UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA
Autorizada pelo Decreto Federal nº 77.496 de 27/04/76
Recredenciamento pelo Decreto n°17.228 de 25/11/2016
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
COORDENAÇÃO DE INICIAÇÃO CIENTÍFICA

XXVI SEMINÁRIO DE INICIAÇÃO CIENTÍFICA DA UEFS
SEMANA NACIONAL DE CIÊNCIA E TECNOLOGIA - 2022

Investigando o impacto de anomalias de código nas práticas do desenvolvimento de software.

Daniel Alves Costa[1]; Jose Amancio Macedo Santos [2]

1. Bolsista PROBIC/UEFS, Graduando em Engenharia de Computação, Universidade Estadual de Feira de Santana, e-mail: dancostafsa@hotmail.com
2. Orientador, Departamento de Tecnologia, Universidade Estadual de Feira de Santana, e-mail: zeamancio@uefs.br

PALAVRAS-CHAVE: Anomalia de código; repositório de software; projeto de software; atributos de software.

INTRODUÇÃO

Com os estudos realizados na área de Engenharia de Software foi visto a necessidade de identificar anomalias presentes nos códigos com o intuito de melhorar o software estudado evitando possíveis erros futuros e grandes dificuldades para manutenção do mesmo quando este estiver em uso (Fowler, 2018).

Nessa necessidade de procurar deixar o código o mais fácil de entender e buscando evitar repetições e prejuízos futuros, a identificação desses erros possui grande relevância, visto que esses problemas afetam a segurança, a integridade, a manutenção, o valor e o desempenho dos softwares que apresentam qualquer um dos tipos de imperfeições.

Essas imperfeições podem ser dadas como code smells. Esses smells são representados como “cheiros” que podem tornar o software suscetível a necessidade de uma refatoração, e quando não feita prejudica a integridade do projeto. Dentro desses smells possuem aqueles considerados “cheiros ruins” os bad smells, onde estes comprometem negativamente o aplicativo tornando necessário muitas correções para o bom funcionamento (Fowler, 2018)(Lanza, 2007).

A partir disto é possível perceber a grande necessidade de se identificar esses code smells para a redução de possíveis danos para as empresas. Logo, este trabalho tem como objetivo buscar a identificação dessas anomalias, utilizando de fatores contextuais já presentes na literatura para relacionar com code smells identificados em diversos softwares de código aberto, para com isso verificar se fatores contextuais podem impactar na incidência de code smells no software.

MATERIAL E MÉTODOS OU METODOLOGIA (ou equivalente)

Em busca dos softwares, na linguagem Java, para ser realizado a investigação de code smells neles, foi utilizado o dataset do GHTorrent (Gousios, 2013), sendo este um projeto de coleta de dados de repositórios públicos disponíveis na hospedagem Github. Com o auxilio do framework Python PyDriller (Spadini, 2018) foi feito um script para a coleta das variáveis de contexto, ele realiza uma mineração de dados em repositórios do Git. Com isso foram extraídas as variáveis:

Tamanho do sistema (loc): total de linhas presentes no software.
Número de mudanças: total de commits realizados no repositório.
Número de contribuidores: total de desenvolvedores que realizaram commits.
Tempo de desenvolvimento: tempo calculado entre o primeiro e o ultimo commit.

Com essas variáveis foi realizada uma distribuição em forma de quartil para cada uma, em função de somente pegar os softwares necessários para a analise foram desconsiderados dois desses quartis, o primeiro devido aos seus baixos valores presentes na distribuição e o terceiro buscando uma distância entre os sistemas presentes no segundo e no último quartil. A partir dessa filtragem foi classificado como inferior os softwares presentes no segundo quartil e como superior os que se encontravam no último quartil.
Com esses fatores contextuais foi realizado associações entre eles, onde pode conter um, dois, três ou os quatro fatores alternando entre si. Utilizando do agrupamento em que os quatro fatores estão presentes, foi criado contextos com ele onde se alterava quando o fator contextual estar inferior e quando ele estar superior como mostrado na Tabela 1, sendo os softwares classificados em algum desses contextos de acordo com suas características.

Tabela 1. Grupo de contextos criados quando alternados os quatro fatores contextuais.

CONTEXTO	Linhas de código	Número de Commits	Número de Contribuidores	Tempo de desenvolvimento
Contexto 1	INF	INF	INF	INF
Contexto 2	INF	INF	INF	SUP
Contexto 3	INF	INF	SUP	INF
Contexto 4	INF	INF	SUP	SUP
Contexto 5	INF	SUP	INF	INF
Contexto 6	INF	SUP	INF	SUP
Contexto 7	INF	SUP	SUP	INF
Contexto 8	INF	SUP	SUP	SUP
Contexto 9	SUP	INF	INF	INF
Contexto 10	SUP	INF	INF	SUP
Contexto 11	SUP	INF	SUP	INF
Contexto 12	SUP	INF	SUP	SUP
Contexto 13	SUP	SUP	INF	INF
Contexto 14	SUP	SUP	INF	SUP
Contexto 15	SUP	SUP	SUP	INF

Contexto 16	SUP	SUP	SUP	SUP

Legenda: INF – Inferior, SUP – Superior.

Para a extração de code smell presentes nos softwares foi utilizado a ferramenta de mineração de repositórios de software Repository Miner (Mendes, 2017). Com ele foi possível a extração de 7 tipos de code smell diferentes: god class, data class, brain class, brain method, complex method, featury envy e long method. Esses sete tipos de smells foram considerados para o estudo.

RESULTADOS E/OU DISCUSSÃO (ou Análise e discussão dos resultados)

Buscando identificar o padrão dos dados coletados dos softwares foi aplicado o teste de normalidade de Shapiro-Wilk, esse teste foi escolhido por ser um dos mais poderosos para esta verificação (Yap, 2011). Com o resultado do teste foi mostrado que os dados não seguem uma distribuição normal. Devido a esse resultado foi utilizado o teste de hipótese de Mann-Whitney, esse teste é utilizado para determinar se duas amostras independente pertencem à mesma população, verificando se existe igualdade entre as medianas (Kothari, 2004), e por ser um teste não-paramétrico, ele não necessita de distribuição normal, indo de acordo com os dados que estão sendo utilizados.
Para o teste de Mann-Whitney se assumiu duas hipóteses que foram testadas buscando o resultado para o experimento. As hipóteses adotadas foram as seguintes:

- Hipótese Nula: Sistemas com tempo de desenvolvimento superior e inferior não apresentam diferença na distribuição do percentual do code smell.
- Hipótese Alternativa: Sistemas com tempo de desenvolvimento superior e inferior apresentam diferença na distribuição do percentual do code smell.

Com as hipóteses formuladas os sistemas foram submetidos ao teste de acordo com sua classificação mostrada na Tabela 1. Os principais resultados encontrados para os code smells estudados foram:

- Brain Method: Nesse smell a quantidade de linha de código e o número de commits possuem impacto significativo para sua incidência.
- Brain Class: Nesse smell a quantidade de linha de código e o tempo de desenvolvimento possuem impacto significativo para sua incidência.
- Complex Method: Nesse smell nenhuma variável de contexto possui impacto significativo para sua incidência.
- God Class: Nesse smell a quantidade de linha de código, o número de commits e o tempo de desenvolvimento possuem impacto significativo para sua incidência.
- Long Method: Nesse smell a quantidade de linha de código possui impacto significativo para sua incidência.
- Data Class: Nesse smell a quantidade de linha de código possui impacto significativo para sua incidência.
- Feature Envy: Nesse smell a quantidade de linha de código e o tempo de desenvolvimento possuem impacto significativo para sua incidência.

Esses resultados mostram que os fatores contextuais da quantidade de linhas de código, número de commits e o tempo de desenvolvimento do software possui uma grande relevância para a incidência de code smells nos softwares estudados.

CONSIDERAÇÕES FINAIS (ou Conclusão)

A partir desse estudo apresentado foi perceptível que os fatores contextuais estudados nesse projeto podem afetar negativamente o desenvolvimento de um software, podendo causar a aparição de anomalias, como é o caso dos code smells. Tendo como maior relevância os fatores de quantidade de linhas de código, o número de commits realizado no sistema e o tempo que levou para ele ser desenvolvido. Com isso mostra que estar atento a esses quesitos pode evitar que code smells apareçam em um sistema, fazendo com que o código seja mais "limpo", com um melhor desempenho e melhor aproveitado pelo cliente.

REFERÊNCIAS

FOWLER, M. 2018. Refactoring: improving the design of existing code. Addison-Wesley Professional.

LANZA, Michele; MARINESCU, Radu. Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media, 2007.

SPADINI, D. PyDriller documentation! Pydriller, 2018. Disponivel em: https://pydriller.readthedocs.io/en/latest/index.html#. Acesso em: 15 de agosto de 2022.

GOUSIOS, Georgios. The GHTorent dataset and tool suite. In: 2013 10th Working Conference on Mining Software Repositories (MSR). IEEE, 2013. p. 233-236.

MENDES, Thiago et al. RepositoryMiner-uma ferramenta extensível de mineração de repositórios de software para identificação automática de Dívida Técnica. CBSoft 2017-Sessao de Ferramentas (), p. 12, 2017.

YAP, Bee Wah; SIM, Chiaw Hock. Comparisons of various types of normality tests. Journal of Statistical Computation and Simulation, v. 81, n. 12, p. 2141-2155, 2011.

KOTHARI, Chakravanti Rajagopalachari. Research methodology: Methods and techniques. New Age International, 2004.