Associação Propagadora Esdeva
Centro Universitário Academia – UniAcademia
Curso de Engenharia de Software
Trabalho de Iniciação Científica – Artigo

GIT Viewer:
Uma plataforma para análise de dados do GitHub

Cláudio Oliveira[1], Rogério Tostes[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

Atualmente, existe um gama enorme de repositórios de código fonte que podem ser acessados e estudados abertamente, principalmente os de código aberto. O grande ponto é, que para explicar fenômenos dentro da engenharia de software precisamos de muitos dados históricos, e os projetos de código aberto são uma grande oportunidade de explorar esse universo e auxiliar a entendermos melhor o processo de manutenção, evolução e decaimento de um software, com base no histórico de outros softwares. Com a análise da manutenção de software, baseado nos dados históricos contidos em repositórios de código fonte, é possível detectar e predizer evoluções e/ou decaimento em sistemas de software, através de técnicas de mineração de dados. Nesse trabalho foi desenvolvida uma plataforma de análise de repositórios de código fonte, com o intuito de ajudar os gerentes de projetos durante o gerenciamento do ciclo de vida dos softwares mantidos. Os objetivos da plataforma são identificar efeitos no software que podem levar ao seu decaimento, advindos de falhas de codificação, projeto ou arquitetura. Para demostrar o funcionamento da plataforma, será apresentado um caso de uso, analisando um projeto de código aberto disponível no GitHub. Considerando o repositório de código fonte como sendo uma base histórica, através da mineração de dados é possível extrair informações úteis, que auxiliem no processo de manutenção de software, tornando-o menos custoso, pois as informações contidas podem ajudar a identificar erros ou padrões que levam ao retrabalho, ou ao decaimento de software e que por vezes são informações descartadas pelas empresas.

Palavras-chave: Mineração de repositório de código fonte. GitHub. Contribuição de desenvolvedores de software.

[1] Discente do Curso de Engenharia de Software do Centro Universitário Academia – UniAcademia.
[2] Docente do Curso de Engenharia de Software do Centro Universitário Academia – UniAcademia. Orientador.

1 INTRODUÇÃO

A manutenção de software tem sido uma área de grande interesse para os pesquisadores e os profissionais da computação, pois é inevitável a ocorrência de mudanças ao longo do tempo para assegurar a utilidade e a qualidade do software. Apesar de estudos como os de Lehman (1996), Basili, Briand e Melo (1996) e Belady e Lehman (1976), iniciados ainda na década de 70 terem grande importância e impacto para a área de engenharia de software, pois definiram modelos, métricas e postulados sobre o processo de manutenção e evolução de software, as organizações ainda têm dificuldades em implementar e gerir as manutenções, pois não deixam de estar envolvidos grandes investimentos financeiros.

Na verdade grande parte do orçamento de software nas grandes empresas, é atribuída à manutenção e evolução de software, em detrimento do desenvolvimento de novos sistemas, conforme Silva (2013). Esse gasto está ligado as mudanças que podem surgir da identificação de defeito no software; do levantamento de novos requisitos por conta da necessidade do software em representar tarefas do mundo real, que está em constante mudança, adequando sua utilização ao novo ambiente; da necessidade de melhorias de desempenho ou da sua associação a outros sistemas de software.

Dentro do processo de manutenção de software, os repositórios de código fonte são considerados depósitos de informações históricas sobre as atividades de desenvolvimento e de manutenção de um software, contendo dados passíveis de análise, conforme Colaço Jr (2010): i) bases de código estático; ii) histórico de versões do software; iii) rastros de execução de programas; iv) relatórios de erros; v) listas de discussão e; vi) logs de implantação de sistemas.

Considerando o repositório de código fonte como sendo uma base histórica, através da mineração de dados é possível extrair informações úteis, que auxiliem no processo de manutenção de software, tornando-o menos custoso, pois as informações ali contidas podem ajudar a identificar erros ou padrões que levam ao retrabalho, ou ao decaimento de software e que por vezes são informações descartadas pelas empresas. Temos hoje uma grande quantidade de dados atrelados aos projetos de software, e a cada dia esses dados vêm aumentado, assim ter o conhecimento do histórico do projeto é muito importante para a gestão do software.

A ideia da mineração de dados não é nova, pois a mineração de dados é o processo de encontrar anomalias, padrões e correlações em grandes conjuntos de dados para prever resultados. Com uma variedade de técnicas, podemos usar essas informações para aumentar a renda com o software, cortar custos, melhorar o relacionamento com os clientes, reduzindo riscos. O campo Mineração de Repositórios de Software (MSR) tornou-se crítico para apoiar a manutenção, melhorar a qualidade do processo de software e validar várias ideias ou técnicas de pesquisa. De modo geral, pretende-se

utilizar as informações contidas em repositórios de código fonte, para compreender os aspectos que afetam a qualidade da manutenção, o custo associado a está atividade e seu reflexo na comunidade de desenvolvimento.

Esse trabalho é continuação do trabalho desenvolvido em Souza *et al.* (2020). Além dessa introdução, na seção 2 apresentaremos o referencial teórico do trabalho, já a seção 3 apresenta os trabalhos relacionados. A seção 4 explora as características do GitHub que podem ser analisadas pela plataforma desenvolvida e detalhes técnicos da mesma. A seção 5 demostramos a utilização da ferramenta por meio de um caso de uso, com as análises geradas e os resultados que podem ser colhidos. Por fim, a seção 6 apresenta as limitações do trabalho, os próximos passos e as considerações finais.

2 REFERENCIAL TEÓRICO

No final da década de 1960, o termo “manutenção de software” era definido como qualquer atividade de desenvolvimento realizada no software após a entrada inicial e implantação do software. Isto é, a manutenção do sistema é o estágio final de desenvolvimento, durante o qual apenas pequenos ajustes e correções de erros são feitos.

Com o tempo, as pessoas perceberam que o processo não era muito eficaz, principalmente porque os requisitos feitos apenas na fase inicial do projeto mudavam como frequência ao longo do projeto. A fim de entender a natureza dessas mudanças no processo de desenvolvimento, Lehman (1980) conduziu alguns estudos empíricos sobre o desenvolvimento de sistemas da IBM.

Lehman (1980) definiu três tipos de programas. A classe S contém programas que podem especificar formalmente suas funções e ter soluções precisas. Questões famosas, como caixeiro-viajante, n-rainha, tráfego máximo etc., são exemplos que podem ser usados como especificações de programa do tipo S. Esses programas são estáticos (letra S vem de *Static*), porque uma vez determinada a solução para o problema, ela atenderá plenamente às suas necessidades.

A categoria P contém programas projetados para resolver problemas práticos, que também podem ser especificados formalmente, mas não podem ser resolvidos com precisão absoluta (letra P vem de *real world Problem solution*). Exemplos dessa categoria incluem programas de previsão do tempo, jogadores de xadrez, programadores de voos e rotas de trem. Devido à complexidade, os resultados obtidos por tais procedimentos não podem resolver completamente o problema. Portanto, o

programa tipo P também deve definir claramente seus requisitos e alterá-los apenas com o objetivo de melhorar seu desempenho ou a qualidade dos resultados.

Os programas da Classe E são aqueles programas que não podem ser totalmente definidos pela especificação. Esses programas são inseridos no mundo real para simplificar as tarefas em seu domínio, alterando assim a forma como as atividades são realizadas. Exemplos de aplicações da Classe E incluem: sistemas operacionais, sistemas de controle aéreo, mercados financeiros etc. Programas nesta categoria modificam seu domínio depois que o aplicativo é implementado inicialmente, a demanda aumentará naturalmente com o tempo para atender aos novos requisitos das atividades alteradas. Esses procedimentos continuarão a mudar após a implementação inicial, portanto, estão em processo de desenvolvimento contínuo (letra E vem de *Evolution*).

Portanto, os sistemas do tipo E são mais sujeitos a alterações do que os programas do tipo S e P. Lehman (1996) conduziu seu trabalho para a análise evolutiva de programas do tipo E, tentando descrever um processo evolutivo, atualmente conhecidas como as leis da evolução de software:

- Mudança continua: Softwares devem ser continuamente adaptados senão, com o tempo deixaram de atender as necessidades para a qual foram desenvolvidos ou se tornará obsoleto.

- Complexidade crescente: Conforme o software sofre alterações, sua estrutura original passa a ser descaracterizada e sua complexidade aumenta, de modo que também aumente gradativamente os custos de sua manutenção até o momento que a manutenção passe a se tornar inviável.

- Auto regulação: Durante o ciclo de vida os resultados serão quase constantes, de modo que à substituição de recursos ou pessoas tenham efeitos imperceptíveis na evolução do software.

- Conservação de familiaridade: A evolução de um programa é limitada pela familiaridade que seus desenvolvedores têm com o sistema. Isto é, a medida que o software evolui, todos os que estão a ele associados, devem manter o domínio de seu conteúdo e comportamento para conseguir uma evolução satisfatória.

- Crescimento contínuo: Adições de novas funcionalidades ou melhorias em funções pré-existentes ocasionadas pelo domínio operacional em que o software está inserido deverão ser realizadas para manter a contínua satisfação dos usuários, ocasionando no aumento do conteúdo funcional do software.

- Qualidade declinante: A qualidade do software diminui com a sua evolução, por possíveis problemas no desenvolvimento e agravado pela possibilidade de mudanças externas.

- *Feedback*: Os processos de evolução do software constituem um complexo sistema de realimentação que é constantemente realizado por seus usuários, podendo ser estes retornos positivos e negativos entre as versões.

A mineração de repositórios de código fonte auxilia a validar as leis de evolução de software, que por sua vez apresentam métricas e medidas que podem auxiliar os gerentes de projetos a entenderem a dinâmica do sistema mantido, assim como a do grupo mantenedor, ou seja, da equipe de desenvolvimento do software, sendo esse o objeto explorado por este trabalho.

3 TRABALHOS RELACIONADOS

Começando pelo trabalho Robbes (2007), aborda-se que sistema de controle de versão apesar de úteis para avaliar a evolução de um sistema de software, as informações que eles contêm são limitadas de várias maneiras. Com base nisso, propõem um repositório de informações alternativo que armazena mudanças incrementais no sistema em estudo, recuperadas da IDE usado para construir o software.

No trabalho de Poncin, Serebrenik e Van Den Brand (2011), foi desenvolvido o FRASR (*Framework for Analyzing Software Repositories*), aumentando o framework de mineração de processos ProM. Isto é, aplicaram técnicas de *Process Mining*, originalmente desenvolvidas para análise de processos de negócios, para mineração de repositórios de software.

Já Ali, Guéhéneuc e Antoniol (2012), apresentaram o Trustrace, uma abordagem de recuperação de rastreabilidade baseada em confiança, para mineração de repositórios de software, baseado em técnicas de recuperação da informação. O MetricMiner (SOKOL, ANICHE e GEROSA, 2013), é uma aplicação web que ajuda em algumas etapas da mineração de repositórios de software, como cálculo de métricas, extração de dados e inferência estatística, diferente do trabalho aqui desenvolvido o foco é no código da aplicação.

Já o SonarQube (CAMPBELL e PAPAPETROU, 2013), trata-se de uma aplicação web que analisa o código fonte e extrai uma variedade de relatórios sobre o sistema, como resultados de métricas de código e dependências estruturais entre as classes. O

foco dessa ferramenta é apoiar a equipe de desenvolvimento e acompanhar a qualidade do código escrito.

Por fim, no trabalho de Kalliamvakou et al. (2014), os pesquisadores exploraram as informações armazenadas nos logs de eventos do GitHub, tentando entender como seus usuários utilizam o site para colaborar no software. Além disso, forneceram um conjunto de recomendações para pesquisadores de engenharia de software sobre como abordar os dados no GitHub, sendo algumas dessas recomendações seguidas para criação da plataforma GIT Viewer, objeto desse trabalho.

4 MINERAÇÃO DE REPOSITÓRIO DE SOFTWARE

Normalmente sistemas de software possuem uma longa história de desenvolvimento, com vários desenvolvedores trabalhando em diferentes partes do sistema. Mesmo assim, nenhum desenvolvedor desconhece completamente o código do sistema, mesmo que quem iniciou o projeto não faz mais parte da equipe.

O processo de mineração de repositório de software analisa a evolução do software e pesquisas neste campo revelam informações úteis para o desenvolvimento de projetos específicos, ou modelos de desenvolvimento de software que podem ser estendidos para outros sistemas (SOKOL, ANICHE e GEROSA, 2013).

O termo "repositório de software" abrange todos os artefatos produzidos durante o desenvolvimento de um sistema de software, desde arquivos com código-fonte do sistema que podem ser armazenados em um sistema de controle de versão, até mensagens em listas de e-mails trocadas entre os desenvolvedores (SOKOL, ANICHE e GEROSA, 2013).

O sistema de controle de versão é uma ferramenta usada no desenvolvimento diário de software, onde os desenvolvedores podem rastrear mudanças no código que estão manutenindo. Um conjunto de alterações feitas no código são registradas em *commits*, e o sistema de controle de versão armazena esses grupos de alterações, como uma linha do tempo.

Em (KAGDI, COLLARD e MALETIC, 2007), alguns trabalhos de mineração de repositório de software (MRS) foram avaliados. Por meio da análise desses trabalhos, os autores propuseram uma taxonomia para classificar as pesquisas nessa área. Esta classificação é baseada em quatro aspectos da pesquisa, considerando: i) Fonte de informação dos dados; ii) objetivos de trabalho, iii) o método usado para analisar os dados e; iv) a granularidade dos dados analisados.

Essa pesquisa se baseai nessa taxonomia, onde a fonte de informações é o GitHub. A saber, o GitHub é uma plataforma de hospedagem de código-fonte com controle de versão usando o Git. Ele permite que programadores, utilitários ou qualquer usuário cadastrados na plataforma contribuam em projetos privados e/ou de código fonte aberto de qualquer lugar do mundo. A seguir listaremos os recursos disponíveis no GitHub que podem ser explorados via API (*Application Programming Interface*):

1 - Repositórios:

- Pesquisar por nome de repositório, descrição ou conteúdo do arquivo README;
- Pesquisar com base no conteúdo de um repositório;
- Pesquisar nos repositórios de um usuário ou organização;
- Pesquisa por tamanho de repositório;
- Pesquisar por número de seguidores;
- Pesquisar por número de garfos;
- Pesquisar por número de estrelas;
- Pesquisar quando um repositório foi criado ou atualizado pela última vez;
- Pesquisar por idioma;
- Pesquisa por tópico;
- Pesquisar por número de tópicos;
- Pesquisa por licença;
- Pesquisar por repositório público ou privado;
- Pesquisar com base no fato de um repositório ser um espelho;
- Pesquisar com base em se um repositório está arquivado;
- Pesquisar com base no número de problemas com good first issue ou help wanted rótulos;

2 – Commits:

- Pesquisar por autor ou committer;

- Pesquisar por data de criação ou do commit;
- Filtrar commits de merge;
- Pesquisar por hash;
- Pesquisar por principal;
- Pesquisar por árvore;
- Pesquisar nos repositórios de usuário ou um a organização;
- Filtrar repositório públicos ou privados;

3 – Código:

- Pesquisar pelo conteúdo do arquivo ou caminho do arquivo;
- Pesquisar por localização de arquivo;
- Pesquisar por tamanho de arquivo;
- Pesquisar por nome de arquivo;
- Pesquisar por extensão de arquivo;

4 - Problemas de pesquisa e solicitações de pull:

- Pesquisar somente problemas e pull requests;
- Pesquisar por título, texto ou comentários;
- Pesquisar nos repositórios de um usuário ou uma organização;
- Pesquisar por estado aberto ou fechado;
- Pesquisar por repositório público ou privado;
- Pesquisar por autor;
- Pesquisar por responsável;
- Pesquisar por menção;
- Pesquisar por menção da equipe;
- Pesquisar por autor do comentário;

- Pesquisar por um usuário envolvido em um problema ou uma pull request;
- Procurar problema e pull requests vinculados;
- Pesquisar por etiqueta;
- Pesquisar por marco;
- Pesquisar por quadro de projeto;
- Pesquisar por status do commit;
- Pesquisar por SHA do commit;
- Pesquisar por nome do branch;
- Pesquisar por linguagem;
- Pesquisar por número de comentários;
- Pesquisar por número de interações;
- Pesquisar por número de reações;
- Pesquisar por pull requests de rascunho;
- Pesquisar por status de revisão e revisor da pull request;
- Pesquisar por data da criação ou da última atualização de um problema ou uma pull request;
- Pesquisar por data de encerramento de um problema ou uma pull request;
- Pesquisar por data do merge da pull request;
- Pesquisar somente pull request com merge ou sem merge;
- Pesquisar com base no fato de o repositório estar arquivado;
- Pesquisar conversas bloqueadas e não bloqueadas;
- Pesquisar por metadados ausentes;

5 - Usuários:

- Pesquisar pelo nome da conta, nome completo ou e-mail público;
- Pesquisar pelo número de repositórios que um usuário possui;

- Pesquisa por localização;
- Pesquisa por idioma do repositório;
- Pesquisar por quando uma conta de usuário foi criada;

6 - Tópicos:

- Limitar a pesquisa com qualificadores de pesquisa;
- Pesquisar repositórios por tópico;

Esses recursos foram explorados na ferramenta desenvolvida “GIT Viewer” e serão apresentados na próxima seção, abordando as telas da plataforma e os dados que foram obtidos do GitHub durante a análise de um repositório.

5 DEMONSTRAÇÃO DA PLATAFORMA

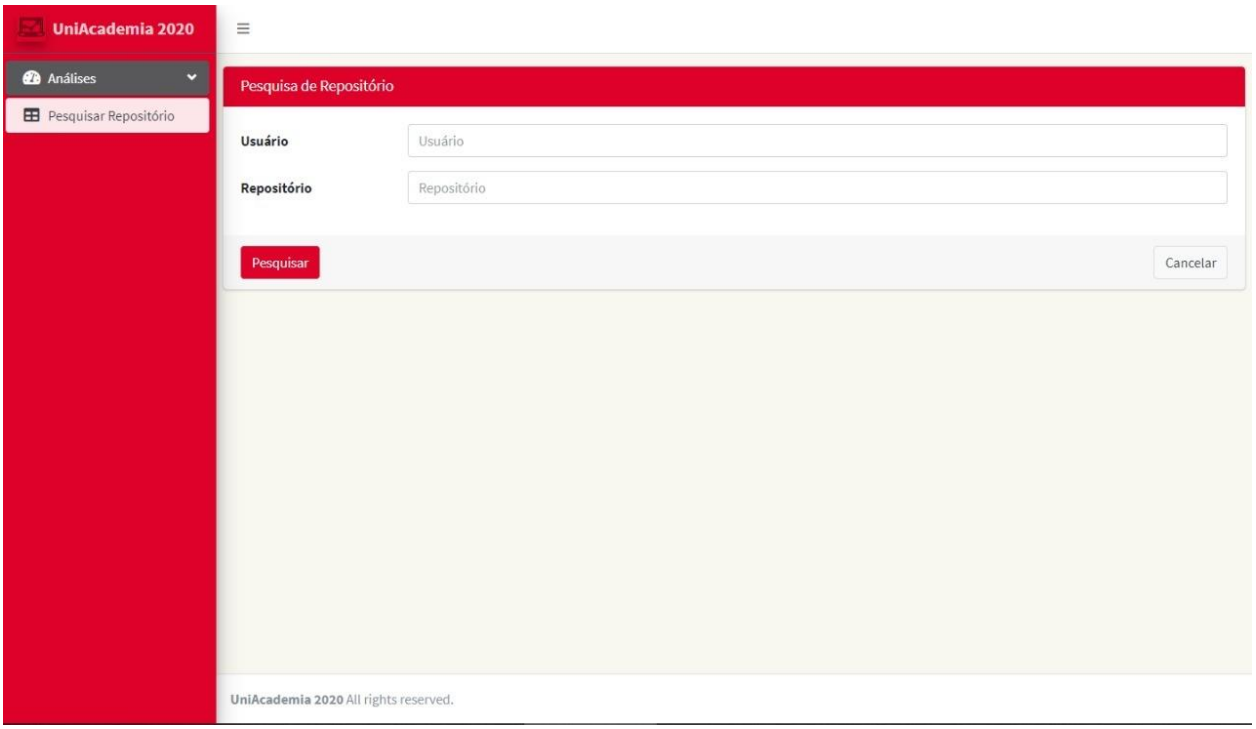
A plataforma desenvolvida possui um fluxo de execução para análise dos dados do repositório conforme apresenta a Figura 1. Seguindo os passos da Figura 1, o ponto inicial é definir o nome do usuário do GitHub, seguido pelo nome do repositório a ser analisado. O preenchimento dessas informações podem ser feitas na tela apresentada na Figura 2.

Figura 1: Fluxo de execução da plataforma GIT Viewer.

Fonte: Elaboração própria.

Conforme apresenta a Figura 2, ao abrir a plataforma GIT Viewer, deve-se definir o repositório para análise.

Figura 2: Definição do repositório no GIT Viewer.



Fonte: Elaboração própria.

Após definir os dados do repositório, a plataforma apresenta um conjunto de informações sobre o repositório em análise, contendo o id do repositório no GitHub, o nome, que é composto pelo nome de usuário e nome do repositório separador por uma barra, número de vezes que o repositório foi marcado com estrela por outros usuários do GitHub, número de *issues* abertas e o número de forks que esse repositório possui. Essas informações podem ser vistas na Figura 3.

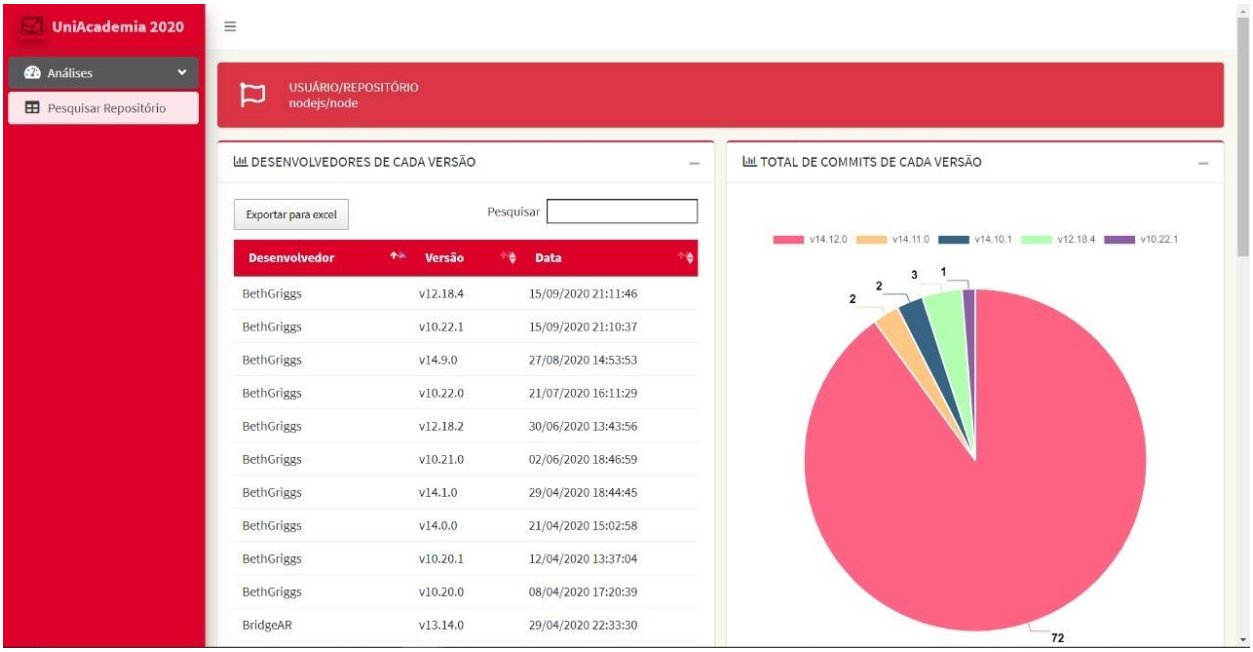
Figura 3: Informações preliminares do repositório no GIT Viewer.

Fonte: Elaboração própria.

Com os dados confirmados é possível obter a análise histórica do repositório usando a API do GitHub. Com base nas consultas realizadas, o GIT Viewer apresenta a lista de desenvolvedores de cada versão, quais foram as versões lançadas no repositório e a data de lançamento de cada uma. Além disso, o usuário pode extrair essas informações em planilhas do excel, para posterior análise. Essas informações são apresentadas em

uma *grid*. Ao lado dessa *grid*, temos um gráfico de pizza representando o total de *commits* por versão, onde cada versão recebe uma cor diferente e o número total, conforme Figura 4.

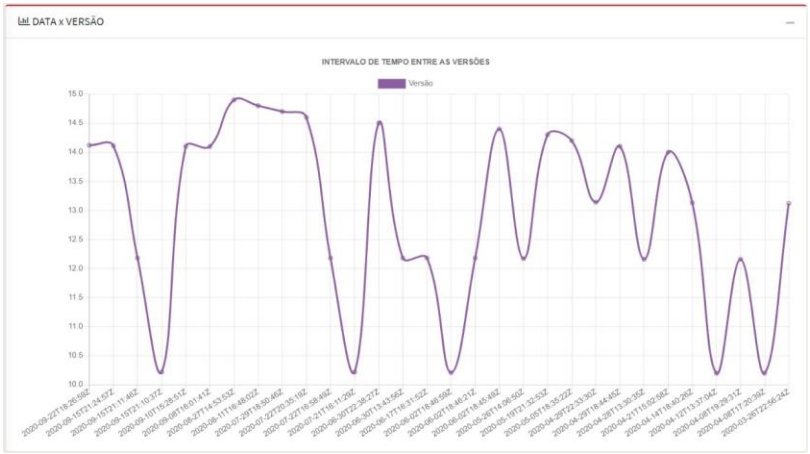
Figura 4: Análise histórica do repositório no GIT Viewer.



Fonte: Elaboração própria.

Outra análise apresentada pela plataforma é o intervalo de datas entre o lançamento de cada versão. O repositório analisado foi o “node”, do usuário “nodejs”, onde o mesmo apresenta diferentes versões sendo lançadas em concomitante. Como pode ser observado na Figura 5, dentro do intervalo analisado foram lançadas versões no node 10.0.+, 12.0.+, 13.0.+, 14.0.+ e 14.5.+, em dias alternados.

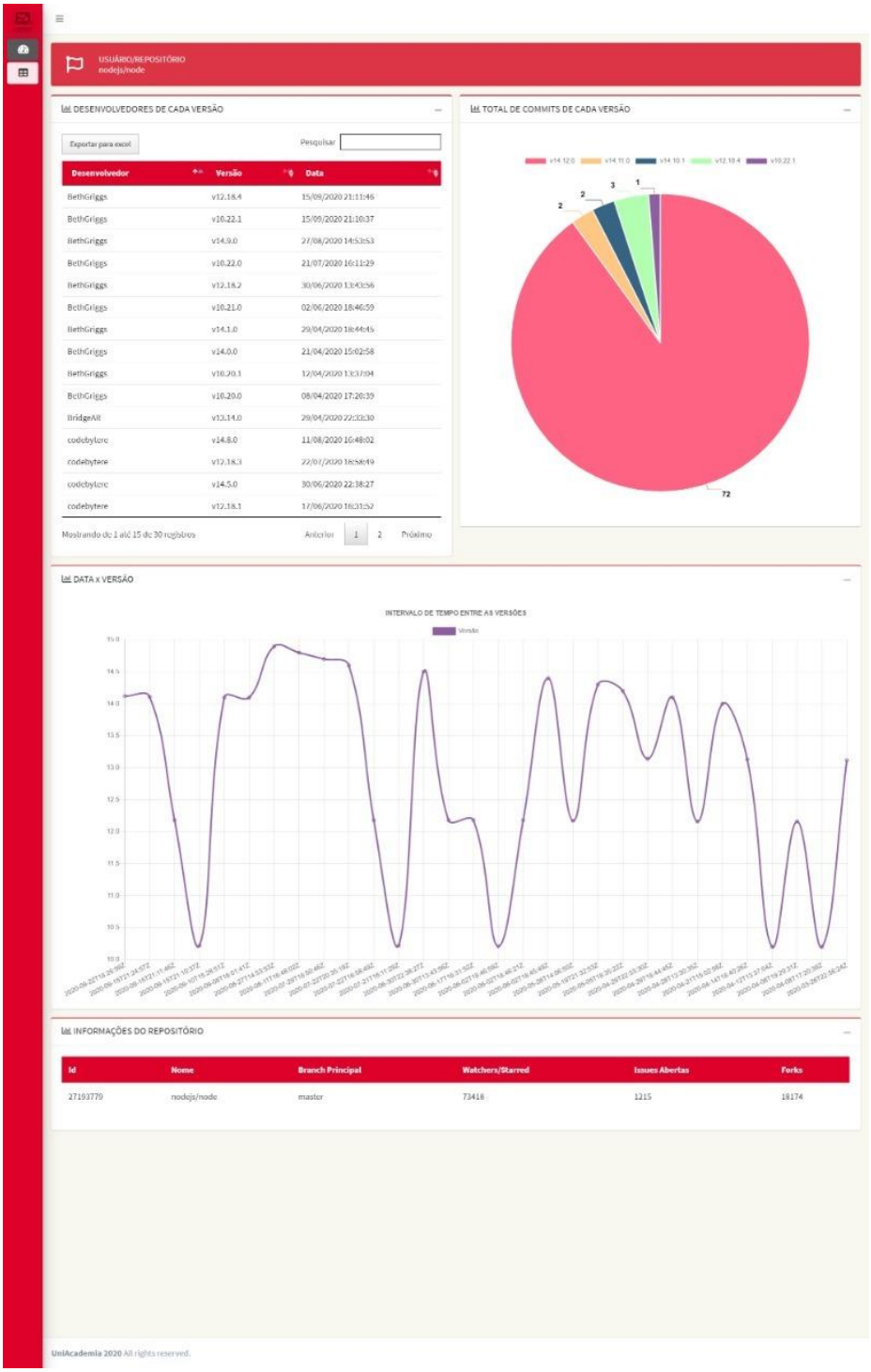
Figura 5: Análise gráfica do repositório no GIT Viewer.



Fonte: Elaboração própria.

Conforme apresenta a Figura 5, os dados de versões lançadas são referentes aos últimos seis meses de desenvolvimento, contados a partir do dia atual da análise. Uma visão completa da interface da plataforma está disponível na Figura 6.

Figura 6: Interface completa do GIT Viewer.



Fonte: Elaboração própria.

O código fonte da plataforma está disponível em <https://github.com/ces-jf/IC-2020> e um vídeo de demonstração do uso pode ser visto no endereço <https://youtu.be/2yAOt1lfyAw>.

6 CONSIDERAÇÕES FINAIS

Esse artigo apresenta o resultado do desenvolvimento de uma iniciação científica (IC) desenvolvida no Centro Universitário Academia (UniAcademia) pelos alunos do curso de Bacharelado em Engenharia de Software. Ao longo da IC foi estudado como capturar dados do GitHub a partir da API disponibilizada.

Como resultado foi desenvolvido uma plataforma web que lê os dados da API do GitHub e apresenta algumas métricas que demostram a evolução do software a partir dos dados do repositório. Apesar de disponível, a plataforma ainda apresenta algumas limitações, tais como a análise de repositórios privados e de personalização das consultas pelos usuários. Essas limitações pretendemos explorar em versões futuras da plataforma.

Entender a dinamia de uma equipe de desenvolvimento e do ciclo de vida de um software não é uma tarefa trivial, pois envolve vários elementos, a MRS é um caminho que permite explorar a partir de dados históricos como a equipe e o software se comportaram ao longo do tempo, auxiliando a melhorar o processo de desenvolvimento e da gestão de equipes.

ABSTRACT

Currently, there is a huge range of source repositories that can be accessed and studied openly especially open-source ones. The big point is, that to explain phenomena within software engineering we need a lot of historical data, and open-source projects are a great opportunity to explore this universe and help us better understand the process of maintenance, evolution, and decay of software, based on the history of other software. With the analysis of software maintenance, based on historical data contained in source code repositories, it is possible to detect and predict evolutions and/or decay in software systems, through data mining techniques. In this work, a platform for the analysis of source code repositories was developed to help project managers during the lifecycle management of the maintained software. The platform's objectives are to identify effects on the software that may lead to its decay, resulting from failures in coding, design, or architecture. To demonstrate the platform's operation, a use case will be presented, analyzing an open-source project available at GitHub. Considering the source code repository as a historical base, through data mining it is possible to extract useful information that helps in the software maintenance process, making it less costly, because the information contained can help identify errors or patterns that lead to rework or software decay and that are sometimes information discarded by companies.

REFERÊNCIAS

ALI, Nasir; GUÉHÉNEUC, Yann-Gaël; ANTONIOL, Giuliano. **Trustrace: Mining software repositories to improve the accuracy of requirement traceability links**. IEEE Transactions on Software Engineering, v. 39, n. 5, p. 725-741, 2012.

BASILI, Victor R.; BRIAND, Lionel C..; MELO, Walcélio L. **A validation of object-oriented design metrics as quality indicators.** IEEE Transactions on software engineering, v. 22, n. 10, p. 751-761, 1996.

BELADY, Laszlo A..; LEHMAN, Meir M. **A model of large program development**. IBM Systems journal, v. 15, n. 3, p. 225-252, 1976.

CAMPBELL, G.; PAPAPETROU, Patroklos P. **SonarQube in action**. Manning Publications Co., 2013.

COLAÇO JR, M. **Mineração de Repositórios de Software: A Computação ajudando à Computação**. SQL Magazine, 2010.

KAGDI, Huzefa; COLLARD, Michael L.; MALETIC, Jonathan I. **A survey and taxonomy of approaches for mining software repositories in the context of software evolution**. Journal of software maintenance and evolution: Research and practice, v. 19, n. 2, p. 77-131, 2007.

KALLIAMVAKOU, Eirini et al. **The promises and perils of mining GitHub**. In: Proceedings of the 11th working conference on mining software repositories. 2014. p. 92-101.

LEHMAN, Manny M. **Laws of software evolution revisited**. In: European Workshop on Software Process Technology. Springer, Berlin, Heidelberg, 1996. p. 108-124.

LEHMAN, Meir M. **Programs, life cycles, and laws of software evolution**. Proceedings of the IEEE, v. 68, n. 9, p. 1060-1076, 1980.

PONCIN, Wouter; SEREBRENIK, Alexander; VAN DEN BRAND, Mark. **Process mining software repositories**. In: 2011 15th European Conference on Software Maintenance and Reengineering. IEEE, 2011. p. 5-14.

ROBBES, Romain. **Mining a change-based software repository**. In: Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007). IEEE, 2007. p. 15-15.

SILVA, Maria Goreti Simão da. **Mineração de repositórios de software modelos de previsão de pedidos de evolução de software**. 2013. Tese de Doutorado. Faculdade de Ciências e Tecnologia.

SOKOL, Francisco Zigmund; ANICHE, Mauricio Finavaro; GEROSA, Marco Aurélio.

MetricMiner: Supporting researchers in mining software repositories. In: 2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM). IEEE, 2013. p. 142-146.

SOUZA, Bianca Morais et al. **Mineração de Repositório de Software: Investigando a qualidade do software em projetos de código aberto**. ANALECTA-Centro de Ensino Superior de Juiz de Fora, v. 5, n. 5, 2020.