Um Modelo de Machine Learning para Identificar Long Methods e Feature Envy em Repositórios de Código Aberto

PUC-Rio

Departamento de Informática
Pós-Graduação em Informática

Professores

Juliana Alves Pereira
Clarisse Sieckenius de Souza

Aluno

Vinícius Souza Martins- 2211658

Turma

2023.2

Data de Entrega

09 de fevereiro de 2024

1
Introdução

A importância de práticas de design de qualidade no desenvolvimento de software é amplamente reconhecida. Conforme descrito por Fowler *et al.* (FOWLER et al., 1999), o conceito de "code smells" refere-se a indicações de possíveis problemas no design ou implementação do código, sugerindo que a refatoração pode ser benéfica para tornar o software mais compreensível e fácil de manter. A refatoração é o processo de melhoria da estrutura do software sem alterar seu comportamento, sendo cada "code smell" associado a um tipo específico de elemento do sistema, como classes ou métodos.

Existem ferramentas no mercado desenvolvidas com o objetivo de detectar code smells e auxiliar o desenvolvedor na refatoração do código. Grande parte dessas ferramentas utlizam-se de métricas para realizar detecções e classificar o tipo de smell. Buscando uma nova abordagem, nossa equipe se baseou em uma pesquisa realizada por desenvolvedores, incluindo a aluna da PUC-Rio Joanne Carneiro da Silva Ribeiro, que ainda não foi publicada. Utilizamos os resultados dessa pesquisa para desenvolver um modelo de Machine Learning que detecta dois tipos de code smells a nível de método, long method e feature envy, em projetos open-source do GitHub que utilizam a linguagem Java.

O emprego desses modelos de Machine Learning contribuem para que desenvolvedores e empresas possam alcançar uma maior precisão na identificação de code smells, ao invés de dependerem de softwares que rotulam code smells baseando-se em métricas, que ferramentas tradicionais podem nem sempre capturar a complexidade e o contexto do código.

2
Estudo Original

O estudo original tem como objetivo analizar a qualidade do dataset MLCQ desenvolvido por Lech Madeyski e Tomasz Lewowski (MADEYSKI; LEWOWSKI, 2020). Este dataset contém 4.770 trechos de código java extraídos com auxílio de uma ferramenta desenvolvida pela empresa code quest de 523 projetos open-source obtidos através do GitHub. Para a análise destes trechos de código, 26 desenvolvedores contribuíram com suas análises, classificando os code smells, e, ao final, foi contabilizado 14.739 análises. Os códigos foram classificados automaticamente pela ferramenta que extraiu os trechos de código de acordo com os code smells:

- Data class
- Blob
- Feature Envy
- Long Method

Os desenvolvedores então classificaram os trechos de código conforme a severidade de acordo com a sua experiência:

- Crítical
- Major
- Minor
- None (indica que o desenvolvedor não reconhece o code smell)

O desenvolvimento do dataset também contou com um questionário detalhado que os desenvolvedores preencheram sobre o seu passado relacionado com seu conhecimento em desenvolvimento de software. Com base neste questionário, optou-se por dividir os desenvolvedores em 4 grupos conforme o seu nível de experiência:

- Junior: até 3 anos

– Intermediário: entre 3 e 8 anos
– Sênior: mais de 8 anos

Para confirmar o code smells a serem incluídos no estudo, optou-se por utilizar somente trechos de código classificados com 2 mais classificações semelhantes entre os desenvovledores.

Com esses dados já obtidos do dataset, os autores então utilizaram-se de ferramentas para extrair métricas e classificar code smells. As ferramentas empregadas no estudo tiveram objetivos específicos, conforme detalhado a seguir:

– CK e Organic: Extrair métricas de nível de classe e método
– Organic: classificação de 4 code smells (long method, feature envy, data class e blob)
– PMD: classificação de 3 code smells (long method, data class e blob)
– Designate: classificação de 2 code smells (long method e feature envy)

Em posse de todos os dados, foi realizada as seguintes etapas:

– limpeza e preparação dos dados
– análise exploratória dos dados
– feature engineering (remoção de duplicatas, colunas irrelevantes, análise de correlação e remoção de características altamente correlacionadas)
– setup do ambiente para modelagem
– treinamento e comparação entre modelos de machine learning
– ajuste fino dos principais 5 modelos de machine learning
– identificação das features mais relevantes para 4 modelos (SVM, Linear Regression, Decision Tree e Random Forest)
– Criação de um modelo de conjuntos
– Avaliação do modelo de conjuntos
– Salvamento do modelo

Para o nosso trabalho, olhamos para o item: Identificação das features mais relevantes para 4 modelos (SVM, Linear Regression, Decision Tree e Random Forest). Com base nos resultados apresentados, realizamos o treinamento focado nestas features, para gerar um modelo para que outros desenvolvedores possam aplicar em repositórios de github.

3
Sobre o Modelo

Com base no objetivo mencionado no capítulo anterior, podemos elencar algumas propriedades do modelo.

3.1
Requisitos funcionais

Os requisitos funcionais são os requisitos que definem o que o modelo de machine learning deve fazer. Os requisitos funcionais do modelo são:

1. O modelo deve ser capaz de prever a presença de dois tipos de code smells: long method e feature envy;
2. O modelo deve gerar um csv contendo as predições.

3.2
Requisitos não funcionais

Os requisitos não funcionais definem como o sistema opera ao nível de serviço e qualidade. Os requisitos funcionais são:

1. O modelo deve ser capaz de processar 100MB de dados em menos de 5 minutos;
2. O modelo, contendo todas os dados necessários para sua utilização, deve ser capaz de rodar em sistemas Windows e MacOS.
3. O modelo deve ter instruções para a preparação do ambiente, coleta dos dados, unificação dos dados e sua utilização.

3.3
Atores

Podemos identificar os seguintes atores para interagir com o modelo:

- Desenvolvedores que atuam em projetos open-soruce no GitHub e buscam corrigir problemas de design e má implementação no código
- Pesquisadores na área de Machine Learning
- Pesquisadores na área de refatoração

4
Criação do Modelo

O objetivo é disponibilizar uma metodologia para que, ao final, o desenvolvedor possa utilizar o modelo treinado, para detectar code smells (feature envy e long method) em projetos open-source no github. Mas, caso o desenvolvedor queira replicar o treinamento ou explorar os dados e fazer um novo modelo, os dados estão disponíveis, assim como a criação do modelo. O modelo é desenvolvido na linguagem Python.

Para ser possível a criação do modelo e da metodologia, o projeto contou com as fases da figura 4.1.

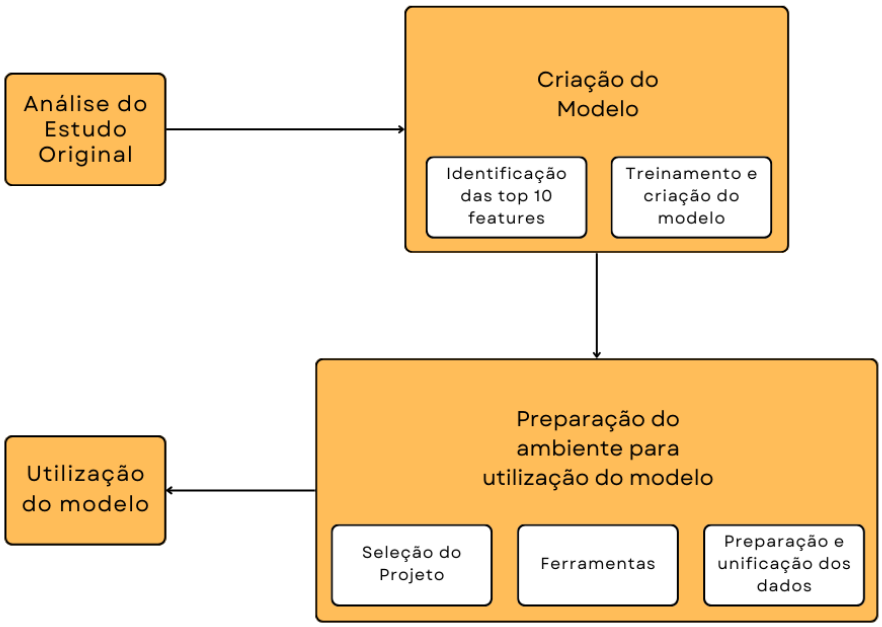

Figure 4.1: Fases do projeto

4.1
Top 10 features identificadas e utilizadas

Com base no estudo original, foram identificadas as top 10 features para cada tipo de code smell (feature envy e long method) de acordo com seu nível de importância (consulte as FIGURAS 7.1 e 7.2 no anexo para mais detalhes). Uma das features encontrada entre o top 10 é "agreemnent_experts" que corresponde a concordância entre desenvolvedores que analisaram os trechos de código no desenvolvimento do dataset MLCQ. Como nosso objetivo é analisar um novo projeto, removemos essa feature do modelo. Assim, focaremos nas outras 9 features.

As features utilizadas para o code smell feature envy foram:

- "smell_Designite_num_aglomeration": Total de code smells detectado pelo Designate
- "smell_PMD_num_agglomeration": Total de code smells detectado pelo PMD
- "LOC": Linhas de código do projeto
- "Stars": Número de estrelas do projeto
- "Number_of_Contributors": Número de contrubuidores do projeto
- "Commits": Número de commits do projeto
- "smell_Organic_featureenvy": Organic detectou ou não o code smell feature envy
- "smell_Designite_agglomeration": Designate detectou 2 ou mais code smells
- "smell_Designite_longmethod": Designate detectou ou não o code smell long method

As features utilizadas para o code smell long method foram:

- "smell_Designite_num_aglomeration": Total de code smells detectado pelo Designate

- "smell_PMD_num_agglomeration": Total de code smells detectado pelo PMD
- "LOC": Linhas de código do projeto
- "Stars": Número de estrelas do projeto
- "Number_of_Contributors": Número de contribuidores do projeto
- "Commits": Número de commits do projeto
- "smell_Organic_longmethod": Ferramenta Organic detectou ou não o code smell long method
- "smell_Designite_longmethod": Ferramenta Designate detectou ou não o code smell long method
- "smell_PMD_longmethod": Ferramenta PMD detectou ou não o code smell long method

Com as features selecionadas, podemos treinar e salvar o modelo.

4.2 Treinamento e criação do modelo

Para treinar o modelo foi utilizado o "out.csv", material do estudo original contendo todas as features extraidas das ferramentas já mencionadas.

4.2.1 Bibliotecas

Para treinar foram necessárias a seguintes bibliotecas:

- numpy
- pandas
- matplotlib
- pycaret
- os
- optuna

4.2.2
Preparação dos dados

Para a preparação dos dados, é realizada a leitura e análise inicial dos dados, verificando o seu formato e identificando linhas duplicadas para remoção. Logo em seguida é feita a seleção de features conforme o modelo específico (long method ou feature envy), e ajustamos a coluna alvo (longmethod_label) para o final do dataframe.

4.2.3
Configuração do ambiente, treinamento, ajuste fino, modelo de conjuntos

Após a preparação dos dados, realizamos as seguintes etapas:

- Utilização do PyCaret para treinar diversos modelos e selecionar os cinco melhores.
- Ajuste fino dos cinco melhores modelos usando a biblioteca optuna.
- Criação de um modelo de conjuntos.

4.2.4
Modelo finalizado

Após realizar todas as etapas, o modelo que melhor apresentou resultado é salvo para uso futuro.

5
Manual do Usuário

5.1
Preparação do ambiente para utilização do modelo

Para que o usuário possa utilizar o modelo, é necessário que selecione qual projeto no GitHub gostaria de analisar, baixar o repositório e utilizar as ferramentas para extração dos dados.

5.1.1
Seleção do projeto no GitHub e extração dos dados do projeto

Para utilizar o modelo é necessário que o usuário selecione um projeto desenvolvido em java no GitHub e faça o download do repositório.

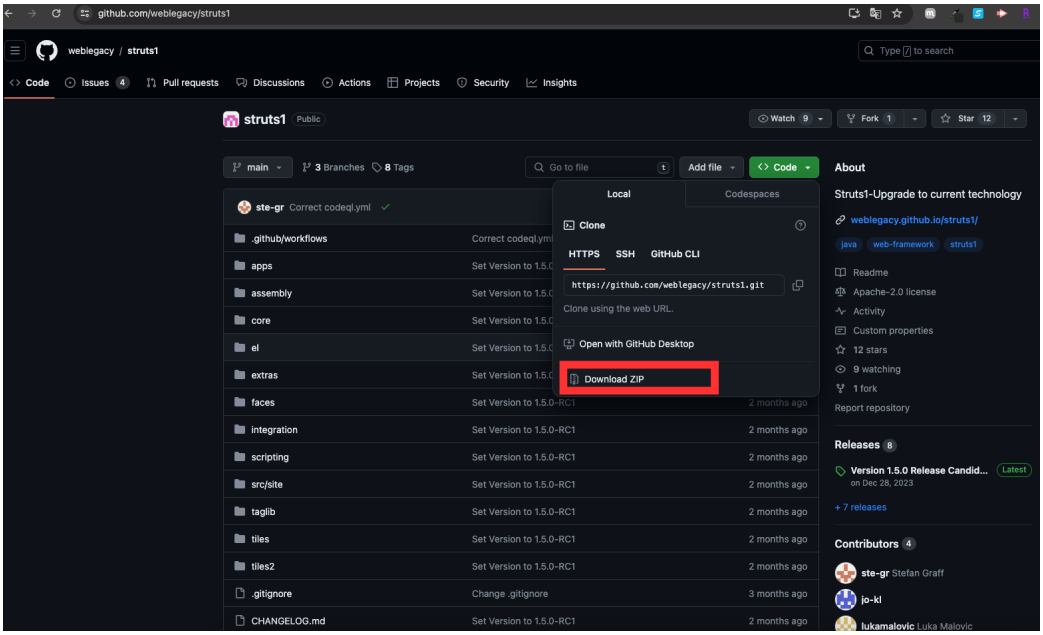

Figure 5.1: Local para download do repositório do GitHub

Abra o código "get_repository_info.py" e insira o endereço do repositório e o token da sua conta no GitHub.

Figure 5.2: Local onde deve ser inserido o token a url do repositório pelo usuário

Execute o código e o arquivo "repository_info.csv" será criado. Ele contém os dados para a extração das seguintes features: commits, stars, LOC e number_of_contributors.

5.1.2 Ferramentas

Antes da utilização das ferramentas, o usuário deve copiar todos os arquivos .java para uma pasta única. O sistema operancional para rodar as ferramentas é o Windows.

5.1.2.1 PMD

Para utilizar a ferramenta PMD e obter os dados, o usuário deve fazer o download através do site https://pmd.github.io/ e, em seguida, realizar as seguintes etapas:

- Extair o arquivo para, por exemplo, C:\pmd-bin-7.0.0-rc4
- Executar a linha de comando:
- "<caminho_para_pmd.bat_pasta_bin_na_pasta_raiz_pmd> check -d <caminho_para_a_pasta_com_os_códigos> -R rulesets/java/design.xml -f csv -r <caminho_do_arquivo_csv/report.csv>"

Após executar a linha de comando, o arquivo "report.csv" é criado. Ele contém os dados para a extração das seguinte features: smell_PMD_num_agglomeration e smell_PMD_longmethod.

5.1.2.2 Organic

Para utilizar a ferramenta Organic e obter os dados, o usuário deve acessar o repositório do OPUS através do link https://github.com/opus-research/organic e baixar o arquivo .jar.

Para executar a ferramenta, deve executar o seguinte comando:

- "java -jar <caminho_para_o_organic-v0.1.2.jar> -src <CAMINHO_PROS_ARQUIVOS_DE_CODIGO> -sf output.json"

O arquivo "output.json" é criado. Ele contém os dados para a extração das seguintes features: smell_Organic_featureenvy e smell_Organic_longmethod.

5.1.2.3 Desginate

A ferramenta Designate apresenta diversas versões, a necessária para obter os dados é a DesignateJava e pode ser obtida através do endereço https://www.designite-tools.com/.

Com o DesignateJava.jar, execute o seguinte comando:

- "java -jar <caminho_para_o_DesigniteJava.jar> -i <caminho_com_os_arquivos> -o <caminho_saida_com_os_smells>"

O Designate gera diversos csv, mas, para o treinamento do modelo, somente foi utilizado os seguintes: ImplementationSmells.csv e DesignSmells.csv.

Através desses dois csvs, iremos obter os dados para a extração das seguintes features: smell_Designite_longmethod, smell_Designite_num_aglomeration e smell_Designite_agglomeration.

5.1.3 Preparação e unificação dos dados

Em posse de todos os dados gerados pelas ferramentas e scripts, crie uma pasta com todos as saídas obtidas para que os dados sejam unificados de maneira que o modelo possa ser aplicado.

Para fazer a unificação dos dados, utilizamos o script "merge.py".

Rode o script "merge.py" e teremos a saída do arquivo "result.csv" com os dados unificados com todas as features necessárias para aplicação do modelo.

Caso o usuário tenha alterado algum nome das saídas geradas anteriormente, será necessário alterar o caminho no código.

5.2 Utilização do modelo

Para utilizar o modelo, abra o código "apply_model.py" e edite o nome do modelo para qual modelo vai utilizar (model_featureenvy3_f9.pkl ou model_method3_f9.pkl). Execute o código.

```python
import pandas as pd
import pandas as pd
from joblib import load

# Carregar os dados
data = pd.read_csv('result.csv')

# Nome do modelo
model_name = 'model_featureenvy3_f9.pkl'  # Substitua pelo nome do modelo desejado

# Selecionar as características com base no modelo
if model_name == 'model_longmethod3_f9.pkl':
    features = data[['smell_Designite_num_aglomeration', 'smell_PMD_num_aglomeration',
                     'LOC', 'Stars', 'Number_of_Contributors', 'Commits', 'smell_Organic_longmethod',
                     'smell_Designite_longmethod', 'smell_PMD_longmethod']]
elif model_name == 'model_featureenvy3_f9.pkl':
    features = data[['smell_Designite_num_aglomeration', 'smell_PMD_num_aglomeration',
                     'LOC', 'Stars', 'Number_of_Contributors', 'Commits', 'smell_Organic_featureenvy',
                     'smell_Designite_aglomeration', 'smell_Designite_longmethod']]

# Carregar o modelo
model = load(model_name)

# Fazer previsões
predictions = model.predict(features)

# Salvar os resultados
output = pd.DataFrame({'Type Name': data['Type Name'], 'Prediction': predictions})
output.to_csv('prediction_results.csv', index=False)
```

Figure 5.3: Script "apply_model.py"

Usamos novamente a biblioteca pandas para manipulação de dados e joblib para carregar nosso modelo treinado.

O modelo é carregado e faz as previsões, gerando um arquivo "prediction_results.csv" com a previsão se possui ou não o code smell para cada arquivo java.

5.3 Observação

Caso encontre dificuldades ao rodar os códigos, utilize o Google Colab[1].

[1] https://colab.research.google.com/

6
Cenários

Nessa seção, apresentamos: (i) 2 cenários distintos utilização do usuário com sucesso, e (ii) 2 cenários distintos que o usuário encontrou dificuldades na sua utilização.

6.1
Cenários de sucesso

- Um desenvolvedor atuante em um grande projeto no GitHub instalou corretamente as ferramentas, leu o manual, rodou o modelo e obteve a saída com as previsões do modelo, obtendo novas informações sobre code smells referente ao projeto que atua. Com essas informações, ele pode priorizar as partes do código a serem melhoradas, garantindo uma manutenibilidade e a qualidade geral do software.
- Um pesquisador atuante na área de Machine Learning replicou a criação do modelo, instalou as ferramentas, coletou e unificou os dados e rodou o modelo com o objetivo de comparar com outros modelos de machine learning treinados para detectar code smells. Com os resultados obtidos, o pesquisador pode então provar algumas hipóteses que tinha levantado sobre a acurácia e desempenho de diferentes modelos existentes na área.

6.2
Cenários com dificuldade

- Um desenvolvedor se interessou sobre o modelo e quis utilizá-lo para o seu pequeno código java. Ao tentar utilizar, descobriu que não é possível, pois para rodar o modelo é necessário ser um projeto no GitHub. O desenvolvedor não desistiu e criou uma conta no GitHub e criou o seu próprio projeto. Quando finalmente conseguiu utilizar o modelo, viu que os resultados não eram confiantes pois as features extraídas do repositório eram novas e geravam um bias nas previsões do modelo.

– Um desenvolvedor ao tentar rodar o script "merge.py" encontrou erros devido a problemas de biblioteca que o impediram de processar e unificar os dados para serem aplicados ao modelo. Analisando os erros que apareciam, precisou instalar versões mais antigas de bibliotecas para que, finalmente, conseguisse rodar o script.

7
Considerações finais

O modelo apresentado continua em desenvolvimento. Novos estudos estão sendo conduzidos para o seu aprimoramento, com adição de novas features através de dados obtidos de eye tracker, e buscando a facilidade da sua utilização, para que, em breve, seja utilizado de maneira simples pelo usuário.

Anexos

Encontra-se em anexo as 10 melhores features para os dois tipos de code smells que o modelo de machine learning foi desenvolvido.

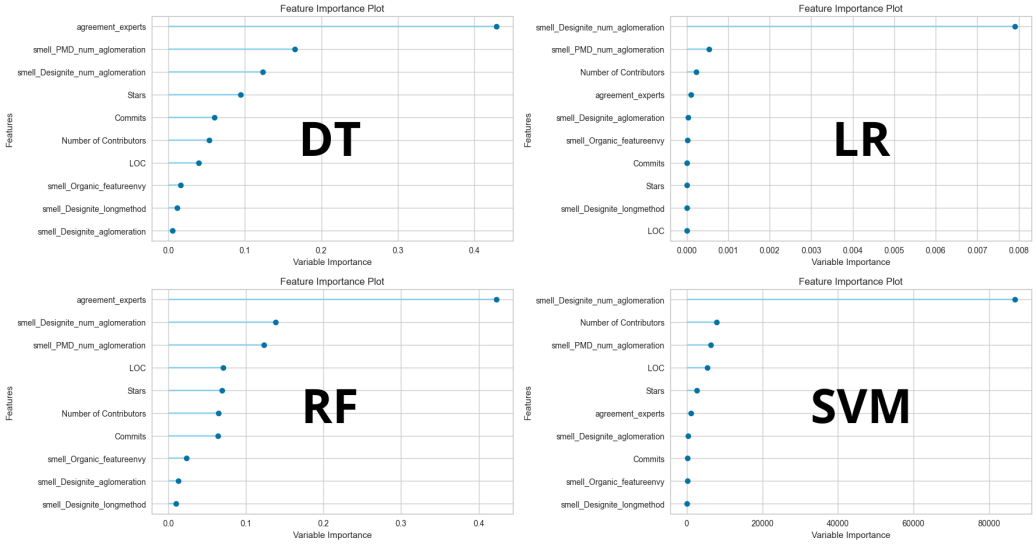

Figure 7.1: 10 melhores features para os modelos (decision tree, linear regression, SVM e random forest) aplicados ao code smell feature envy

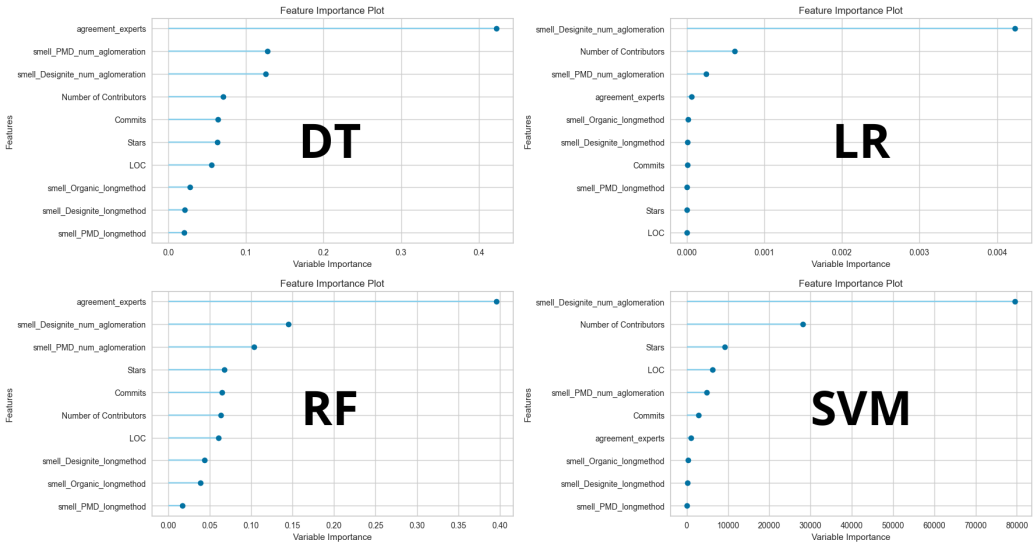

Figure 7.2: 10 melhores features para os modelos (decison tree, linear regression, SVM e randon forest) aplicados ao code smell long method

8
Bibliography

FOWLER, M. et al. Refactoring: Improving the design of existing code addison-wesley professional. **Berkeley, CA, USA**, 1999. Cited in page 1.

MADEYSKI, L.; LEWOWSKI, T. Mlcq: Industry-relevant code smell data set. In: **Proceedings of the Evaluation and Assessment in Software Engineering**. [S.l.: s.n.], 2020. p. 342–347. Cited in page 2.