Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Ciência da Computação

O impacto de fatores contextuais na incidência de *code smells*: um estudo exploratório baseado em mineração de repositórios de software

Elivelton Cerqueira de Jesus

Feira de Santana

2023

Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Ciência da Computação

Elivelton Cerqueira de Jesus

O impacto de fatores contextuais na incidência de *code smells*: um estudo exploratório baseado em mineração de repositórios de software

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientador: José Amancio Macedo Santos

Feira de Santana

2023

Ficha Catalográfica – Biblioteca Central Julieta Carteado

J56i Jesus, Elivelton Cerqueira de
O impacto de fatores contextuais na incidência de *code smells*: um estudo exploratório baseado em mineração de repositórios de software/ Elivelton Cerqueira de Jesus.- Feira de Santana, 2023.

75f.: il
Orientador: Santos, José Amancio Macedo
Dissertação (Mestrado) – Universidade Estadual de Feira de Santana, Programa de Pós-graduação em Ciência da Computação , 2023.

1. *Code Smells* - Software 2. *Code Smells* – Incidência – Fatores contextuais. 3. Mineração de repositório - Software. I. Santos, José Amancio Macedo, orient. II. Universidade Estadual de Feira de Santana. III. Título.

CDU: 681.3

Tatiane Souza Santos - Bibliotecária CRB5/1634

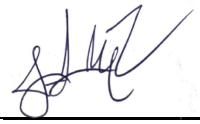
Elivelton Cerqueira de Jesus

O impacto de fatores contextuais na incidência de code smells: um estudo exploratório baseado em mineração de repositórios de software

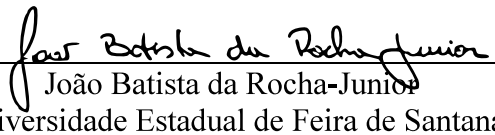
Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Feira de Santana, 14 de dezembro de 2022

BANCA EXAMINADORA



José Amancio Macedo Santos (Orientador(a))
Universidade Estadual de Feira de Santana



João Batista da Rocha-Junior
Universidade Estadual de Feira de Santana

Ivan do Carmo Machado
Universidade Federal da Bahia

Abstract

Context: Code Smells is a metaphor created to describe code structures resulting from potentially inappropriately applied programming design or practice. Studies present code smells as an indication of threat to software quality. However, the adoption of the concept of code smells in the practice of software development cannot yet be considered a reality, at least in certain contexts. Due to the nature of the software, establishing precise contexts for the adoption of concepts, methods and techniques is not trivial. Several factors can be associated with the context in which a software is developed. These factors can be technical, human or social. Although studies related to contextual factors have received attention in recent years, few studies address the issue considering the relationship between contextual factors and code smells.

Objective: The objective of this work is to analyze the relationship between the incidence of code smells and contextual factors in the software. More specifically, the work aims to explore how different types of code smells are related to certain factors, independently or combined to create different contexts.

Method: For this, 419 systems are being used, considering 4 contextual factors which are: System Size, Number of Changes, Number of Contributors and Development Time. Seven types of code smells were considered, which are widely discussed in the literature: Brain Class, Brain Method, Complex Method, Data Class, Feature Envy, God Class, and Long Method. We performed a process of extracting contextual factors through a software repositories mining tool and performed a classification process with the objective of grouping these contextual factors. After that, we started the process of analyzing the collected results, using resources of inferential statistics.

Results: The results indicate that System Size is the contextual factor that most strongly impacts the incidence of different types of code smells. In some situations, as with Data Class, Brain Class, Feature Envy and Brain Method, System Size impacts the incidence of these code smells, regardless of the context in which it is observed. Other factors also impact differently the incidence of some types of code smells studied. For example, the Time of Development contextual factor impacts the incidence of God Class and Brain Class. For these two code smells, Development Time impacts incidence, regardless of the combination with the other contextual

factors. This evidences the strong relationship between the factor and the code smells in question.

Conclusion: This study contributes to expanding empirical data on the relevance of contextual factors in relation to code smells. It also presents a dataset on code smells and contextual factors of 419 software obtained from repositories mining. As it is presented as an exploratory study, the main finding of this work is the demonstration that the quality of software projects is related to the context in which the software is developed. From this perspective, adopting the concept of code smell in the practice of software development, without taking into account the context in which it is developed, can lead to biased results, or even distorted from the reality of how code smells affect or arise in systems.

Keywords: code smell, software design, contextual factors, software repository mining

Resumo

Contexto: *Code Smell* é uma metáfora criada para descrever estruturas de código resultantes de design ou prática de programação aplicadas de forma potencialmente inadequada. Estudos apresentam *code smells* como indício de ameaça à qualidade do software. Entretanto, a adoção do conceito de *code smells* na prática do desenvolvimento de software ainda não pode ser considerada uma realidade, pelo menos em determinados contextos. Devido à natureza do software, estabelecer contextos precisos para adoção de conceitos, métodos e técnicas não é trivial. Diversos fatores podem estar associados ao contexto em que um software é desenvolvido. Estes fatores podem ser técnicos, humanos ou sociais. Apesar dos estudos relacionados a fatores contextuais terem recebido atenção nos últimos anos, poucos trabalhos abordam o tema considerando a relação entre fatores contextuais e *code smells*.

Objetivo: O objetivo deste trabalho é analisar a relação entre a incidência de *code smells* e fatores contextuais do software. Mais especificamente, o trabalho visa explorar como diferentes tipos de *smells* estão relacionados com determinados fatores, de forma independente ou combinados para formar diferentes contextos.

Método: Estão sendo utilizados 419 sistemas, considerando 4 fatores contextuais que são: Tamanho do Sistema, Número de Mudanças, Número de Contribuidores e Tempo de Desenvolvimento. Foram considerados 7 tipos de *code smells*, que são amplamente discutidos na literatura: *Brain Class, Brain Method, Complex Method, Data Class, Feature Envy, God Class, e Long Method*. Executamos um processo de extração dos fatores contextuais através de uma ferramenta de mineração de repositórios de software e realizamos um processo de classificação com o objetivo de realizar um agrupamento desses fatores contextuais. Após isso, iniciamos o processo de análise dos resultados coletados, utilizando recursos da estatística inferencial.

Resultados: Os resultados indicam que Tamanho de Sistema é o fator contextual que impacta mais fortemente na incidência de diferentes tipos de *code smells*. Em algumas situações, como aconteceu com *Data Class, Brain Class, Feature Envy* e *Brain Method*, Tamanho do Sistema impacta na incidência destes *code smells*, independente da combinação deste fator com os outros estudados. Outros fatores também impactam de forma diferente na incidência de alguns tipos de *code smells* estudados. Por exemplo, o fator contextual Tempo de Desenvolvimento impacta na incidência de *God Class* e *Brain Class*. Para esses dois *code smells*, Tempo de

Desenvolvimento impacta na incidência, independente da combinação com os outros fatores contextuais. Isso evidencia a forte relação entre o fator e os *code smells* em questão.

Conclusão: Este estudo contribui para ampliar o conjunto de dados empíricos sobre a relevância dos fatores contextuais em relação aos *code smells*. Apresenta também um conjunto de dados sobre *code smells* e fatores contextuais de 419 softwares obtidos a partir de mineração de repositórios. Por ser apresentado como estudo exploratório, o principal achado deste trabalho está na demonstração de que a qualidade de projetos de software tem relação com o contexto em que os softwares são desenvolvidos. Nessa perspectiva, adotar o conceito de *code smell* na prática do desenvolvimento de software, sem levar em conta o contexto em que este é desenvolvido, pode levar a resultados enviesados, ou até distorcidos da realidade de como os *code smells* afetam ou surgem no sistemas.

Palavras-chave: *code smell*, projeto de software, fatores contextuais, mineração de repositório de software

Prefácio

Esta dissertação de mestrado foi submetida à Universidade Estadual de Feira de Santana (UEFS) como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

A dissertação foi desenvolvida no Programa de Pós-Graduação em Ciência da Computação (PGCC), tendo como orientador o Prof. Dr. **José Amancio Macedo Santos**.

Agradecimentos

Ao meu orientador, Prof. Dr. José Amancio Macedo Santos pela oportunidade deste trabalho. Obrigado pela confiança e por me atender da maneira mais rápida possível todas as vezes que incomodei. Agradeço por todo o conhecimento compartilhado de forma fenomenal, e por me apresentar a pesquisa em *code smells* com a qual desejo contribuir com muito mais. Você ganhou um fã!

Agradeço aos professores do programa de Pós-Graduação em Ciência da Computação e também a UEFS, pelos ensinamentos que obtive nesta instituição formidável.

E agradeço a todos, que de alguma forma contribuíram para a realização deste trabalho. A vocês, o meu muito obrigado!

"Dedico este trabalho à minha família, que tiraram todas as pedras do caminho para que eu conseguisse chegar até este momento. 99% deste trabalho só existe por causa de vocês e eu não poderia ser mais abençoado que isto"

Sumário

Alinhamento com a Linha de Pesquisa

Linha de Pesquisa: Software e Sistemas Computacionais

A pesquisa contribui para área da engenharia de software apresentando novas evidências sobre a qualidade de software e como ela pode ser impactada pelo processo de desenvolvimento. Para isso, utilizamos mineração sobre um extenso conjunto de repositórios de softwares.

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

"Um bom começo é a metade."
– Aristóteles

Code smells é uma metáfora criada para descrever estruturas de código resultantes de design ou prática de programação aplicadas de forma potencialmente inadequada. Este conceito foi criado por Fowler (1999) que forneceu 22 definições informais de *code smells* e cenários no qual podem ser encontrados. Mais tarde, Marinescu criou regras de detecção baseadas em métricas de código e thresholds para identificar as partes com potenciais problemas (Marinescu, 2001, 2002, 2004; Lanza e Marinescu, 2006). Esses trabalhos foram importantes para apresentar *code smells* como indício de um problema no design do software.

Muitos trabalhos que foram desenvolvidos mostram evidências do impacto dos *code smells* na qualidade do software. Isso indica que a comunidade científica vem estudando os efeitos relacionados à adoção de *code smells* no desenvolvimento (Santos et al., 2018). Entretanto, a adoção do conceito na prática do desenvolvimento de software ainda não pode ser considerada uma realidade, pelo menos em determinados contextos (de Mello et al., 2019).

Devido à natureza do software, estabelecer contextos precisos para adoção de conceitos, métodos e técnicas não é trivial. Mesmo assim, é importante compreender como estes afetam a atividade. Sem isso, a adoção de *code smells* na prática, ou mesmo de outras soluções relacionadas ao design e a qualidade, podem ser enviesadas, pois não há "bala de prata" na engenharia de software. Tudo depende dos diversos fatores contextuais que podem existir no processo (Brooks e Kugler, 1987). Estes fatores são técnicos, humanos ou sociais, pois são consequências dos sistemas de software serem desenvolvidos em diferentes ambientes, para diversos fins e por equipes com diversas culturas (Zhang et al., 2013).

O impacto dos fatores contextuais na qualidade do software já vem sendo estudado

pela comunidade. Zhang et al. (2013) buscaram compreender se as distribuições de métricas são impactadas por seis fatores contextuais que são: domínio do aplicativo, linguagem de programação, idade, tempo de desenvolvimento, número de alterações e número de downloads. Foi descoberto que todos os seis fatores afetam a distribuição de 20 métricas. Além disso, o fator "linguagem de programação" é o que mais afeta, impactando em 35 métricas.

Alguns autores criaram ferramentas cuja função é unicamente estipular *thresholds* para ferramentas de detecção de *code smells* a partir do contexto de um grupo definido de softwares (Fontana et al., 2015b). Outros autores usavam *thresholds* variáveis, pois perceberam que o uso desses valores em contextos inadequados diminui a precisão da ferramenta de detecção (de Paulo Sobrinho et al., 2018).

Apesar disso, poucos trabalhos abordam o tema sobre a relação de *code smells* com diversos fatores contextuais. Dessa forma, ainda há dificuldades em compreender a relação entre esses atributos que são inerentes ao contexto em que o software é construído, com a qualidade do projeto do software, mesmo para aqueles fatores que já foram estudados.

O objetivo deste trabalho é analisar a relação entre a incidência de *code smells* e fatores contextuais do software. Mais especificamente, o trabalho visa explorar como diferentes tipos de c*code smells* estão relacionados com determinados fatores, de forma independente ou combinados para formar diferentes contextos. Para isso, estão sendo utilizados 419 sistemas, considerando 4 fatores contextuais que são: Tamanho do Sistema, Número de Mudanças, Número de Contribuidores e Tempo de Desenvolvimento. Detectamos para o estudo 7 tipos de *code smells*: *Brain Class, Brain Method, Complex Method, Data Class, Feature Envy, God Class,* e *Long Method*.

Dos 4 fatores contextuais investigados neste trabalho, 3 deles já foram estudados em conjunto por Zhang et al. (2013). Os fatores são Tamanho do sistema, Número de mudanças e Tempo de desenvolvimento. Zhang et al. (2013) estudaram os fatores contextuais e a correlação com métricas de manutenção de software. Em nosso trabalho, consideramos *code smells* como métrica de qualidade, uma vez que a metáfora é proposta com o objetivo de mensurar qualidade de projetos de software, além de ser amplamente aceita pela comunidade.

Também adotamos Número de Contribuidores como um fator contextual, uma vez que observamos estudos que também consideram seu impacto no desenvolvimento de software (Carmel e Bird, 1997; Meneely e Williams, 2009; Bird et al., 2010). Além disso, também levamos em conta a possibilidade de observar um característica importante de softwares open source hospedados no Github, que é a colaboração entre desenvolvedores (Cosentino et al., 2017; Zöller et al., 2020).

Adotamos 7 tipos de *code smells*, pois são os detectáveis pelo Repository Miner (Mendes et al., 2017), ferramenta adotada pela detecção. Os *code smells* escolhidos têm sido amplamente estudados na literatura (Santos et al., 2013; Arcelli Fontana et al., 2016; Liu et al., 2018). Neste estudo, avaliamos se a incidência dos *code smells*

estudados está correlacionada com algum fator contextual, ou com o agrupamento destes fatores contextuais.

A principal contribuição deste trabalho é a avaliação de um conjunto de evidências empíricas sobre a relação entre *code smells* e fatores contextuais, analisando o quão impactante pode ser o contexto do desenvolvimento na qualidade do software. Mesmo para os fatores e *code smells* que já vem sendo investigados, são necessários mais estudos para ampliar o espaço amostral sobre o qual há algum entendimento em relação ao efeito dos fatores contextuais e atributos de qualidade. Isso é parte da natureza de trabalhos baseados no paradigma experimental (Juristo e Vegas, 2009). Além disso, este conhecimento empírico favorece uma maior utilização do conceito de *code smell* na prática de desenvolvimento de software, o que é uma demanda identificada por estudos secundários recentes na área (Santos et al., 2018; Sharma e Spinellis, 2018).

Outra importante contribuição deste trabalho para a comunidade acadêmica é a disponibilização de um conjunto de dados sobre *code smells* e fatores contextuais de mais de 400 softwares obtidos a partir de mineração de repositórios de software. A disponibilização do conjunto de softwares do estudo, obtidos a partir dos repositórios de código aberto, permite minerar um extenso conjunto de informações, colaborando assim com a formação deste campo de conhecimento e oferecendo mais sustentação aos pesquisadores da área.

A dissertação está estruturada nos seguintes capítulos:

- No Capítulo 2, apresentamos a fundamentação teórica em que discutimos o conceito de *code smell*, discutimos também algumas características de trabalhos que envolvem mineração e por fim, elencamos alguns trabalhos relacionados.
- No Capítulo 3 é apresentado o método de pesquisa, em que é mostrado como escolhemos os fatores contextuais. Também apresentamos as definições dos *code smells* e após isso, o *dataset* da pesquisa. Também descrevemos como realizamos a coleta de dados referente aos fatores contextuais e o processo de classificação dos softwares a partir destes. Seguimos com o estudo realizando uma amostragem e extraindo os *code smells* dessa amostra e, por fim, analisamos os os dados coletados de todo o processo.
- No Capítulo 4, apresentamos os resultados encontrados e o impacto que os fatores podem gerar na incidência dos *code smells*.
- No Capítulo 5 apresentamos uma discussão sobre estes resultados e limitações da pesquisa.
- No Capítulo 6 apresentamos as considerações finais do estudo e indicações de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

"Knowing how things work is the basis for appreciation, and is thus a source of civilized delight."
– William Safire

No final da década de 90, Fowler (1999) escreveu um livro sobre refatoração, no qual afirmou que refatorar é "o processo de mudar um sistema de software de forma que não altere o comportamento externo do código, mas melhore sua estrutura interna". Com base nisso, apresentou diversas técnicas para melhoria do código através da refatoração. Além das técnicas, foram apresentadas estruturas de códigos que sugerem possibilidades de melhorias. Essas indicações de possíveis problemas na estrutura foram chamadas de *code smells*. Ao total, foram apresentadas 22 definições informais de *code smells* e cenários nos quais podem ser encontrados. Esse livro se tornou base para muitos estudos sobre *code smells*.

Apesar disso, o intuito do trabalho feito por Fowler (1999) era dar uma sugestão de quando a estrutura pode sofrer melhorias, e não fornecer critérios precisos para detecção. Isso fez com que o processo de detecção fosse subjetivo e dependesse da percepção de quem estava detectando. Esse cenário mudou quando Marinescu criou regras de detecção baseadas em métricas de código (Marinescu, 2001, 2002, 2004; Lanza e Marinescu, 2006). Essas estratégias consistem em um mecanismo genérico para analisar um determinado trecho de código-fonte usando métricas e *thresholds*.

Como é descrita por Lanza e Marinescu (2006), "a métrica é o mapeamento de uma característica particular de uma entidade medida por um valor numérico". Para a sua pesquisa, Lanza e Marinescu (2006) utilizaram métricas de design empregadas para mensurar a qualidade, o tamanho e a complexidade do software. *Thresholds* assumem o papel de valores de referência, definindo quando as métricas possuem valor alto ou baixo. Ao utilizar as métricas e *thresholds*, foi possível estabelecer parâmetros mais claros para detectar *code smells* presentes no código.

O mecanismo de detecção baseado em métricas é composto por um conjunto de operadores lógicos (como "e" e "ou"), que reúnem diferentes condições de filtragem em uma regra estruturada. Uma condição de filtragem é composta por métricas e seus *thresholds*, na qual ocorre a verificação se a métrica está abaixo ou acima dos *thresholds* estipulados. Após a realização de todas as condições de filtragem e da relação através dos operadores lógicos, ocorre uma resposta booleana, indicando se há algum potencial problema no trecho de código especificado. Para cada *code smell*, há um mecanismo de análise diferente. A visão esquemática para essa estratégia é ilustrado na Figura 2.1 a seguir.

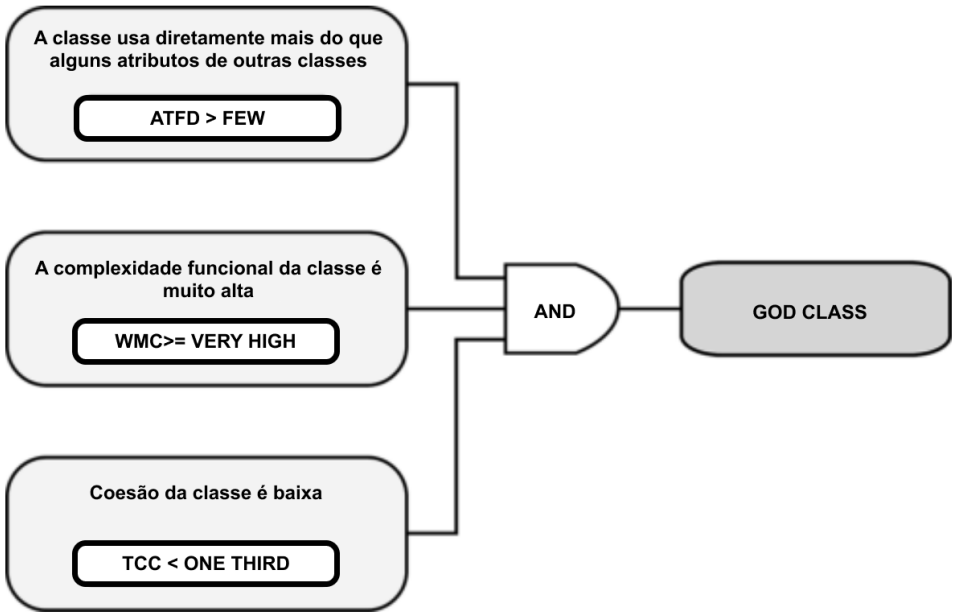


Figura 2.1: Estratégia de detecção para *God Class* (Lanza e Marinescu, 2006)

Lanza e Marinescu (2006) propuseram o agrupamento das métricas ATFD (*Access To Foreign Data*), WMC (*Weighted Method Count*) e TCC (*Tight ClassCohesion*) como forma de caracterizar o *code smell God Class*. A métrica ATFD conta quantos atributos de outras classes são acessados diretamente ou via métodos de acesso (*getters*). WMC é o cálculo realizado para se obter complexidade ciclomática da classe, através dos seus métodos. Por fim, TCC é o número relativo de métodos diretamente conectados via acessos de atributos.

Dessa forma, a heurística proposta para detecção do *code smell* é definida conforme Figura 2.1. A figura indica que se a métrica ATFD possuir valor igual ou maior que *FEW*, a métrica WMC possuir valor igual ou maior que *VERY HIGH* e a métrica TCC for menor que *ONE THIRD* para determinada classe, ela é identificada como um *code smell*. Os valores dos *thresholds FEW, VERY HIGH e ONE THIRD* são definidos de acordo com um *benchmarking* proposto por Lanza e Marinescu (2006), que considera algumas características de contexto dos software, como tamanho e outras métricas de complexidade usadas no livro.

Assim como o trabalho de Fowler (1999), o trabalho de Lanza e Marinescu (2006) se tornou base não só para pesquisas na área, mas também para o desenvolvimento de ferramentas para detecção automática de *code smells*, como inCode (Marinescu et al., 2010), Iplasma (Cristina et al., 2005) e Repository Miner (Mendes et al., 2017). Esses dois estudos foram muito importantes para apresentar *code smells* como uma ameaça à qualidade de software. Para entender o tamanho do impacto, pesquisadores vêm realizando estudos em diversas perspectivas diferentes. Apresentaremos a seguir, algumas dessas perspectivas.

2.1 *Code Smells* e a qualidade de software

Muitas pesquisas foram realizadas para entender como *code smells* impactam no design e qualidade do código. Khomh et al. (2009) investigaram se classes com *code smells* são mais propensas a mudanças do que as classes que não possuem *code smells*. Para realizar essa avaliação, investigaram a relação entre 29 *code smells* e o histórico de mudança de dois sistemas distintos. Para gerar o histórico de mudanças, foram observadas as alterações realizadas em classes de diversas versões desses 2 sistemas. Após avaliação dos históricos e realização de testes estatísticos para entender a relação entre as classes alteradas e os *code smells*, os autores reuniram evidências empíricas de que os *code smells* impactam negativamente, tornando as classes mais voltadas a sofrer alterações. Além disso, perceberam que alguns *code smells* específicos, como *Message Chains*, causam um impacto significativamente maior nas mudanças do que os demais *code smells* avaliados.

Ahmed et al. (2017) estudaram como os code smells se relacionam com *Merge Conflicts*, acreditando que, com a presença de alguns tipos de *code smells*, há uma probabilidade maior de também ter *Merge Conflicts*. Para isso, analisaram 143 repositórios do GitHub e refizeram 6.979 conflitos para obter métricas sobre alterações no código. Para cada conflito, foram extraídos os *code smells* no código e após isso estudaram a relação entre ambas. Identificaram que entidades com instâncias de *code smells* a nível de método como *Blob Operation* e *Internal Duplication* estão altamente relacionados com *Merge Conflicts*. Concluíram que *code smells* afetam não só a manutenção do código, como também as atividades relacionadas ao projeto, impactando dessa forma, na qualidade do programa.

Cedrim et al. (2017) estudaram os efeitos benéficos e prejudiciais da refatoração em *code smells*. Analisaram com que frequência os tipos de refatoração comumente usados afetam 13 tipos de *code smells*. Para isso, avaliaram os históricos de versões de 23 projetos e obtiveram uma base de 16.566 refatorações. Concluíram que, embora 79,4% das refatorações tenham tocado em elementos que possuíam *code smells*, na maioria das vezes (57%) não reduziram suas ocorrências. Apenas 9,7% das refatorações removeram os *code smells*, enquanto 33,3% introduziram novos, indicando que o processo de refatoração adiciona mais *code smells* do que elimina. Além disso, mais de 95% desses *code smells* causados pela refatoração não foram removidos pos-

teriormente, o que sugere que o processo de refatorar tende a introduzir, com mais frequência, *code smells* que persistem no sistema ao longo do tempo.

Yamashita e Moonen (2012) realizaram um estudo de caso para identificar como os *code smells* impactam na capacidade de manutenção do código. Essa avaliação foi realizada a partir de duas etapas: No primeiro momento, os autores planejaram manutenções para quatro sistemas Java. Após isso, observaram e entrevistaram os desenvolvedores que realizaram essa manutenção durante 14 dias. Eles concluíram que *code smells* oferecem informações sobre aspectos como encapsulamento, herança e simplicidade do código, que podem ser melhorados através da refatoração. Porém nem todos os problemas podem ser identificados através das métricas de *code smells*, por isso, há necessidade de combinar abordagens diferentes para realizar uma manutenção geral.

2.2 Utilização da mineração de repositórios para estudos com *code smells*

Várias pesquisas sobre *code smells* utilizam a mineração como recurso para execução dos estudos. Com a mineração de repositórios, é possível encontrar padrões e informações úteis sobre os softwares (Hassan, 2008). Peters e Zaidman (2012), através de um estudo de caso, investigaram a vida útil dos *code smells* e o comportamento de refatoração dos desenvolvedores em 7 sistemas *open source*. Foi desenvolvida uma ferramenta para mineração de repositórios que calcula o tempo de vida das instâncias de *code smells* nestes projetos. Com isso, perceberam que a vida útil está diretamente ligada a facilidade de refatoração e importância que é dada pelos desenvolvedores, implicando que na maioria das vezes, estes não se preocupam em resolver os problemas arquiteturais.

Tufano et al. (2017) realizaram um estudo sobre o histórico de alterações de 200 projetos de código aberto e investigaram quando os *code smells* são introduzidos pelos desenvolvedores e qual o motivo da origem destes. Para isso, foi desenvolvido uma estratégia para identificar a inserção do *code smells* dentro dos mais de 500 mil *commits*, além da análise manual de mais de 9 mil *commits*. Eles chegaram à conclusão de que a maioria das instâncias de *code smells* são inseridas desde o momento da criação e não durante o processo de evolução como é de senso comum. Além disso, foi notado que muitas destas instâncias poderiam ser evitadas se fosse realizado um processo de verificação antes da confirmação do *commit*. Também foi percebido que há mais de 400 casos de inserção de *code smells* durante a refatoração, embora o principal motivo para a refatoração seja justamente o contrário.

Fontana et al. (2015a) focaram na detecção de *code smells*, com atenção para a maneira como se relacionam e co-ocorrem, para então avaliar a dívida técnica que é causada em um cenário arquitetônico. Com um *dataset* composto por 74 sistemas, eles definiram e identificaram os tipos de relações entre os *code smells* e mediram

a frequência que aconteciam. Para isso, desenvolveram uma ferramenta que analisa sistemas Java e é capaz de detectar 6 tipos de *code smells*. Descobriram que 26% das instâncias de *God Class* usam pelo menos uma *Data Class*. Também observaram que 53% e 70% dos métodos afetados respectivamente por *Shotgun Surgery* e *Dispersed Coupling* chamam uma classe ou método afetado por um *code smells*. Isso confirmou a hipótese de que os *code smells* tendem a se agrupar e interagir de maneiras diferentes, e que esse fato causa mais efeito sobre a manutenção do que cada *code smell* isoladamente. Além disso, observaram que o *Brain Method* tem a maior quantidade de co-ocorrências. Dessa forma, concluíram que *Brain Method, God Class, Shotgun Surgery, Dispersed Coupling* são os *code smells* que mais afetam na degradação da arquitetura.

2.3 Trabalhos relacionados: relação entre *code smells* e fatores contextuais

Alguns trabalhos estudam ferramentas de detecção de *code smells* e indicam uma possível influência do contexto na inconsistência dos resultados obtidos. Paiva et al. (2017) compararam quatro ferramentas de detecção de *code smells*. Três dessas ferramentas utilizavam a mesma técnica de detecção, nomeadamente, as estratégias baseadas em métricas de Lanza e Marinescu (2006) . Todas as quatro ferramentas foram utilizadas para detectar três *code smells* (*God Class, God Method* e *Feature Envy*) presentes em dois sistemas de software que possuem características diferentes, com o objetivo de calcular precisão e concordância. Os resultados apontam que as ferramentas de detecção apresentam níveis de precisão diferentes entre os *code smells* e os sistemas. Enquanto para o primeiro sistema, a precisão das 4 ferramentas para os 3 *code smells* variou entre 0 e 100%, para o segundo sistema, esse dado variou entre 0 e 85%. Além disso, a concordância geral dos sistemas, variou entre 83% e 98%. Isso indica que apesar de uma boa concordância entre os sistemas, há diferentes níveis de precisão em diferentes contextos. Essas diferenças de valores estão diretamente ligadas às mudanças de *thresholds* entre as ferramentas de detecção.

Fontana et al. (2011) apresentaram seis ferramentas de detecção de *code smells* e realizaram um experimento para detectar onze *code smells* no mesmo sistema com essas ferramentas. Perceberam que há inconsistências significativas nos resultados encontrados. Os autores supõem que essas inconsistências podem acontecer devido ao fato de que, apesar de usarem as mesmas métricas, os *thresholds* definidos são diferentes. Além disso, apontaram a dificuldade para conseguir visualizar as regras utilizadas na definição dos *thresholds*. Por fim, indicaram que seria benéfico ter a possibilidade de definir e alterar os *thresholds* de acordo com os contextos.

Essa característica também foi observada por Rasool e Arshad (2015). Os autores realizaram uma revisão sistemática para entender o estado da arte em relação a ferramentas e técnicas para detecção de *code smells*. Entre as observações realizadas, os autores afirmam que a precisão das técnicas de detecção de *code smells* baseadas

em métricas é dependente da escolha adequada dos *thresholds*, que geralmente é realizada empiricamente de forma não confiável. Além disso, observaram que mudanças de valores entre ferramentas impactam diretamente no resultado das detecções.

Fontana et al. (2015b) propuseram um método para derivar os *thresholds* para detecção dos *code smells*. Esses dados seriam baseados a partir dos valores de métricas, observadas nos sistemas onde serão detectados os *code smells*. Esses *thresholds* são gerados a partir do conjunto de dados dos sistemas a se observar, permitindo que esses valores sejam definidos de forma adequada ao contexto do conjunto selecionado. Os autores indicaram que a motivação para o desenvolvimento do método foi que ao se utilizar abordagens empíricas para definição dos *thresholds* podem ser produzidos muitos falsos negativos ou falsos positivos, caso os *thresholds* não estejam adequados. Para ilustrar o funcionamento desse método, utilizaram dados de 74 sistemas e extraíram *thresholds* para detecção de 5 *code smells*.

Assim como os *code smells*, os fatores contextuais e seus impactos na qualidade de software vêm sendo estudados. Zhang et al. (2013) buscaram entender se as distribuições das métricas para manutenção de software são afetadas por seis fatores contextuais: domínio do aplicativo, linguagem de programação, idade, tempo de desenvolvimento, número de alterações e número de downloads. Para isso, selecionaram 320 sistemas de software e calcularam 39 métricas desses sistemas. Após isso, realizaram testes estatísticos para determinar se havia diferenças significativas entre as distribuições, em relação a cada um dos seis fatores contextuais. Descobriram que todos os seis fatores contextuais afetam a distribuição de 20 métricas. Além disso, o fator "linguagem de programação" é o que mais afeta, impactando em 35 métricas.

Ogheneovo et al. (2014) estudaram a correlação entre a complexidade do software e o custo de manutenção no projeto de três sistemas operacionais. Para realizar essa pesquisa, utilizaram o fator de contexto "linhas de código" como base para mensurar a complexidade, com a justificativa de que o software se torna mais complexo conforme cresce de tamanho. Utilizaram um modelo de estimativa de custos, para extrair o custo de manutenção. Os resultados indicaram que há uma correlação forte. À medida que as linhas de código do software aumentam, também há aumento nos custos para mantê-lo. Isso acontece pois é necessário aumentar a equipe e o tempo necessário para entender o código, à proporção que o software vai ficando mais complexo.

Behnamghader et al. (2017) investigaram como os *commits* afetam a qualidade de software. Analisaram um total de 19.580 *commits* de 38 sistemas Java, para entender melhor como cada *commit* impacta no sistema. Para este estudo, os autores consideraram nove métricas de qualidade de software, entre elas, *code smells*. Em relação a *code smells*, os resultados mostram que, em média, 46% dos *commits* impactantes, ou seja, *commits* que realizam alterações importantes no sistema, introduzem novos *code smells* ou resolvem os existentes. Isso indica que, de forma constante, os desenvolvedores estão realizando mudanças em trechos de código com *code smells*, através de *commits* importantes.

Este presente estudo tem a finalidade de investigar a relação de mais alguns fatores contextuais, que até então foram abordados em outras áreas referentes a qualidade de software, com a incidência de *code smells*. Até onde sabemos, este é o primeiro trabalho a propor uma investigação detalhada sobre a relação entre 4 fatores contextuais com a incidência de 7 *code smells*, dividindo os sistemas em diferentes contextos. Além disso, também estamos analisando essa relação considerando um conjunto de mais de 400 sistemas, o que é um volume superior aos outros estudos observados.

Capítulo 3

Metodologia

Neste capítulo é apresentado o design do estudo. Inicialmente, apresentamos a questão de pesquisa que serviu como base para o desenvolvimento do estudo. Em seguida, os fatores contextuais escolhidos são explicados. Além disso, é descrito o processo de escolha da base de dados e da coleta de dados referentes aos fatores contextuais. Também apresentamos como os dados foram classificados, a seleção da amostra para estudo e como os *code smells* foram extraídos dos softwares para posterior análise. Finalmente, explicamos as estratégias de análise utilizadas. Na Figura 3.1, é possível visualizar de forma resumida os passos executados.

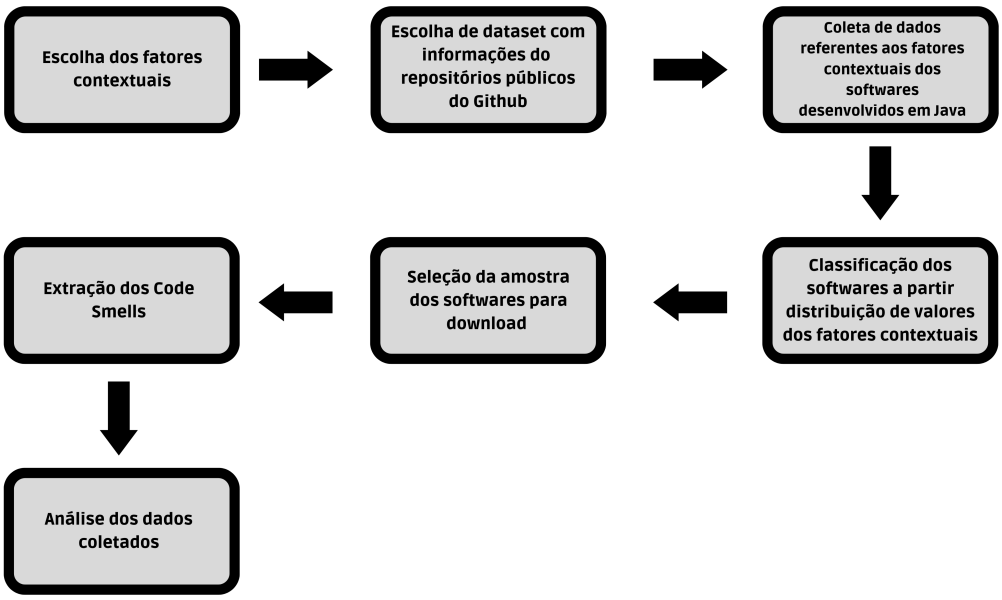


Figura 3.1: Visão esquemática das etapas realizadas na Metodologia

3.1 Questão de Pesquisa

Para orientar no objetivo de investigar a relação entre a incidência de *code smells* e fatores contextuais de software, desenvolvemos a seguinte questão de pesquisa:

Fatores contextuais do software podem impactar na incidência de code smells de forma diferente, de acordo com o tipo de code smell?

Esta questão visa entender se softwares que possuem fatores contextuais diferentes podem estar ou não mais propensos a *code smells*. Há a necessidade de termos mais estudos empíricos para colaborar com a construção do conhecimento na área. Além disso, compreender o comportamento dos *code smells* em relação a características inerentes ao desenvolvimento de software pode auxiliar na adoção prática do conceito de *code smells*, pois torna mais assertiva a compreensão do impacto na tarefa de construção e na qualidade final do projeto.

Devido a essas características do trabalho, este pode ser enquadrado como um estudo exploratório que visa evidenciar a importância de tratar do tema em busca da melhoria da qualidade do software.

Dessa forma, com essa questão, estamos observando se fatores também podem impactar em *code smells*. Para isso, vamos observar se impactam de forma distintas em *code smells* de tipos diferentes.

3.2 Fatores Contextuais

Nesta seção apresentamos os fatores contextuais considerados para a realização do estudo. Para a escolha desses fatores, recorremos a pesquisas na literatura que já estudaram fatores contextuais, além de outros trabalhos que abordaram as características do software e a sua relação com a qualidade. Abaixo, apresentamos os fatores contextuais selecionados e a motivação desta seleção.

Tamanho do sistema: Ogheneovo et al. (2014) utilizaram linhas de código como fator de contexto em seu estudo sobre a relação entre complexidade de software e custo de manutenção. Concluíram que à medida que o software cresce em tamanho, a complexidade também tende a aumentar, o que, consequentemente, torna mais difícil realizar manutenções.

Número de mudanças: Behnamghader et al. (2017) investigaram como as mudanças realizadas no código-fonte do software impactam na qualidade. Consideraram *commits* como um dos fatores e analisaram 19.580 *commits* de 38 softwares em Java. Concluíram que vários fatores relacionados aos *commits*, como a realização de *commits* muito cedo ou frequentes ao desenvolver, podem degradar a qualidade do sistema.

Número de contribuidores: Meneely e Williams (2009) estudaram a participação de contribuidores no desenvolvimento de softwares de código aberto e concluíram que

sistemas no qual 9 ou mais contribuidores participaram do desenvolvimento, eram 16 vezes mais propensos a ter uma vulnerabilidade na segurança do software.

Tempo de desenvolvimento: Canfora e Cimitile (2001) discutiram de maneira geral, a manutenção de software, apresentando sua relevância, os problemas e as soluções disponíveis. Uma de suas percepções, é que, softwares que levam longos períodos em desenvolvimento, tendem a oferecer maior dificuldade no momento da manutenção. Observação também apresentada por Lehman (1979).

3.3 *Code Smells*

Nesta seção, descrevemos os *code smells* utilizados neste estudo. Adotamos estes *code smells*, devido a possibilidade de detecção da ferramenta utilizada, o Repository Miner (Mendes et al., 2017). Além disso, os *code smells* escolhidos já foram utilizados em outros estudos encontrados na literatura (Santos et al., 2013; Arcelli Fontana et al., 2016; Liu et al., 2018).

Data Class: Refere-se a classes que detém dados, mas não possuem uma funcionalidade clara. Apesar disso, outras classes dependem desta classe. A falta de métodos funcionalmente relevantes pode indicar que dados e comportamentos relacionados não são mantidos em um só lugar, o que indica falta de encapsulamento (Lanza e Marinescu, 2006).

God Class: Indica classes que tendem a centralizar as funções o sistema. São classes que realizam muitas atividades de forma independente, delegando apenas pequenos detalhes para classes triviais. Além disso, há a quebra de encapsulamento, pois acessam diretamente muitos atributos de outras classes. Isso tem um impacto negativo na reutilização e na compreensão dessa classe (Lanza e Marinescu, 2006).

Brain Class: São classes complexas que tendem a acumular uma quantidade excessiva de funcionalidade, geralmente na forma de vários métodos afetados pelo *Brain Method*. Apesar de se assemelhar bastante com *God Class*, são *code smells* distintos. *Brain Classes* são classes bastantes complexas que não acessam dados de outras classes, enquanto *God Classes* estão ligadas não só a complexidade, mas também ao seu padrão de comportamento de realizar justamente esses acessos a dados (Lanza e Marinescu, 2006).

Complex Method: São métodos que possuem complexidade ciclomática alta, ou seja, executam diferentes conjuntos de comandos dentro do seu código (Sharma et al., 2017).

Long Method: São métodos excessivamente grandes, portanto, usam muitas variáveis e parâmetros temporários, tornando-os mais propensos a erros. Tentem a afetar a compreensibilidade e testabilidade do código.(Lanza e Marinescu, 2006)

Feature Envy: refere-se a métodos que acessam diretamente ou através de métodos de acessos (*getters*), muitos dados de outras classes. Isso pode ser um sinal de que o método deve ser movido para outra classe (Lanza e Marinescu, 2006).

Brain Method: é um método que executa muitos papéis e tende a centralizar a funcionalidade de uma classe, tornando-se difícil de manter ou entender (Lanza e Marinescu, 2006).

3.4 Escolha do *dataset* e coleta de dados referentes aos fatores contextuais

Para a coleta de dados, fizemos o download de uma base de dados do GHTorrent (Gousios, 2013) , com nome e endereço de repositórios de software no Github[1]. O GHTorrent é um projeto que coleta dados de repositórios públicos de software disponíveis no Github. O Github é a hospedagem de codificação colaborativa mais utilizada no mundo. Em maio de 2021, possuía mais de 200 milhões de repositórios e mais de 65 milhões de desenvolvedores na plataforma. Os dados dos softwares no GitHub são disponibilizados pelo GHTorrent com o propósito de colaborar para o ambiente de pesquisa na área de engenharia de software. No momento do download realizado neste trabalho, em maio de 2021, o GHTorrent continha dados de 6.760.351 softwares.

Para este trabalho, decidimos utilizar softwares desenvolvidos em Java. A justificativa para esta escolha é pelo fato de Java ser uma das linguagens de programação mais utilizadas nas últimas décadas. Além disso, a maioria das ferramentas que detectam *code smells* são desenvolvidas para essa linguagem (Fernandes et al., 2016). Com isso, coletamos informações somente dos sistemas em java, totalizando 631.274 softwares.

Também foi feita a extração dos dados referentes aos fatores contextuais. Para realizar essa extração, criamos um *script* em *Python*, utilizando o *framework* chamado Pydriller (Spadini et al., 2018). O Pydriller é uma ferramenta que realiza mineração de dados em repositórios Git e facilita a extração de informações. Abaixo, apresentamos como extraímos cada fator contextual

Tamanho do sistema: Para esse fator, consideramos o total de linhas de código presentes no software. Como essa informação não está acessível diretamente através do Pydriller, para captar esse dado utilizamos a quantidade de linhas modificadas em todos os *commits*, somando as quantidades de linhas adicionadas e subtraindo a quantidade de linhas excluídas.

Número de mudanças: Para esse fator, contabilizamos o total de *commits* realizados no repositório.

Número de contribuidores: Para esse fator, levamos em conta o total de desenvolvedores que realizaram *commits* nos repositórios.

Tempo de desenvolvimento: Foi calculado o tempo existente entre o primeiro *commit* realizado e o último *commit*. Esse dado foi registrado em formato de horas.

[1]https://github.com/

Em razão do *dataset* GHTorrent ter sido criado em 2015, foi necessário verificar quais repositórios ainda estavam disponíveis. Para realizar essa checagem e coleta dos fatores contextuais, utilizamos 10 máquinas virtuais hospedadas na DigitalOcean[2], no qual o *script* em *Python*, criado com o Pydriller, era executado em partes diferentes do *dataset* GHTorrent. Os *scripts* foram executados e recuperaram dados dos fatores contextuais de 452.843 softwares. Para os outros 178.431 não conseguimos recuperar informações por estarem indisponíveis no Github. Estes foram excluídos da seleção.

3.5 Classificação dos softwares

Após a coleta dos fatores contextuais dos 452.843 softwares, para cada fator contextual, observamos a distribuição de valores relacionados aos mais de 450 mil softwares cujos dados foram coletados. Desconsideramos os softwares dentro do 1º quartil de cada fator, devido aos baixos valores presentes nessa parte da distribuição. Suspeitamos que parte desses softwares poderiam representar sistemas muito simples (denominados *toy systems*) ou mesmo sistemas incompletos. Por exemplo, sistemas dentro do 1º quartil do fator contextual tamanho do sistema, tinham 1.300 linhas ou menos, valor quatro vezes menor do que a mediana da distribuição (4.829 linhas). Outro exemplo é em relação ao tempo de desenvolvimento, no qual os sistemas do 1º quartil tinham menos de 269 horas, sendo que a mediana está em 3.316 horas.

Também desconsideramos o 3º quartil, com o intuito de criar uma distância de valores entre os sistemas que estão no 2º quartil e sistemas que se encontrem no último quartil. Uma vez que o objetivo da pesquisa é comparar como diferentes características de contexto impactam na existência de *code smells*, decidimos selecionar softwares com valores de contextos diferentes. Além disso, para garantir que dentro de um contexto, existam sistemas com características muito próximas, também não levamos em conta *outliers*.

Em seguida, classificamos os softwares resultantes dos filtros aplicados. Usamos o termo *INFERIOR* para descrever o grupo de softwares que se encontram no 2º quartil em relação ao fator de contexto observado. De forma análoga, classificamos os softwares cujo valor referente ao fator de contexto em análise está enquadrado nos valores do último quartil usando o termo *SUPERIOR*. A Figura 3.2 demonstra a classificação.

Linhas de código: Os valores encontrados foram: 1 (mínimo), 1.363(25%), 4.829 (mediana), 28.642 (75%) e 69.560 (máximo). Sistemas que tinham entre 1.363 e 4.829 linhas, foram classificados com indicação de que possuem número *INFERIOR* de linhas de código, enquanto aqueles que tinham entre 28.642 e 69.560 linhas foram classificados com indicação de que possuem número *SUPERIOR* de linhas de código.

[2]https://www.digitalocean.com/

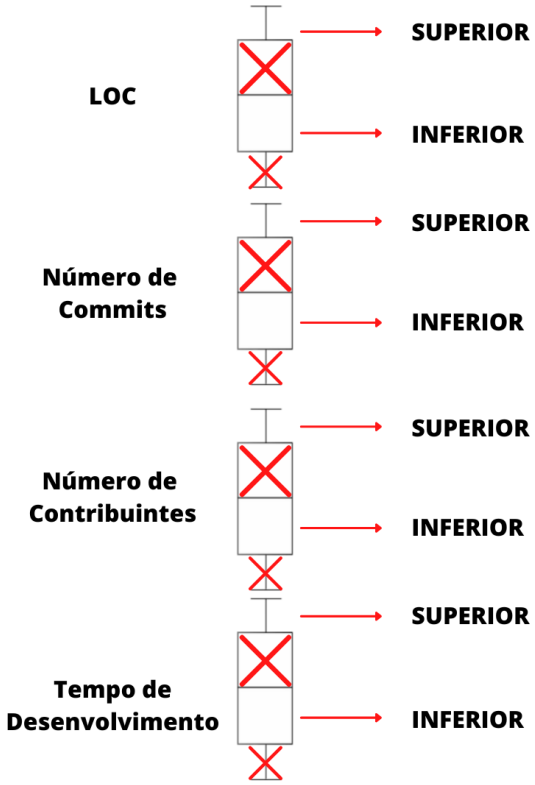


Figura 3.2: Visão esquemática da classificação realizada com a distribuição dos fatores contextuais

Número de commits: Os valores de *commits* foram: 2 (mínimo), 7 (25%), 25 (mediana), 148 (75%) e 359 (máximo). Softwares que tinham entre 7 e 25 *commits* foram classificados com indicação de que possuem número *INFERIOR* de *commits*. Softwares que tinham entre 148 e 359 *commits* no repositório, foram classificados com indicação de que possuem número *SUPERIOR* de *commits*.

Número de contribuidores: Os valores referentes ao número de contribuidores foram: 1 (mínimo), 1(25%), 2 (mediana), 7 (75%) e 16 (máximo). Como na distribuição desse grupo os valores de mínimo, 25% e mediana são iguais, todos os softwares com 1 e 2 foram classificados com indicação de que possuem número *INFERIOR* de contribuidores. Aqueles que possuíam entre 7 e 16 contribuidores, foram classificados com indicação de que possuem número *SUPERIOR* de contribuidores.

Tempo de desenvolvimento: Os valores relacionados ao tempo de desenvolvimento foram: 1 (mínimo), 269 (25%), 3.316 (mediana), 16.421 (75%) e 40.647 horas (máximo). Os softwares entre 269 e 3.316 horas foram classificados com indicação de que possuem tempo de desenvolvimento *INFERIOR*. Já os sistemas entre 16.421 e 40.647 horas foram classificados com indicação de que possuem tempo de desenvolvimento *SUPERIOR*.

A Tabela 3.1 resume os valores dos fatores contextuais resultantes da classificação

realizada.

Tabela 3.1: Valores coletados dos fatores de contextuais.

Fator Contextual	***INFERIOR***		***SUPERIOR***	
	De	Até	De	Até
Linhas de Código	1363	4829	28642	69560
Número de commits	7	25	148	359
Número de Contribuintes	1	2	7	16
Tempo de desenvolvimento	269	3316	16421	40647

Consideramos, nas fases seguintes, somente sistemas que receberam a classificação *INFERIOR* ou *SUPERIOR* dos 4 fatores contextuais, ou seja, apenas sistemas que estavam dentro do 2º quartil ou do 4º quartil de todos os fatores contextuais. Essa estratégia visou garantir que os softwares selecionados sempre tivessem uma classificação, seja ela *INFERIOR* ou *SUPERIOR*, para todos os fatores contextuais analisados neste estudo. Após separar os sistemas que se encontravam neste cenário, foram selecionados um total de 9.058 softwares.

3.6 Contexto

Apresentamos a seguir, uma definição que será importante para as próximas fases deste trabalho. Sendo assim, criamos esta seção para esclarecer o papel do contexto dentro do projeto.

Dentro de cada grupo, podemos criar contextos de softwares. Para isso, selecionamos os fatores que serão utilizados no grupo, combinando-os através da classificação realizada (*INFERIOR* e *SUPERIOR*). Então, consideramos os softwares que se encontram na interseção desta combinação. Um exemplo de contexto é apresentado na Figura 3.3.

Ao selecionar fatores contextuais diferentes em cada grupo, podemos criar possibilidades de contextos diferentes, sendo compostos dos mesmos fatores contextuais, mas com classificações (*INFERIOR* e *SUPERIOR*) diferentes. Por exemplo, na combinação entre os fatores Tamanho do Sistema e Número de Mudanças, encontramos sistemas no qual Tamanho do Sistema é *INFERIOR* e Número de Mudanças também é *INFERIOR* (Linha 1 da Tabela 3.2). Também há sistemas com Tamanho do Sistema é *SUPERIOR* e Número de Mudanças também *SUPERIOR* (linha 4 da Tabela 3.2). Esse grupo é exemplificado na Tabela 3.2.

Em outra combinação, formada por Número de Contribuidores e Tempo de Desenvolvimento, temos contextos completamente diferentes. Há sistemas com Número de Contribuidores *INFERIOR* e Tempo de Desenvolvimento *INFERIOR* (linha 1 da Tabela 3.3), e também há sistemas com Número de Contribuidores *SUPERIOR* e Tempo de Desenvolvimento *SUPERIOR* (linha 4 na Tabela 3.3) como pode ser visto na Tabela 3.3.

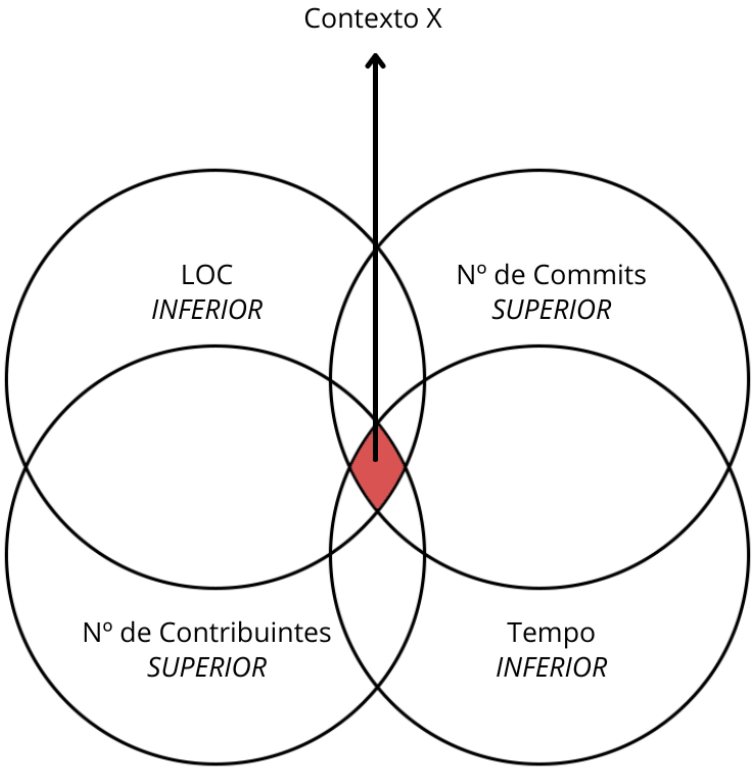


Figura 3.3: Visão esquemática da classificação realizada com a distribuição dos fatores contextuais

Dessa forma, é visto que os contextos da combinação entre Tamanho do Sistema e Número de Mudanças e os contextos da combinação entre Número de Contribuintes e Tempo de Desenvolvimento, são diferentes pois são criados a partir de fatores contextuais diferentes.

3.7 Seleção da amostra e extração dos *Code Smells*

Após a etapa de classificação, utilizamos softwares em que os 4 contextos estão agrupados. Para dividir os softwares e escolhemos de maneira aleatória 30 softwares de cada contexto. Com essa combinação é possível gerar 16 contextos diferentes, apresentados na Tabela 3.4. Resolvemos adotar esta combinação, como base para amostra, por ser o único no qual há softwares divididos em todos os contextos. Essa etapa de amostragem ocorre devido ao fato da impossibilidade de realizar o

Tabela 3.2: Contextos formados a partir de Tamanho do Sistema e Número de Mudanças

Tamanho do Sistema	**Número de Mudanças**
INF	*INF*
INF	*SUP*
SUP	*INF*
SUP	*SUP*

Tabela 3.3: Contextos formados a partir de Número de Contribuidores e Tempo de Desenvolvimento

Número de Contribuidores	**Tempo de Desenvolvimento**
INF	*INF*
INF	*SUP*
SUP	*INF*
SUP	*SUP*

download e extração dos *code smells* de mais de 9 mil softwares.

Tabela 3.4: Contextos criados ao agrupar os 4 fatores contextuais (*INF = INFERIOR, SUP = SUPERIOR*)

Tamanho do Sistema	**Número de Mudanças**	**Número de Contribuidores**	**Tempo de desenvolvimento**
INF	*INF*	*INF*	*INF*
INF	*INF*	*INF*	*SUP*
INF	*INF*	*SUP*	*INF*
INF	*INF*	*SUP*	*SUP*
INF	*SUP*	*INF*	*INF*
INF	*SUP*	*INF*	*SUP*
INF	*SUP*	*SUP*	*INF*
INF	*SUP*	*SUP*	*SUP*
SUP	*INF*	*INF*	*INF*
SUP	*INF*	*INF*	*SUP*
SUP	*INF*	*SUP*	*INF*
SUP	*INF*	*SUP*	*SUP*
SUP	*SUP*	*INF*	*INF*
SUP	*SUP*	*INF*	*SUP*
SUP	*SUP*	*SUP*	*INF*
SUP	*SUP*	*SUP*	*SUP*

Durante o processo de seleção dos softwares, observamos algumas características que poderiam criar algum viés na análise de dados. Destacamos a identificação de termos como “teste” e “demo” no nome e adotamos o critério de excluir softwares que continham estes termos no nome. Acreditamos que softwares que contenham esses termos podem estar incompletos ou podem não apresentar a versão final. Abaixo, no Algoritmo 1, é possível visualizar um trecho do código que faz essa validação:

A escolha de 30 softwares por contexto resultaria em 480 softwares para o estudo. Porém, alguns contextos apresentaram menos de 30 softwares. Esses contextos têm em comum Número de Mudanças *INFERIOR* e Número de contribuidores *SUPERIOR* . Acreditamos que isso ocorre devido às suas características. Nos casos em

Algorithm 1 Verificar se o nome era Teste ou Demo

```
def verificarNomeTD(nome):
if "teste" in nome or "demo" in nome then
    return True
else
    return False
end if
```

que Número de Mudanças é *INFERIOR* e número contribuidores *SUPERIOR*, os softwares são mais raros, pois, quanto mais contribuidores há no repositório, os *commits* realizados tendem a aumentar. No final do processo, realizamos o download dos 419 sistemas selecionados aleatoriamente. Na Figura 3.4, há uma visão esquemática do processo para diminuir de 6.760.351 softwares que estavam no *dataset* do GHTorrent até os 419 softwares da amostragem deste trabalho

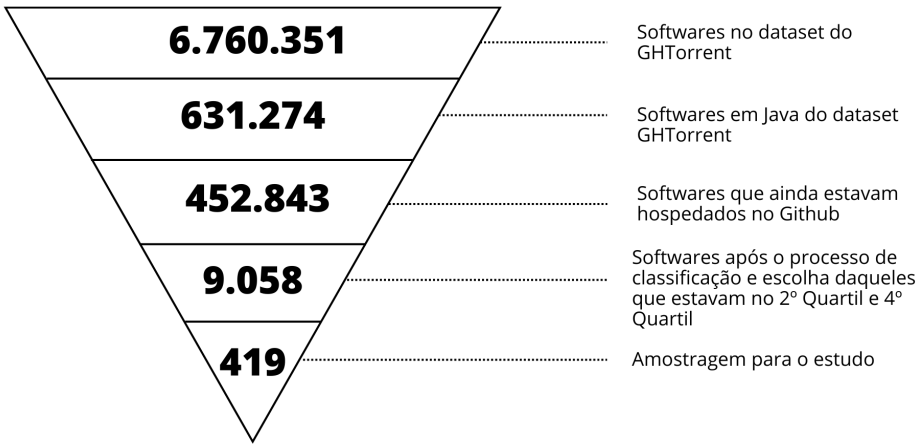


Figura 3.4: Visão esquemática das etapas realizadas para filtragem dos softwares

Utilizando o Repository Miner (Mendes et al., 2017), extraímos os *code smells* dos 419 sistemas selecionados. O Repository Miner é uma ferramenta para mineração de repositórios de software. Dentro das suas funções, ele é capaz de detectar 7 tipos de *code smells* diferentes: *God Class, Data Class, Brain Class, Complex Method, Feature Envy, Long Method* e *Brain Method*. Consideramos para realização do estudo, os 7 tipos de *code smells* detectáveis.

3.8 Estratégia de Análise

Nesta seção, vamos apresentar a estratégia de análise, visando responder a questão de pesquisa Q1, apresentada anteriormente. Para realizar a análise, examinamos cada repositório e verificamos a existência de *code smells*, utilizando o Repository

Miner. Após isso, calculamos o percentual de classes com um determinado *code smell* em relação ao total de classes no repositório. A escolha de utilizar o percentual visa normalizar a informação, pois a quantidade de classes varia muito de um sistema para o outro.

Para representar esse dado, criamos variáveis. Para *Brain Class*, criamos a variável *percentClassesWithBrainClass*. Dessa forma, a cada um dos 419 softwares é associado um valor de *percentClassesWithBrainClass*. Da mesma forma, para *God Class*, denominamos *percentClassesWithGodClass* e seguimos o mesmo padrão para todos outros cinco *code smells*. As variáveis são apresentadas a seguir:

- *percentClassesWithBrainClass*
- *percentClassesWithGodClass*
- *percentClassesWithDataClass*
- *percentClassesWithLongMethod*
- *percentClassesWithComplexMethod*
- *percentClassesWithBrainMethod*
- *percentClassesWithFeatureEnvy*

A partir desse ponto, realizamos análises para identificar se há diferença significativa no percentual de classes com *code smells* comparando os diferentes contextos. Dessa forma, averiguamos a relação entre os fatores contextuais e os *code smells*, observando se um determinado fator contextual é capaz de causar diferença no percentual de classes com *code smells*. Essa diferença pode indicar que o fator contextual analisado é capaz de influenciar na incidência de um determinado *code smell*.

Examinamos o impacto dos fatores contextuais sobre cada *code smells*, testando hipóteses que criamos para cada um individualmente. Para verificar como cada fator contextual isolado impacta no percentual de um *code smell*, aplicamos o teste de Mann-Whitney (Teste U) usando uma taxa de 5% para nível de significância (p-$value$=0.05). Esta escolha foi feita após a aplicação do teste Shapiro-Wilk de normalidade, no qual observamos que, para todos os casos, as distribuições dos contextos não eram normais. O teste de Shapiro Wilk é um dos mais poderosos para verificação de teste de normalidade e apresenta bons resultados para amostras n>=30 (Yap e Sim, 2011). O Teste de Mann-Whitney é usado para determinar se duas amostras independentes, que podem ser não pareadas, pertencem à mesma população ao verificar se há igualdade entre as medianas (Kothari, 2004). Por ser um teste não-paramétrico, não assume distribuição normal, o que está alinhado com as características das amostras do estudo.

Inicialmente, fizemos a comparação da distribuição do percentual de classes com *code smells* entre sistemas cujo um determinado fator contextual assume valor *INFERIOR*, com aqueles que assumem valor *SUPERIOR*. Para cada *code smell*, fizemos essa comparação com os 4 fatores contextuais do estudo. Um exemplo desta comparação, pode ser vista Tabela 3.5 a seguir:

Tabela 3.5: Comparação entre *INFERIOR* e *SUPERIOR* para os fatores contextuais

Fator Contextual	**Comparação**	**Hipótese nula é falsa**	***p-value***
Tamanho do Sistema	INF x SUP	Sim	6,10e-08
Nº de Mudanças	INF x SUP	Sim	0,0272
Nº de Contribuidores	INF x SUP	Não	0,7748
Tempo de Desenvolvimento	INF x SUP	Sim	0,0009

Para os fatores cujos resultados da análise apontam diferença significativa no percentual de um *code smell*, aprofundamos a análise considerando também o efeito a partir da combinação com outro fator contextual. Ou seja, a partir do exemplo da Tabela 3.5, em que Tamanho do Sistema, Número de Mudanças e Tempo de Desenvolvimento geraram diferenças significativas no percentual de classes com *code smells*, nos aprofundamos fazendo a análise de cada um, combinando com os outros fatores do estudo. Por exemplo, em relação ao Tamanho do Sistema, analisamos da seguinte forma:

- Tamanho do Sistema x Número de Mudanças
- Tamanho do Sistema x Número de Contribuidores
- Tamanho do Sistema x Tempo de Desenvolvimento

Para realizar esta análise focamos apenas naqueles contextos em que o fator analisado alterna e outro fator permanece com a classificação *INFERIOR* ou *SUPERIOR*, que são 2 casos dentre as combinações entre dois fatores. Assim, podemos avaliar somente o contexto em que o fator analisado pode influenciar. Por exemplo, em relação a Tamanho do Sistema x Número de mudanças, analisamos as diferenças do percentual de classes com *code smells* entre as seguintes combinações:

- (Tamanho do Sistema *INFERIOR* e Número de Mudanças *INFERIOR*) X (Tamanho do Sistema *SUPERIOR* e Número de Mudanças *INFERIOR*);
- (Tamanho do Sistema *INFERIOR* e Número de Mudanças *SUPERIOR*) X (Tamanho do Sistema *SUPERIOR* e Número de Mudanças *SUPERIOR*);

Nestas combinações, apenas Tamanho do Sistema alterna enquanto Número de Mudanças permanece estático. Dessa forma, conseguimos indicar de maneira mais assertiva se o Tamanho do Sistema é um fator com impacto determinante na incidência do *code smell* em estudo ou não. No cenário em que os dois fatores alternam, não é possível apontar qual dos fatores está influenciando de fato na diferença do percentual.

Com essa combinação entre fatores contextuais, podemos observar de forma mais clara o impacto de cada fator contextual e conjecturar sobre alguns cenários. Por exemplo, podemos encontrar o cenário em que um fator contextual impacta independente do valor que o outro fator assume. Este cenário foi encontrado em relação ao Tamanho do Sistema para *Data Class* e pode ser visto na Tabela 3.6. As cores

nas linhas data tabela são usadas para efeito visual de separar os fatores de contexto em observação.

Tabela 3.6: Resultado da combinação de Tamanho do Sistema para *Data Class*

Tamanho do sistema alterna	**Fatores que permanecem com classificação fixa** Número de Mudanças	Número de Contribuidores	Tempo de Desenvolvimento	**H0DC é falsa**	***p-value***
	INF	-	-	Sim	0,0132
	SUP	-	-	Sim	7,21e-08
INF X SUP	-	INF	-	Sim	0,0064
	-	SUP	-	Sim	7,45e-09
	-	-	INF	Sim	1,75e-06
	-	-	SUP	Sim	0,0004

Também podemos encontrar o cenário em que o fator contextual só impacta em contextos específicos. Um exemplo deste cenário pode ser visto na tabela 3.7, no qual mostramos o resultado encontrado na análise de Número de Mudanças para *God Class*. Neste caso, *God Class* só é impactado por Número de Mudanças nos contextos em que Tamanho do Sistema assume valor SUPERIOR e quando Número de Contribuidores assume valor INFERIOR.

Tabela 3.7: Resultado da combinação de Número de Mudanças para *God Class*

Nº de Mudanças alterna	**Fatores que permanecem com classificação fixa** Tamanho do sistema	Número de Contribuidores	Tempo de Desenvolvimento	**H0GC é falsa**	***p-value***
	INF	-	-	Não	0,3425
	SUP	-	-	Sim	0,0007
INF X SUP	-	INF	-	Sim	0,0151
	-	SUP	-	Não	0,6049
	-	-	INF	Não	0,0543
	-	-	SUP	Não	0,4578

Dessa forma, combinar com outros fatores contextuais, pode dar um indicativo do quão impactante é cada fator contextual em relação aos *code smells* estudados. Outros cenários também são encontrados e podem ser vistos na seção de resultados.

Para efeito de simplificação do processo de apresentação dos resultados, não aprofundamos a análise quando percebemos que não há impacto significativo no percentual de *code smells* para um fator contextual específico. Percebemos que, quando o fator não impacta individualmente, fará pouca diferença se está ou não relacionado com outros fatores. De uma forma geral, ele seguirá não influenciando no impacto quando colocado em evidência. Nas próximas seções, apresentamos os resultados encontrados para cada um dos *code smells* estudados.

Capítulo 4

Resultados

Neste capítulo, discutimos os resultados encontrados no estudo. Cada subseção trata de um *code smell* diferente, no qual analisamos o impacto dos fatores estudados sobre este. Assim como dito anteriormente, a estratégia adotada é a de aprofundar a análise relacionada a um determinado fator contextual a partir da observação do seu impacto na incidência de um *code smell*. Aprofundamos a análise observando o impacto de um determinado fator combinado com os outros fatores contextuais. Dessa forma, é possível avaliar o quão impactante é este fator contextual sobre o *code smell*, em contextos diferentes.

4.1 *Data Class*

Para o *code smell Data Class* criamos a variável *percentClassesWithDataClass*, que representa o percentual de classes com *Data Class* dentro de cada software. Sendo assim, comparamos a distribuição de *percentClassesWithDataClass* entre diversos contextos para verificar a hipótese apresentada abaixo:

"Há diferenças na distribuição do *percentClassesWithDataClass* entre sistemas de contextos diferentes".

Para essa hipótese, foram adotadas as seguintes hipóteses nula e alternativa:

H0DC: Não há diferença significativa na distribuição de percentClassesWithDataClass para softwares em contextos diferentes;

HADC: Há diferença significativa na distribuição de percentClassesWithDataClass para softwares em contextos diferentes;

4.1.1 Análise sobre os fatores contextuais isolados, considerando *Data Class*

Para essa análise, comparamos a distribuição *percentClassesWithDataClass* para os sistemas cujo determinado fator contextual assume valor *SUPERIOR*, com aqueles

que assumem valor *INFERIOR*. A partir desse ponto, testamos a hipótese nula para validar as diferenças entre as duas distribuições.

Como pode ser visto na Tabela 4.1, através do teste de Mann-Whitney, apenas o fator Tamanho do Sistema apresenta uma diferença estatisticamente significativa (*p-value* <0,05) entre os sistemas de valor *INFERIOR* e *SUPERIOR*, para a distribuição de *percentClassesWithDataClass*. Para os outros fatores contextuais, não foi observado indícios de impacto significativo, devido aos *p-value*s apresentados serem maiores do que o estipulado como limite.

Quando observado o número de sistemas com *Data Class*, é possível observar que aqueles que possuem um Tamanho do Sistema *SUPERIOR*, estão mais propensos a ter *Data Class*, sendo que 102 dos 191 analisados (53,40%), tinham pelo menos uma instância do *code smell*. Enquanto para Tamanho do Sistema INFERIOR, apenas 41 dos 228, (17,98%) possuíam *Data Class*.

Apesar do estudo ser voltado para o percentual de classes com *code smells*, e não o percentual de sistemas com *code smells*, acreditamos que este valor pode ser um indicativo de qual classificação do fator contextual está gerando um número maior de incidência de *Data Class*. Neste caso, por exemplo, é apresentado que o Tamanho do Sistema *SUPERIOR* indica ter um número maior de sistemas com *Data Class*.

Tabela 4.1: Comparação entre *INFERIOR* e *SUPERIOR* dos fatores para *Data Class*

Fator Contextual	**# de sistemas INF**	**# de sistemas SUP**	**# (%) de sistemas - Fator INF**	**# (%) de sistemas - Fator SUP**	**H0DC é falsa**	***p-value***
Tamanho do Sistema	228	191	41 (17,98%)	102 (53,40%)	Sim	4,10e-09
Nº de Mudanças	179	240	49 (27,37%)	94 (39,16%)	Não	0,1279
Nº de Contribuidores	240	179	89 (37,08%)	54 (30,16%)	Não	0,1905
Tempo de Desenvolvimento	218	201	73 (33,48%)	102 (34,82%)	Não	0,8735

Como somente o fator Tamanho do Sistema apresentou uma hipótese nula falsa indicando diferença significativa no *percentClassesWithDataClass*, iremos aprofundar nos resultados encontrados apenas sobre este fator. Desta forma, realizamos análises agrupando Tamanho do Sistema com cada um dos outros três fatores contextuais estudados.

4.1.2 Análise sobre Tamanho do Sistema agrupado com outro fator contextual

Nesta seção, analisamos Tamanho do Sistema agrupado com os outros fatores contextuais, porém realizamos esse agrupamento considerando apenas um outro fator contextual por vez. Dessa maneira, realizamos análise sobre:

- Tamanho do Sistema X Número de Mudanças;
- Tamanho do Sistema X Número de Contribuidores;
- Tamanho do Sistema X Tempo de Desenvolvimento.

Quando realizamos essas combinações, geramos contextos de sistemas diferentes. Por exemplo, considerando o Tamanho do Sistema x Número de Mudanças, há sistemas de tamanho menor, com poucas mudanças para chegar até este ponto (Tamanho do sistema *INFERIOR* e Nº de Mudanças *INFERIOR*). Há sistemas que também têm tamanho menor, mas que passaram por muitas mudanças (Tamanho do sistema *INFERIOR* e Nº de Mudanças *SUPERIOR*). Também há o contexto de sistemas que já são considerados grandes, mas que sofreram poucas mudanças desde a sua entrada no repositório (Tamanho do Sistema *SUPERIOR* e Nº de Mudanças *INFERIOR*). Por fim, os sistemas que são grandes e que passaram por muitas mudanças (Tamanho do sistema *SUPERIOR* e Nº de Mudanças *SUPERIOR*).

A mesma estratégia de combinação é usada para analisar a relação Tamanho do Sistema X Número de Contribuidores e Tamanho do Sistema X Tempo de Desenvolvimento.

Para realizar esta análise focaremos somente naqueles contextos em que Tamanho do Sistema alterna e outro fator permanece com a classificação *INFERIOR* ou *SUPERIOR*, que são 2 casos dentre as combinações possíveis para cada cruzamento. Assim, podemos avaliar somente o contexto em que Tamanho do Sistema pode influenciar. Por exemplo, em relação a Tamanho do Sistema x Número de Mudanças, analisamos as diferenças de *percentClassesWithDataClass* entre as seguintes combinações:

- (Tamanho do Sistema *INFERIOR* e Número de Mudanças *INFERIOR*) X (Tamanho do Sistemas *SUPERIOR* e Número de Mudanças *INFERIOR*);
- (Tamanho do Sistema *INFERIOR* e Número de Mudanças *SUPERIOR*) X (Tamanho do Sistemas *SUPERIOR* e Número de Mudanças *SUPERIOR*);

Nestas combinações, apenas Tamanho do Sistema alterna enquanto Número de Mudanças permanece estático. Dessa maneira, conseguimos indicar de maneira mais assertiva se o Tamanho do Sistema é um fator com impacto mais determinante na incidência do *code smell* em estudo. No cenário em que os dois fatores alternam, não é possível apontar qual dos fatores influencia de fato na diferença entre *percentClassesWithDataClass*.

Na Tabela 4.2 abaixo, é apresentado resultado encontrado para esta análise. É possível visualizar o valor assumido pelo outro fator contextual, enquanto Tamanho do Sistema alternou, além do resultado da avaliação da hipótese e o *p-value* encontrado.

Como pode ser observado, em todas as comparações, a hipótese nula é falsa, ou seja, houve diferença significativa entre as distribuições quando Tamanho do Sistema alternou e Número de Mudanças assumiu valor *INFERIOR*. Também ocorreu quando Número de Mudanças assumiu valor *SUPERIOR*.

O mesmo cenário foi encontrado tanto para a comparação com Número de Contribuidores quanto para a comparação com Tempo de Desenvolvimento. Houve diferença significativa (hipótese nula falsa) quando Tamanho do Sistema alternou e Número de Contribuidores e Tempo de Desenvolvimento assumiram valor *INFERIOR*. Também houve diferenças quando Tamanho do Sistema alternou e os fatores assumiram valor *SUPERIOR*.

Tabela 4.2: Resultado da combinação de Tamanho do Sistema para *Data Class*

Tamanho do sistema alterna	**Fatores que permanecem com classificação fixa**			**H0DC é falsa**	***p-value***
	Número de Mudanças	Número de Contribuidores	Tempo de Desenvolvimento		
INF X SUP	INF	-	-	Sim	0,0132
	SUP	-	-	Sim	7,21e-08
	-	INF	-	Sim	0,0064
	-	SUP	-	Sim	7,45e-09
	-	-	INF	Sim	1,75e-06
	-	-	SUP	Sim	0,0004

Dessa maneira, foi observado que, com apenas dois fatores contextuais sendo considerados, Tamanho do Sistema é impactante independente do valor assumido pelo outro fator. Ou seja, Tamanho do Sistema irá gerar diferenças no percentual de classes com *Data Class* mesmo em cenários de contextos diferentes.

Para efeitos de simplificação, não mostramos as análises combinando um fator contextual com outros 2 e 3 fatores contextuais. Essas combinações geram um volume muito grande de informação que percebemos não agregar novas percepções ao estudo. Dessa maneira, apresentaremos apenas os resultados até a combinação com apenas outro fator. Utilizemos essa mesma estratégia de suprimir estas combinações para todos os outros *code smells*, pelo mesmo motivo.

4.2 *God Class*

Para *God Class*, criamos a variável *percentClassesWithGodClass* responsável por representar o percentual de classes com *God Class* dentro de cada software. A partir deste ponto, assim como na análise apresentada anteriormente (para o *code*

smell Data Class), comparamos a distribuição de *percentClassesWithGodClass* entre diversos contextos para verificar a hipótese apresentada abaixo:

"Há diferenças na distribuição do *percentClassesWithGodClass* entre sistemas de contextos diferentes".

As seguintes hipóteses nula e alternativa foram adotadas:

H0GC: Não há diferença significativa na distribuição de percentClassesWithGodClass para softwares em contextos diferentes;

HAGC: Há diferença significativa na distribuição de percentClassesWithGodClass para softwares em contextos diferentes;

4.2.1 Análise sobre os fatores contextuais isolados, considerando *God Class*

Para analisar o impacto dos fatores contextuais em relação a *God Class*God Class, inicialmente comparamos a distribuição *percentClassesWithGodClass* entre os sistemas cujos fatores contextuais são classificados como INFERIOR e SUPERIOR de forma isolada. Após este passo, testamos a hipótese nula para validar as diferenças entre as distribuições.

Para *God Class*, foi indicado através do teste de Mann-Whitney, que há diferenças (*p-value* <0,05) na distribuição entre os sistemas para os fatores Tamanho do Sistema, Número de Mudanças e Tempo de Desenvolvimento.

Em todos os 3 casos, há um indicativo de que, quando assumem valor SUPERIOR, os sistemas estão mais propensos a ter *God Class*. Para Tamanho do Sistema, quando assumiu valor SUPERIOR, 102 dos 191 sistemas (53,40%) possuem *God Class*. Para Número de Mudanças, 99 dos 240 Sistemas (41,25%). Já em relação a Tempo de Desenvolvimento, 87 dos 201 (43,28%). Os dados são apresentados na Tabela 4.3.

Tabela 4.3: Comparação entre *INFERIOR* e *SUPERIOR* dos fatores para *God Class*

Fator Contextual	**# de sistemas INF**	**# de sistemas SUP**	**# (%) de sistemas - Fator INF**	**# (%) de sistemas - Fator SUP**	**H0GC é falsa**	***p-value***
Tamanho do Sistema	228	191	45 (19,73%)	102 (53,40%)	Sim	6,10e-08
Nº de Mudanças	179	240	48 (26,81%)	99 (41,25%)	Sim	0,0272
Nº de Contribuidores	240	179	83 (34,58%)	64 (35,75%)	Não	0,7748
Tempo de Desenvolvimento	218	201	60 (27,52%)	87 (43,28%)	Sim	0,0009

Nas subseções seguintes, apresentamos os resultados encontrados para cada um dos três fatores contextuais ao combinar com outros fatores contextuais, além da relação destes com *God Class*.

4.2.2 Análise sobre Tamanho do Sistema agrupado com outro fator contextual

Agrupamos Tamanho do Sistema com outro fator contextual, considerando para teste de hipótese os contextos em que Tamanho do Sistema alterna para comparar *INFERIOR* com *SUPERIOR*, enquanto o outro fator contextual assume uma determinada classificação. Dessa forma, analisamos a distribuição de *percentClassesWithGodClass* considerando os seguintes contextos:

- Tamanho do Sistema X Número de Mudanças;
- Tamanho do Sistema X Número de Contribuidores;
- Tamanho do Sistema X Tempo de Desenvolvimento;

Como pode ser visto na Tabela 4.4, houve diferença na distribuição de *percentClassesWithGodClass* em todos os contextos, exceto naquele em que Número de Mudanças assumiu valor *INFERIOR*, indicando que Tamanho do Sistema pode não ser um fator impactante para a incidência de *God Class* em sistemas que sofreram poucas mudanças durante a sua construção.

Tabela 4.4: Resultado da combinação de Tamanho do Sistema para *God Class*

Tamanho do sistema alterna	**Fatores que permanecem com classificação fixa**			**H0GC é falsa**	***p-value***
	Número de Mudanças	Número de Contribuidores	Tempo de Desenvolvimento		
INF X SUP	INF	-	-	Não	0,2542
	SUP	-	-	Sim	5,56e-09
	-	INF	-	Sim	8,64e-05
	-	SUP	-	Sim	0,0001
	-	-	INF	Sim	3,90e-06
	-	-	SUP	Sim	0,0026

Apesar de não ser comprovado diferença quando Número de mudanças assume valor *INFERIOR*, é observado que quando Tamanho do Sistema alterna entre valores *SUPERIOR* e *INFERIOR* e Número de Mudanças assume valor *SUPERIOR*, há diferenças na incidência de *God Class*.

Além disso, também há diferenças para Número de Contribuidores e Tempo de Desenvolvimento quando ambos assumem valor *INFERIOR* e também quando assumem valor *SUPERIOR*. Isso indica que, Tamanho do Sistema impacta na incidência de *God Class* nos softwares independente de qual valor Número de Contribuidores e Tempo de Desenvolvimento assumam.

4.2.3 Análise sobre Número de Mudanças agrupado com outro fator contextual

Analisamos o Número de Mudanças combinados com mais um outro fator. Para esta combinação, as possibilidade de contexto são:

- Número de Mudanças x Tamanho do Sistema
- Número de Mudanças x Número de Contribuidores
- Número de Mudanças x Tempo de Desenvolvimento

Como pode ser visto na Tabela 4.5, ao considerar Número de Mudanças x Tamanho do Sistema (presente no primeiro bloco branco), quando Tamanho do Sistema assume valor *SUPERIOR*, a hipótese nula é falsa. Isso indica que há diferenças no percentual de classes com *God Class* para sistemas em que o Número de Mudanças é *SUPERIOR* ou *INFERIOR*, desde que o Tamanho do Sistema seja *SUPERIOR*. Para sistemas com Tamanho do Sistema é *INFERIOR* não é percebido estatisticamente se há diferença de incidência do *code smell God Class* quando o Número de Mudanças é *SUPERIOR* ou *INFERIOR*.

Para os contextos relacionados a Número de Mudanças x Número de Contribuidores (bloco cinza na Tabela 4.5), há diferenças significativas entre as distribuições de *percentClassesWithGodClass* quando o Número de Mudanças alterna e o Número de Contribuidores é *INFERIOR*.

Já para Número de Mudanças x Tempo de Desenvolvimento (último bloco branco na Tabela 4.5), tanto no contexto em que Tempo de Desenvolvimento é *INFERIOR*, quanto naquele em este assume valor *SUPERIOR*, a hipótese nula não pode ser descartada, indicando que não há diferenças entre as distribuições.

Tabela 4.5: Resultado da combinação de Número de Mudanças para *God Class*

Nº de Mudanças alterna	**Fatores que permanecem com classificação fixa**			**H0GC é falsa**	***p-value***
	Tamanho do sistema	Número de Contribuidores	Tempo de Desenvolvimento		
INF X SUP	INF	-	-	Não	0,3425
	SUP	-	-	Sim	0,0007
	-	INF	-	Sim	0,0151
	-	SUP	-	Não	0,6049
	-	-	INF	Não	0,0543
	-	-	SUP	Não	0,4578

Para esta análise, os resultados indicam que Número de Mudanças pode ser um fator impactante para o surgimento de *God Class* em sistemas grandes (Tamanho do Sistema *SUPERIOR*). Além disso, há o indicativo que também pode ser um diferencial quando são considerados sistemas com poucos colaboradores (Número de Contribuidores *INFERIOR*).

4.2.4 Análise sobre Tempo de Desenvolvimento agrupado com outro fator contextual

A última análise em relação a *God Class* é baseada em Tempo de Desenvolvimento. Assim como nos outros contextos, inicialmente avaliamos Tempo de Desenvolvimento agrupado com mais um outro fator contextual. Dessa maneira, analisamos os seguintes contextos:

- Tempo de Desenvolvimento x Tamanho do Sistema
- Tempo de Desenvolvimento x Número de Mudanças
- Tempo de Desenvolvimento x Número de Contribuidores

Como pode ser visto na Tabela 4.6, para os contextos relacionados a Tempo de Desenvolvimento, há diferenças entre as distribuições de *percentClassesWithGodClass* em quase todos os contextos estudados em que este alterna. A única exceção é quando Tamanho do Sistema assume valor *SUPERIOR*.

Tabela 4.6: Resultado da combinação de Tempo de Desenvolvimento para *God Class*

Tempo de Desenvolvimento alterna	**Fatores que permanecem com classificação fixa**			**H0GC é falsa**	***p-value***
	Tamanho do sistema	Nº de Mudanças	Nº de Contribuidores		
INF X SUP	INF	-	-	Sim	0,0054
	SUP	-	-	Não	0,0609
	-	INF	-	Sim	0,0161
	-	SUP	-	Sim	0,0241
	-	-	INF	Sim	0,0233
	-	-	SUP	SIm	0,0152

Isso indica que, agrupado com somente com mais um fator contextual do estudo, Tempo de Desenvolvimento é impactante na incidência de *God Class*. Exceto em Sistemas grandes (Tamanho do Sistema *SUPERIOR*), em que a hipótese nula não pode ser descartada.

4.3 Brain Class

Para *Brain Class*, a variável criada foi *percentClassesWithBrainClass*, responsável por retratar o percentual de classes com esse *code smell* em cada software. Dessa forma, comparamos os softwares através da distribuição *percentClassesWithBrainClass* e verificamos a hipótese apresentada abaixo:

"Há diferenças na distribuição do *percentClassesWithBrainClass* entre sistemas de contextos diferentes".

Para essa hipótese, foram adotadas as seguintes hipóteses nula e alternativa:

H0BC: Não há diferença significativa na distribuição de percentClassesWithBrainClass para softwares em contextos diferentes;

HABC: Há diferença significativa na distribuição de percentClassesWithBrainClass para softwares em contextos diferentes;

4.3.1 Análise sobre os fatores contextuais isolados, considerando *Brain Class*

Inicialmente, comparamos através do teste de Mann-Whitney a diferença de distribuição para *percentClassesWithBrainClass* entre sistemas cujo fator contextual tem valor *INFERIOR* e sistemas em que o fator contextual assume valor *SUPERIOR*. Após isso, através do teste de hipótese, avaliamos qual hipótese é aceita levando em conta as duas distribuições.

Dessa forma, como pode ser visto na Tabela 4.7, os fatores contextuais Tamanho do Sistema e Tempo de Desenvolvimento apresentam *p-value* menor que 0,05, indicando que a hipótese nula é falsa. Isso indica que há diferenças significativas na incidência do *code smell Brain Class* comparando sistemas com Tamanho de Sistema *SUPERIOR* e *INFERIOR*. O mesmo ocorre também para a variável Tempo de Desenvolvimento: a incidência do *code smell Brain Class* é significativamente diferente entre sistemas com Tempo de Desenvolvimento *SUPERIOR* e *INFERIOR*.

Para Tempo de Desenvolvimento, também é observado que aqueles que possuem mais tempo têm mais tendência a possuir *Brain Class*, sendo encontrado em 65 dos 201 sistemas (32,33%). Para Tempo de Desenvolvimento *INFERIOR*, apenas 41 dos 218 sistemas (18,80%) possuem *Brain Class*.

Tabela 4.7: Comparação entre *INFERIOR* e *SUPERIOR* dos fatores para *Brain Class*

Fator Contextual	**# de sistemas INF**	**# de sistemas SUP**	**# (%) de sistemas - Fator INF**	**# (%) de sistemas - Fator SUP**	**H0BC é falsa**	***p-value***
Tamanho do Sistema	228	191	23 (10,08%)	83 (43,45%)	Sim	8,00e-12
Nº de Mudanças	179	240	36 (20,11%)	70 (29,16%)	Não	0,0699
Nº de Contribuidores	240	179	68 (28,33%)	38 (21,22%)	Não	0,0819
Tempo de Desenvolvimento	218	201	41 (18,80%)	65 (32,33%)	Sim	0,0014

Devido a apenas Tamanho do Sistema e Tempo de Desenvolvimento apresentarem hipótese nula falsa nesta etapa, iremos aprofundar apenas nestes dois fatores. Desta forma, analisamos primeiramente o impacto do Tamanho do Sistema sobre a incidência de *Brain Class* e por último apresentamos a análise relacionada a Tempo de Desenvolvimento.

4.3.2 Análise sobre Tamanho do Sistema agrupado com outro fator contextual

Nesta seção, analisamos Tamanho do Sistema combinado com apenas um outro fator contextual por vez. Dessa maneira, avaliamos o impacto sobre *percentClassesWithBrainClass* em relação a:

- Tamanho do Sistema X Número de Mudanças;
- Tamanho do Sistema X Número de Contribuidores;
- Tamanho do Sistema X Tempo de Desenvolvimento.

A mesma estratégia de combinação que foi utilizada anteriormente com outros *code smells*, é também vista aqui, em que focamos somente naqueles contextos em que Tamanho do Sistema alterna e outro fator permanece com a classificação *INFERIOR* ou *SUPERIOR*. Sendo assim, são analisados 2 casos dentre as combinações possíveis para cada combinação.

Como pode ser observado na Tabela 4.8, em todas as comparações, a hipótese nula é falsa. Houve diferenças significativas entre as distribuições quando qualquer um dos outros 3 fatores assumiu valor *INFERIOR* ou valor *SUPERIOR*.

Tabela 4.8: Resultado da combinação de Tamanho do Sistema para *Brain Class*

Tamanho do sistema alterna	**Fatores que permanecem com classificação fixa**			**H0BC é falsa**	***p-value***
	Número de Mudanças	Número de Contribuidores	Tempo de Desenvolvimento		
INF X SUP	INF	-	-	Sim	0,0016
	SUP	-	-	Sim	1,60e-09
	-	INF	-	Sim	2,49e-05
	-	SUP	-	Sim	4,09e-08
	-	-	INF	Sim	2,25e-06
	-	-	SUP	Sim	1,33e-06

Dessa maneira, quando comparado com apenas mais um fator contextual, Tamanho do Sistema é impactante. Ou seja, Tamanho do Sistema irá gerar diferenças no percentual de classes com *Brain Class* mesmo em cenários de contextos diferentes.

4.3.3 Análise sobre Tempo de Desenvolvimento agrupado com outro fator contextual

Assim como fizemos na análise com Tamanho do Sistema, agrupamos Tempo de Desenvolvimento com outros fatores para que seja possível realizar análise sobre diferentes perspectivas em relação a este fator contextual. Inicialmente, agrupamos Tempo de Desenvolvimento com mais um fator contextual.

Considerando os outros três presentes neste estudo, é possível realizar análises da seguinte maneira:

- Tempo de Desenvolvimento x Tamanho do Sistema
- Tempo de Desenvolvimento x Número de Mudanças
- Tempo de Desenvolvimento x Número de Contribuidores

Na tabela 4.9 abaixo, é possível visualizar o resultado dessas análises. Como pode ser visto, quando Tempo de Desenvolvimento alterna e Tamanho do Sistema assumem valor *INFERIOR*, a hipótese nula não pode ser rejeitada (*p-value* > 0,05), indicando que não há diferenças significativas entre as distribuições de *percentClassesWithBrainClass* neste contextos.

Já para o contexto em que Tempo de Desenvolvimento alterna e Tamanho do Sistema é *SUPERIOR*, a hipótese nula é falsa, indicando que há diferença entre as distribuições referentes a *percentClassesWithBrainClass*.

Tanto em relação a Número de Mudanças, quanto Número de Contribuidores, a hipótese nula é falsa (*p-value* <0,05) independente se o valor do fator contextual é *INFERIOR* ou *SUPERIOR*.

Tabela 4.9: Resultado da combinação de Tempo de Desenvolvimento para *Brain Class*

Tempo de Desenvolvimento alterna	**Fatores que permanecem com classificação fixa**			**H0BC é falsa**	***p-value***
	Tamanho do sistema	Nº de Mudanças	Nº de Contribuidores		
INF X SUP	INF	-	-	Não	0,0573
	SUP	-	-	Sim	0,0108
	-	INF	-	Sim	0,0135
	-	SUP	-	Sim	0,0398
	-	-	INF	Sim	0,0448
	-	-	SUP	Sim	0,0119

Portanto, Tempo de Desenvolvimento indica ser um fator contextual impactante em relação a *Brain Class* quando agrupado com outro fator contextual, exceto no cenário em que Tamanho do Sistema assume valor *INFERIOR*. Ou seja, isso indica

que o tempo que o software leva para ser desenvolvido é relevante para a incidência de *Brain Class*, exceto no cenário em que o sistema é pequeno.

4.4 *Complex Method*

Em relação a *Complex Method*, criamos a variável *percentClassesWithComplexMethod* que apresenta o percentual de classes com *Complex Method* dentro de cada software. Sendo assim, comparamos a distribuição de *percentClassesWithComplexMethod* entre diversos contextos para verificar a hipótese apresentada abaixo:

"Há diferenças na distribuição do *percentClassesWithComplexMethod* entre sistemas de contextos diferentes".

Para essa hipótese, foram adotadas as seguintes hipóteses nula e alternativa:

H0CM: Não há diferença significativa na distribuição de percentClassesWithComplexMethod para softwares em contextos diferentes;

HACM: Há diferença significativa na distribuição de percentClassesWithComplexMethod para softwares em contextos diferentes;

4.4.1 Análise sobre os fatores contextuais isolados, considerando Complex Method

Assim como foi feito com os outros *code smells*, utilizamos o teste de Mann-Whitney para verificar a diferença na distribuição de *percentClassesWithComplexMethod* entre sistemas cujo fator contextual tem valor *INFERIOR* e sistemas em que assume valor *SUPERIOR*.

Como pode ser visto na tabela 4.10, não há diferença significativa em nenhum dos fatores contextuais. Dessa forma, a incidência de *Complex Method* não está ligada a nenhum dos fatores contextuais estudados.

Tabela 4.10: Comparação entre *INFERIOR* e *SUPERIOR* dos fatores para *Complex Method*

Fator Contextual	**# de sistemas INF**	**# de sistemas SUP**	**# (%) de sistemas - Fator INF**	**# (%) de sistemas - Fator SUP**	**H0CM é falsa**	***p-value***
Tamanho do Sistema	228	191	147 (64,47%)	163 (85,34%)	Não	0,2428
Nº de Mudanças	179	240	114 (63,68%)	196 (81,66%)	Não	0,0898
Nº de Contribuidores	240	179	175 (72,91%)	135 (75,41%)	Não	0.3864
Tempo de Desenvolvimento	218	201	161 (73,85%)	149 (74,12%)	Não	0.1719

Como não há um fator contextual em destaque para *Complex Method*, para efeitos de simplicidade, não nos aprofundaremos no debate em relação a *Complex Method*, pois, realizamos os testes relacionados e observamos que, de uma forma geral, há raras diferenças significativas quando combinado com outros fatores, reforçando o que foi observado individualmente.

4.5 *Long Method*

Considerando *Long Method*, criamos a variável *percentClassesWithLongMethod* que representa o percentual de classes que possuem *Long Method* dentro de cada software. Desta forma, comparamos a distribuição de *percentClassesWithLongMethod* entre contextos diferentes, verificando a hipótese a seguir:

"Há diferenças na distribuição do *percentClassesWithLongMethod* entre sistemas de contextos diferentes".

Abaixo pode ser visualizado as hipóteses nula e alternativa:

H0LM: Não há diferença significativa na distribuição de percentClassesWithLongMethod para softwares em contextos diferentes;

HALM: Há diferença significativa na distribuição de percentClassesWithLongMethod para softwares em contextos diferentes;

4.5.1 Análise sobre os fatores contextuais isolados, considerando *Long Method*

Como ponto de partida do processo, foi comparada a distribuição de *percentClassesWithLongMethod* entre os sistemas de classificação *INFERIOR* e *SUPERIOR*

para cada um dos fatores contextuais apresentados no estudo. Através do teste de Mann-Whitney foi constatado que a hipótese nula é falsa, indicando que há diferença entre essas distribuições para o Tamanho do Sistema, no qual apresenta *p-value* igual a 0,0393, como pode ser visto na Tabela 4.11.

Em relação ao número de sistemas com *Long Method*, aqueles que possuem Tamanho do Sistema SUPERIOR são mais impactados, indicando que 85,34% dos sistemas apresentam *Long Method*. Apesar disso, há pouca diferença em relação aos sistemas que possuem Tamanho do Sistema *INFERIOR*, sendo que 60,08% dos sistemas possuem *Long Method*.

Tabela 4.11: Comparação entre *INFERIOR* e *SUPERIOR* dos fatores para *Long Method*

Fator Contextual	**# de sistemas INF**	**# de sistemas SUP**	**# (%) de sistemas - Fator INF**	**# (%) de sistemas - Fator SUP**	**H0LM é falsa**	***p-value***
Tamanho do Sistema	228	191	137 (60,08%)	163 (85,34%)	Sim	0,0393
Nº de Mudanças	179	240	117 (65,36%)	70 183 (76,25%)	Não	0,6363
Nº de Contribuidores	240	179	171 (71,25%)	129 (72,06%)	Não	0,3550
Tempo de Desenvolvimento	218	201	160 (73,39%)	140 (69,65%)	Não	0,5354

Na próxima sessão, nos aprofundamos nas análises focadas para o Tamanho do Sistema e avaliamos o impacto deste em relação a *Long Method*.

4.5.2 Análise sobre Tamanho do Sistema agrupado com outro fator contextual

Agrupamos Tamanho do Sistema com outro fator contextual, levando em conta os contextos em que Tamanho do Sistema alterna para comparar *INFERIOR* com *SUPERIOR*, enquanto o outro fator contextual assume uma classificação fixa. Dessa forma, analisamos a distribuição de *percentClassesWithLongMethod* considerando os seguintes contextos:

- Tamanho do Sistema X Número de Mudanças;
- Tamanho do Sistema X Número de Contribuidores;
- Tamanho do Sistema X Tempo de Desenvolvimento.

Como pode ser visto na Tabela 4.12, para Tamanho do Sistema x Número de Mudanças não pode ser comprovada diferença entre *percentClassesWithLongMethod* quando

Número de Mudanças assume valor *INFERIOR* e também quando assume valor *SUPERIOR*. Para os contextos em que são considerados Tamanho do Sistema X Número de Contribuidores é encontrado o mesmo cenário, não podendo descartar a hipótese nula em ambos os casos.

Para Tamanho do Sistema x Tempo de Desenvolvimento, não pode ser descartado a hipótese nula quando Tamanho do Sistema alterna e Tempo de Desenvolvimento é *INFERIOR*. Porém a hipótese nula é falsa quando Tempo de Desenvolvimento é *SUPERIOR*, o que indica que há diferenças entre as distribuições de *percentClassesWithLongMethod* para Tamanho do Sistema *INFERIOR* e *SUPERIOR*

Tabela 4.12: Resultado da combinação de Tamanho do Sistema para *Long Method*

Tamanho do sistema alterna	**Fatores que permanecem com classificação fixa**			**H0LM é falsa**	***p-value***
	Número de Mudanças	Número de Contribuidores	Tempo de Desenvolvimento		
INF X SUP	INF	-	-	Não	0,0709
	SUP	-	-	Não	0,3584
	-	INF	-	Não	0,0709
	-	SUP	-	Não	0,3001
	-	-	INF	Não	0,3337
	-	-	SUP	Sim	0,0419

Dessa forma, para *Long Method*, quando considerando Tamanho do Sistema agrupado com apenas mais um fator agrupado, há impacto em apenas um contexto, em que Tamanho do Sistema alterna e Tempo de Desenvolvimento é *SUPERIOR*, indicando que, quando o sistema levar muito tempo para ser desenvolvido, o tamanho que este tem será determinante para a incidência de *Long Method*.

4.6 *Feature Envy*

Para *Feature Envy*, criamos *percentClassesWithFeatureEnvy*, responsável por retratar o percentual de classes com *Feature Envy* em cada software. Após isso comparamos a distribuição *percentClassesWithFeatureEnvy* entre softwares de contextos diferentes para verificarmos a hipótese apresentada abaixo:

"Há diferenças na distribuição do *percentClassesWithFeatureEnvy* entre sistemas de contextos diferentes".

Para essa hipótese, foram adotadas as seguintes hipóteses nula e alternativa:

H0FE: Não há diferença significativa na distribuição de *percentClassesWithFeatureEnvy* para softwares em contextos diferentes;

HAFE: Há diferença significativa na distribuição de *percentClassesWithFeatureEnvy* para softwares em contextos diferentes;

4.6.1 Análise sobre os fatores contextuais isolados, considerando *Feature Envy*

Com o intuito de analisar o impacto de *Feature Envy*, comparamos a distribuição *percentClassesWithFeatureEnvy* entre os sistemas classificados como *INFERIOR* e *SUPERIOR* para cada um dos fatores contextuais. Após este passo, testamos a hipótese nula para validar as diferenças entre as distribuições. Como pode ser visto na Tabela 4.13, após a aplicação do teste de Mann-Whitney, a hipótese nula é descartada apenas para Tamanho do Sistema, cujo o *p-value* é menor que 0,05.

Observando o número de sistemas com *Feature Envy*, é possível visualizar que, apesar de serem poucos sistemas, aqueles que possuem Tamanho SUPERIOR contém mais *Feature Envy* do que aqueles que possuem Tamanho *INFERIOR*. Para os de Tamanho *SUPERIOR* são 36 de 191 (18,84%), enquanto os de Tamanho *INFERIOR* são 6 de 228 (2,63%)).

Tabela 4.13: Comparação entre *INFERIOR* e *SUPERIOR* dos fatores para *Feature Envy*

Fator Contextual	**# de sistemas INF**	**# de sistemas SUP**	**# (%) de sistemas - Fator INF**	**# (%) de sistemas - Fator SUP**	**H0FE é falsa**	***p-value***
Tamanho do Sistema	228	191	6 (2,63%)	36 (18,84%)	Sim	8,11e-08
Nº de Mudanças	179	240	13 (7,26%)	29 (12,08%)	Não	0,1264
Nº de Contribuidores	240	179	23 (9,58%)	19 (10,61%)	Não	0,7477
Tempo de Desenvolvimento	218	201	27 (12,38%)	15 (7,46%)	Não	0,0687

Seguindo a estratégia adotada na análise dos outros *code smells*, nos aprofundamos apenas no fator contextual referente a Tamanho do Sistema, que o fator contextual que apresenta indícios de que pode gerar algum impacto.

4.6.2 Análise sobre Tamanho do Sistema agrupado com outro fator contextual

Agrupamos Tamanho do Sistema com outro fator contextual, considerando para teste de hipótese os contextos em que Tamanho do Sistema alterna para efeitos de comparação entre *INFERIOR* e *SUPERIOR*. Em relação ao outro fator contextual, este assume uma determinada classificação. Dessa forma, analisamos a distribuição de *percentClassesWithFeatureEnvy* considerando os seguintes contextos:

- Tamanho do Sistema X Número de Mudanças;
- Tamanho do Sistema X Número de Contribuidores;
- Tamanho do Sistema X Tempo de Desenvolvimento.

Após a aplicação do teste, é possível identificar que a hipótese nula é falsa em todos os contextos testados. Independente do valor assumido (*INFERIOR* ou *SUPERIOR*) por Número de Mudanças, Número de Contribuintes e Tempo de Desenvolvimento, ao alterar Tamanho do Sistema, haverá diferenças na distribuição de *percentClassesWithFeatureEnvy*, como pode ser observado na Tabela 4.14.

Tabela 4.14: Resultado da combinação de Tamanho do Sistema para *Feature Envy*

Tamanho do sistema alterna	**Fatores que permanecem com classificação fixa**			**H0FE é falsa**	***p-value***
	Número de Mudanças	Número de Contribuidores	Tempo de Desenvolvimento		
INF X SUP	INF	-	-	Sim	0,0056
	SUP	-	-	Sim	8,74e-06
	-	INF	-	Sim	0,0002
	-	SUP	-	Sim	4,92e-05
	-	-	INF	Sim	1,33e-05
	-	-	SUP	Sim	0,0014

Sendo assim, considerando estes contextos, Tamanho do Sistema se mostra influente em relação a incidência de *Feature Envy*, impactando em diferentes contextos, independente do do valor assumido pelo outro fator (se *SUPERIOR* ou *INFERIOR*).

4.7 *Brain Method*

Em relação a *Brain Method*,foi criada a variável *percentClassesWithBrainMethod* que apresenta o percentual de classes com *Brain Method* dentro de cada software. Sendo assim, comparamos a distribuição de *percentClassesWithBrainMethod* entre diversos contextos para verificar a hipótese apresentada abaixo:

"Há diferenças na distribuição do *percentClassesWithBrainMethod* entre sistemas de contextos diferentes".

Para essa hipótese, foram adotadas as seguintes hipóteses nula e alternativa:

H0BM: Não há diferença significativa na distribuição de percentClassesWithBrainMethod para softwares em contextos diferentes;

HABM: Há diferença significativa na distribuição de percentClassesWithBrainMethod para softwares em contextos diferentes;

4.7.1 Análise sobre os fatores contextuais isolados, considerando *Brain Method*

Comparamos através do teste de Mann-Whitney a diferença de distribuição para *percentClassesWithBrainMethod* entre sistemas cujo fator contextual tem valor *INFERIOR* e sistemas em que o fator contextual assume valor *SUPERIOR*. Após este passo, através do teste de hipótese, avaliamos qual hipótese é aceita levando em conta as duas distribuições.

Para *Brain Method*, após o teste através do teste de Mann-Whitney, foi indicado que há diferenças na distribuição entre os sistemas (*p-value* <0,05) para os fatores Tamanho do Sistema, Número de Mudanças, como pode ser visto na Tabela 4.15.

Em relação a quantidade de sistemas com *Brain Method* por classificação de cada fator, há indícios de que aqueles que possuem Tamanho do Sistema *SUPERIOR* apresentam mais *code smells* do tipo *Brain Method*: 121 dos 191 (63,68%) sistemas possuem *Brain Method*. Para Tamanho do Sistema *INFERIOR*, 71 dos 228 (31,14%) sistemas possuem *Brain Method*.

Para Número de Mudanças, é observado que aqueles que possuem mais mudanças têm maior tendência a possuir *Brain Method*, sendo que em 127 dos 240 (52,91%) sistemas há pelo menos uma instância de *Brain Method*. Para Número de Mudanças *INFERIOR*, 65 dos 179 (36,31%) sistemas possuíam *Brain Method*.

Tabela 4.15: Comparação entre *INFERIOR* e *SUPERIOR* dos fatores para *Brain Method*

Fator Contextual	**# de sistemas INF**	**# de sistemas SUP**	**# (%) de sistemas - Fator INF**	**# (%) de sistemas - Fator SUP**	**H0BM é falsa**	***p-value***
Tamanho do Sistema	228	191	71 (31,14%)	121 (63,68%)	Sim	5,01e-05
Nº de Mudanças	179	240	65 (36,31%)	127 (52,91%)	Sim	0,0168
Nº de Contribuidores	240	179	113 (47,08%)	79 (44,13%)	Não	0,6864
Tempo de Desenvolvimento	218	201	94 (43,11%)	98 (48,75%)	Não	0,2057

Apresentaremos nas subseções seguintes, os resultados encontrados para Tamanho do Sistema e Número de mudanças, ao combinar com outros fatores contextuais.

4.7.2 Análise sobre Tamanho do Sistema agrupado com outro fator contextual

Seguindo a estratégia definida, iniciamos analisando Tamanho do Sistema agrupado com os outros fatores contextuais, considerando apenas um outro fator contextual por vez. Dessa maneira, Como pode ser visto na Tabela 4.16, em todos os contextos criados a partir destes agrupamentos, independente do fator contextual e da classificação assumida, as hipóteses nulas são falsas, apontando para um cenário em que há diferenças significativas entre a distribuição de *percentClassesWithBrainMethod* independente da classificação (*INFERIOR* ou *SUPERIOR*) e do fator contextual do estudo considerado.

Tabela 4.16: Resultado da combinação de Tamanho do Sistema para *Brain Method*

Tamanho do sistema alterna	**Fatores que permanecem com classificação fixa**			**H0BM é falsa**	***p-value***
	Número de Mudanças	Número de Contribuidores	Tempo de Desenvolvimento		
INF X SUP	INF	-	-	Sim	0,0092
	SUP	-	-	Sim	0,0072
	-	INF	-	Sim	0,0026
	-	SUP	-	Sim	0,0057
	-	-	INF	Sim	0,0497
	-	-	SUP	Sim	0,0001

Dessa maneira, Tamanho do Sistema indica ser um fator impactante na incidência de *Brain Method* quando relacionado com mais um fator contextual, independente de qual classificação é assumida por este.

4.7.3 Análise sobre Número de Mudanças agrupado com outro fator contextual

Para a análise referente a Número de Mudanças, agrupamos com outro fator contextual, sendo que Número de Mudanças alterna para comparar *INFERIOR* com *SUPERIOR*, enquanto o outro fator contextual assume uma determinada classificação. Como pode ser visto na Tabela 4.17, que o único contexto em que há diferença significativa (hipótese nula falsa), se dá quando Números de Mudanças alterna e Número de Contribuidores é *SUPERIOR*.

Tabela 4.17: Resultado da combinação de Número de Mudanças para *Brain Method*

Nº de Mudanças alterna	**Fatores que permanecem com classificação fixa**			**H0BM é falsa**	***p-value***
	Tamanho do sistema	Número de Contribuidores	Tempo de Desenvolvimento		
	INF	-	-	Não	0,1696
	SUP	-	-	Não	0,2501
INF X SUP	-	INF	-	Não	0,9863
	-	SUP	-	Sim	7,40e-05
	-	-	INF	Não	0,0542
	-	-	SUP	Não	0,1621

Tanto para Número de Contribuidores *INFERIOR*, quanto para os contextos que são formados por Tamanho do Sistema e Tempo de Desenvolvimento, a hipótese nula não pode ser descartada. Dessa forma, o único cenário em que o Número de Mudanças feitas no sistema pode gerar a incidência de *Brain Method*, é aquele em que há vários contribuidores no projeto.

Capítulo 5

Discussão

Nesta seção, discutimos os principais *findings* do estudo, baseado no conjunto de análises realizadas. Na seção 5.1 sintetizamos as observações relacionadas aos fatores contextuais e o impacto na incidência de cada um dos *code smells* investigados. Na Seção 5.2, discutimos a questão a partir da observação de como cada fator impacta na incidência de um grupo de *code smells*. Além disso, essa seção propõe conjecturas sobre as observações realizadas, visando produzir *insights* para novas análises. Na seção 5.3, abordamos a relação entre o estudo apresentado e a necessidade de incorporar elementos de contexto às ferramentas de detecção de *code smells*. O tema vem sendo discutido e o estudo apresentado nesta dissertação cria evidências empíricas sobre as relações entre alguns fatores contextuais e alguns tipos de *code smells*.

5.1 Influência de fatores contextuais sobre cada *code smell* estudado

Nesta seção, discutimos a relação dos fatores contextuais com relação a cada *code smell*, de forma independente.

Data Class. Considerando *Data Class*, é possível observar que há um impacto do Tamanho do Sistema sobre a incidência do *code smell*. Isso é reforçado quando analisamos este fator em conjunto com os outros fatores de contexto. Independente do contexto em que foi analisado, Tamanho do Sistema gera uma diferença significativa em relação ao percentual de classes com *Data Class*. Para os outros fatores contextuais estudados, não foi possível observar diferença significativa na incidência de *Data Class*.

God Class. *God Class* foi o *code smell* mais impactado por diferentes fatores contextuais. Observamos diferença significativa na incidência do *code smell God Class* para 3 dos 4 fatores contextuais estudados. Tamanho do Sistema, Número de Mudanças e Tempo de Desenvolvimento são fatores em que foi observada diferença

significativa no percentual de classes com *God Class*, comparando as amostras com valores *SUPERIOR* e *INFERIOR*, para cada um destes fatores. Ao analisar como cada fator impacta em *God Class*, é possível perceber que cada um interfere de uma maneira diferente.

Tamanho do Sistema impacta no percentual de *God Class* em praticamente todos os cenários, exceto quando houve poucas mudanças nos sistemas, ou seja, a variável Número de Mudanças é *INFERIOR*. Ainda sim, este é apenas um contexto dentro dos outros analisados que indicam uma considerável influência em relação a *God Class*.

Em relação a Número de Mudanças, apesar de isoladamente apresentar impacto na incidência de *God Class*, só encontramos diferenças significativas em dois contextos específicos. O primeiro contexto aponta para um impacto do Número de Mudanças quando os sistemas são grandes, ou seja, o fator contextual Tamanho de Sistema é *SUPERIOR*. Já o segundo contexto, indica impacto do Número de Mudanças quando o fator contextual Número de Contribuidores é *INFERIOR*, isto é, quando há poucos contribuidores no projeto.

Tempo de Desenvolvimento, assim como no resultado que foi encontrado para Tamanho do Sistema, também impacta no percentual de *God Class* em quase todos os cenários, exceto quando Tamanho do Sistema é grande (*SUPERIOR*). Dessa forma, se apresenta como um fator contextual importante devido a quantidade de cenários encontrados em que há impacto do fator.

Como dito anteriormente, *God Class* é o *code smell* que mais sofre impacto pelos fatores contextuais estudados. Isso vai de encontro a atenção que é dada pela comunidade, pois, este *code smell* é amplamente estudado por diversos autores, que buscam compreender desde o seu processo de detecção, até o efeito sobre a qualidade do código (M Santos et al., 2014; Santos et al., 2017; Alkharabsheh, 2021; Alkharabsheh et al., 2022). Futuramente, iremos nos aprofundar neste *code smell*, com o intuito de entender melhor em que os fatores podem estar influenciando em relação a ocorrência de *God Class*.

Brain Class. Para *Brain Class*, os testes executados apontam que este pode ser influenciado por Tamanho do Sistema e Tempo de Desenvolvimento. Ao analisar cada fator combinado com outros fatores contextuais, Tamanho do Sistema é impactante em qualquer contexto, gerando diferenças no percentual de *Brain Class* em todos os testes. Considerando Tempo de Desenvolvimento, encontramos um cenário similar, sendo que em quase todos os testes também foram encontradas diferenças no percentual de *Brain Class*, com exceção apenas do contexto em que Tamanho do Sistema é *INFERIOR*.

Complex Method. De todos os 7 *code smells* presentes no estudo, *Complex Method* foi o único que não apresentou diferença nos percentuais de classes afetadas para nenhum dos fatores contextuais isolados. Dessa forma, não aprofundamos a discussão neste *code smell*, pois apesar de existir diferenças em alguns casos quando

combinados, além de serem raros, não pode ser traçado nenhum padrão específico.

Long Method. Para o *code smell Long Method*, observamos o impacto apenas do fator Tamanho do Sistema. Ao analisar este fator, combinado com outros fatores contextuais, assim como aconteceu com outros *code smells*, Tamanho do Sistema impactou em praticamente todos os contextos, exceto quando o Tempo de Desenvolvimento do Software era grande.

Feature Envy. Os estudos realizados com *Feature Envy*, indicam um impacto do Tamanho do Sistema no percentual de classes detectadas com *Feature Envy*. Além disso, ao analisar de forma combinada com outros fatores contextuais, foi percebido um impacto em todos os cenários analisados, mostrando que este fator irá gerar diferenças na incidência de *Feature Envy*, independente do valor assumido pelos outros fatores.

Brain Method. Por último, *Brain Method* foi impactado por Tamanho do Sistema e Número de Mudanças. Tamanho do Sistema, quando combinado com outros fatores contextuais, foi capaz de impactar no percentual de classes com *Brain Method* em todos os cenários. Já Número de Mudanças só não impactou no contexto em que o número de contribuidores é alto. Em todos os outros foi capaz de gerar diferenças na incidência de Brain Method.

5.2 Influência de fatores contextuais sobre o conjunto dos *code smells* estudados

Nesta seção, discutimos a relação dos fatores contextuais, considerando o conjunto de *code smells* investigado.

Após o resumo dos resultados encontrados, podemos realizar algumas observações com o intuito de elucidar a resposta à questão de pesquisa apresentada. De uma forma geral, vimos que os fatores contextuais não só impactam na incidência dos *code smells*, como também impactam de forma diferente, de acordo ao *code smell* e ao fator que está sendo relacionado.

Por exemplo, Tamanho do Sistema é um fator importante, pois afeta praticamente todos os *code smells* do estudo. Porém, afetou de forma diferente alguns *code smells* em relação a outros. Em algumas situações, como aconteceu com *Data Class, Brain Class, Feature Envy* e *Brain Method*, Tamanho do Sistema impacta na incidência destes *code smells*, independente do contexto em que é observado.

Em relação a *God Class*, Tamanho do Sistema gera diferenças no percentual de *God Class* em quase todos os contextos, com exceção apenas quando Número de Mudanças é *INFERIOR*. Para *Long Method*, Tamanho do Sistema impacta somente em um contexto específico quando Tempo de Desenvolvimento do Sistema é grande.

Sendo assim, é possível ver que Tamanho do Sistema impacta com muita intensidade em 4 *code smells*. Mas essa intensidade é diferente para *God Class*, em que já não há

total impacto e também *Long Method*, em que só é capaz de impactar em cenários específicos.

Por fim, Tamanho do Sistema só não impactou em *Complex Method*. Este *code smell*, inclusive, não foi impactado por nenhum fator contextual. Algumas conjecturas podem ser feitas. Pode existir a possibilidade de, por característica, ter *code smells* que podem não sofrer influência de nenhum fator contextual. Eles podem surgir de maneira natural, independente do contexto em que o software é desenvolvido. Há também a possibilidade de ele só não ser impactado por esses 4 fatores contextuais que selecionamos, mas ser impactados por outros fatores contextuais diferentes.

Além das observações sobre Tamanho do Sistema, os outros fatores contextuais também impactam de forma mais significativa na incidência de alguns *code smells*. O fator contextual Tempo de Desenvolvimento impacta na incidência de *God Class* e *Brain Class*. Para esses dois *code smells*, Tempo de Desenvolvimento impacta em quase todos os contextos. Isto significa que, apesar de não ter impactado em uma grande quantidade de *code smells* diferentes, quando gerou impacto em um *code smell*, foi capaz de gerar em diversos contextos.

O fator contextual Número de Mudanças gera diferenças na incidência de *God Class* e *Brain Method*. Porém, Número de Mudanças só gerou impacto em contextos específicos, indicando que este fator contextual pode depender de outras características para poder influenciar de fato em algum *code smell*.

Dos 4 fatores estudados, Número de Contribuidores não indicou impactar em nenhum *code smell* do estudo. Ainda assim, há questões que precisam ser aprofundadas sobre esse fator contextual. Como primeiro ponto, nós conjecturamos que Número de Contribuidores pode impactar em *code smells* diferentes deste que levamos em conta no estudo. Além disso, em algumas situações, Número de Contribuidores gerou a situação ideal para que Número de Mudanças se tornasse um fator impactante.

Em relação a *Brain Method*, por exemplo, o único contexto em que Número de Mudanças gerou diferenças significativas, foi quando o Número de Contribuidores assume valor SUPERIOR, isto é, há impacto no percentual de classes com *Brain Method* quando Número de Mudanças alterna, porém, isso só acontece quando há muitos contribuidores no projeto. Um cenário parecido é encontrado para *God Class*, em que Número de Mudanças só impacta em dois contextos. Um destes cenários, é quando Número de Contribuidores assume valor INFERIOR.

Sendo assim, apesar de não ser capaz de gerar impacto nos *code smells* estudados quando alterna, Número de Contribuidores pode contribuir para que Número de Mudanças seja um fator impactante. Isso indica que este fator contextual merece atenção futura para entendermos este fenômeno.

5.3 Principais Achados

Este trabalho, como um estudo exploratório, apresenta um conjunto empírico com o intuito de discutir um problema importante: o papel do contexto na qualidade do software.

Nós apresentamos várias relações entre fatores contextuais e alguns tipos diferentes de *code smells*, evidenciando que há, no mínimo, algum indício de que a qualidade do projeto software de uma forma geral, pode ser afetada pelo contexto em que o software está sendo construído (Dósea et al., 2018).

Nessa perspectiva, adotar o conceito de *code smell* na prática do desenvolvimento de software, sem levar em conta o contexto em que este é desenvolvido, pode levar a resultados enviesados, ou então até distorcidos da realidade de como os *code smells* afetam ou surgem no sistemas. Desta forma, levar em conta os fatores contextuais inerentes ao software pode ajudar a compreender como estes impactam na incidência de *code smells*. Entender esta relação pode nos aproximar de uma adoção mais assertiva do conceito de *code smells* pela indústria do desenvolvimento de software, tanto na perspectiva teórica, quanto prática.

O resultado deste trabalho corrobora com o que já foi encontrado sobre a influência dos fatores contextuais na qualidade do software como um todo. Ogheneovo et al. (2014) indicaram que o Tamanho do Sistema, impacta na complexidade do software. Canfora e Cimitile (2001) apontaram que, softwares com muito tempo de desenvolvimento, tendem a oferecer dificuldade na manutenção. Behnamghader et al. (2017) concluíram que o Número de Mudanças realizadas no software (*commits*), podem degradar a qualidade do sistema. Além disso, Zhang et al. (2013) também já apontaram o impacto de 6 fatores contextuais sobre métricas de manutenção de software.

O que apresentamos em nosso estudo, é mais um indício sobre a relevância dos fatores contextuais, desta vez sobre outro parâmetro para a qualidade de software, que são os *code smells*.

5.4 Ameaças à validade

Uma das ameaças à validade do trabalho diz respeito à escolha das ferramentas de detecção de *code smells*. O Repository Miner foi utilizado para extrair as informações referentes aos 7 *code smells* do estudo. Para extrair os fatores contextuais, utilizamos a ferramenta Pydriller. Tanto o software utilizado para detecção de *code smells*, quanto o software utilizado para identificação dos fatores de contexto limitam a extração de dados pelas suas heurísticas. Devido às limitações de recursos, optamos por enfatizar o extenso conjunto de dados a ser analisados, em detrimento do uso de várias ferramentas de extração de dados. A escolha de utilizar apenas uma ferramenta para extrair os *code smells* e utilizar apenas uma ferramenta para extrair os fatores contextuais se dá pela dificuldade de executar várias ferramentas no conjunto de softwares que utilizamos no nosso estudo, pois, com o Pydriller,

extraímos dados dos fatores contextuais de 452.843 softwares softwares. Com o Repository Miner, baixamos e executamos a extração de *code smells* de mais de 400 softwares. Executar essas ações mais uma vez com outras ferramentas seria inviável de acordo ao tempo disponível. Um outro aspecto que deve ser considerado com relação à escolha das ferramentas é que adotamos duas ferramentas que são utilizadas pela comunidade acadêmica (Pecorelli et al., 2020; Amanatidis et al., 2020; de Freitas Farias et al., 2020; Allamanis et al., 2021).

Outra ameaça à validade ocorre em relação à escolha dos *code smells*. A escolha dos *code smells* foi limitada pela adoção do Repository Miner. Entretanto, vale ressaltar que os *code smells* utilizados neste estudo também são amplamente adotados pela comunidade. Santos et al. (2018) identificam os *code smells* adotados como sendo bastante estudados. Pode-se argumentar que outros *code smells* deveriam ser estudados, Entretanto, apesar dos *code smells* que investigamos serem bastante estudados e conhecidos, os estudos feitos não tratam das relações dos *code smells* com fatores contextuais. Além disso, é preciso considerar a necessidade de mitigar os vieses das heurísticas de detecção de *code smells*. Neste sentido, escolhemos uma ferramenta que adota a heurística de Lanza e Marinescu que é utilizado por diversas ferramentas e diversos estudos (Cristina et al., 2005; Marinescu et al., 2010; D'Ambros et al., 2010; Olbrich et al., 2010; Gradišnik et al., 2019).

Pode ser considerado também uma ameaça à validade a escolha dos fatores contextuais. Devido a existência de inúmeros fatores contextuais inerentes ao software, torna-se difícil de mensurar quais são os fatores contextuais ideais para cada cenário de estudo. Além disso, há também dificuldades para extração de muitos outros fatores. Por esse motivo, nos limitamos a escolher aqueles que a ferramenta coletava e já foram trabalhados em outros artigos (Canfora e Cimitile, 2001; Zhang et al., 2013; Ogheneovo et al., 2014; Behnamghader et al., 2017). Como objetivo não é definir quais são os fatores contextuais que impactam em determinado *code smell*, mas sim mostrar que os fatores contextuais devem ser discutidos em relação a *code smells*, acreditamos que estas escolhas não geraram grandes impactos nos resultados apresentados neste estudo.

Por fim, outra ameaça à validade do estudo diz respeito à generalização dos resultados. Algumas das descobertas podem não ser diretamente aplicáveis a diferentes softwares. Devido a natureza do software, é inviável considerar em um trabalho todas características possíveis. Justamente por isso, essa análise é voltada para essa população. Para mitigar, escolhemos um conjunto de softwares que pode ser considerado grande, quando comparado com outros estudos disponíveis na literatura. Por exemplo, em um trabalho de mineração de dados sobre repositórios de software, Fontana et al. Fontana et al. (2015a) utilizaram 74 softwares no seu estudo. Já o Tufano et al. (2017) utilizaram 200 softwares. Neste estudo, nós utilizamos 419 softwares.

Capítulo 6

Conclusões

Neste trabalho, realizamos um estudo exploratório, com 419 softwares recuperados do Github, a partir de um processo filtragem baseado em fatores de contextos de mais de 450 mil softwares. Os fatores de contexto estudados foram Tamanho do Software, Número de Mudanças, Número de Contribuidores e Tempo de Desenvolvimento.

Este estudo contribui para ampliar os dados empíricos sobre a relevância dos fatores contextuais em relação aos *code smells*. Apresentamos evidência de que fatores contextuais podem impactar na incidência de *code smells* de formas diferentes, indicando que, a depender do contexto do software, podem ser esperado mais alguns *code smells* do que outros. Os *code smells* investigados foram: *Brain Class, Brain Method, Complex Method, Data Class, Feature Envy, God Class,* e *Long Method*.

Observamos que tamanho do sistema é o fator que mais tem impacto na incidência de *code smells*. Entretanto, percebemos que, a incidência de *code smells* também depende da combinação de alguns fatores contextuais. Por exemplo, a incidência de *Long Method* em relação ao Tamanho do Sistema é maior quando Tempo de Desenvolvimento for classificado como *SUPERIOR*. A seção 5.3 explora outros cenários observados. Com relação aos *code smells*, notamos que *God Class* é o mais afetado por diferentes fatores contextuais. *Complex Method* não foi afetado por nenhum fator.

Uma outra contribuição do trabalho para a comunidade acadêmica foi a disponibilização de um extenso dataset produzidos durante essa pesquisa. Disponibilizamos um conjunto de dados sobre *code smells* e fatores contextuais de 419 softwares obtidos a partir de mineração de repositórios. Esse conjunto de dados poderá ser utilizado pela comunidade acadêmica para novos estudos que envolvem o contexto do software e o *code smell*.

Como trabalhos futuros, propomos investigar a eficiência das ferramentas de detecção de *code smells* em contextos diferentes. Como já foi apresentado no estudo da Fontana et al. (2015b), conjecturamos que seja necessário realizar ajustes nos *thresholds*, pois, a depender do contexto de software, é possível que as ferramentas

percam eficiência. Para comprovar essa conjectura, é possível utilizar a base atual para medir o quão eficiente são as ferramentas em cada um dos contextos estudados.

Também pretendemos ampliar o conjunto de *code smells* a serem observados. A evolução do trabalho neste sentido é relativamente simples, uma vez que demanda apenas o uso de outras ferramentas de detecção de *code smells*. Outra alternativa a partir do *dataset* produzido, é a investigação de outros fatores contextuais.

Outras oportunidades podem ser consideradas, como a realização de *surveys*, visando identificar, por exemplo, aspectos humanos ou organizacionais relacionados com equipes atuando nos softwares investigados. Dessa forma, consideramos que muitas outras possibilidades de estudos podem ser desenvolvidas a partir deste *dataset* e da estratégia de análise adotada neste trabalho.

Visando apoiar a comunidade científica com novas oportunidades de pesquisa que podem emergir desse conjunto de dados, disponibilizamos informações de fatores contextuais sobre os mais de 450 mil softwares cujas informações foram coletadas. Além disso, também disponibilizamos os softwares analisados neste estudo, bem como as informações sobre os *code smells* detectados. Os dados estão disponíveis no link: `https://zenodo.org/record/7374205#.ZBiFSnbMJD8`

Referências

Ahmed, I., Brindescu, C., Mannan, U. A., Jensen, C., e Sarma, A. (2017). An empirical examination of the relationship between code smells and merge conflicts. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, páginas 58–67. IEEE.

Alkharabsheh, K. (2021). An empirical study on the co-occurrence of design smells in the same software module: God class case study. In *2021 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, páginas 1–6. IEEE.

Alkharabsheh, K., Alawadi, S., Ignaim, K., Zanoon, N., Crespo, Y., Manso, E., e Taboada, J. A. (2022). Prioritization of god class design smell: A multi-criteria based approach. *Journal of King Saud University-Computer and Information Sciences*.

Allamanis, M., Jackson-Flux, H., e Brockschmidt, M. (2021). Self-supervised bug detection and repair. *Advances in Neural Information Processing Systems*, 34:27865–27876.

Amanatidis, T., Mittas, N., Moschou, A., Chatzigeorgiou, A., Ampatzoglou, A., e Angelis, L. (2020). Evaluating the agreement among technical debt measurement tools: building an empirical benchmark of technical debt liabilities. *Empirical Software Engineering*, 25(5):4161–4204.

Arcelli Fontana, F., Mäntylä, M. V., Zanoni, M., e Marino, A. (2016). Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering*, 21(3):1143–1191.

Behnamghader, P., Alfayez, R., Srisopha, K., e Boehm, B. (2017). Towards better understanding of software quality evolution through commit-impact analysis. In *2017 IEEE International conference on software quality, reliability and security (QRS)*, páginas 251–262. IEEE.

Bird, C., Nagappan, N., Murphy, B., Gall, H., e Devanbu, P. (2010). An analysis of the effect of code ownership on software quality across windows, eclipse, and firefox. *Technical report, University of California*.

Brooks, F. e Kugler, H. (1987). *No silver bullet*. April.

Canfora, G. e Cimitile, A. (2001). Software maintenance. In *Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals*, páginas 91–120. World Scientific.

Carmel, E. e Bird, B. J. (1997). Small is beautiful: a study of packaged software development teams. *The Journal of High Technology Management Research*, 8(1):129–148.

Cedrim, D., Garcia, A., Mongiovi, M., Gheyi, R., Sousa, L., de Mello, R., Fonseca, B., Ribeiro, M., e Chávez, A. (2017). Understanding the impact of refactoring on smells: A longitudinal study of 23 software projects. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, páginas 465–475. ACM.

Cosentino, V., Izquierdo, J. L. C., e Cabot, J. (2017). A systematic mapping study of software development with github. *IEEE Access*, 5:7173–7192.

Cristina, M., Radu, M., Mihancea, F., et al. (2005). iplasma: an integrated platform for quality assessment of object-oriented design. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*, páginas 77–80.

D'Ambros, M., Bacchelli, A., e Lanza, M. (2010). On the impact of design flaws on software defects. In *2010 10th International Conference on Quality Software*, páginas 23–31. IEEE.

de Freitas Farias, M. A., de Mendonça Neto, M. G., Kalinowski, M., e Spínola, R. O. (2020). Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary. *Information and Software Technology*, 121:106270.

de Mello, R., Uchôa, A., Oliveira, R., Oizumi, W., Souza, J., Mendes, K., Oliveira, D., Fonseca, B., e Garcia, A. (2019). Do research and practice of code smell identification walk together? a social representations analysis. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, páginas 1–6. IEEE.

de Paulo Sobrinho, E. V., De Lucia, A., e de Almeida Maia, M. (2018). A systematic literature review on bad smells—5 w's: which, when, what, who, where. *IEEE Transactions on Software Engineering*.

Dósea, M., Sant'Anna, C., e da Silva, B. C. (2018). How do design decisions affect the distribution of software metrics? In *Proceedings of the 26th Conference on Program Comprehension*, páginas 74–85.

Fernandes, E., Oliveira, J., Vale, G., Paiva, T., e Figueiredo, E. (2016). A review-based comparative study of bad smell detection tools. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, páginas 1–12.

Fontana, F. A., Ferme, V., e Zanoni, M. (2015a). Towards assessing software architecture quality by exploiting code smell relations. In *Proceedings of the Second International Workshop on Software Architecture and Metrics*, páginas 1–7. IEEE Press.

Fontana, F. A., Ferme, V., Zanoni, M., e Yamashita, A. (2015b). Automatic metric thresholds derivation for code smell detection. In *2015 IEEE/ACM 6th International Workshop on Emerging Trends in Software Metrics*, páginas 44–53. IEEE.

Fontana, F. A., Mariani, E., Mornioli, A., Sormani, R., e Tonello, A. (2011). An experience report on using code smells detection tools. In *2011 IEEE fourth international conference on software testing, verification and validation workshops*, páginas 450–457. IEEE.

Fowler, M. (1999). Refactoring: Improving the design of existing code. In *11th European Conference. Jyväskylä, Finland.*

Gousios, G. (2013). The ghtorent dataset and tool suite. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, páginas 233–236. IEEE.

Gradišnik, M., Beranič, T., Karakatič, S., e Mausaš, G. (2019). Adapting god class thresholds for software defect prediction: A case study. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, páginas 1537–1542. IEEE.

Hassan, A. E. (2008). The road ahead for mining software repositories. In *2008 Frontiers of Software Maintenance*, páginas 48–57. IEEE.

Juristo, N. e Vegas, S. (2009). Using differences among replications of software engineering experiments to gain knowledge. In *2009 3Rd international symposium on empirical software engineering and measurement*, páginas 356–366. IEEE.

Khomh, F., Di Penta, M., e Gueheneuc, Y.-G. (2009). An exploratory study of the impact of code smells on software change-proneness. In *2009 16th Working Conference on Reverse Engineering*, páginas 75–84. IEEE.

Kothari, C. R. (2004). *Research methodology: Methods and techniques*. New Age International.

Lanza, M. e Marinescu, R. (2006). *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media.

Lehman, M. M. (1979). On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1:213–221.

Liu, H., Xu, Z., e Zou, Y. (2018). Deep learning based feature envy detection. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, páginas 385–396.

M Santos, J. A., de Mendonça, M. G., Dos Santos, C. P., e Novais, R. L. (2014). The problem of conceptualization in god class detection: agreement, strategies and decision drivers. *Journal of Software Engineering Research and Development*, 2(1):1–33.

Marinescu, R. (2001). Detecting design flaws via metrics in object-oriented systems. In *Proceedings 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems. TOOLS 39*, páginas 173–182. IEEE.

Marinescu, R. (2002). Measurement and quality in objectoriented design.

Marinescu, R. (2004). Detection strategies: Metrics-based rules for detecting design flaws. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, páginas 350–359. IEEE.

Marinescu, R., Ganea, G., e Verebi, I. (2010). Incode: Continuous quality assessment and improvement. In *2010 14th European Conference on Software Maintenance and Reengineering*, páginas 274–275. IEEE.

Mendes, T., Novais, R., Mendonca, M., Carvalho, L., e Gomes, F. (2017). Repositoryminer-uma ferramenta extensível de mineração de repositórios de software para identificação automática de dívida técnica. *CBSoft 2017-Sessao de Ferramentas ()*, página 12.

Meneely, A. e Williams, L. (2009). Secure open source collaboration: an empirical study of linus' law. In *Proceedings of the 16th ACM conference on Computer and communications security*, páginas 453–462.

Ogheneovo, E. E. et al. (2014). On the relationship between software complexity and maintenance costs. *Journal of Computer and Communications*, 2(14):1.

Olbrich, S. M., Cruzes, D. S., e Sjøberg, D. I. (2010). Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems. In *2010 IEEE international conference on software maintenance*, páginas 1–10. IEEE.

Paiva, T., Damasceno, A., Figueiredo, E., e Sant'Anna, C. (2017). On the evaluation of code smells and detection tools. *Journal of Software Engineering Research and Development*, 5(1):1–28.

Pecorelli, F., Palomba, F., Khomh, F., e De Lucia, A. (2020). Developer-driven code smell prioritization. In *Proceedings of the 17th International Conference on Mining Software Repositories*, páginas 220–231.

Peters, R. e Zaidman, A. (2012). Evaluating the lifespan of code smells using software repository mining. In *2012 16th European Conference on Software Maintenance and Reengineering*, páginas 411–416. IEEE.

Rasool, G. e Arshad, Z. (2015). A review of code smell mining techniques. *Journal of Software: Evolution and Process*, 27(11):867–895.

Santos, J. A., de Mendonça, M. G., e Silva, C. V. (2013). An exploratory study to investigate the impact of conceptualization in god class detection. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, páginas 48–59.

Santos, J. A. M., Rocha-Junior, J. B., e de Mendonça, M. G. (2017). Investigating factors that affect the human perception on god class detection: an analysis based on a family of four controlled experiments. *Journal of Software Engineering Research and Development*, 5(1):1–39.

Santos, J. A. M., Rocha-Junior, J. B., Prates, L. C. L., do Nascimento, R. S., Freitas, M. F., e de Mendonça, M. G. (2018). A systematic review on the code smell effect. *Journal of Systems and Software*, 144:450–477.

Sharma, T., Fragkoulis, M., e Spinellis, D. (2017). House of cards: code smells in open-source c# repositories. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, páginas 424–429. IEEE.

Sharma, T. e Spinellis, D. (2018). A survey on software smells. *Journal of Systems and Software*, 138:158 – 173.

Spadini, D., Aniche, M., e Bacchelli, A. (2018). Pydriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, páginas 908–911.

Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., e Poshyvanyk, D. (2017). When and why your code starts to smell bad (and whether the smells go away). *IEEE Transactions on Software Engineering*, 43(11):1063–1088.

Yamashita, A. e Moonen, L. (2012). Do code smells reflect important maintainability aspects? In *2012 28th IEEE international conference on software maintenance (ICSM)*, páginas 306–315. IEEE.

Yap, B. W. e Sim, C. H. (2011). Comparisons of various types of normality tests. *Journal of Statistical Computation and Simulation*, 81(12):2141–2155.

Zhang, F., Mockus, A., Zou, Y., Khomh, F., e Hassan, A. E. (2013). How does context affect the distribution of software maintainability metrics? In *2013 IEEE International Conference on Software Maintenance*, páginas 350–359. IEEE.

Zöller, N., Morgan, J. H., e Schröder, T. (2020). A topology of groups: What github can tell us about online collaboration. *Technological Forecasting and Social Change*, 161:120291.