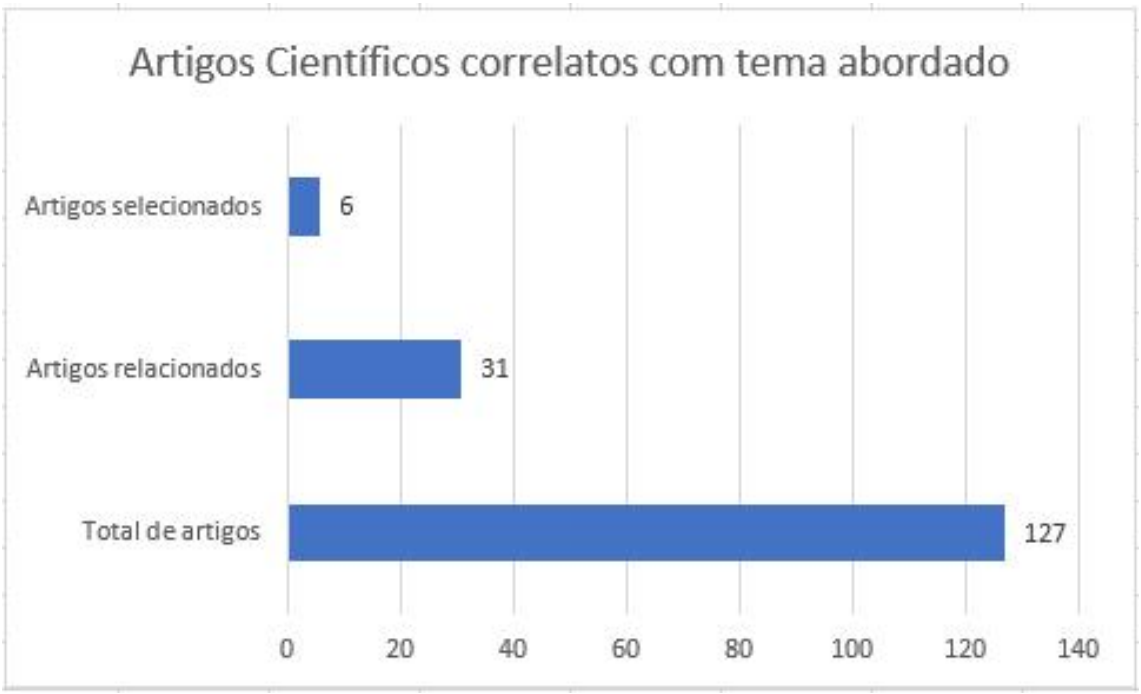
De acordo com a Tabela 1, pode-se concluir que, apesar de utilizarem uma abordagem diferente, todas podem auxiliar na análise do histórico de um determinado projeto. O foco da ferramenta GitLogAnalysis é a extração de dados através do histórico do git log, independendo da linguagem de programação que foi usada na construção do projeto.

Algo que não foi observado nas demais ferramentas comparadas, foi a disponibilidade *online*. A Ferramenta GitLogAnalysis foi desenvolvida pensando também em preencher este espaço, visto que esta disponibilidade entrega mais facilidade de utilização para o usuário, e a manutenção dos dados extraídos não ficam restritos à uma máquina local.

Para a realização da pesquisa, foi utilizada a *String* de busca: *(git AND log AND métricas) AND "manutenção de software" AND visualização*

A busca foi feita em 14 de março de 2022, e limitou-se à base do Google Scholar <https://scholar.google.com.br/> por se tratar de uma base cujo o acesso é livre.

Figura 2. Gráfico representado a busca dos trabalhos relacionados.



Fonte: Elaboração Própria

Os resultados obtidos mostram que, ao todo, 127 artigos foram retornados, sendo que desses, apenas 31 estavam relacionados indiretamente ao tema do trabalho, e desses, apenas 6 estavam diretamente relacionados ao trabalho. O critério de seleção dos artigos foi baseado em dois principais pontos:

- Estar relacionado com a análise e o acompanhamento sobre a evolução e o ciclo de vida de um software;
- Apresentar uma solução que não fosse baseada somente em análise de código fonte, mas também considerar dados extraídos do repositório de software

Os 31 artigos indiretamente relacionados, em sua maioria, abordam soluções e estudos voltados para a implementação de código, arquitetura de código, arquitetura de projetos, métricas de softwares específicos, evolução de softwares baseada somente na análise de código fonte.

Entretanto, os artigos que foram selecionados atendiam aos dois critérios descritos, por isso contribuíram para a construção desta ferramenta e deste trabalho.

3 METODOLOGIA

A ferramenta GitLogAnalysis foi construída para plataforma Web, contando com uma Interface e uma *Application Programming Interface* (API). A Interface foi desenvolvido em *Angular*[6], na versão 8.23. A escolha deste framework para a criação do projeto de Interface foi baseada nos seguintes pontos:

1. Familiaridade com a arquitetura de projetos Web, baseando-se em arquivos HTML com *TypeScript*

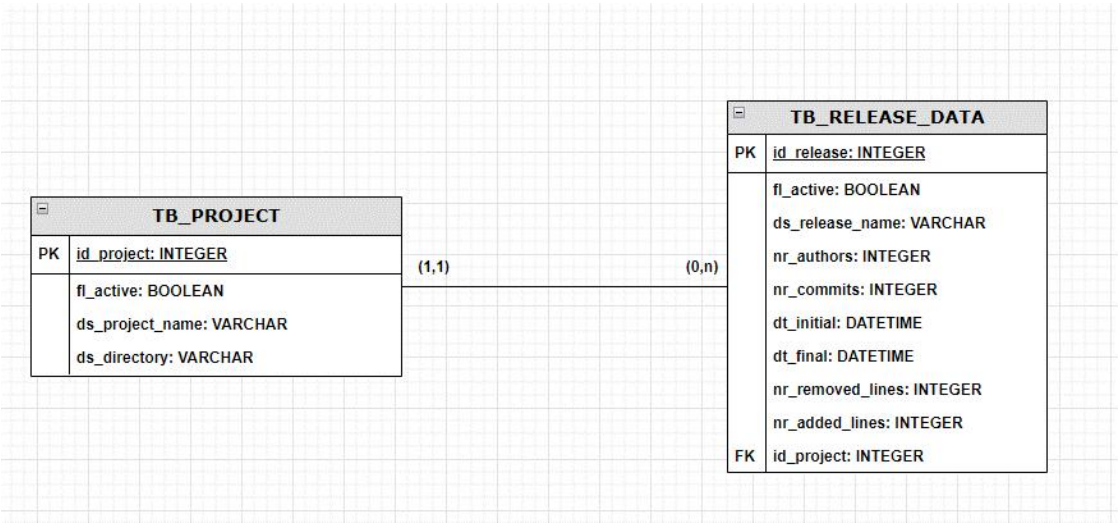
2. Permite criar aplicações baseadas em *components*;

3. Projeto com *single-page application* (SPA), com *lazy-load* implementado, que entrega mais performance à ferramenta, pois carrega apenas os módulos que a página acessada utiliza;

A API foi construída em *.NET Core 3.1*[7], utilizando o C#, na versão 7. A utilização de uma API permite que uma ou mais Interfaces possam guardar, buscar, deletar ou atualizar os registros de seu Banco de dados, sem que a Interface conheça a implementação destas operações.

Para armazenar os dados extraídos pela ferramenta, utilizou-se o *MySQL*[8], que é um sistema de gerenciamento de banco de dados (SGBD) *open-source*. o *MySQL* está presente no mercado desde 1995. A escolha desse SGBD foi baseada na disponibilidade do mesmo em diferentes sistemas operacionais, compatibilidade com a linguagem de programação utilizada na API e pelo baixo custo de manutenção.

Figura 3. Diagrama de Tabelas Relacionais da aplicação.



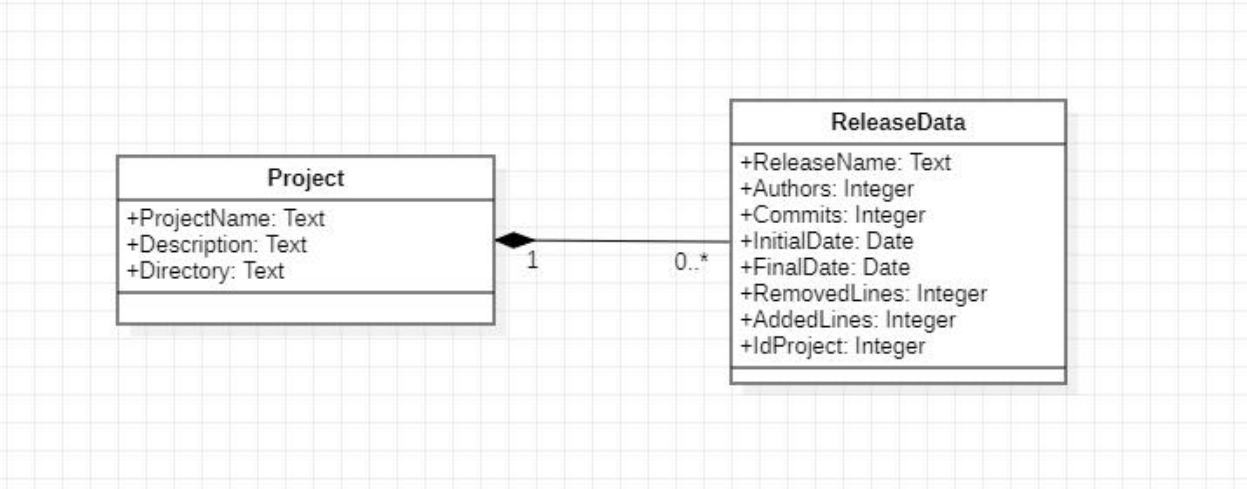
Fonte: Elaboração Própria.

[6] Angular. Disponível em <https://angular.io/>. Acesso em: 08 de março de 2022
[7] .NET Core 3.1. Disponível em <https://dotnet.microsoft.com>. Acesso em: 08 de março de 2022
[8] MySQL. Disponível em <https://www.mysql.com/>. Acesso em: 08 de março de 2022

O sistema possui duas tabelas, onde podemos realizar a manutenção dos registros através das operações de inserção e consulta ao banco de dados pela própria ferramenta GitLogAnalysis. Cada tabela é representada pela sua respectiva classe modelo, conforme Figura 3

A Classe Project possui os atributos *ProjectName*, *Description* e *Directory*,que permitem a identificar pasta que o projeto foi salvo, para então o GitLogAnalysis fazer a busca e o reconhecimento dos dados do repositório.

Figura 4. Diagrama de classes da aplicação.



Fonte: Elaboração Própria.

Com o intuito de facilitar o entendimento da aplicação, foi disponibilizada uma documentação da API gerada pelo Swagger. Ela pode ser consultada na página inicial da API. Esta documentação fornece informações sobre os *payloads* que os *endpoints* esperam em cada requisição, o retorno dessas requisições, e exemplos de chamadas a estes *endpoints*.

4 RESULTADOS E DISCUSSÃO

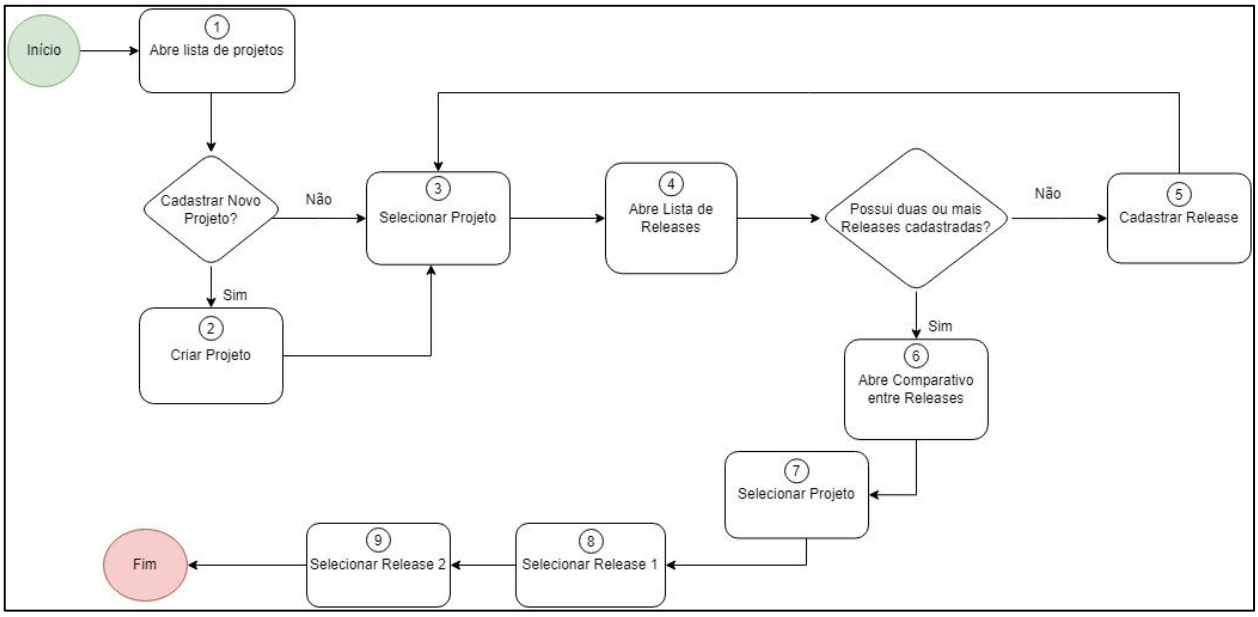
A Ferramenta GitLogAnalysis foi construída para auxiliar na extração e visualização de dados de um repositório do GitHub, visando facilitar a análise acerca da evolução do projeto através de métricas observadas no processo de desenvolvimento.

O GitLogAnalysis extrai os seguintes dados dos repositórios: Autores, Linhas Adicionadas, Linhas Removidas e a quantidade de *commits* feitos no período informado no cadastro da release que será analisada.

Para verificar a utilização e os resultados que a ferramenta GitLogAnalysis pode trazer, foi feito um estudo de caso utilizando o repositório público do Sistema Operacional Linux, presente no GitHub.

Foi criado um fluxograma para mostrar a execução da ferramenta, que pode ser observado na Figura 5.

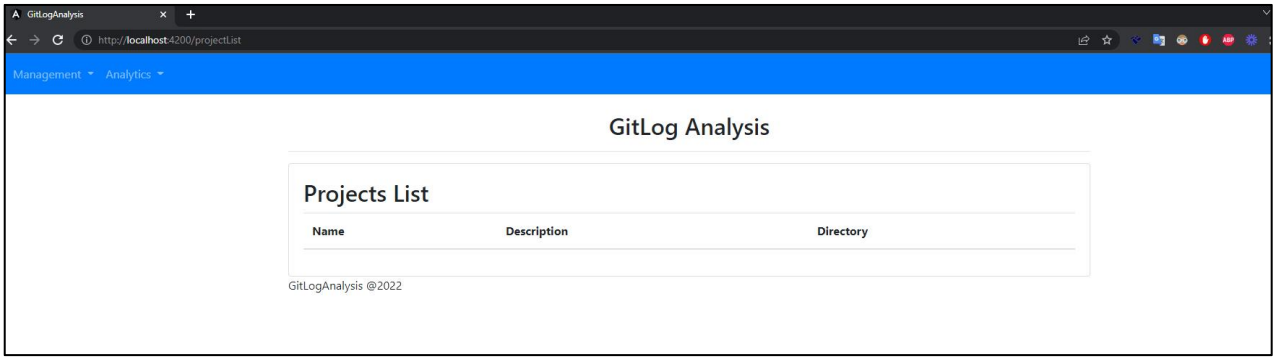
Figura 5. Fluxograma de uso da ferramenta.



Fonte: Elaboração Própria.

Ao iniciar a ferramenta GitLogAnalysis, a primeira tela que é exibida é a que mostra todos projetos cadastrados, conforme Figura 6. Como foi observado, não há projetos cadastrados até o momento, o que nos impede de prosseguir com a análise.

Figura 6. Tela com a lista de projetos cadastrados.

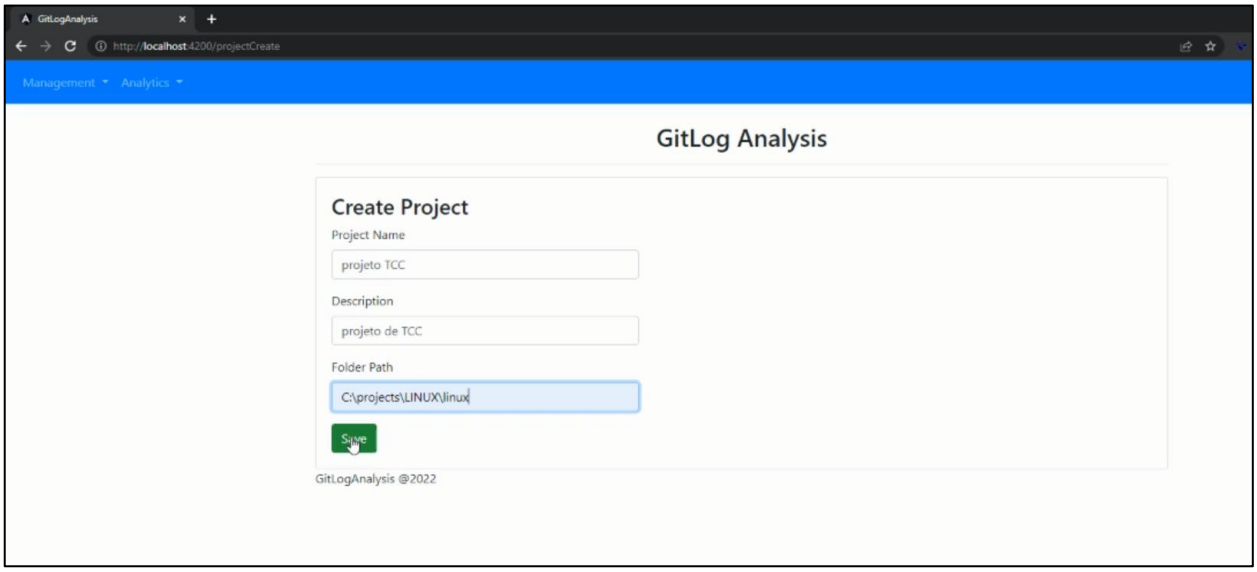


Fonte: Elaboração Própria.

Na Figura 7, vemos o formulário para cadastrar um projeto. Ao finalizar este

preenchimento, um Projeto é gravado no banco de dados com as seguintes informações: Nome, descrição e diretório. O próximo passo é cadastrar as releases, no mínimo duas, para que seja possível realizar a análise.

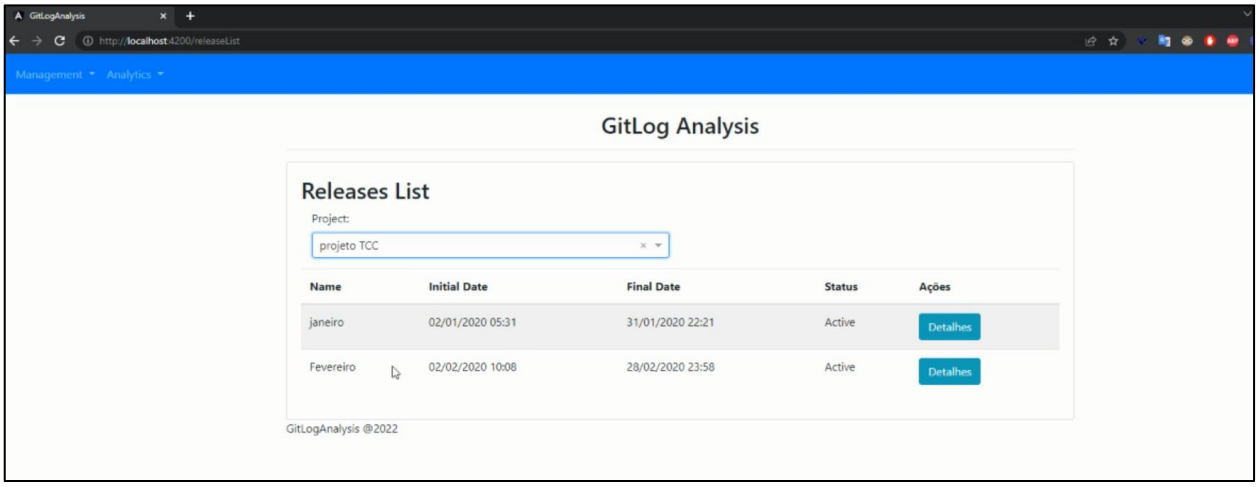
Figura 7. Tela de Cadastro de Projeto.



Fonte: Elaboração Própria.

Na Figura 8, vê-se a tela de Releases Cadastradas, separadas por seus respectivos projetos. Como fora mostrado nos diagramas presentes na seção 3 deste artigo, cada Release deve obrigatoriamente ser referenciada a um único Projeto.

Figura 8. Tela de Releases Cadastradas.

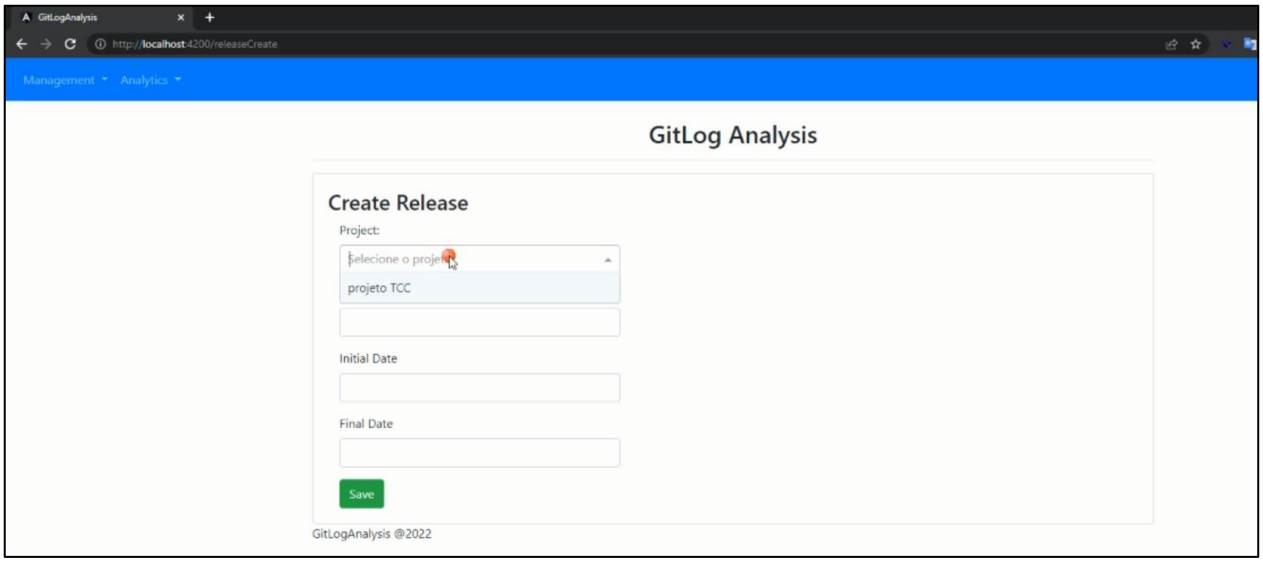


Fonte: Elaboração Própria.

Na Figura 9, observa-se a tela onde é feito o cadastro da Release. É neste momento que delimitamos o período que será usado para análise. Para seguir, deve-se

selecionar o Projeto ao qual a Release pertence, o nome que será dado à Release, a data inicial do período a ser analisado e a data final do período a ser analisado.

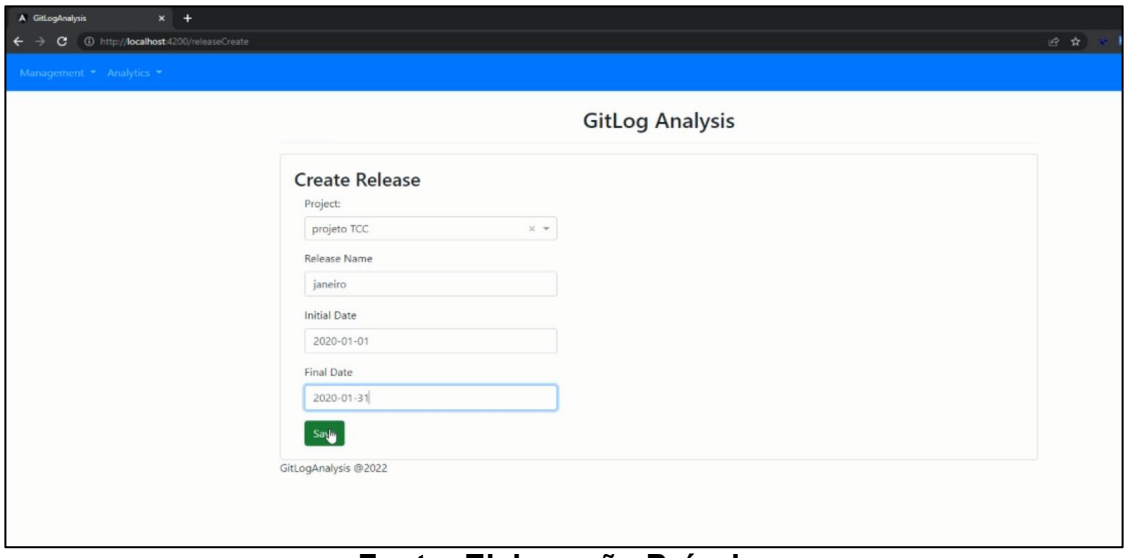
Figura 9. Tela de Releases Cadastradas.



Fonte: Elaboração Própria.

Conforme mostra a Figura 10, este é o exemplo de formulário preenchido completamente. Para este estudo de caso, foram cadastradas duas releases, seguindo o JSON mostrado na Figura 11. Os campos que não são apresentados no formulário, são campos que são preenchidos pela própria com base na extração dos dados pela própria ferramenta. São estes os campos que serão gravados com dados da análise: *Commits*, *Authors*, *Removed Lines* e *Added Lines*.

Figura 10. Tela de Cadastro de Releases.



Fonte: Elaboração Própria.

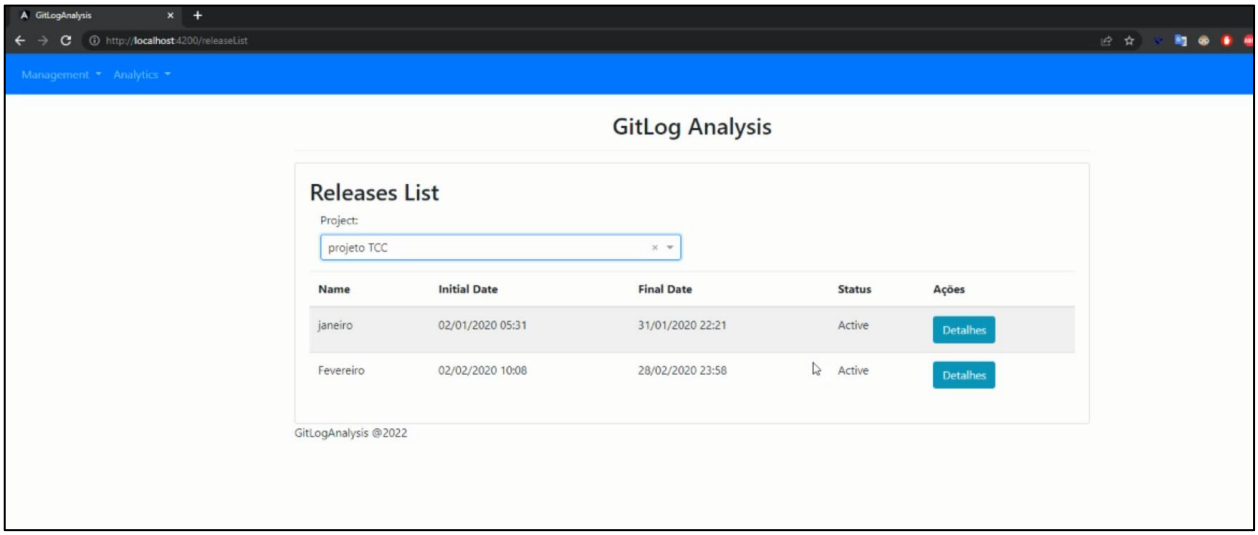
Figura 11. Tela de Releases Cadastradas.


```
{
  "releaseName": "janeiro",
  "authors": 1248,
  "commits": 7477,
  "initialDate": "2020-01-02T05:31:40",
  "finalDate": "2020-01-31T22:21:21",
  "removedLines": 16012,
  "addedLines": 19855,
  "idProject": 1,
  "project": null,
  "id": 1,
  "active": true
},
{
  "releaseName": "Fevereiro",
  "authors": 1008,
  "commits": 5424,
  "initialDate": "2020-02-02T10:08:24",
  "finalDate": "2020-02-28T23:58:56",
  "removedLines": 11655,
  "addedLines": 14490,
  "idProject": 1,
  "project": null,
  "id": 2,
  "active": true
},
```


Fonte: Elaboração Própria.

Conforme mostra a Figura 12, ao selecionar o projeto, é possível ver as duas releases que foram cadastradas anteriormente. As Releases são listadas na ordem em que foram cadastradas.

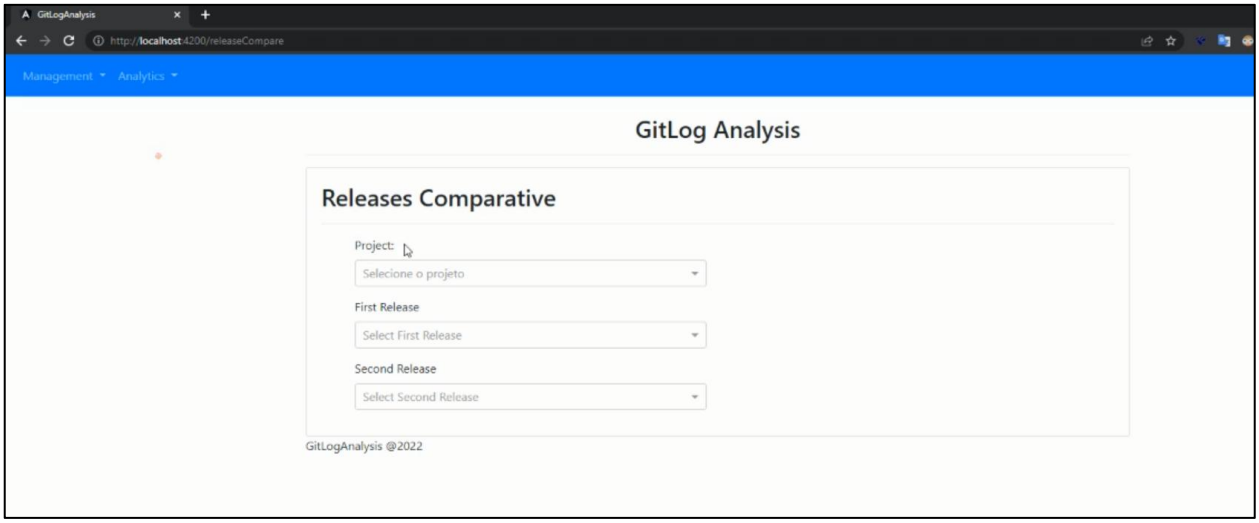
Figura 12. Tela de Releases Cadastradas.



Fonte: Elaboração Própria.

Na Figura 13, é exibida a tela onde é feito o comparativo das Releases. É necessário selecionar o Projeto a ser analisado, em primeiro lugar. Logo em seguida, seleciona-se a primeira Release e, por fim, a segunda release a ser analisada.

Figura 13. Tela de Comparação das Releases.

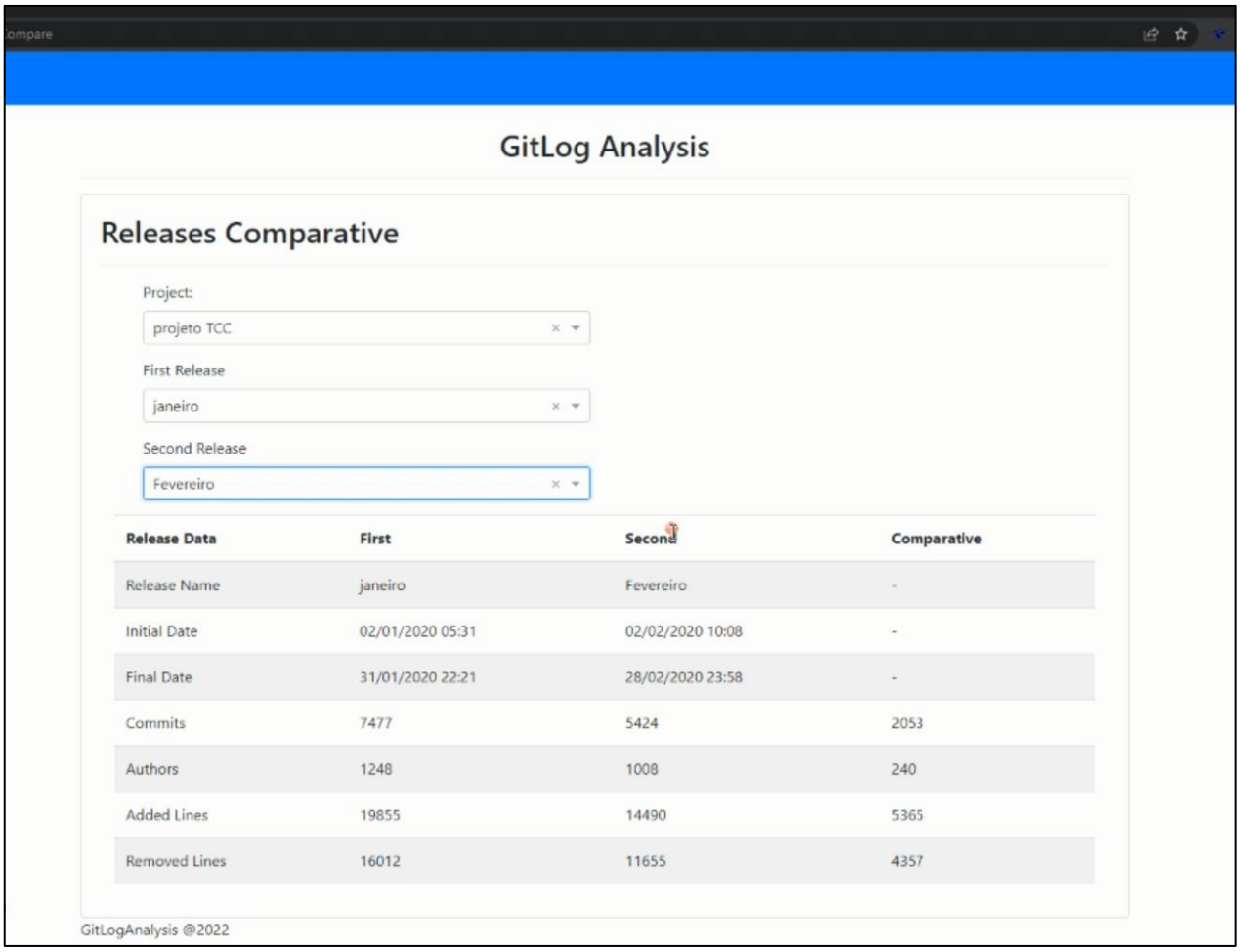


Fonte: Elaboração Própria.

Após selecionar todas as informações, a ferramenta executa um cálculo com base nas Releases que foram gravadas no banco de dados, e é retornado o resultado final da análise entre as duas Releases selecionadas na coluna com o título "*Comparative*".

Na Figura 10, é exibida a tela com o resultado final de todo o processo descrito. Após selecionar o Projeto e as duas Releases a serem analisadas, a ferramenta GitLogAnalysis exibe uma tabela com as informações extraídas desta comparação. No caso abaixo (Figura 14), observou-se que a Release nomeada de "janeiro", teve 2053 *commits* a mais que a Release nomeada "Fevereiro". Ainda aqui, foi possível ver também que a Release "janeiro" registrou 240 autores a mais que a Release "Fevereiro", 5365 linhas adicionadas a mais que a Release "Fevereiro", e também teve 4357 linhas removidas a mais que a mesma.

Figura 14. Tela de Comparação das Releases, com o comparativo entre Releases.



Fonte: Elaboração Própria.

Uma demonstração de uso da ferramenta está disponível em https://youtu.be/s4iShpd28jM. O código fonte da ferramenta encontra-se disponível para download em https://github.com/ces-jf/GitLogAnalysis. Na próxima seção abordaremos algumas considerações sobre este trabalho, as ameaças à validade e os trabalhos futuros.

5 CONSIDERAÇÕES FINAIS

Todo software está sujeito a passar por mudanças, refatorações e melhorias. Em alguns casos, o software pode necessitar até mesmo de ser refeito. Para que essas alterações sejam cada vez mais assertivas, é necessário que se tenha visão do passado desse software. É de extrema importância que o software seja acompanhado por todos aqueles que o desenvolvem ou o gerenciam. Por esta razão, uma ferramenta que permite ao seu usuário consultar um determinado período de desenvolvimento, dá grande vantagem nessa busca pela qualidade de software e evolução do seu processo de desenvolvimento.

No decorrer do processo de desenvolvimento algumas dificuldades foram encontradas, como:

- Integração entre o *bash* do *Git* e a API: O *bash* do G*it* é uma aplicação construída para atender à plataforma *Linux*, nativamente. Houve certa dificuldade para executar os comandos dentro do repositório, a princípio. Após estudos, optou-se por utilizar o *PowerShell* para rodar os comandos.

- Manipulação da massa de dados extraídos: A primeira versão da aplicação não estava gravando os registros corretamente no banco de dados, devido ao formato da devolução dos dados feita pelo *PowerShell*. Só foi possível realizar a gravação no Banco de Dados após tratar esses dados mudando seu formato.

Embora estas ferramentas possam ser de grande ajuda, é necessário que haja a compreensão de que elas possuem limitações e necessitam ser melhoradas e evoluídas, para que alcancem resultados cada vez mais relevantes.

Algumas limitações da ferramenta são:

- Integração apenas com o sistema *Git*;
- Uma única exibição de relatório, sem opção de manipulação da exibição
- Restrita a atuação em repositórios locais
- Criada para atender às necessidades de um projeto maior, a ferramenta, que compõe o projeto *Mineração de Repositórios de Software: Investigando a Qualidade de Software em Projetos de Código Aberto*, limitou-se a atende-lo.

Como objetivos futuros para a ferramenta GitLogAnalysis, pretende-se melhorar as seguintes funcionalidades:

- Criação de relatórios manipuláveis por parâmetros.
- Exportação dos resultados em arquivos nos formatos de PDF, CSV e JSON.
- Implementar visualização de gráficos.
- Promover integração com plataformas online de repositórios.
- Expandir área de atuação permitindo analisar repositórios que usam outros Sistemas de Controle de Versão.

Por fim, com esta ferramenta os usuários podem monitorar, principalmente, o andamento da evolução do projeto, visando ampliar a visão de determinados cenários

em períodos específicos, tais como, número de pessoas que participaram do desenvolvimento, número de *commits*, linhas adicionadas e removidas neste período.

REFERÊNCIAS

ANGULAR. **Angular, Página inicial**. Disponível em: <https://angular.io>. Acesso em: 08 de março 2022.

Anslow, C., Marshall, S., Noble, J., and Biddle, R. (2013). **"Sourcevis: Collaborative software visualization for co-located environments"**. In: Software Visualization (VISSOFT), 2013 First IEEE Working Conference On, p. 1-10.

CHACON, Scott; STRAUB, Ben; **Pro Git**. California: APRESS, 2014.

GITHUB. **Github, Página inicial**. Disponível em: <https://github.com>. Acesso em: 08 de março 2022.

Lehman, Meir M., et al. **"Metrics and laws of software evolution-the nineties view."** Proceedings Fourth International Software Metrics Symposium. IEEE, 1997.

MYSQL. **MySQL: About**, 2022. Sobre. Disponível em: < https://www.mysql.com/about/>. Acesso em: 09 de abril 2022.

PINZGER, M. (2005). **ArchView - Analyzing Evolutionary Aspects of Complex Software Systems**. PhD thesis.

SOKOL, Francisco. MetricMiner: uma ferramenta web de apoio à mineração de repositórios de software. **São Paulo: Universidade de São Paulo**, 2012.

Souza, B. M., Junqueira, A. L. C., Testoni, G. R., Terra, A., & Sirqueira, T. M. F. (2020). **Mineração de Repositório de Software: Investigando a qualidade do software em projetos de código aberto**. ANALECTA-Centro Universitário Academia, 5(5).

SWAGGER. **Swagger, Página inicial**. Disponível em: < https://swagger.io/>. Acesso em: 22 de junho 2022.

Tavares, J., David, J., Araújo, M., Braga, R., Campos, F., & Carneiro, G. (2015, May). **GiveMe Views: uma ferramenta de suporte a evolução de software baseada na análise de dados históricos**. In *Anais do XI Simpósio Brasileiro de Sistemas de Informação* (pp. 55-62). SBC.

TAVARES, Jacimar Fernandes et al. **Uma Infraestrutura baseada em Múltiplas Visões Interativas para Apoiar Evolução de Software**. **iSys-Brazilian Journal of Information Systems**, v. 8, n. 1, p. 65-101, 2015.