Analyzing the Impact of Inter-smell Relations on Software Maintainability

An Empirical Study with Software Product Lines

Júlio Martins
Universidade Federal do Ceará (UFC)
Programa de Pós-Graduação em Computação (PCOMP)
Quixadá, CE, Brasil
juliomserafim@gmail.com

Carla Ilane Moreira Bezerra
Universidade Federal do Ceará (UFC)
Programa de Pós-Graduação em Computação (PCOMP)
Quixadá, CE, Brasil
carlailane@ufc.br

Anderson Uchôa
Departamento de Informática, PUC-Rio
Rio de Janeiro, RJ, Brasil
auchoa@inf.puc-rio.br

ABSTRACT

A Software Product Line (SPL) consists of a systematic reuse strategy to construct systems with less effort as long as they belong to the same family that share the same components and belong to the same domain of Marketplace. In this context, to support large-scale reuse, components of a Software Product Line should be easy to maintain. Thus, developers should be more concerned with anomalies known as code smells and more than that, co-occurrences known as Inter-smell deserve to be further studied to verify their real impact on maintainability in SPL. Thus, this paper conducts a study to investigate the impact of Inter-smell occurrences on maintainability in MobileMedia and Health Watcher SPLs. The results show that the presence of co-occurrences of Inter-smell did not negatively impact the maintenance of MobileMedia and Health Watcher SPLs, unlike results found in other studies in the literature, and even more, our results indicate that the metric Lack of Cohesion of Methods is one of the most important for the maintainability of object-oriented SPLs.

CCS CONCEPTS

• **Software and its engineering → Software product lines; Software Maintenance**;

KEYWORDS

Software product line engineering. Code Smell. Maintainability. Software - QualityControl.

ACM Reference format:
Júlio Martins, Carla Ilane Moreira Bezerra, and Anderson Uchôa. 2019. Analyzing the Impact of Inter-smell Relations on Software Maintainability. In *Proceedings of Brazilian Symposium on Information Systems, Aracaju, Sergipe, Brazil, 20–24 May, 2019 (SBSI'19),* 8 pages.
https://doi.org/10.475/123_4

SBSI'19, 20–24 May, 2019, Aracaju, Sergipe, Brazil

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

1 INTRODUÇÃO

Linhas de Produto de Software (LPS) representam um conjunto de sistemas de software que compartilham de um conjunto de *features* comuns e gerenciadas, que satisfazem as necessidades de um segmento de mercado particular ou missão [22]. As LPSs têm se consolidado como um paradigma de desenvolvimento de software que promove o reuso de software de maneira sistematizada diminuindo o esforço de manutenção [22]. Portanto, as *features* de uma LPS devem ser fáceis de manter e evoluir. Consequentemente os desenvolvedores devem identificar eventuais estruturas de código anômalas que são prejudiciais para a manutenção de LPSs. Essas estruturas são conhecidas como *code smells*.

Code smells são estruturas de código anômalas que representam sintomas de problemas que afetam a manutenibilidade dos sistemas em diversos níveis, tais como classes e métodos [9, 13]. Por exemplo, *Feature Envy* [9] ocorre quando parte do código de uma *feature* deve ser extraído para outra *feature*, já que ela está mais interessada em dados da antiga *feature*. *Code smells* afetam qualquer sistema incluindo LPSs [6]. No entanto, o efeito negativo de um *code smells* em LPS é ainda mais importante do que um sistema único. Esse efeito também pode incluir a propagação de problemas de manutenção para vários produtos da LPS [7]. Assim, os desenvolvedores de LPS precisam priorizar os *code smells* para identificação e eliminação.

Estudos anteriores têm investigado a relação entre *code smells* e problemas de manutenibilidade do código fonte [7, 21, 34]. Yamashita et al. [35] observaram que as atuais abordagens investigam apenas a ocorrência de *code smells* individuais e não os efeitos das relações entre *code smells* e problemas de manutenibilidade do código fonte. Por outro lado, Pietrzak and Walter [21] investigaram as relações entre *code smells* que ocorrem quando existem um relacionamento de dependências entre elas, *Inter-smell*. Os autores também definem seis tipos de relações, i.e., *Inter-smell* e afirmam que o estudo dessas associações e dependências entre *code smells* podem resultar em uma melhor compreensão sobre potenciais problemas que essas anomalias podem causar para a manutenibilidade. Um exemplo de relação *inter-smell* é uma *God Class* que pode conter um *Duplicated Code* [21].

Fernandes et al. [7] afirmam que ocorrências individuais de *code smells* não são suficientes para caracterizar problemas de manutenibilidade em LPSs. Assim, os autores avaliam o impacto de certos *code smells* em LPSs que podem estar interligadas, formando as chamadas aglomerações de anomalias, i.e., um grupo de dois ou mais elementos de código anômalos que estão direta ou indiretamente

relacionados através da estrutura do programa de um sistema [18]. Os autores também sugerem que aglomerações são mais efetivas do que ocorrências individuais de *code smells* para identificar problemas de instabilidade em LPSs. Embora estudos anteriores tenham investigado a relação entre *code smells* e problemas de manutenibilidade, a maioria desses estudos investigam apenas a ocorrência de *code smells* de forma individual e não as suas coocorrências ou agrupamentos. Além disso, os estudos que analisam aglomerações em LPS [7] não investigam as relações *Inter-smell* e nem o seu impacto para a manutenibilidade em LPSs orientadas a objeto.

Este artigo endereça as limitações mencionadas anteriormente por meio de um estudo empírico. Primeiro, este estudo é realizado com diferentes *releases* de duas LPSs orientada a objetos (OO), e implementadas em Java: *MobileMedia* e *Health Watcher*. *MobileMedia* é uma LPS de aplicações que manipulam foto, música e vídeo em dispositivos móveis, como celulares [8]. *Health Watcher* é uma LPS de sistemas WEB, que visa permitir que pessoas registrem suas queixas relativas a problemas de saúde [4]. Segundo, foram coletados cinco tipos de *code smells* em cada uma das LPSs. Terceiro, foram identificadas coocorrências de *code smells* conhecidas como relações *Inter-smell* por meio de métricas [14]. Finalmente, foram realizadas refatorações para remoção dos *Inter-smell* comparando as *releases* antes e depois das refatorações para verificar o impacto dessas anomalias para a manutenibilidade utilizando métricas conhecidas [3]. Os resultados estão sumarizados a seguir:

- O número de ocorrências de *Duplicated Code* aumenta à medida que o número de ocorrências de *Long Method* também aumenta.
- A métrica LCOM é uma das mais importantes métricas para a manutenibilidade de LPSs OO.
- A presença de ocorrências de *Inter-smell* não provocou impacto negativo para a manutenibilidade e qualidade do código das LPSs *MobileMedia* e *Health Watcher*.

Este estudo está organizado da seguindo forma. A Seção 2 apresenta a fundamentação teórica para apoiar a compreensão deste estudo. A Seção 3 descreve o método de pesquisa. A Seção 4 apresenta os resultados do estudo. A Seção 5 apresenta e discute os resultados do estudo. A Seção 6 discute ameaças à validade do estudo. A Seção 7 compara este estudo com estudos anteriores. Finalmente, a Seção 8 apresenta as conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A Seção 2.1 discute sobre *code smells* e refatoração em LPSs. A Seção 2.2 apresenta as relações *inter-smell* em LPSs. Finalmente, a Seção 2.3 discute e apresentam as métricas utilizadas neste estudo.

2.1 *Code Smells* e Refatoração

Code Smells são indicadores de problemas de manutenção em sistemas [9]. Esses indicadores podem afetar qualquer sistema [15, 18], incluindo LPSs [7]. Fowler [9] define um catálogo de 22 tipos de *Code smells* que afetam diferentes níveis do código fonte. Por exemplo, *Long Method* e *Feature Envy* afetam o código fonte no escopo de métodos. Por outro lado, *Lazy Class*, *God Class* e *Data Class* representam indicadores no escopo da classe. Estudos anteriores [15, 18] fornecem evidências de que *code smells* são indicadores suficientes de partes de código afetadas pela pobre decomposição de *features*, no caso de sistemas de software únicos. É extendida essa suposição para LPSs orientadas a objetos, com base em estudos que investigam *code smells* em LPS implementados através de outros mecanismos, tais como mecanismos de composição [6, 7]. A Tabela 1 descreve os cinco tipos de *code smells* analisados neste estudo.

Tabela 1: Tipo de *code smells* analisados neste estudo

Tipo de *Code Smells*	Descrição
Long Method [9, 13]	Um método muito longo e complexo, cujas declarações podem ser extraídas para novos métodos, para que cada método seja mais fácil de entender e alterar, se necessário.
Feature Envy [9, 13]	Instruções de um método que deve ser movido para outro método, às vezes localizado em outra classe, cujas *features* são mais compartilhadas e usadas.
God Class [9, 13]	Uma classe muito grande e complexa, que geralmente concentra muitos recursos do sistema.
Type Checking [9, 13]	Uma estrutura de código caracterizada por vários desvios condicionais, geralmente estruturados como uma instrução *switch*. Cada condição deve ser encapsulada em alguns métodos.
Duplicated Code [9, 13]	Uma mesma expressão ou código repetido está em dois métodos de uma mesma classe.

Refatoração é uma prática comum usada por desenvolvedores para eliminar *code smells* [17, 27]. Refatoração é uma transformação de programa destinada a melhorar a estrutura de um programa enquanto preserva seu comportamento observável [9]. Refatorações podem ter finalidades diferentes. Por exemplo, *Move Method* [9] é frequentemente aplicado por desenvolvedores para eliminar o *code smells Feature Envy*. Essa refatoração consiste em mover um método de uma classe para outra, afim de remover dependências excessivas entre as duas classes. Outro exemplo é *Extract Subclass* [9] que pode ser aplicado visando eliminar o *code smells God Class*. Uma *God Class* é caracterizada por uma classe com tamanho e complexidade excessivos.

2.2 Relações *Inter-smell*

Relações *Inter-smell* ocorrem quando existem relações e dependências entre dois ou mais *code smells*. Por exemplo uma mesma classe que é *God Class* e também possui um *Duplicated Code* [21]. Pietrzak and Walter [21] definem alguns tipos de relações *Inter-smell* objetivando uma maior precisão na detecção de *Code smells* e no impacto negativo que essas anomalias podem causar em sistemas. Tabela 2 descreve os tipos de relações *Inter-smell* definidas por [21] e analisadas nesse estudo.

Processo de identificação de relações *inter-smell*. A identificação das relações *Inter-smell* é realizada de forma manual por meio da utilização de três tipos de métricas [14, 21]. A explicação das métricas é dada a seguir:

- *exist(CS)*: corresponde ao número de entidades (classes ou métodos) onde o *code smell* (CS) foi encontrado.
- *co-exist(CS1, CS2)*: corresponde ao números de classes ou métodos na qual ambos *code smells* foram encontrados.
- $exist - overlap(CS1, CS2) = \frac{co - exist(CS1, CS2)}{exist(CS1)}$: corresponde ao número de classes ou métodos na qual ambos os *code smells CS1, CS2* foram encontrados dividido pelo número de classes ou métodos na qual o *code smell CS1* foi

Tabela 2: Relações *inter-smell* analisados neste estudo

Tipo de Relações	Descrição
Plain Support	Representa a relação mais simples de ser identificada. Essa relação ocorre quando a presença de um *Code Smell* A em uma entidade implica fortemente que o *Code Smell* B também esteja presente nessa mesma entidade. Essa relação pode ser representada com uma seta que indica a direção da relação: $(A \rightarrow B)$.
Mutual Support	Ocorre quando existe um *Plain Support* simétrico, na qual é impossível indicar a direção da relação. Essa relação pode ser representada através de uma seta com duas extremidades: $(A \leftrightarrow B)$.
Rejection	Corresponde o oposto do *Plain Support*. Isso significa que se um *Code Smell* A está presente em uma entidade, muito provavelmente o *Code Smell* B não está presente nessa mesma entidade. Essa relação pode ser representada com uma seta "negativa": $(A \nRightarrow B)$.
Inclusion	Uma relação mais forte que a *Plain Support*, nesta relação a presença de *Code Smell* A em uma entidade implica obrigatoriamente que o *Code Smell* B também estará presente nessa entidade: $(A \Rightarrow B)$.

encontrado, i.e., a métrica co-exist(CS1,CS2) dividido pela métrica *exist(CS1)*.

Primeiramente, as métricas *exist(CS)* e *co-exist(CS1,CS2)* são computadas. Em seguida, os valores dessas métricas são utilizadas para compor o calculo da métrica $exist-overlap(CS1, CS2)$. Essa métrica é equivalente a métrica *co-exist* de um CS1 e CS2 dividido pelo *exist* do CS1. A Figura 1 ilustra um exemplo prático de como utilizar as métricas para identificar relações *inter-smell*.

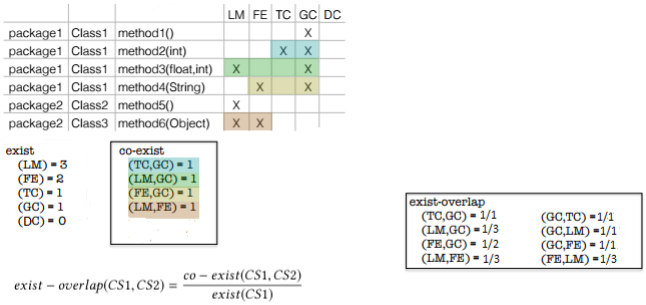

Figura 1: Processo para identificar relações

De acordo com a Figura 1 a métrica $exist - overlap(LM, GC)$ é equivalente a 1/3 pois o *co-exist(LM,GC)* = 1 e o número de ocorrências de LM = 3. O inverso também acontece, o $exist - overlap(GC, LM)$ é equivalente a 1/1 pois o *co-exist(LM,GC)* = 1 e o número de ocorrências de GC = 1. O escopo para a utilização dessas métricas são as classes ou métodos que contém os *code smells*. Assim, as relações *Inter-smell* são facilmente identificadas:

- *Plain Support* $(A \rightarrow B)$: se $exist - overlap(A, B) > exist - overlap(B, A)$
- *Mutual Support* $(A \leftrightarrow B)$: se $exist - overlap(A, B) \approx exist - overlap(B, A)$
- *Rejection* $(A \nRightarrow B)$: se $co - exist(A, B) \approx 0$
- *Inclusion* $(A \Rightarrow B)$: se $exist - overlap(A, B) \approx 1$

Este estudo leva em consideração as relações *Inter-smell* descritas na Tabela 2, para analisar qual o efeito dessas relações para a qualidade do código e manutenibilidade das LPS *MobileMedia* e da LPS *Health Watcher*.

2.3 Medição de Manutenibilidade

As métricas propostas por Chidamber and Kemerer [3] são pioneiras na área de métricas orientadas a objeto e apresentam uma base teórica para medir código orientado a objeto. Neste estudo a suíte *CK metrics* [3] foi escolhida para ser utilizada na verificação da qualidade interna da LPSs *MobileMedia* e *Health Watcher* uma vez que essas métricas já terem sido utilizadas em diversos trabalhos [5, 16, 31]. A Tabela 3 descreve as métricas utilizadas neste estudo.

Tabela 3: Métricas de manutenibilidade utilizadas

Tipo de Métricas	Descrição
Weighted Methods per Class (WMC)	Somatório da complexidade ciclomática de todos os métodos locais definidos em uma classe
Depth of Inheritance Tree (DIT)	O número de níveis que uma subclasse herda de métodos e atributos de uma superclasse na árvore de herança.
Number of Children (NOC)	O número de subclasses diretas de uma determinada classe.
Coupling between Objects (CBO)	O número de classes chamadas por uma determinada classe. CBO mede o grau de acoplamento entre classes.
Response for a Class (RFC)	Número total de métodos que podem potencialmente ser executados para responder uma mensagem recebida pelo objeto da classe
Lack of Cohesion of Methods (LCOM)	Mede a coesão de uma classe. Quanto maior o valor desta métrica, menos coesa é a classe

Para calcular as métricas foi utilizada a ferramenta ckjm. Essa ferramenta calcula as métricas propostas por Chidamber and Kemerer [3] processando o bytecode de arquivos Java compilados. Em seguida, as métricas são extraídas classe por classe [26]. Em Singh and Malhotra [24], os autores afirmam que as métricas apresentadas na Tabela 3 são negativamente correlacionadas com a manutenibilidade de software, i.e., quanto menor o valor das métricas, maior a manutenibilidade do software.

Para avaliar a manutenibilidade das LPSs a partir dos valores das métricas, este estudo utilizou a mesma abordagem de Tarwani and Chug [28], na qual os autores utilizam o somatório de todas as métricas para medir e comparar a manutenibilidade, assim, quanto menor o valor do somatório das métricas, maior a manutenibilidade. As LPSs *MobileMedia* e *Health Watcher* são medidas com seus respectivos códigos originais, i.e, antes do processo de remoção dos *Inter-smell*, para que após o processo de remoção dessas coocorrências, seja medida novamente a qualidade do código dessas LPSs com o objetivo de comparar o código original com o código refatorado para que assim, seja avaliado o impacto dos *Inter-smell* para a manutenibilidade.

3 MÉTODO DO ESTUDO

A Seção 3.1 introduz o objetivo do estudo e as questões de pesquisa. A Seção 3.2 descreve cada etapa e procedimento do estudo, desde a coleta de dados até a análise de dados.

3.1 Objetivo e Questões de Pesquisa

O estudo das coocorrências e relações entre *code smells* é uma área de pesquisa relativamente nova. Porém, já existem trabalhos na literatura sobre esse tema [21, 35]. No entanto, nenhum desses trabalhos analisaram o impacto de coocorrências ou relações entre

code smells sobre a qualidade e manutenibilidade do código em LPSs orientada a objetos. Devido ao conhecimento empírico limitado sobre este assunto, este estudo avalia *o impacto de relações inter-smell na manutenibilidade de LPS orientada a objetos*. O objetivo de estudo é descrito a seguir [32]: *analisar* LPSs orientada a objetos; *com o proposito de* avaliar o impacto de relações *inter-smell*; *com respeito a* manutenibilidade de LPSs; *sobre ponto de vista de* pesquisadores; *no contexto de* duas LPSs orientada a objetos. A Tabela 4 apresenta as três perguntas de pesquisa (QPs). Cada QP é detalhada a seguir.

Tabela 4: Questões de pesquisa

QP	Descrição
QP1	Quais são os *code smells* mais frequentes em LPSs orientada a objetos?
QP2	Qual o padrão que pode ser observado nos *code smells* a partir das relações *Inter-smell* de LPSs orientada a objetos?
QP3	Qual o impacto das relações *Inter-smell* para a qualidade e manutenibilidade do código de LPSs orientada a objetos?

QP1 visa investigar as ocorrências de cinco tipos de *code smells* (ver Tabela 1) e verificar quais são os tipos de *code smells* mais frequentes nas LPSs OO *MobileMedia* e *Health Watcher*. O objetivo é buscar um melhor entendimento sobre a estrutura do código de cada uma das *releases*, a partir da detecção dos *code smells*. Foram mensuradas as ocorrências, i.e., quantos *code smells* afetam cada uma das *releases* de cada LPS. Foram utilizadas as ferramentas JDeodorant e PMD para quantificar as ocorrências de *code smells*.

Por meio da QP2 foram investigadas as coocorrências de *code smells* e suas relações *Inter-smell* [21]. O objetivo é identificar as relações *Inter-smell*, a partir dos *code smells* detectados na QP1, para investigar como essas anomalias estão agrupadas em cada uma das *releases* das LPSs *MobileMedia* e *Health Watcher*. Esse conhecimento é essencial para verificar se (i) a presença de algum *code smell* específico implica na presença de outro *code smell*; e (ii) se a presença de um determinado *code smell* implica na ausência de algum outro *code smell*. As relações *Inter-smell* foram mensuradas conforme apresentado na Seção 2.2.

Finalmente, QP3 avalia se as relações *Inter-smell* são prejudiciais ou não para a manutenibilidade de cada uma das *releases* das LPSs *MobileMedia* e *Health Watcher*. Com isso, foi analisado o impacto das relações *Inter-smell* para a manutenibilidade das duas LPSs. Foi executada a comparação da qualidade de código das classes com ocorrências de *Inter-smell* e as mesmas classes, desta vez, sem a presença dessas relações, que foram descaracterizadas após aplicação de operações de refatoração. A manutenibilidade foi mensurada, por meio de cinco métricas de software (ver Seção 2.3) .

3.2 Passos e Procedimentos do Estudo

Esta seção descreve os cinco passos do estudo, visando apoiar a investigação sobre o impacto de relações *inter-smell* sobre a manutenibilidade de LPSs orientada a objetos.

Passo 1. Selecionar LPSs orientada a objetos. Foram selecionadas LPSs orientada a objetos desenvolvidas em Java dada a alta popularidade da linguagem[1]. Foi conduzida uma busca na literatura *ad-hoc* para identificar LPSs orientada a objetos. Foram priorizadas LPS utilizadas em estudos anteriores, com código fonte disponível para análise, e que apresentassem duas ou mais *releases*. Como resultado, foram encontradas duas LPSs: MobileMedia [2, 5, 8] e Health Watcher [10, 25]. Foi confirmado manualmente que cada uma dessas LPS foram implementadas usando orientação a objetos. As duas LPSs selecionadas são descritas a seguir:

- *MobileMedia* compõe produtos para gerenciar mídia em dispositivos móveis, como fotos, músicas e vídeos. O objetivo é oferecer suporte a usuários de dispositivos móveis com vários recursos, como criação, edição e exclusão de vários tipos de mídia. MobileMedia possui um conjunto de oito *releases*.
- *Health Watcher* compõe produtos para permitir o uso restrito de reclamações e consultas sobre o sistema de saúde pública de uma cidade. O objetivo é permitir que o cidadão registre suas queixas relativas a problemas de saúde pública. Health Watcher possui um conjunto de três *releases*.

Passo 2. Detectar e coletar code smells por LPS. Foram detectados cinco tipos de *code smells*: *Long Method*, *Feature Envy*, *God Class*, *Type Checking* e *Duplicated Code*. A Tabela 1 descreve os cinco *code smells* coletados. Os *code smells* foram coletados por meio de duas ferramentas, JDeodorant [29] e PMD[2]. Essas ferramentas foram escolhidas pela disponibilidade e ampla utilização em trabalhos anteriores para detecção de *code smells* [20, 23].

Passo 3. Detectar e coletar relações Inter-smell. Após coletar a ocorrências individuais de *code smells*. Foram detectadas as relações de coocorrências dos *code smells* nas duas LPSs analisadas. Em seguida, foram detectadas quatro tipos de relações *Inter-smell*: *Plain Support*, *Manual Support*, *Rejection* e *Inclusion*. A Seção 2.2 descreve os quatro tipos de relações detectadas, assim como as métricas e procedimentos adotados para coletar essas relações.

Passo 4. Medir e remover as ocorrências de Inter-smell. Foi mensurada a qualidade interna do código fonte de cada classe de todas as *releases* das LPSs *MobileMedia* e *Health Watcher* contendo as relações *Inter-smell*, apenas essas classes tiveram as métricas coletadas. Foi utilizada a ferramenta ckjm [26] para coletar um conjunto de seis métricas conhecidas da literatura: *Weighted Methods per Class (WMC)*, *Depth of Inheritance Tree (DIT)*, *Number of Children (NOC)*, *Coupling between Objects (CBO)*, *Response for a Class (RFC)* e *Lack of Cohesion of Methods (LCOM)* (ver Tabela 3).

Afim de analisar o impacto de relações *inter-smell* em cada LPS, todas as classes e métodos que possuíam coocorrência de *code smells* foram refatoradas, de modo que apenas um *code smell* individual permanecesse. O objetivo foi descaracterizar a coocorrência de *code smells*. As operações de refatoração foram feitas de forma automática utilizando a ferramenta JDeodorant [30]. A escolha do *code smell* a ser removido foi feita de forma aleatória. Após o processo de refatoração, foi verificado se realmente as coocorrências detectadas foram removidas (ver Seção 2.2). No entanto, não foi verificado o surgimento de novos *code smells* ou novas coocorrências em decorrência da remoção dessas anomalias utilizando a ferramenta

[1] https://www.tiobe.com/tiobe-index/

[2] https://pmd.github.io/

JDeodorant no processo automático de refatoração.

Passo 5. Analisar o impacto de relações Inter-smell sobre a manutenibilidade da LPS. Foram comparadas todas as *releases* de cada LPS. Por exemplo, a *release* 1 original da LPS *MobileMedia* foi comparada com a *release* 1 refatorada da LPS *MobileMedia* e esse processo continuou para as sete *releases* restantes da LPS *MobileMedia*. O mesmo processo foi aplicado para a LPS *Health Watcher*, as três *releases* originais foram comparadas com as três refatoradas. A comparação foi realizada através dos resultados das métricas de qualidade coletadas. As comparações foram realizadas apenas nas classes que tinham coocorrências de *code smells*. Para avaliar a manutenibilidade das LPSs a partir dos valores das métricas, este estudo utilizou a abordagem adotada por Tarwani and Chug [28]. Essa abordagem utiliza o somatório de todas as métricas para medir e comparar a manutenibilidade. De modo que quanto menor o valor do somatório, maior a manutenibilidade.

4 RESULTADOS

A Seção 4.1 apresenta os tipos de *code smells* mais frequentes em LPSs orientada a objetos (LPSs OO) **(QP_1)**. A Seção 4.2 discute os padrões que podem ser observados de code smells em relações *inter-smell* **(QP_2)**. A Seção 4.3 avalia o impacto das relações *inter-smell* na manutenibilidade de LPSs OO **(QP_3)**.

4.1 Code smells mais frequentes em LPSs OO

Tabela 5 apresenta o número de ocorrências de *code smells* em cada uma das oito *releases* da LPS *MobileMedia*. A primeira coluna lista cada uma das *releases*. A segunda, terceira, quarta, quinta e sexta colunas apresentam a frequência da cada *code smells* analisado, isto é, *God Class* (GC), *Feature Envy* (FE), *Long Method* (LM), *Duplicated Code* (DC) e *Type Checking* (TC).

Tabela 5: Ocorrências de *code smells* na LPS *MobileMedia*

Release	GC	FE	LM	DC	TC
1	4	3	6	15	0
2	4	3	9	58	0
3	4	3	10	72	0
4	4	3	11	82	0
5	4	3	9	119	0
6	2	0	14	206	0
7	2	1	15	231	1
8	2	1	21	297	0

Os resultados da Tabela 5 revelam algumas descobertas interessantes. Foi observado que o *code smell* mais frequente entre todas as *releases* da LPS *MobileMedia*, foi DC juntamente com LM. Além disso, nas *releases* 1, 2, 3, 4, 6, 7 e 8 à medida que o número de LM aumenta, o número de ocorrências de DC também obtêm um certo crescimento. Semelhante a tabela anterior, a Tabela 6 apresenta o número de ocorrências de *code smells* em cada uma das *releases* da LPS *HealthWatcher*. Similar a LPS *MobileMedia*, o *code smell* mais frequente em *Health Watcher* foi o *Duplicated Code* juntamente com *Long Method*. Além disso, também foi observado que à medida que o número de LM aumenta, o número de ocorrências de DC também obtêm um crescimento nas *releases* 1, 2 e 3.

Tabela 6: Ocorrências de *code smells* na LPS *Health Watcher*

Release	GC	FE	LM	DC	TC
1	0	4	73	378	3
2	0	4	87	783	3
3	0	0	88	804	4

Portanto, a partir da QP_1 podemos concluir que os *code smells* mais frequentes em *MobileMedia* e *Health Watcher* são: *Duplicated Code* e *Long Method*. O número de ocorrências de *Duplicated Code* aumenta à medida que o número de ocorrências de *Long Method* também aumenta.

4.2 Padrões de *code smells* em relações *Inter-smell*

Para responder essa pergunta, é realizada uma apresentação de como as ocorrências de *code smells* estão organizadas e são apresentados indícios dessas ocorrências a partir das relações *Inter-smell*.

Após o processo manual de identificação das relações *Inter-smell* na LPS *MobileMedia*, foram feitas algumas observações em cada uma dessas relações:

- ***Plain Support***: Nesta relação, os *Inter-smell* que mais apareceram foram: $GC \rightarrow LM$, $LM \rightarrow DC$, $GC \rightarrow DC$ e $FE \rightarrow GC$. Em sete das oitos *releases*, a ocorrência de $GC \rightarrow LM$ foi identificada, porém a ocorrência de $LM \rightarrow DC$ foi detectada em todas as *releases*. Diferente de todas as outras *releases*, a *release* 7 foi a única em que foi encontrado apenas uma ocorrência para a relação *Plain Support* ($LM \rightarrow DC$).
- ***Mutual Support***: Nenhuma ocorrência de *Inter-smell* foi detectada para essa relação.
- ***Rejection***: Para esta relação foi observado que se existir ocorrências dos *code smells* LM, GC, DC e FE em uma entidade de código fonte (método ou classe) provavelmente não existirá ocorrência do *code smell* TC nessa entidade. Também foi observado que nas *releases* 6, 7 e 8 foi encontrada a ocorrência: $FE \nRightarrow GC$, isso pode ser explicado observando a Tabela 5, o número de ocorrências de GC e FE diminuíram em relação as outras *releases*.
- ***Inclusion***: Essa relação é mais forte que *Plain Support* e ela foi detectada nas *releases* 2, 5 e 8. Na *release* 2 foi detectada a ocorrência ($GC \Rightarrow DC$), enquanto que nas *releases* 5 e 8 foram detectadas as ocorrências ($GC \Rightarrow LM$) e ($GC \Rightarrow DC$) respectivamente. Esse resultado sugere que a presença do *code smell* GC pode indicar fortemente a ocorrência de outro *code smell* na mesma entidade, mais precisamente as ocorrências dos *code smells* DC e LM.

Após o processo manual de identificação das relações *Inter-smell* na LPS *Health Watcher*, foram feitas algumas observações em cada uma dessas relações:

- ***Plain Support***: Nesta relação, os *Inter-smell* que mais apareceram foram: $FE \rightarrow LM$, $LM \rightarrow DC$, $GC \rightarrow DC$ e $FE \rightarrow DC$. Em todas as três *releases*, a ocorrência de $LM \rightarrow DC$ foi identificada. Diferente de todas as outras *releases*, a *release* 3 foi a

única em que foram encontradas ocorrências de *Inter-smell* com o *code smell* TC.

- ***Mutual Support***: Nenhuma ocorrência de *Inter-smell* foi detectada para essa relação.
- ***Rejection***: Para esta relação foi observado que nas *releases* 1 e 2 as ocorrências de LM, DC e FE em uma entidade de código fonte (método ou classe) provocou a não ocorrência do *code smell* TC nessa entidade. Enquanto que na *release* 3, as ocorrências de ($FE \rightarrow DC$) e ($FE \rightarrow LM$) não foram detectadas. Diferente do que aconteceu com as *releases* 1 e 2 onde ambas as ocorrências de *Plain Support* foram detectadas. Isso pode ser explicado pela ausência do *code smell* FE na *release* 3, como pode ser observado na Tabela 6. O caso interessante, foi a rejeição do *code smell* GC para os outros *code smells*.
- ***Inclusion***: Essa relação é mais forte que a *Plain Support* e ela foi detectada nas *releases* 1 e 2. Em ambas as *releases* foram detectadas as ocorrências de ($FE \Rightarrow LM$), ($FE \Rightarrow DC$). Esse resultado sugere que a presença do *code smell* FE implica fortemente na ocorrência dos *code smells* LM e DC.

Portanto, a partir da QP_2 podemos concluir que na LPS *MobileMedia* a presença dos *code smells* LM, GC, DC e FE indica a não ocorrência de *Type Checking*. A presença de GC indica fortemente a ocorrência de DC e LM na mesma entidade, assim, uma simples refatoração de GC pode indicar a eliminação de LM e DC. Na LPS *Health Watcher* a presença de LM indica a provável ocorrência de DC. A ocorrência de FE implica fortemente na presença de LM e DC no mesmo método e a presença dos *code smells* TC,FE,LM e DC indicou a não presença de GC.

4.3 Impacto de relações *Inter-smell* em LPSs OO

Para avaliar a manutenibilidade das LPS com e sem os *Inter-smell*, i.e, antes e depois do processo de refatoração, foram utilizadas as métricas apresentadas na Seção 2.3. Singh and Malhotra [24], os autores afirmam que essas métricas são negativamente correlacionadas com a manutenibilidade de software, i.e, quanto menor o valor das métricas, maior a manutenibilidade do software. Para avaliar a manutenibilidade das LPSs a partir dos valores das métricas, foi utilizada a mesma abordagem de Tarwani and Chug [28], na qual os autores utilizam o somatório de todas as métricas para medir e comparar a manutenibilidade.

A Tabela 7 apresenta como a manutenibilidade aumentou ou diminuiu nas *releases* da LPS *MobileMedia*. A primeira coluna lista cada uma das *releases*. A segunda e terceira colunas listam respectivamente a quantidade de relações *inter-smell* antes e depois do processo de refatoração. Nesse caso o valor de manutenibilidade que é o somatório das métricas,está entre 0 e 2000, na qual 0 descreve uma *release* que é muito fácil de manter e 2000 indica uma *release* com a mais alta dificuldade para realizar tarefas de manutenção.

Observando a Tabela 7, nas *releases* 1, 2, 3, 4, 7 e 8 a remoção de *Inter-smell* trouxe um impacto negativo para a manutenibilidade. Principalmente para as *releases* 3 e 4 a remoção dos *Inter-smell* trouxe um impacto negativo na manutenibilidade muito maior comparado com os resultados encontrados para as outras *releases*. Dentre as *oito releases* analisadas, apenas nas *releases* 5 e 6 a remoção de *Inter-smell* significou uma melhoria da manutenibilidade.

Tabela 7: Manutenibilidade da LPS *MobileMedia* com e sem *Inter-smell*

Release	Com *Inter-smell*	Sem *Inter-smell*
1	1092	1262
2	955	1254
3	1213	1867
4	1311	1978
5	1008	924
6	1509	1101
7	793	891
8	1163	1250

Semelhante a tabela anterior, a Tabela 8 apresenta como a manutenibilidade aumentou ou diminuiu nas *releases* da LPS *Health Watcher*. Lembrando que quanto mais o somatório de cada *release* se aproximar de 0 (zero), maior será a capacidade de manutenção. Como pode ser observado na Tabela 8, em todas as *releases* a remoção de *Inter-smell* trouxe um impacto negativo para a manutenibilidade. Em nenhuma *release* da LPS *Health Watcher* foi encontrado que a remoção de *Inter-smell* trouxe impacto positivo para a manutenibilidade.

Tabela 8: Manutenibilidade da LPS *Health Watcher* com e sem *Inter-smell*

Release	Com *Inter-smell*	Sem *Inter-smell*
1	2741	3421
2	3130	3715
3	3009	4579

Este trabalho analisou oito *releases* da LPS *MobileMedia* e três *releases* da LPS *Health Watcher*. Em cada uma das *releases* foram detectadas coocorrências de *code smells* conhecidas como *Inter-smell* e após esse passo foram realizadas refatorações para remoção dos *Inter-smell* comparando as *releases* antes e depois das refatorações para verificar o impacto dessas anomalias para a manutenibilidade. Foram utilizadas métricas de manutenibilidade para comparar a manutenibilidade entre as *releases* originais e *releases* refatoradas.

Portanto, a partir da QP_3 podemos concluir que em seis *releases* da LPS *MobileMedia* (1, 2, 3, 4, 7 e 8) e nas três *releases* da LPS *Health Watcher* a presença de ocorrências de *Inter-smell* não provocou impacto negativo para a manutenibilidade e qualidade do código. Assim a presença de *Inter-smell* não significou uma piora na manutenibilidade das LPSs *MobileMedia* e *Health Watcher*.

5 DISCUSSÃO

Na Tabela 7 pode ser observado que das oito *releases* da LPS *MobileMedia*, seis mostraram que a presença de ocorrências de *Inter-smell* não significou impacto negativo para a manutenibilidade. As *releases* 3, 4 e 6 apresentaram valores mais discrepantes, e isso pode ser explicado pelos valores da métrica LCOM, nessas *releases*. Essa métrica teve valores bem discrepantes tanto nas *releases* 3 e 4 que mostram o impacto positivo dos *Inter-smell* quanto para a *release* 6 que mostrou impacto negativo da presença de *Inter-smell*. Como

pode ser observado na Tabela 8, todas as três *releases* com *Inter-smell* tiveram um índice de manutenibilidade maior do que aquelas que não tinha ocorrências de *Inter-smell*, mostrando que a presença de *Inter-smell* não resultou em pior manutenibilidade. A métrica LCOM fez grande diferença nos resultados, indicando que essa pode ser uma importante métrica para a manutenibilidade de LPSs orientada a objeto.

Trabalhos anteriores mencionam que *code smells* tem baixo impacto ou não impactam projetos de software: [11, 33]. Enquanto que outros trabalhos observam que *code smells* tem um impacto significativo para projetos de software: [12, 19]. Da mesma forma que esses trabalhos tiveram resultados diferentes, Fernandes et al. [7] identificaram que aglomerações de *code smells* trazem impacto negativo para à manutenibilidade. Neste trabalho foi encontrado que as coocorrências de *code smells* conhecidas como *Inter-smell* não tiveram impacto negativo para a manutenibilidade ou a qualidade do código em *releases* das LPSs *MobileMedia* e *Health Watcher*.

6 AMEAÇAS À VALIDADE

Esta seção discute as ameaças à validade do estudo de acordo com a classificação de Wohlin et al. [32].

Validade Interna. Em relação ao número pequeno de *releases*, deve ser destacado que o número de LPSs orientada a objetos disponíveis para a pesquisa é bem limitado. Para minimizar esse problema, uma das LPSs escolhidas (*Health Watcher*) possui cerca de 6000 linhas de código por *release*. A detecção de relações *Inter-smell* foi realizada de forma manual, porém, as métricas utilizadas foram validadas anteriormente em trabalho na literatura [14]. Para mitigar o problema da identificação manual de relações *Inter-smell*, para cada *release* o processo foi realizado duas vezes com o objetivo de confirmar os resultados encontrados.

Validade de Construção. Os *code smells* foram identificados automaticamente pelas ferramentas JDeodorant e PMD, diminuindo a chance de erros na detecção. Mesmo assim, as estratégias implementadas por essas ferramentas podem ser um potencial fator de ameaça à validade. Desse modo, outras ferramentas de detecção poderiam utilizar outras estratégias de detecção diferentes das estratégias utilizadas pelas duas ferramentas deste estudo. Assim, isso poderia provocar uma variação do conjunto de *code smells* identificados e consequentemente afetaria a detecção de relações *Inter-smell*.

Validade Externa. Embora as LPSs analisadas pertençam a diferentes domínios, os resultados podem servir apenas para LPSs orientada a objetos escritas em Java. Como trabalho futuro é necessário fazer uma análise também para LPSs escritas em outros paradigmas e outras linguagens.

7 TRABALHOS RELACIONADOS

A comparação dos trabalhos relacionados ao estudo, foi dividida em três tópicos.

Ocorrências individuais de code smells. Ouni et al. [19] afirmam que a presença de *code smells* aumenta de forma significativa o custo de manutenção de sistemas e assim os autores propõem uma nova abordagem de refatoração para eliminar os *smells* mais prejudiciais. Por outro lado, trabalhos como o de Yamashita and Counsell [33] e de Hall et al. [11] mostraram que *code smells* não têm um grande impacto negativo em projetos de software. Entretanto, nenhum dos trabalhos acima investiga as coocorrências de *code smells* e seu impacto.

Coocorrências de code smells em sistemas tradicionais. Abbes et al. [1] analisaram as interações entre *code smells* e seus efeitos. Eles concluíram que quando os *code smells* apareciam isolados, não possuíam nenhum impacto sobre a manutenibilidade, mas quando apareciam interligados, traziam um grande esforço para a manutenção. Yamashita and Moonen [34] analisam o impacto das relações *Inter-Smell* na manutenibilidade de quatro sistemas industriais de tamanho médio escritos em Java. Os autores detectam significantes relações entre *Feature Envy*, *God Class* e *Long Method* e concluem que relações *Inter-Smell* são associadas com problemas durante as atividades de manutenção. Entretanto, nenhum dos trabalhos acima investiga as coocorrências de *code smells* no contexto de LPSs. Assim, este trabalho investiga o impacto de coocorrências de *code smells* conhecidas como *Inter-smell* para a manutenibilidade e qualidade do código no domínio de LPS, analisando as *releases* de *MobileMedia* e *Health Watcher*.

Coocorrências de code smells em LPSs. Fernandes et al. [7] avaliam o impacto de aglomerações de *code smells* para o problema de manutenção, conhecido como instabilidade. Os resultados encontrados sugerem que aglomerações são mais efetivas do que ocorrências individuais de *code smells* para identificar instabilidade ou estruturas que geralmente prejudicam a manutenibilidade da LPS. No entanto, os autores levam em consideração LPSs implementadas utilizando programação orientada a *feature* e apesar de levar em consideração aglomerações, não levam em consideração as relações *Inter-smell*. Oizumi et al. [18] investigaram aglomerações de *code smells* como indicadores de problemas de design. Os autores chegaram a conclusão de que determinadas aglomerações são indicadores consistentes de problemas de design após analisar um número expressivo de aglomerações em sete sistemas de software diferentes, incluindo a LPS *MobileMedia*. Este trabalho também avalia o impacto de relações de *code smells* para a manutenibilidade de LPSs, porém os tipos de relações avaliadas serão relações *Inter-smell* e as LPSs utilizadas neste trabalho são as versões orientada a objetos das LPSs *MobileMedia* e *Health Watcher*.

8 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho realizou um estudo empírico para avaliar o impacto de coocorrências de *code smells* conhecidas como relações *Inter-smell* para a manutenibilidade de duas LPSs orientada a objetos implementadas em Java. Foram utilizadas as LPSs orientada a objetos *MobileMedia* e *Health Watcher*. Todas as *releases* dessas LPSs foram comparadas para verificar qual impacto de anomalias conhecidas como *Inter-smell* para a manutenibilidade. A estratégia utilizada neste trabalho foi comparar cada *release* utilizando um conjunto de métricas de manutenibilidade conhecidas na literatura.

Como resultados do estudo foi verificado que as relações entre *code smells* não trouxeram impacto negativo para a manutenibilidade das *releases* das LPSs *MobileMedia* e *Health Watcher*. Diferente do que foi encontrado em Fernandes et al. [7], no qual os autores identificaram que aglomerações de *code smells* trazem impacto negativo para à manutenibilidade. No entanto, os autores utilizaram LPSs orientada a *features* e outros tipos de *code smells*.

Além disso, as descobertas deste trabalho sugerem que a métrica *LCOM*, que mede a coesão de uma classe, é fundamental para qualidade do código e o índice de manutenibilidade nos produtos de uma LPS. Ainda, as descobertas apontam que os engenheiros de aplicação devem se preocupar mais com a coesão das classes na implementação da aplicação, preservando assim a manutenibilidade dos produtos e beneficiando de forma direta a reusabilidade.

Alguns trabalhos futuros identificados a partir das descobertas deste trabalho, são: (i) análise de outras LPSs implementadas em diferentes paradigmas; (ii) reprodução do estudo com ferramentas que detectam outros *code smells*; e (iii) replicação do estudo utilizando mais métricas para avaliar a manutenibilidade juntamente com desenvolvedores especialistas no domínio.

REFERÊNCIAS

[1] Marwen Abbes, Foutse Khomh, Yann-Gael Gueheneuc, and Giuliano Antoniol. 2011. An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. In *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 181–190.

[2] W Abdelmoez, Hatem Khater, and Noha El-shoafy. 2012. Comparing maintainability evolution of object-oriented and aspect-oriented software product lines. In *Proceedings of the 8th International Conference on Informatics and Systems (INFOS)*. IEEE, SE–53.

[3] Shyam R Chidamber and Chris F Kemerer. 1994. A metrics suite for object oriented design. *Transactions on Software Engineering (TSE)* 20, 6 (1994), 476–493.

[4] Robert Dyer, Hridesh Rajan, and Yuanfang Cai. 2012. An exploratory study of the design impact of language features for aspect-oriented interfaces. In *Proceedings of the 11th annual international conference on Aspect-oriented Software Development*. ACM, 143–154.

[5] Robert Dyer, Hridesh Rajan, and Yuanfang Cai. 2012. An exploratory study of the design impact of language features for aspect-oriented interfaces. In *Proceedings of the 11th Annual International Conference on Aspect-oriented Software Development (AOSD)*. ACM, 143–154.

[6] Wolfram Fenske and Sandro Schulze. 2015. Code smells revisited: A variability perspective. In *Proceedings of the 8th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*. ACM, 3.

[7] Eduardo Fernandes, Gustavo Vale, Leonardo Sousa, Eduardo Figueiredo, Alessandro Garcia, and Jaejoon Lee. 2017. No Code Anomaly is an Island. In *International Conference on Software Reuse (ICSE)*. Springer, 48–64.

[8] Eduardo Figueiredo, Nelio Cacho, Claudio Sant'Anna, Mario Monteiro, Uira Kulesza, Alessandro Garcia, Sérgio Soares, Fabiano Ferrari, Safoora Khan, Fernando Castor Filho, and others. 2008. Evolving software product lines with aspects: an empirical study on design stability. In *Proceedings of the 30th International Conference on Software Engineering (ICSE)*. ACM, 261–270.

[9] Martin Fowler. 1999. *Refactoring*. Addison-Wesley Professional.

[10] Phil Greenwood, Thiago Bartolomei, Eduardo Figueiredo, Marcos Dosea, Alessandro Garcia, Nelio Cacho, Cláudio Sant'Anna, Sergio Soares, Paulo Borba, Uirá Kulesza, and others. 2007. On the impact of aspectual decompositions on design stability: An empirical study. In *European Conference on Object-Oriented Programming*. Springer, 176–200.

[11] Tracy Hall, Min Zhang, David Bowes, and Yi Sun. 2014. Some code smells have a significant but small effect on faults. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 4 (2014), 33.

[12] Foutse Khomh, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2012. An exploratory study of the impact of antipatterns on class change-and fault-proneness. *Empirical Software Engineering (ESE)* 17, 3 (2012), 243–275.

[13] Michele Lanza and Radu Marinescu. 2007. *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media.

[14] Angela Lozano, Kim Mens, and Jawira Portugal. 2015. Analyzing code evolution to uncover relations. In *Proceedings of the 2nd Workshop on Patterns Promotion and Anti-patterns Prevention (PPAP)*. IEEE, 1–4.

[15] Isela Macia, Joshua Garcia, Daniel Popescu, Alessandro Garcia, Nenad Medvidovic, and Arndt von Staa. 2012. Are automatically-detected code anomalies relevant to architectural modularity?. In *Proceedings of the 11th International Conference on Aspect-Oriented Software Development (AOSD)*. 167–178.

[16] Ruchika Malhotra and Anuradha Chug. 2016. An empirical study to assess the effects of refactoring on software maintainability. In *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 110–117.

[17] Emerson Murphy-Hill, Chris Parnin, and Andrew P Black. 2012. How we refactor, and how we know it. *IEEE Trans. Softw. Eng. (TSE)* 38, 1 (2012), 5–18.

[18] Willian Oizumi, Alessandro Garcia, Leonardo da Silva Sousa, Bruno Cafeo, and Yixue Zhao. 2016. Code anomalies flock together: Exploring code anomaly agglomerations for locating design problems. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. IEEE, 440–451.

[19] Ali Ouni, Marouane Kessentini, Slim Bechikh, and Houari Sahraoui. 2015. Prioritizing code-smells correction tasks using chemical reaction optimization. *Software Quality Journal* 23, 2 (2015), 323–361.

[20] Thanis Paiva, Amanda Damasceno, Juliana Padilha, Eduardo Figueiredo, and Claudio Sant'Anna. 2015. Experimental Evaluation of Code Smell Detection Tools. In *Workshop on Software Visualization, Evolution and Maintenance (VEM)*. ACM, 15.

[21] Błażej Pietrzak and Bartosz Walter. 2006. Leveraging code smell detection with inter-smell relations. *Extreme Programming and Agile Processes in Software Engineering* (2006), 75–84.

[22] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. 2005. *Software product line engineering:* foundations, principles and techniques. Springer Science & Business Media.

[23] Beijun Shen and Tong Ruan. 2008. A case study of software process improvement in a Chinese small company. In *International Conference on Computer Science and Software Engineering, 2008*, Vol. 2. IEEE, 609–612.

[24] Yogesh Singh and Ruchika Malhotra. 2012. *Object-oriented software engineering*. PHI Learning Pvt. Ltd.

[25] Sergio Soares, Eduardo Laureano, and Paulo Borba. 2002. Implementing distribution and persistence aspects with AspectJ. In *ACM Sigplan Notices*, Vol. 37. ACM, 174–190.

[26] Diomidis Spinellis. 2005. Tool writing: a forgotten art?(software tools). *IEEE Software* 22, 4 (2005), 9–11.

[27] Konstantinos Stroggylos and Diomidis Spinellis. 2007. Refactoring–does it improve software quality?. In *5th WoSQ*. 10.

[28] Sandhya Tarwani and Anuradha Chug. 2016. Sequencing of refactoring techniques by Greedy algorithm for maximizing maintainability. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 1397–1403.

[29] Nikolaos Tsantalis, Theodoros Chaikalis, and Alexander Chatzigeorgiou. 2008. JDeodorant: Identification and removal of type-checking bad smells. In *Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 329–331.

[30] Nikolaos Tsantalis, Davood Mazinanian, and Shahriar Rostami. 2017. Clone refactoring with lambda expressions. In *Proceedings of the 39th International Conference on Software Engineering (ICSE)*. IEEE Press, 60–70.

[31] Anderson Uchôa, Eduardo Fernandes, Ana Carla Bibiano, and Alessandro Garcia. 2017. Do Coupling Metrics Help Characterize Critical Components in Component-based SPL? An Empirical Study. In *Proceedings of the 5th Workshop on Software Visualization, Evolution and Maintenance (VEM)*. 36–43.

[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[33] Aiko Yamashita and Steve Counsell. 2013. Code smells as system-level indicators of maintainability: An empirical study. *Journal of Systems and Software (JSS)* 86, 10 (2013), 2639–2653.

[34] Aiko Yamashita and Leon Moonen. 2013. Exploring the impact of inter-smell relations on software maintainability: An empirical study. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. IEEE, 682–691.

[35] Aiko Yamashita, Marco Zanoni, Francesca Arcelli Fontana, and Bartosz Walter. 2015. Inter-smell relations in industrial and open source systems: A replication and comparative analysis. In *Proceedings of the 31st International Conference onSoftware Maintenance and Evolution (ICSME)*. IEEE, 121–130.