Commit Explorer: Uma Solução *Open Source* para Extração e Avaliação de *Commits* e Códigos

Mateus Florentino Back
Campus Paranavaí
Instituto Federal do Paraná
Paranavaí, PR, Brasil
Email: mateusfback14@gmail.com.br

Frank Willian Cardoso de Oliveira
Campus Paranavaí
Instituto Federal do Paraná
Paranavaí, PR, Brasil
Email: frank.willian@ifpr.edu.br

Marcelo Figueiredo Terenciani
Campus Paranavaí
Instituto Federal do Paraná
Paranavaí, PR, Brasil
marcelo.terenciani@ifpr.edu.br

***Abstract*—This paper presents the development and evaluation of Commit Explorer, an open-source tool for extracting, analyzing, and visualizing data from GitHub repositories, with a focus on educational contexts. The work is grounded in literature on software quality, static code analysis, and teaching practices supported by distributed version control systems. Data collection involved semi-structured interviews with Software Engineering instructors and the implementation of a functional prototype. The tool integrates a customized PMD configuration to assess code quality and provides interactive dashboards for metric visualization. Expected outcomes include supporting educators in monitoring students' coding practices, streamlining assessments, and fostering improved programming skills through structured feedback.**

***Keywords*—software quality, commit analysis, static code analysis, PMD, software engineering education**

***Resumo*—Este trabalho apresenta o desenvolvimento e a avaliação do Commit Explorer, uma ferramenta *open source* voltada à extração, análise e visualização de dados provenientes de repositórios GitHub, com foco no contexto educacional. A pesquisa é fundamentada na literatura sobre qualidade de software, análise estática de código e práticas de ensino apoiadas por sistemas de controle de versão. A coleta de dados para o desenvolvimento da aplicação envolveu entrevistas semiestruturadas com docentes de Engenharia de Software e a implementação de um protótipo funcional. A ferramenta integra uma configuração personalizada do PMD para avaliar a qualidade do código e disponibiliza *dashboards* interativos para visualização de métricas. Os resultados esperados incluem apoiar docentes no acompanhamento das práticas de programação dos estudantes, agilizar o processo avaliativo e promover a melhoria das habilidades de codificação por meio de *feedback* estruturado.**

***Palavras-chave*—qualidade de software, análise de commits, análise estática de código, PMD, ensino de engenharia de software**

I. Introdução

O uso de sistemas de controle de versão, como o Git, e de plataformas de hospedagem de código, como o GitHub, é fundamental na indústria de software. Essas ferramentas permitem gerenciar alterações, colaborar de forma distribuída e manter o histórico completo dos projetos, consolidando-se como práticas essenciais para o desenvolvimento moderno de aplicações.

No contexto educacional, o acompanhamento manual de múltiplos repositórios e *commits* em turmas numerosas apresenta desafios significativos. Cada projeto possui um histórico próprio, com diferentes estilos de escrita de código, estruturas de pastas e padrões de versionamento. A análise manual demanda tempo elevado, carece de critérios padronizados e pode gerar avaliações subjetivas, dificultando a identificação de boas práticas ou de problemas recorrentes.

Ferramentas de análise estática, como o PMD, surgem como alternativas promissoras por oferecer diagnósticos objetivos sobre a qualidade do código, identificando *code smells* e calculando métricas como complexidade ciclomática. No entanto, sua aplicação direta em ambientes educacionais exige ajustes para evitar falsos positivos e adequar as regras ao nível de maturidade dos estudantes.

Diante desse cenário, este trabalho apresenta o desenvolvimento de uma solução *open source* para extração, análise e visualização de *commits* e código-fonte hospedados no GitHub, o Commit Explorer. O sistema integra análise estática customizada a um painel interativo, fornecendo indicadores como frequência de *commits*, clareza das mensagens e métricas de qualidade do código. Sua disponibilização como software livre favorece a colaboração de docentes, discentes e comunidade externa, estimulando a evolução contínua e a adaptação a diferentes contextos educacionais.

II. Fundamentação Teórica

O uso de sistemas de controle de versão como o Git e plataformas colaborativas como o GitHub tornou-se essencial no desenvolvimento de software moderno, permitindo gerenciar alterações de forma distribuída, colaborar em equipe e manter um histórico completo e auditável das modificações realizadas [1]. No ensino de programação, essas ferramentas aproximam

o estudante das práticas adotadas na indústria, favorecem o trabalho em grupo e permitem registrar não apenas o produto final, mas todo o processo de desenvolvimento.

No contexto acadêmico, o acompanhamento da evolução de projetos por meio do histórico de *commits* possibilita identificar padrões de trabalho, estilos de codificação e eventuais problemas de organização. Ferramentas como o GitHub oferecem recursos complementares, como *issues*, *pull requests* e integração contínua, que, quando utilizados adequadamente, contribuem para uma gestão mais eficiente de atividades práticas. Entretanto, a análise manual desses dados em turmas numerosas demanda tempo elevado e podendo gerar avaliações subjetivas [2].

Boas práticas de *commits* incluem granularidade adequada e mensagens claras, conforme os princípios de *Clean Code* [3]. *Commits* pequenos e frequentes facilitam a rastreabilidade, enquanto *commits* extensos, pouco frequentes ou com mensagens genéricas dificultam a manutenção e a revisão, podendo inclusive indicar desorganização ou plágio. No ensino, o monitoramento desses aspectos é relevante para fornecer avaliação construtiva e orientar melhorias nas práticas de versionamento dos alunos.

Ferramentas de análise estática, como o PMD, surgem como alternativas promissoras para automatizar parte dessa avaliação. O PMD é capaz de detectar *code smells*, como variáveis não utilizadas, duplicações de código e complexidade excessiva, além de calcular métricas como a complexidade ciclomática [4], [5]. Essas métricas permitem identificar trechos de código potencialmente problemáticos, que podem comprometer a manutenibilidade e legibilidade do software. Estudos indicam que a aplicação de análise estática em ambientes educacionais, aliada a um *feedback* estruturado, pode contribuir para o aprendizado dos estudantes e reduzir a carga de trabalho docente [6].

Entretanto, a aplicação direta de regras industriais em contextos educacionais pode gerar ruído e falsos positivos, especialmente quando o código é produzido por iniciantes. Configurações personalizadas são necessárias para ajustar o nível de rigor e relevância das regras ao estágio de aprendizado dos alunos [7]. Isso exige uma abordagem criteriosa, que considere tanto a detecção de problemas recorrentes quanto a clareza das mensagens e a coerência do histórico de desenvolvimento.

O conceito de *refactoring* [8] complementa essa perspectiva, reforçando a importância de melhorar gradualmente a estrutura interna do código sem alterar seu comportamento externo. Ensinar boas práticas de refatoração e integrá-las ao fluxo de desenvolvimento estimula a produção de código mais limpo e sustentável. Quando combinado com avaliação formativa [9], o controle de versão e a análise estática personalizada permitem acompanhar a evolução das habilidades dos alunos, identificar pontos críticos e orientar intervenções pedagógicas mais assertivas.

Dessa forma, a combinação de controle de versão, análise estática adaptada ao contexto acadêmico e visualização de métricas fundamenta a proposta do Commit Explorer. A ferramenta integra esses elementos em uma solução *open source* que visa apoiar docentes no acompanhamento do processo de desenvolvimento, oferecendo indicadores objetivos, relatórios interativos e um ambiente que favorece a aprendizagem contínua.

III. Metodologia

A pesquisa adotou abordagem incremental e interativa [10], com foco na construção de uma ferramenta educacional *open source* para análise de *commits* no GitHub. Foram realizadas três entrevistas semiestruturadas [11] com docentes do curso de Engenharia de Software do Instituto Federal do Paraná - Campus do Paraná, no mês de março de 2025, para identificar critérios e necessidades na avaliação de atividades práticas com GitHub.

O desenvolvimento seguiu três etapas principais: levantamento de requisitos, implementação da solução e avaliação inicial. Na primeira etapa, foram consolidados requisitos funcionais e não funcionais a partir das entrevistas. Na segunda, foi implementada uma API REST em Java/Spring Boot para extração e análise de *commits*, utilizando o JGit e o PMD com regras personalizadas ao contexto acadêmico, e um *dashboard* web em React.js para exibição das métricas. A terceira etapa compreendeu a validação preliminar da ferramenta junto aos docentes avaliando usabilidade, clareza das métricas e efetividade da análise.

A Figura 1 apresenta a visão geral da metodologia de desenvolvimento, incluindo o fluxo de entrevistas, análise de requisitos, implementação e validação. Essa representação serve como referência visual do processo adotado.

IV. Proposta de Solução

O Commit Explorer tem como objetivo apoiar o ensino de programação por meio da análise automatizada de *commits* em repositórios hospedados no *GitHub*, oferecendo métricas objetivas e visualização interativa para docentes e estudantes. A solução adota arquitetura orientada a domínio (DDD) com separação de comandos e consultas (CQRS), estruturando o sistema em camadas de Domínio, Aplicação, Infraestrutura e Apresentação.

O **backend**, desenvolvido em Java com o *framework* Spring Boot, é responsável pela coleta e processamento dos dados. Ele utiliza a biblioteca JGit para clonar e percorrer repositórios, processando cada *commit* de forma assíncrona. Para cada *commit*, o sistema verifica duplicidade, registra autores ainda

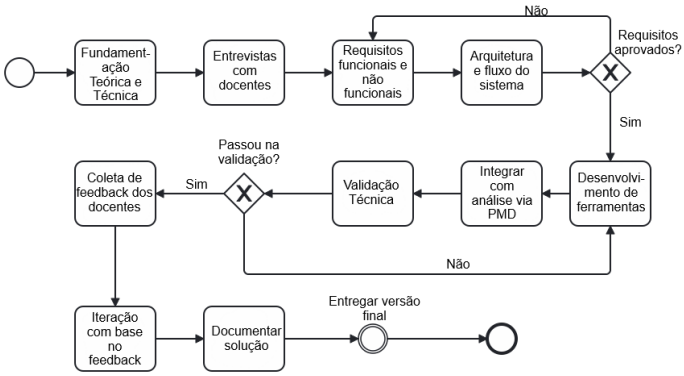

Fig. 1. Metodologia de desenvolvimento do Commit Explorer.

não cadastrados e aplica regras gerais (clareza e padrão das mensagens, frequência e tamanho das contribuições) e regras específicas voltadas ao contexto educacional (correlação com *issues*, coerência entre datas e autores).

Quando arquivos escritos em Java são identificados, o *back-end* aciona a análise estática por meio do PMD, configurado com um conjunto de regras adaptadas ao ambiente acadêmico. Esse processo gera métricas sobre a ocorrência de *code smells*, complexidade ciclomática e demais indicadores de qualidade do código. Todos os resultados obtidos são consolidados e armazenados em um banco de dados relacional, de forma a garantir rastreabilidade e permitir consultas futuras, servindo como base para relatórios e visualizações no painel.

O **frontend**, desenvolvido em React.js, apresenta os resulta dos em um painel *web* responsivo. Ele permite filtrar e visualizar métricas por período, *branch*, autor ou tipo de análise, com dados dispostos em gráficos, tabelas e indicadores de fácil interpretação. A interface foi projetada para oferecer usabilidade simples, com foco em clareza e rapidez na obtenção de informações.

O sistema também prevê funcionalidades adicionais, como:

- Extração de *commits* filtrados por *branch*, autor e intervalo de datas;
- Identificação de *code smells* e cálculo de métricas de complexidade;
- Avaliação da clareza e padronização das mensagens de *commit*;
- Correlação de commits com tarefas e *issues* vinculadas;
- Geração de relatórios consolidados por aluno ou projeto, nos formatos PDF e CSV;
- Visualização de dados históricos para acompanhamento longitudinal do desempenho.

A Figura 2 apresenta o fluxo resumido de operação do Commit Explorer, desde a interação inicial do usuário no painel até a consolidação e disponibilização das métricas.

O projeto é disponibilizado como *software open source*, com repositório público, documentação, guia de contribuição e licença permissiva, possibilitando que a comunidade acadêmica e profissional participe ativamente de sua evolução. Os repositórios estão disponíveis em:

Backend: https://github.com/mateusback/commitExplorer
Frontend: https://github.com/mateusback/commit-explorer-f

V. Resultados e Discussão

A versão inicial do Commit Explorer foi validada com docentes do curso de Engenharia de Software do Instituto Federal do Paraná, por meio da análise de repositórios acadêmicos das disciplinas de Projeto Integrador, Construção de Software e Programação para Web.

Os testes mostraram que a ferramenta possibilita acompanhar métricas como frequência e coesão dos *commits*, clareza das mensagens, presença de *code smells* e evolução do código. Os docentes relataram que o Commit Explorer agiliza a avaliação, facilita a detecção de padrões problemáticos e oferece evidências objetivas para o *feedback*.

Na comparação com a avaliação manual, foram citadas dificuldades como a falta de padronização dos repositórios,

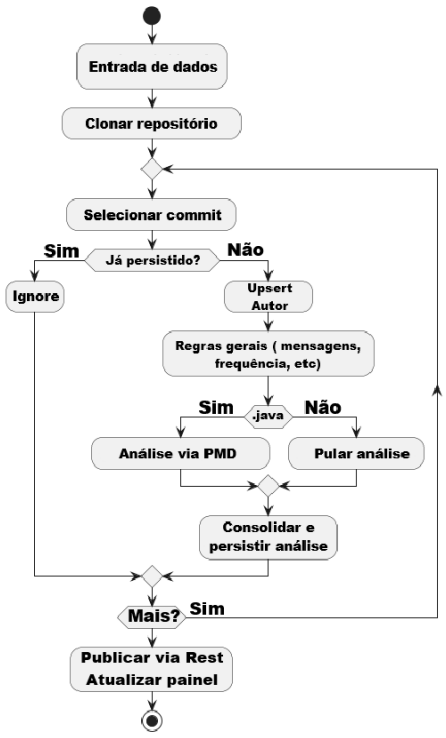

Fig. 2. Fluxo resumido de análise de commits no Commit Explorer.

a complexidade em rastrear autoria individual e a sobrecarga causada pelo grande volume de *commits*. A interface da ferramenta, ilustrada na Figura 3, foi destacada pela clareza das visualizações e pela facilidade em comparar projetos.

Entre as sugestões, destacaram-se a ampliação da detecção de *code smells*, o refinamento das métricas de clareza e autoria, além da integração com *issues* e o Projetos do *Github*. O caráter *open source* foi considerado um ponto forte, pois permite adaptações e integrações em diferentes contextos. Como limitações, mencionou-se a necessidade de expandir o conjunto de regras do PMD e de otimizar o desempenho em repositórios de grande porte.

Adicionalmente, ressaltou-se a importância de incorporar recursos para identificar problemas nas mensagens de *commits*, uso inadequado de padronizações, descumprimento de convenções de linguagem, entre outros aspectos. Ainda assim, a versão atual do *Commit Explorer* oferece uma base para novos testes com docentes e discentes, permitindo que os estudantes utilizem a ferramenta não apenas como suporte à avaliação, mas também como instrumento para reconhecer e corrigir pontos que necessitam de melhoria em suas práticas de desenvolvimento.

VI. Conclusão

O *Commit Explorer* demonstrou potencial para apoiar o ensino de programação por meio da análise automatizada de *commits* e código-fonte hospedados no GitHub. A integração de métricas de versionamento e análise estática permitiu oferecer uma visão objetiva do processo de desenvolvimento, facilitando a avaliação e o *feedback* para estudantes. O desenvolvimento da ferramenta foi orientado por entrevistas com docentes e levantamento bibliográfico, garantindo que suas funcionalidades atendessem às necessidades reais do contexto acadêmico. Mesmo em estágio inicial, a solução evidenciou viabilidade técnica e utilidade prática, especialmente em turmas com grande volume de entregas e projetos colaborativos.

A disponibilização como *software open source* favorece a colaboração de docentes, discentes e comunidade externa, permitindo que a ferramenta evolua de forma contínua e se adapte a diferentes realidades educacionais. A arquitetura modular e a documentação disponível contribuem para a manutenção e a ampliação das funcionalidades ao longo do tempo. Como trabalhos futuros, pretende-se expandir o conjunto de métricas analisadas, incluir suporte a novas linguagens, criar um histórico de avaliações por docente para acompanhamento longitudinal, ampliar recursos de gamificação e otimizar o desempenho da análise em repositórios de grande porte, além de realizar estudos de caso em diferentes instituições para validar e aprimorar a solução em cenários variados.

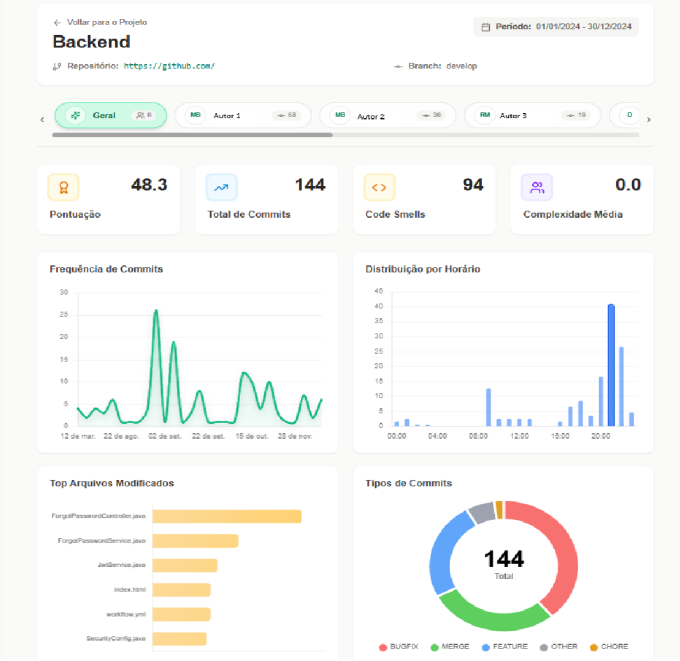

Fig. 3. Dashboard com informações de uma análise.

Referências

[1] S. Chacon and B. Straub, *Pro Git*. Apress, 2021.
[2] D. Spinellis, "Git," *IEEE Software*, vol. 29, no. 3, pp. 100–101, 2012.
[3] R. C. Martin, *Clean code: A handbook of agile software craftsmanship*. Pearson Education, 2008.
[4] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.
[5] A. Kaur and R. Nayyar, "A comparative study of static code analysis tools for vulnerability detection in c/c++ and java source code," *Procedia Computer Science*, vol. 167, pp. 3214–3223, 2020, available at https://doi.org/10.1016/j.procs.2020.04.217.
[6] E. A. AlOmar, S. A. AlOmar, and M. W. Mkaouer, "On the use of static analysis to engage students with software quality improvement: An experience with pmd," *arXiv preprint arXiv:2302.05554*, jul 2023, available at https://doi.org/10.48550/arXiv.2302.05554.
[7] H. Cui, M. Xie, T. Su, C. Zhang, and S. H. Tan, "An empirical study of false negatives and positives of static code analyzers from the perspective of historical issues," *arXiv preprint arXiv:2408.13855*, aug 2024, available at https://doi.org/10.48550/arXiv.2408.13855.
[8] M. Fowler, *Refactoring: Improving the design of existing code*. Addison-Wesley, 1999.
[9] D. J. Nicol and D. Macfarlane-Dick, "Formative assessment and self-regulated learning: A model and seven principles of good feedback practice," *Studies in higher education*, vol. 31, no. 2, pp. 199–218, 2006.
[10] E. Bezerra, *Princípios de Análise e Projeto de Sistemas com UML*, 7th ed. Rio de Janeiro: Elsevier Editora Ltda., 2007.
[11] M. T. D. Fraser and S. M. G. Gondim, "Da fala do outro ao texto negociado: discussões sobre a entrevista na pesquisa qualitativa," *Paidéia*, vol. 14, no. 28, pp. 139–152, 2004.