UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS

UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO

ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

Henrique dos Santos Ramires Granatto

ANÁLISE DE QUALIDADE DE CÓDIGO COM MODELOS DE MACHINE LEARNING

Porto Alegre

2023

UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS

UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO

ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

Henrique dos Santos Ramires Granatto

ANÁLISE DE QUALIDADE DE CÓDIGO COM MODELOS DE MACHINE LEARNING

Trabalho apresentado à disciplina Engenharia de Software Aplicada, pelo Curso de Especialização em Engenharia de Software da Universidade do Vale do Rio dos Sinos - UNISINOS, ministrada pela Profª Drª Josiane Brietzke Porto

Porto Alegre

2023

Análise de Qualidade de Código com Modelos de Machine Learning

Henrique dos Santos Ramires Granatto[1]

[1] Unidade Acadêmica de Pesquisa e Pós-Graduação – Universidade do Vale do Rio do Sinos – (UNISINOS) – Porto Alegre – RS – Brasil

henrique.ramires.granatto@gmail.com

Abstract. *This paper describes the creation of a system for software quality analysis that uses machine learning. A literature review was carried out, and the focus remained on investigating how to use models to make these types of tools more intelligent both for presenting data and suggesting improvements to the code. The results demonstrate that machine learning models do not make these tools obsolete, but complement them.*

Resumo. *Esse artigo descreve a criação de um sistema para análise de qualidade de software que utiliza machine learning. Foi realizada uma revisão da literatura, e o foco manteve-se em investigar como utilizar modelos para tornar esse tipo de ferramenta mais inteligente tanto para a apresentação dos dados, quanto para a sugestão de melhorias no código. Os resultados demonstram que modelos de machine learning não tornam essas ferramentas obsoletas, mas as complementam.*

1. Introdução

A qualidade do código é um pilar fundamental no desenvolvimento de software, influenciando diretamente a eficiência, manutenibilidade e confiabilidade de um sistema. A falta de avaliação constante da qualidade do código pode resultar em bugs recorrentes, retrabalho e, consequentemente, em custos adicionais e atrasos no desenvolvimento de novas funcionalidades. Ferramentas como o SonarQube oferecem uma análise estática da qualidade do código, no entanto, a transição efetiva dos dados fornecidos por essas ferramentas para implementação de melhorias pode ser desafiadora, especialmente para profissionais com menor experiência.

A complexidade e falta de documentação em códigos muitas vezes dificultam a definição e implementação efetiva de melhorias, especialmente para profissionais menos experientes. Apesar da existência de ferramentas de análise estática, há uma lacuna na transformação desses dados em ações práticas. Ao mesmo tempo, observamos o surgimento de modelos de Machine Learning (ML) para linguagem natural e criação de código, agnósticos à linguagem de programação, apresentando-se como uma potencial solução para superar esses desafios.

Neste contexto, surge a oportunidade de explorar abordagens inovadoras que integrem modelos de Machine Learning (ML) para aprimorar a análise da qualidade do código e proporcionar sugestões práticas de aprimoramento. Assim sendo, o objetivo deste trabalho pode ser representado pela seguinte questão de pesquisa: como podemos empregar modelos de Machine Learning (ML) para auxiliar na análise dos dados

provenientes de ferramentas de qualidade de código e gerar sugestões práticas de aprimoramento para o código analisado?

O objetivo principal desse trabalho é implementar uma solução tecnológica que utilize esses modelos para aprimorar a análise da qualidade do código, proporcionando sugestões efetivas de melhorias. Para atingir esse objetivo geral, foram definidos alguns passos importantes. Foi necessário definir qual ferramenta estática seria utilizada para revisar o código, assim como quais tipos de melhorias relatadas pela ferramenta seriam abordadas. Após, qual modelo de Machine Learning (ML) seria utilizado para gerar a análise do código com base no dados retornados pela ferramenta estática e, por fim, se o resultado gerado pelo modelo de fato apresenta uma melhoria na qualidade do código analisado, dentro do escopo definido.

A escolha deste tema é motivada pela necessidade de preencher a lacuna entre a análise estática da qualidade do código e a implementação efetiva de melhorias. A abordagem proposta representa uma inovação frente às soluções existentes, promovendo maior acessibilidade e eficácia na otimização de códigos. A relevância deste trabalho é evidenciada pela crescente complexidade dos projetos de software e pela demanda por práticas mais eficientes na gestão da qualidade do código.

Este artigo está estruturado da seguinte forma: a Seção 2 apresenta uma fundamentação teórica relacionada ao tema. Na Seção 3, descrevemos a metodologia utilizada para alcançar os objetivos propostos. A Seção 4 apresenta os resultados obtidos. Finalmente, a Seção 5 conclui o artigo, resumindo as contribuições do trabalho e delineando possíveis direções para pesquisas futuras.

2. Fundamentação Teórica

Esta seção traz uma breve discussão teórica do problema, com o objetivo de fundamentá-lo nas teorias existentes, bem como os trabalhos relacionados já realizados.

2.1 Code smells

No campo da engenharia de software, o termo code smells refere-se a características ou padrões no código-fonte que indicam a possível presença de problemas ou más práticas de programação. Esses "cheiros de código" não são erros diretos, mas indícios de que algo pode estar errado, sugerindo a necessidade de uma análise mais aprofundada (Fowler et al, 1999).

Dentro do contexto dos code smells, a God Class é um padrão identificado como um indicador de má prática de programação. Esse padrão se refere a uma classe que tem muitos campos de instância, muitos métodos e geralmente faz muitas coisas. Em essência, a God Class é um ponto focal de um sistema, acumulando muitas responsabilidades e, por conseguinte, violando o princípio de responsabilidade única.

2.2 Machine Learning

Machine Learning (ML) é uma subárea da inteligência artificial (IA) que se concentra no desenvolvimento de algoritmos e modelos capazes de aprender padrões e tomar decisões com base em dados, sem uma programação explicita.

Existem três principais tipos de aprendizado de máquina, sendo um deles o Aprendizado Supervisionado, em que o modelo é treinado em um conjunto de dados rotulado, onde a saída desejada é conhecida. Ele aprende a mapear as entradas para as saídas, tornando-se capaz de fazer previsões em novos dados (STOIAN, 2020).

O emprego de técnicas de Machine Learning na detecção de code smells representa uma abordagem inovadora e eficiente para lidar com a complexidade crescente dos sistemas de software modernos. Diversos estudos foram e estão sendo conduzidos para examinar o uso de machine learning para detecção de code smells, como o realizado por Alazba et al em 2021 e Lei et al em 2020.

3. Metodologia

Esta é uma pesquisa qualitativa por se tratar de algo que não pode ser mensurado em aspectos quantitativos, já que o objetivo é avaliar a capacidade de modelos de machine learning de gerarem relatórios analíticos e sugestões de código que agreguem valor para um código que possuí code smells.

Esta pesquisa também pode ser classificada como aplicada e exploratória, pois seu objetivo é "[...] gerar conhecimentos para a aplicação prática, dirigidos à solução de problemas específicos, envolvendo verdades e interesses locais" e "[...] proporcionar maior familiaridade com o problema, com vistas a torná-lo mais explícito ou a construir hipóteses" (Gerhardt e Silveira, 2009, p. 35).

Na primeira fase, foi realizada uma revisão abrangente da literatura existente relacionada à análise de qualidade de software e ao uso de machine learning nesse context. A mesma também pode ser classificada como exploratória por ser realizada com base em materiais já elaborados como livros, artigos e teses (Gil, 2007).

Na segunda fase, o foco foi o desenvolvimento do protótipo da solução tecnológica, que compreende a interface de usuário, a integração com o modelo de machine learning, a integração com a ferramenta de análise de qualidade de código e o processamento dos dados gerados para apresentação.

Na terceira fase, foi realizada a análise e a avaliação dos dados que foram gerados pelo protótipo, a fim de validar se o mesmo alcançou o objetivo esperado, ou seja, se sugestão proposta pelo modelo de machine learning trouxe uma melhore real na qualidade do código.

4. Desenvolvimento e Apresentação dos Resultados

Esta seção apresenta informações sobre a arquitetura utilizada na construção do Sistema, o seu funcionamento e os resultados obtidos. Imagens do software e o repositório com o código fonte se encontram no Apêndice.

4.1 Arquitetura

O sistema foi construído seguindo uma estrutura cliente-servidor, com duas aplicações independentes. Ambas as aplicações foram desenvolvidas utilizando Javascript (NodeJS). O modelo de machine learning utilizado para gerar o relatório foi o ChatGPT,

e a ferramenta utilizada para realizar a análise de código foi o Dr. Tools.A comunicação entre as aplicações é realizada através de API Rest (para o envio do repositório que será analisado) e Socket (para recebimento das atualizações sobre a análise).

Em relação ao frontend, foi utilizado o Vite, que proporciona a implementação de um ambiente frontend que utiliza NodeJS e abstrai os detalhes de implementação do servido. Para o design, foi utilizado o a biblioteca de components Bootstrap. Em relação ao backend, foi utilizado o framework ExpressJS, muito conhecido e utilizado pela comunidade.Para a comunicação com o ChatGPT foi utilizada a biblioteca do OpenAI.

4.2 Funcionamento

O primeiro passo na utilização da ferramenta é o envio do repositório público. No segundo passo, o sistema irá realizar o download do código fonte. No terceiro passo, o sistema irá executar a análise do código utilizando o Dr. Tools. No quarto passo, o sistema irá escolher um ponto do código que contém os smell Long Method e Complex Method. No quinto passo, o sistema irá enviar esse trecho do código, junto com informações sobre os dois tipos de smells, para o ChatGPT, solicitando que o mesmo gere uma análise que contenha: a explicação de cada um dos smells, a explicação do porquê o trecho de código contém esses smells, e uma sugestão de melhora para o código. No sexto passo, a aplicação backend irá enviar todos os dados coletados para a aplicação frontend via socket.

4.3 Resultados

Passar realizar testes e validar a qualidade dos dados entregues pelo sistema, foi solicitada a autorização de um desenvolvedor para que um de seus repositórios de código pessoais pudesse ser analisado. O relatório com os dados foi entregue ao mesmo para que ele pudesse dar o seu parecer.

O desenvolvedor relatou que as informações em texto são sempre claras e relevantes. Em relação a sugestão de código, algumas partes são de fato úteis e aproveitáveis, enquanto outras apresentam uma ideia relevante, mas uma implementação ruim ou pobre.

Quanto desafios, é possível destacar dois principais: a dificuldade de se encontrar modelos de machine learning que possuam um canal de integração oficial, muitos não tem possibilidade de integração ou a integração é através de engenharia reversa; tempo de resposta do ChatGPT para responder as requisições realizadas via API.

5. Considerações Finais

Uma constatação relevante é a observação de que a maioria dos estudos acerca do uso de machine learning para aprimorar a qualidade de código está centrada na detecção de code smells. Tal foco cria a percepção de uma verdadeira corrida tecnológica, onde a substituição das ferramentas existentes é encarada como um imperativo na busca pela eficiência e precisão na identificação desses problemas.

Contudo, ao chegarmos a este ponto, é crucial ressaltar a perspectiva mais ampla que se delineou: os modelos de machine learning não devem ser meramente vistos como

substitutos, mas sim como aliados capazes de transcender a simples detecção de irregularidades no código-fonte. A capacidade desses modelos de não apenas identificar code smells, mas também analisar o contexto do código e oferecer sugestões práticas de melhoria, revela um potencial transformador para a prática de desenvolvimento de software.

O entendimento de que os modelos de machine learning podem contribuir significativamente não apenas na etapa de identificação, mas também na interpretação e sugestão de melhorias práticas é uma virada crucial. Isso sugere uma evolução na abordagem, elevando a automação no desenvolvimento de software a um patamar mais avançado e inteligente.

Nesse contexto, ao considerarmos possíveis linhas de pesquisa futuras, emerge a proposta instigante de explorar o uso de machine learning na análise dos dados retornados pelas ferramentas de análise de código. Esse enfoque visa não apenas apresentar as informações de maneira clara e compreensível, mas também sugerir melhorias práticas de forma acessível. Esta área, ainda pouco explorada no momento, promete abrir novos horizontes e oferecer insights valiosos para aprimorar a eficácia do desenvolvimento de software.

Por se tartar apenas de um protóripo, o sistema criado não está disponível para usuários finais, pois ainda é necessário não só polimento nas questões de usabilidade, mas a ampliação das funcionalidades de maneira geral, por exemplo: o processamento de outros tipos de smells, a possibilidade de criar plugins para outras ferramentas, a possibilidade de criar plugins para outros modelo de Machine Learning (ML) e etc.

Referências

Alazba, A., Aljamaan, H., Alshayeb, M.: Deep learning approaches for bad smell detection: a systematic literature review. Empir. Softw. Eng. 28, 77 (2023).

Fowler, M., Beck, K., 2019. Refactoring: Improving the Design of Existing Code - Second Edition. Pearson.

Gerhardt, T. E.; Silveira, D. T. Métodos de Pesquisa. 1. ed. Porto Alegre: Editora da UFRGS, 2009. p. 120.

Gil, A. Como elaborar projetos de pesquisa. Atlas: São Paulo, 2007.

Lacerda, Guilherme; Petrillo, Fabio; Pimenta, Marcelo; Guéhéneuc, Yann Gaël. Code smells and refactoring: A tertiary systematic review of challenges and observations, Journal of Systems and Software, set. 2020. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121220300881/pdfft?md5=f259cb697bd1c02c96ae39cd7036fb65&pid=1-s2.0-S0164121220300881-main.pdf>. Acesso em: 30 out. 2023.

Lei, M., Li, H., Li, J., Aundhkar, N., Kim, D.-K.: Deep learning application on code clone detection: a review of current knowledge. J. Syst. Softw. 184, 111141 (2022).

Malhotra, Ruchika, Bhawna Jain e Marouane Kessentini. Examining Deep Learning's Capability to Spot Code Smells: A Systematic Literature Review. Cluster Computing, v. 26, n. 6, p. 3473-3501, 2023. Disponível em: <https://link.springer.com/content/pdf/10.1007/s10586-023-04144-1.pdf>. Acesso em: 30 out. 2023.

Shahidi, Mahnoosh; ASHTIANI, Mehrdad; Nasrabadi, Morteza Zakeri. An automated extract method refactoring approach to correct the long method code smell, Journal of Systems and Software, maio 2022. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121222000048/pdfft?md5=d198159d89f27bc4e369f4d029e701ac&pid=1-s2.0-S0164121222000048-main.pdf>. Acesso em: 30 out. 2023.

Silva, Robson Keemps da. SmellGuru: A machine learning-based approach to predict design problems, Repositório Digital da Biblioteca da Unisinos, 9 set. 2022. Disponível em: <http://repositorio.jesuita.org.br/bitstream/handle/UNISINOS/12013/Robson%20Keemps%20da%20Silva_PROTEGIDO.pdf>. Acesso em: 30 out. 2023.

APÊNDICES

Repositório: https://github.com/henriquegranatto/intellicode.git

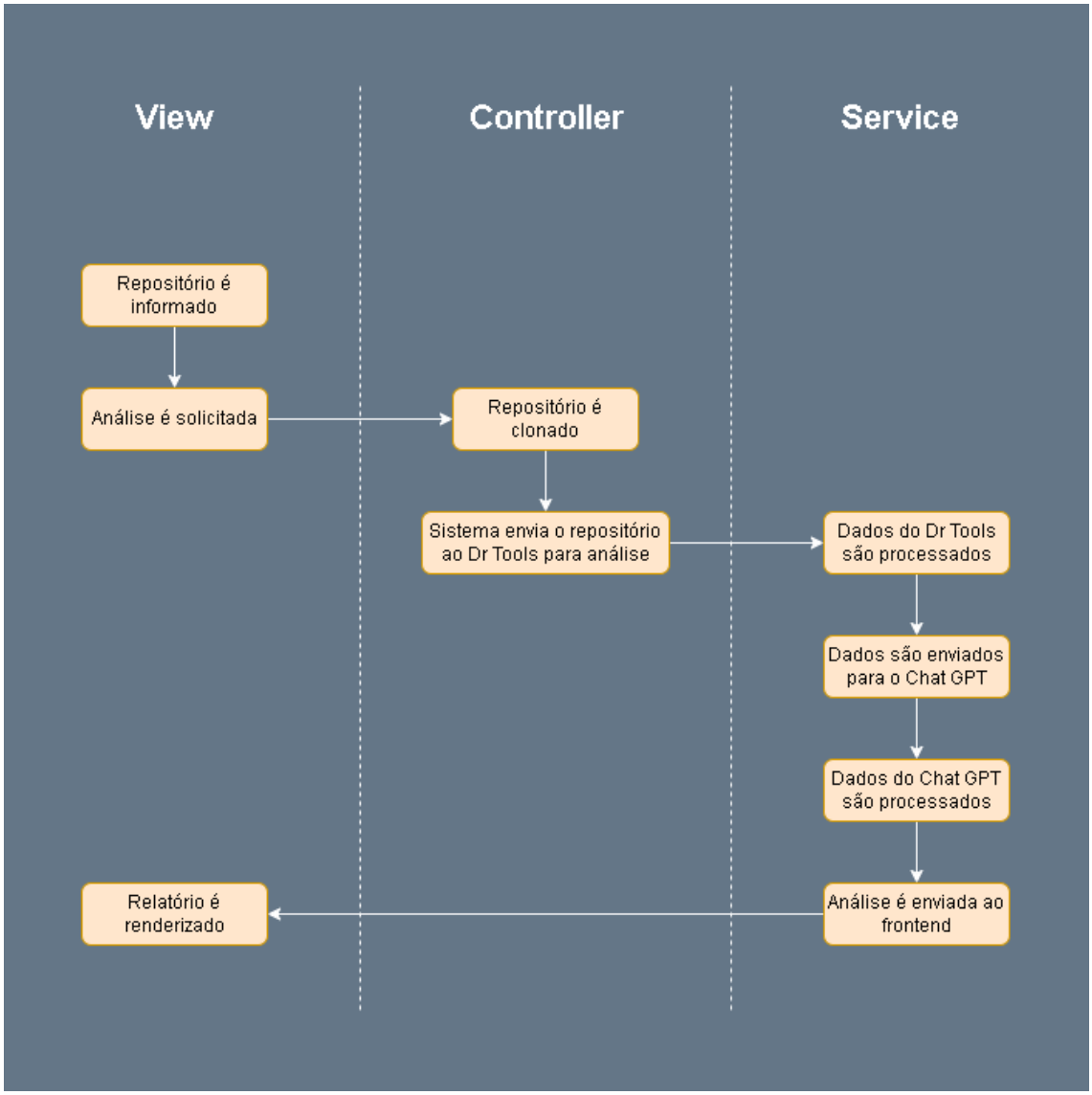

Figure 1. Fluxograma do funcionamento do sistema.

Figure 2. Página inicial.

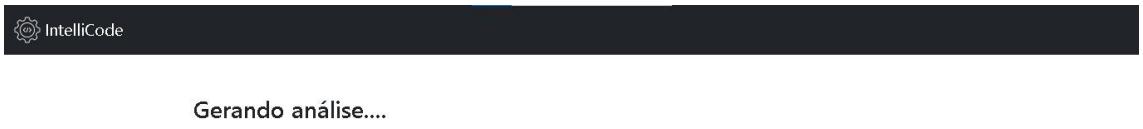

Figure 3. Notificação sobre o status da análise.

IntelliCode

repository/src/com/sistemabancario/Main.java:55 - PopulaContas()

Método Longo

O smell do "Método Longo" é identificado durante a análise de software e refere-se à presença de métodos extensos e complexos em um programa de computador. Esse problema ocorre quando um método se torna excessivamente longo, dificultando a compreensão, manutenção e teste do código. As razões para esse problema surgirem incluem a falta de modularidade, acúmulo de lógica em um único ponto e a violação dos princípios de coesão e baixo acoplamento. O tratamento efetivo para esse smell envolve refatorar o código, dividindo o método longo em partes menores e mais especializadas. Isso não apenas melhora a legibilidade do código, mas também facilita a reutilização e manutenção. As consequências dos métodos longos podem incluir dificuldades na identificação de bugs, aumento do esforço de desenvolvimento e redução da flexibilidade do software para alterações futuras. Portanto, identificar e abordar proativamente o smell do "Método Longo" é fundamental para garantir a qualidade e sustentabilidade do código-fonte.

Método Complexo

O smell do "Método Complexo" ocorre quando um método em um programa de computador se torna excessivamente complexo e intrincado. Esse smell surge devido à presença de lógica complexa, aninhamento excessivo ou uma combinação de múltiplas responsabilidades dentro de um único método. As razões para esse problema surgirem incluem a falta de decomposição de tarefas em partes menores e mais gerenciáveis, além da falta de aplicação de princípios de design como o princípio da responsabilidade única. O tratamento efetivo para o smell do "Método Complexo" envolve a aplicação de técnicas de refatoração, como a extração de métodos ou a reorganização da lógica para reduzir a complexidade. Ao simplificar a estrutura do método, a legibilidade do código é melhorada, tornando-o mais fácil de entender e manter. As consequências desse smell incluem dificuldades na detecção de erros, aumento da complexidade cognitiva para os desenvolvedores e redução da adaptabilidade do código a alterações futuras. Portanto, é crucial abordar proativamente o smell do "Método Complexo" para promover a manutenibilidade e a escalabilidade do software.

Figure 4. Explicação sobre os smells.

Análise do Código

O código citado apresenta tanto o smell do "Método Longo" quanto o smell do "Método Complexo". Isso ocorre porque o método `PopulaContas` é extenso e contém lógica complexa que poderia ser decomposta em partes menores e mais especializadas.

Versão Resolvida

Aqui está uma versão do código citado que resolve esses dois smells:

```
public static void PopulaContas() throws URISyntaxException {
    BancoServicos bancoServicos = new BancoServicos(bancos);
    PessoaServicos pessoaServicos = new PessoaServicos(pessoas);
    ContasServicos contasServicos = new ContasServicos(bancos, pessoas);
    Path caminhoDoArquivo = Path.of(Main.class.getResource(Path.of("dados", "contas.csv").toString()).toURI()).t

    List<String[]> csvData = Helpers.leCSV(caminhoDoArquivo);
    for (String[] colunas : csvData) {
        Pessoa pessoa = pessoaServicos.buscarPorCPF(colunas[1]);
        Banco banco = bancoServicos.buscarPorNumero(Integer.parseInt(colunas[2]));
        int numero = Integer.parseInt(colunas[3]);
        double saldo = Double.parseDouble(colunas[4]);
        String senha = colunas[5];

        if (pessoa == null || banco == null) {
            continue;
        }

        if (colunas[0].equalsIgnoreCase("poupança")) {
            double rendimento = Double.parseDouble(colunas[7]);
            contasServicos.inserirNovo(new ContaPoupanca(pessoa, banco, numero, senha, saldo, rendimento));
        } else if (colunas[0].equalsIgnoreCase("corrente")) {
            double taxaMensal = Double.parseDouble(colunas[6]);
            contasServicos.inserirNovo(new ContaCorrente(pessoa, banco, numero, senha, saldo, taxaMensal));
        }
    }
```

Figure 4. Análise do código e melhorias.