Análise da relação entre *code smells* e métricas qualitativas em repositórios de Sistemas de direção autônoma

Lorrayne Reis Silva [1]

[1]Bacharelado em Engenharia de Software ´

[2]Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

[3]Edifício Fernanda, Rua Claudio Manoel, 1.162, Funcionários
´ 30.140-100 — Belo Horizonte — MG — Brazil

lorrayne.silva.1220819@sga.pucminas.br

Abstract. *Autonomous driving systems have gained prominence in the recent industrial market, being precursors to the performance of autonomous vehicles that does not promote human interference while driving. Thus, repositories for simulating these are often used in the development and benchmarking of tests, but the provision of reliability in these simulations is directly related to qualitative factors. Given the scarcity of studies that delimit on the subject, the present work seeks to understand the qualitative relationships presented by this segment, through code smells and quality attributes. From the evaluation carried out in 1,000 Python language repositories, the classification of the main code smells found and the correlation of these with associated qualitative and realese metrics.*

Resumo. *Sistemas de direção autônoma tem ganhado destaque no mercado industrial recente, sendo percursores para atuação de veículos autônomos que fomentam uma condução sem interferência humana. Assim, repositórios para a simulação desses são frequentemente usados no desenvolvimento e aferição de testes, porém a provisão de confiabilidade nestas simulações esta diretamente relacionada a fatores qualitativos. Dado a escassez de estudos que delimitam sobre o assunto, o presente trabalho busca entender as relações qualitativas apresentadas por esse segmento, por meio de code smells e atributos de qualidade. A partir da avaliação realizada em 1.000 repositórios da linguagem Python, da classificação dos principais code smells encontrados e a correlação desses com métricas qualitativas e realeses associadas.*

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientadora de conteúdo (TCC I): Simone de Assis - simone@pucminas.br
Orientador acadêmico (TCC I): Laerte Xavier - laertexavier@pucminas.br
Orientador do TCC II: (A ser definido no próximo semestre)

Belo Horizonte, 15 de Novembro de 2022.

1. Introdução

A introdução de novas tecnologias aplicadas ao desenvolvimento de *softwares* para sistemas de condução automatizados (ADSs, do inglês *Automated Driving Systems*) proporcionou um novo patamar de direcionamento para a evolução de veículos autônomos (AV, do inglês *Autonomous Vehicles*). *Softwares* ADSs são frequentemente compostos de várias unidades funcionais, conhecidas como recursos ADS, que automatizam tarefas de direção específicas como frenagem de emergência, reconhecimento de sinais de trânsito e controle de cruzamento [Abdessalem et al. 2020]. Estes sistemas possuem aplicabilidade em vias públicas e são *softwares* complexos que devem atender a vários requisitos, como segurança, cumprimento das regras de trânsito e conforto [Luo et al. 2022], atreladamente a sua execução. Assim, como todo *software*, está suscetível a *bugs* e podem causar acidentes fatais.

Algumas abordagens foram propostas na literatura para resolver esta condição. Dentre elas, os resultados do artigo de Stocco et al. (2020) discorrem sobre a detecção de falhas atreladas a disponibilidade de um conjunto rotulado de erros. Por meio de um ambiente de simulação como fator de execução, empregam a classificação de erros para a geração de previsão e autorrecuperação de ADSs [Stocco et al. 2020]. Não obstante, Sharma and Kessentini (2021) apresentam um *dataset* identificativo de *code smells* relacionados a erros decorrentes de métricas de qualidade, coletadas de repositórios do *GitHub*. Assim, repositórios simuladores são frequentemente utilizados durante o desenvolvimento de AVs para previsão de erros de atuação, porém como são fontes *open source* podem não apresentar certo nível qualitativo que confere confiabilidade as simulações. **O problema a ser resolvido pontua-se na escassez informacional da qualidade de repositórios de *softwares open source* simuladores na promoção de *code smells*, que podem ocasionar em simulações de baixa confiabilidade**.

Com intuito de realizar buscas efetivas por *bugs*, Garcia and Chen (2020) realizaram análises em sistemas AVs através da coleta de 499 *bugs* por meio de *commits* de sistemas de *software* de código aberto. Além disso, demonstram que componentes de planejamento têm um alto número de *bugs* e exibem sintomas que são importantes para uma condução segura e correta de veículos [J. Garcia and Chen 2020]. Não obstante, Tang et al. (2021) por meio da avaliação de *issues* do *software open source* Openpilot, categorizaram *bugs* em relação a seu local de ocorrência no *software*, como *model bugs*, *plan/control bugs* e *UI bugs*. Promovendo uma resolução sobre recorrência de *bugs* que refletem na abertura de *issues* com características semelhantes [Tang et al. 2021]. Logo, resolver o problema deste trabalho é importante, pois outras abordagens da literatura não apontam a necessidade atencional da relação entre a *softwares* ADSs *open source* e a qualidade do código na promoção de *code smells* que podem contribuir para simulações falhas no desenvolvimento de AVs.

Dessa forma, **o objetivo geral do trabalho visa identificar o relacionamento entre atributos de qualidade de *softwares* simuladores *open source* ADSs na ocorrência de *code smells***. Para atingir esse objetivo foram estabelecidos três objetivos específicos: 1) Classificação de *code smells* frequentes em repositórios simuladores de ADSs, 2) Relação entre atributos internos de qualidade e a ocorrência de *code smells* classificados 3) Comparação e correlação da tendência quantitativa de *code smells* de acordo com a presença ou não de *releases* nestes repositórios.

Como resultado, espera-se pontuar uma taxonomia de *code smells* frequentemente atrelados a repositórios ADSs. Conjuntamente, a exposição de quais classificações estão diretamente relacionadas a um maior prejuízo para atributos internos de qualidade desses *softwares*. Ademais, procura-se avaliar a existência de uma associação de aumento ou decremento qualitativo relacionada a atuação de *code smells* em métricas qualitativas. Por fim, busca-se sinalizar a tendência de um maior número de *code smells* relacionados a repositórios que não possuem *releases*, em comparação aqueles que possuem e demonstrar a evolução numérica de *code smells* desses últimos no decorrer de *releases*.

Desse modo, as próximas seções do trabalho são segmentadas: em primeira instância, apresentação da Seção 2 que delimita a fundamentação teórica utilizada, em conseguinte a Seção 3, na qual são demonstrados artigos relacionados a temática trabalhada. E por fim, a Seção 4 que apresenta os materiais e métodos trabalhados para a resolução das questões definidas.

2. Fundamentação Teórica

Esta seção apresenta os principais conceitos e técnicas diretamente envolvidos na solução da problemática apresentada. São eles: qualidade de *software*, análise estática de *software*, *code smells* e Sistemas de direção autônoma.

2.1. Qualidade de *software*

Segundo Ian Sommerville (2011), a qualidade de *software* se estabelece principalmente em seu modo ocorrente de execução, estruturação e de organização. Tais características se refletem nos atributos de qualidade de um *software*, como : robustez, complexidade e modularidade. A partir da utilização de padrões durante o desenvolvimento, promovem propriedades determinísticas para o facilitamento de testes e manutenção em produtos gerados. Não obstante, diversos estudos discorrem sobre o relacionamento direto entre a qualidade apresentada por um *software* na criação de relações de dependência que podem fomentar a difusão de *bugs* [Chen et al. 2019].

2.2. Análise estática de *software*

Análises estáticas em *software* referem-se a uma avaliação em código fonte estático com o intuito de checar a existência de possíveis padrões errôneos, violações de programação e a identificação de paradigmas de qualidade em código. Os resultados apresentados por essas podem demonstrar o relacionamento direto entre o código desenvolvido e a qualidade externa demonstrada pelo *software* [Plosch et al. 2008]. Outrossim, tais análises podem apontar características qualitativas relacionadas a maior ou menor consumo de recursos na execução de programas [Chen et al. 2021]. Por fim, são recomendadas pela ISO 26262/2011 para reduzir *bugs* em tempos de execução, pertencente ao seguimento de padrões de segurança automotiva no desenvolvimento de *software* [Imparato et al. 2017].

2.3. *Code smells*

Code smells definem implementações ou escolhas de *design* realizados negativamente que impactam a evolução de *softwares*, contribuindo para reduzir a capacidade de programadores no entendimento e mantenimento de aplicações [Hamdi et al. 2021]. Os mesmos podem apontar problemas com a qualidade do desenvolvimento e concomitantemente contribuem para a identificação de possíveis erros, além da promoção de facilitamento da refatoração a ser realizada por uma equipe técnica [Sharma 2018]. A partir

da identificação de *code smells* e seu tratamento, é possível promover o aumento da qualidade do *software*, torná-lo mais fácil de manter e reduzir as chances de falhas em um sistema [Dewangan et al. 2021].

2.4. Sistemas de direção autônoma

Sistemas de direção autônoma (*ADSs*) são sistemas complexos que possuem segurança crítica diretamente relacionada ao atendimento de requisitos pré-estabelecidos [Luo et al. 2022]. Essa ocorrência pressupõe o recebimento de dados de vários sensores, analisados em tempo real por meio de redes neurais profundas. As mesmas são responsáveis por determinar parâmetros para o acionamento de atuadores de execução [Stocco et al. 2020]. Logo, os *softwares* desses sistemas são de fundamental importância para operação e sua composição decorre de várias unidades de componentes, que automatizam tarefas de direções específicas. Consequentemente, estão envolvidos diretamente na atuação e execução total de funcionamento, com a solicitação de recursos sendo realizada continuamente, além da promoção de abertura e recebimento de dados de componentes advindos de outras camadas [Abdessalem et al. 2020].

3. Trabalhos relacionados

Nesta seção, são apresentados os trabalhos relacionados que concatenam-se ao objeto de estudo do presente trabalho. Assim, para demonstrar a relação direta entre a co-ocorrência de *code smells* em valores qualitativos de um *software*, Martins et al. (2020) realizaram a avaliação de dois *softwares closed source* na decorrência de 11 *releases*, através da verificação de *code smells* independentes e do estabelecimento de relações de co-ocorrência. Assim, avaliaram métricas qualitativas como coesão, acoplamento e complexidade em decorrência anterior e posterior a remoção de *code smells* seletos. Como resultados relevantes ao presente estudo, pontua-se a promoção de *refactoring* desses com tendência a redução significativa da complexidade de um projeto e o aumento de valores de métricas como coesão e acoplamento [Martins et al. 2020a] . Logo, o trabalho se relaciona a este, na questão metodológica de verificação da relação entre *code smells* e atributos de qualidade.

Martins et al. (2020) a partir da análise de projetos de Linhas de Produtos de *Software* (*SPL*, do inglês *Software Product Line*). Na observância de funcionalidades desses *softwares*, postula sobre a interconexão existente entre *inter-smells* e a manutenibilidade desses sistemas. Por meio da utilização de dois SPLs, classificaram cinco *code smells* e realizam a definição de suas inter-relações. Nas quais posteriormente efetuaram refatoração e compararam *releases* antes e depois para verificar o impacto em relação a manutenibilidade utilizando métricas como Número de filhos (NOC, do inglês *Number of Children*)[Martins et al. 2020b]. Outrossim, os principais resultados do artigo, que servem como embasamento dessa discussão, se referem a ponderação de não ocorrência de impactos a qualidade e manutenibilidade com a presença de *inter-smells*. Por fim, a conexão com o atual trabalho concerne-se no mesmo teor avaliativo da decorrência de *smells* em relação a *releases*.

Em detrimento do esforço despendido na coleta de informações de códigos necessárias para análise investigativa por parte de pesquisadores, Sharma and Kessentini (2021) propõem um conjunto de dados denominado *QScore*. Mediante a coleta e

avaliação de qualidade de códigos de 86 mil repositórios de linguagens C e Java minerados do *Github*, com a aplicação de um índice de qualidade e de cálculos de classificação de *code smells*. Para o fomento de sete tipos de arquitetura de *architecture smells*, 20 tipos de *design smells* e 27 métricas de qualidade de código. As análises realizadas são importantes para este trabalho, na promoção de classificações relacionais entre *code smells* e qualidade de métricas [Sharma and Kessentini 2021]. Logo, o trabalho estabelece uma conexão na presente manifestação por meio da promoção de passos executivos no que tange a classificação de *code smells*.

Outro trabalho relacionado, com o intuito de entender *bugs* decorrentes de AVs Garcia and Chen(2020) por meio dos *softwares open source* Apollo e Autoware, realizaram a investigação de 16.851 *commits*, 499 *bugs* e posteriormente classificaram esses conforme ocorrência de operação em componentes diretos do *software*. Como resultado, obtém-se uma taxonomia de erros recorrentes que afetam diretamente a ação desses *softwares* em relação a controle, planejamento e localização. Onde obtiveram a identificação de 13 causas principais, 20 sintomas de *bugs* e 18 categorias de componentes que esses *bugs* geralmente influenciam [J. Garcia and Chen 2020]. Logo, a relação causal com o trabalho valida-se no estabelecimento da ideia principal no que tange a categorização de *code smells*.

Por fim, para fomentar estratégias para refatoração de códigos em *Python*, Chen et al.(2016) promovem a detecção de *code smells* e realizam a classificação dos tipos que possuem maior ocorrência em sistemas dessa linguagem, por meio da ferramenta de detecção de *code smells* denominada *Pysmell*. Os resultados apontam a identificação de 285 instâncias de *smells* e revela a predominância de maior ocorrência dos tipos *Large Class* e *Large Method* [Chen et al. 2016]. Logo, o trabalho relaciona-se a este no que delimita a utilização da mesma ferramenta e do mesmo intuito investigativo, através da verificação de *code smells* que ocorrem majoritariamente em *softwares Python* de *ADSs*.

4. Materiais e Métodos

Este trabalho é classificado como uma pesquisa quantitativa cujo objetivo é descritivo. Tal embasamento pode ser justificado dado a intenção desse em compreender o relacionamento existente entre atributos qualitativos de s*softwares* ADSs e a presença de *code smells* por meio da análise de dados obtidos através da mineração de repositórios no *Github*. Assim, serão levantados dados em relação à influência de *code smells* em atributos internos de qualidade e a evolução do número de *smells* na decorrência de *releases* em tais repositórios. Em conseguinte, essa seção apresenta estratégias execucionais, etapas e tecnologias utilizadas para a realização dos objetivos propostos.

4.1. Procedimentos

Com o objetivo de analisar repositórios relevantes escritos na linguagem *Python*, serão coletados 1.000 repositórios que contribuem com prerrogativas simulatórias para sistemas autônomos por meio do *GitHub*. A partir de *scripts* com a mesma linguagem, no ambiente de desenvolvimento *Pycharm*, os quais serão desenvolvidos para a coleta de repositórios, coleta de releses e geração de métricas. Os dados gerados serão armazenados no formato de arquivos CSV (do inglês, *comma separated values*) e posteriormente analizados atráves do *software* de visualização de dados *PowerBI*.

4.2. Coleta de repositórios

A coleção de repositórios acontecerá atráves da *API* (do inglês, *Application Programming Interface*) *GraphQL* do *Github*, devido a facilidade que essa promove no selecionamento de diversas informações, para a seleção de campos desejados. Tais repositórios serão selecionados por meio da definição do campo *topic* em *autonomous driving* e a definição da linguagem como *Python*. Não obstante, será também coletado o nome do repositório, sua *url* (do inglês, *Uniform Resource Locator*) e o número de *releases* que esse possui. Porteriormente, esses dados serão armazenados em um arquivo CSV e usados para a obtenção de *releases* de acordo com o nome do repositório.

4.3. Coleta de releases

A partir dos nomes de projetos coletados da etapa anterior será realizado em código uma verificação da presença de *releases* em cada repositório coletado, uma vez que nem todos possuem *releases*. Posteriormente em consulta ao CSV gerado, será realizada a coleta de releases decorrentes, por meio da *API REST* do *GitHub*, uma vez que que essa facilita a integração e automatiza os fluxos de trabalho exigidos.

4.4. Aferimento de code smells e métricas

Para a realização do aferimento investigativo de *code smells* será utilizado a ferramenta *Pysmell*, dado que essa promove a classificação de *smells* direcionados a linguagem *Python* e é fácil de ser manuzeada, bastando uma simples configuração. No que tange o aferimento de métricas, com o intuito de coletar diferentes métricas qualitativas serão empregadas duas ferramentas, sendo elas : as bibliotecas *Multimetric* e *Radon* do *Python*. Ambas oferecem uma variedade de métricas para cálculo, demonstradas nas Tabelas 1 e 2 e entre elas as métricas *Halstead*. As métricas *Halstead* foram escolhidas pois medem a complexidade de módulos de um programa e podem determinar uma medida quantitativa da complexidade dos operadores e operandos de um módulo, além de fornecer vários indicadores de complexidade.

Tabela 1. Métricas utilizadas da biblioteca *Radon*

Radon	
Métrica	**Especificação**
Cyclomatic Complexity	Número de decisões que um bloco de código contém
Maintainability Index	Quão mantenível o código-fonte é
LOC	Número total de linhas de código
LLOC	Número de linhas lógicas de código
SLOC	Número de linhas de código-fonte
Multi	Número de linhas que representam sequências de várias linhas
Blanks	Número de linhas em branco (ou apenas com espaços em branco)
Halstead Metrics	Usadas para identificar propriedades mensuráveis do *software*

Tabela 2. Métricas utilizadas da biblioteca *Multimetric*

Multimetric	
Métrica	**Especificação**
tiobe_compiler	Pontuação de *warnings* do compilador de acordo com o *TIOBE*
tiobe_coverage	Cobertura de acordo com TIOBE
tiobe_duplication	Pontuação de duplicações de código de acordo com o *TIOBE*
tiobe_functional	*Score* de defeito funcional de acordo com *TIOBE*
tiobe_security	Pontuação de segurança de acordo com *TIOBE*
tiobe	Índice de qualidade geral de acordo com *TIOBE*

4.5. Análises de resultados

Para averiguar a relação entre *code smells* e as métricas pontuadas serão realizadas correlações *Pearson* por meio de biblotecas da linguagem *Python*. Essa correlação foi escolhida pois os valores avaliativos variam entre -1 a 1, onde um valor em sentido positivo pode indicar que as variáveis estão diretamente relacionadas e um valor em sentido negativo pode apontar variáveis inversamente proporcionais, ademais um valor apontado como 0 é indicativo da não existência de correlação entre as variáveis analisadas.

Conjuntamente serão elaboradas matrizes de correlação por meio do software *PoweBI* que possibilita uma variação de aplicabilidade de gráficos demonstrativos informacionais. Não obstante, serão demonstrados gráficos comparativos em relação ao percentual de ocorrência númerica da classificação de *code smells* frequentemente encontrados em ADSs. Por fim, gráficos de dispersão para demonstrar o comportamento de aumento ou decremento de *code smells* de acordo com o número de *releases*.

4.6. Cronograma Quinzenal

O presente trabalho será executado por meio de ciclos quinzenais com o desenvolvimento de tarefas específicas a cada ciclo. A Tabela 3 apresenta as fases execucionais de acordo com os meses estipulados para a realização do trabalho, do início de Fevereiro a primeira quinzena de Maio. Na primeira fase ocorre a elaboração do algoritmo para a realização da coleta de repositórios, na segunda fase a elaboração do algoritmo referente a coleta de *releases*, na terceira fase ocorre a execução da coleta de dados dos 1000 repositórios estipulados. Na quarta fase ocorre elaboração para a geração de métricas e na quinta fase a construção da visualização de dados. Por fim, ocorre a fase de discussão e avaliação dos resultados encontrados.

	FEV		**MAR**		**ABR**		**MAI**	
FASES	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**
Elaboração do algoritmo para coleta de repositórios	•							
Elaboração do algoritmo para coleta de *releases*		•						
Execução da coleta de dados			•					
Elaboração do algoritmo para geração de métricas				•				
Execução da geração de métricas					•			
Elaboração do painel de visualização de dados						•		
Discussão e avaliação dos resultados							•	

Tabela 3. Cronograma execucional do trabalho proposto

Referências

Abdessalem, R. B., Panichella, A., Nejati, S., Briand, L. C., and Stifter, T. (2020). Automated repair of feature interaction failures in automated driving systems. page 88–100.

Chen, C., Shoga, M., and Boehm, B. (2019). Exploring the dependency relationships between software qualities. pages 105–108.

Chen, L., Chen, T., Fan, G., and Yin, B. (2021). Static analysis of resource usage bounds for imperative programs. pages 580–581.

Chen, Z., Chen, L., Ma, W., and Xu, B. (2016). Detecting code smells in python programs. pages 18–23.

Dewangan, S., Rao, R. S., Mishra, A., and Gupta, M. (2021). A novel approach for code smell detection: An empirical study. *IEEE Access*, 9:162869–162883.

Hamdi, O., Ouni, A., AlOmar, E. A., and Mkaouer, M. W. (2021). An empirical study on code smells co-occurrences in android applications. pages 26–33.

Imparato, A., Maietta, R. R., Scala, S., and Vacca, V. (2017). A comparative study of static analysis tools for autosar automotive software components development. pages 65–68.

J. Garcia, Y. Feng, J. S. S. A. Y. X. and Chen, Q. A. (2020). A comprehensive study of autonomous vehicle bugs. *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 385–396.

Luo, Y., Zhang, X.-Y., Arcaini, P., Jin, Z., Zhao, H., Ishikawa, F., Wu, R., and Xie, T. (2022). Targeting requirements violations of autonomous driving systems by dynamic evolutionary search (hop at gecco'22). page 33–34.

Martins, J., Bezerra, C., Uchôa, A., and Garcia, A. (2020a). Are code smell co-occurrences harmful to internal quality attributes? a mixed-method study. page 52–61.

Martins, J., Bezerra, C., Uchôa, A., and Garcia, A. (2020b). Are code smell co-occurrences harmful to internal quality attributes? a mixed-method study. page 52–61.

Plosch, R., Gruber, H., Hentschel, A., Pomberger, G., and Schiffer, S. (2008). On the relation between external software quality and static code analysis. pages 169–174.

Sharma, T. (2018). Detecting and managing code smells: Research and practice. pages 546–547.

Sharma, T. and Kessentini, M. (2021). Qscored: A large dataset of code smells and quality metrics. pages 590–594.

Stocco, A., Weiss, M., Calzana, M., and Tonella, P. (2020). Misbehaviour prediction for autonomous driving systems. page 359–371.

Tang, S., Zhang, Z., Tang, J., Ma, L., and Xue, Y. (2021). Issue categorization and analysis of an open-source driving assistant system. pages 148–153.