UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM ENGENHARIA DE SOFTWARE

JÚLIO SERAFIM MARTINS

ANALISANDO O IMPACTO DE INTER-SMELL NA MANUTENIBILIDADE DE LINHAS DE PRODUTO DE SOFTWARE: UM ESTUDO DE CASO

QUIXADÁ – CEARÁ

2017

JÚLIO SERAFIM MARTINS

ANALISANDO O IMPACTO DE INTER-SMELL NA MANUTENIBILIDADE DE LINHAS DE PRODUTO DE SOFTWARE: UM ESTUDO DE CASO

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Orientadora: Profa. Dra. Carla Ilane Moreira Bezerra

QUIXADÁ – CEARÁ

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M343a Martins, Júlio Serafim.
Analisando o impacto de inter-smell na manutenibilidade de linhas de produto de software : um estudo de caso / Júlio Serafim Martins. – 2017.
59 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2017.
Orientação: Profa. Dra. Carla Ilane Moreira Bezerra.

1. Engenharia de linha de produto de software. 2. Code smell. 3. Manutenibilidade. 4. Software - Controle de qualidade. I. Título.

CDD 005.1

JÚLIO SERAFIM MARTINS

ANALISANDO O IMPACTO DE INTER-SMELL NA MANUTENIBILIDADE DE LINHAS DE PRODUTO DE SOFTWARE: UM ESTUDO DE CASO

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Aprovada em: ___ /___/____

BANCA EXAMINADORA

Profa. Dra. Carla Ilane Moreira Bezerra (Orientadora)
Universidade Federal do Ceará – UFC

Prof. Msc. Carlos Diego Andrade de Almeida
Universidade Federal do Ceará - UFC

Prof. Msc. Regis Pires Magalhães
Universidade Federal do Ceará - UFC

A Deus.

Aos meus pais, irmãos e amigos.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pois sem ele nada disso seria possível. Obrigado meu Deus!
Gostaria de agradecer meus pais por toda a educação que eles me deram e por todos os sacrifícios que foram feitos para sempre dar o melhor para mim e para meu irmão Jonas. Eu amo vocês Raimunda Neuma Serafim e Francisco Jucelino Martins da Cruz!
Também agradeço a todos os meus professores do Colégio Santa Clara e do Instituto Federal do Ceará - Campus Canindé.
Ao professor Samy Soares por ter sido o meu primeiro orientador no programa PET e por ter me dado importantes conselhos que me ajudaram ao longo da minha caminhada na graduação.
À professora Carla Ilane por toda paciência, dedicação e sinceridade durante a execução deste trabalho. Certamente eu sou um ser humano melhor depois do convívio com essa pessoa maravilhosa.
À Universidade Federal do Ceará - Campus Quixadá e a todos os professores incríveis desta instituição de ensino. Sinto-me muito orgulhoso em dizer que fui formado na UFC - Quixadá.
Aos meus amigos da graduação e amigos do PET, meu muito obrigado pela contribuição de vocês na minha jornada.

“ Querido Deus, Tu és minha proteção, a minha fortaleza. Tu és o meu Deus, eu confio em Ti.”

(Salmo 91:2)

RESUMO

Uma Linha de Produto de Software (LPS) consiste em uma estratégia de realizar o reuso de forma sistemática para a construção de sistemas com menos esforço desde que estes pertençam a uma mesma família que compartilha os mesmos componentes, e que pertence a um mesmo domínio de mercado. O reuso de software é considerado uma das formas mais poderosas para enfrentar os desafios no desenvolvimento de sistemas de software por benefícios como qualidade de software, diminuição do tempo de entrega e diminuição dos custos. Nesse contexto, para suportar o reuso em larga escala, os componentes de uma Linha de Produto de Software (LPS) devem ser fáceis de manter. Assim, os desenvolvedores devem se preocupar ainda mais com as anomalias conhecidas como *code smells*, e ainda, as co-ocorrências conhecidas como *Inter-smell*, que merecem ser mais estudadas para verificar o seu real impacto na manutenibilidade em LPS. Nesse contexto, este trabalho realiza um estudo de caso para investigar o impacto de ocorrências de *Inter-smell* na manutenibilidade de duas LPSs orientadas a objetos. Inicialmente foi realizada a detecção dos *code smells God Class*, *Duplicated Code*, *Long Method*, *Feature Envy* e *Type Cheking*. Posteriormente foi realizada a identificação das relações *Inter-smell*, medição da qualidade de cada *release* com a presença dessas anomalias e a remoção dessas relações para comparar as *releases* com e sem ocorrências de *Inter-smell*. Os resultados do estudo mostram que a presença de co-ocorrências de *Inter-smell* não impactaram negativamente a manutenibilidade das LPSs estudadas, diferente dos resultados encontrados em outros trabalhos da literatura, e mais ainda, nossos resultados indicam que a métrica *Lack of Cohesion of Methods* LCOM é uma importante métrica para a manutenibilidade de LPSs orientada a objetos.

Palavras-chave: Engenharia de linha de produto de software. *Code smell*. Manutenibilidade. Software - Controle de qualidade.

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. Software reuse is considered one of the most powerful ways to address the challenges in developing software systems for benefits such as software quality, shorter lead times and lower costs. In this context, to support large-scale reuse, components of a Software Product Line (SPL) 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 case study to investigate the impact of Inter-smell occurrences on maintainability in MobileMedia and HealthWatcher SPL. First the detection of code smells, God Class, Duplicated Code, Long Method, Feature Envy, and Type Cheking were detected. After this step, the identification of the Inter-smell relationships was measured, the quality of each release was checked against the presence of these anomalies and the removal of these relationships to compare releases with and without occurrences of Inter-smell.

The results show that the presence of co-occurrences of Inter-smell did not negatively impact the maintenance of MobileMedia and Health Watcher SPL, unlike results found in other studies in the literature, and even more, our results indicate that the metric LOCK of Cohesion of Methods LCOM is one of the most important for the maintainability of object-oriented SPL.

Keywords: Software product line engineering. Code Smell. Maintainability. Software - Quality Control.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LPS	Linha de Produto de Software
GC	*God Class*
DC	*Duplicated Code*
FE	*Feature Envy*
LM	*Long Method*
WMC	*Weighted Methods per Class*
DIT	*Depth of Inheritance Tree*
NOC	*Number of Children*
CBO	*Coupling between Objects*
RFC	*Response for a Class*
LCOM	*Lack of Cohesion of Methods*

SUMÁRIO

1 INTRODUÇÃO

A competitividade no desenvolvimento de software tem se tornado cada vez mais uma preocupação para empresas de todos os tamanhos e segmentos de mercado e a abordagem de Linha de Produto de Software (LPS) ganhou considerável atenção nos últimos anos por seus benefícios voltados ao custo, tempo de mercado e também outros objetivos de negócio (LINDEN; SCHMID; ROMMES, 2007).

Uma Linha de Produto de Software é um conjunto de aplicações com componentes compartilhados, sendo que cada aplicação possui suas próprias restrições e requisitos (SOMMERVILLE, 2011). A engenharia da LPS promove a produção de uma família de produtos de software a partir de *features* comuns, ao invés de produzir um produto de cada vez do zero. A diferença entre os produtos da linha ocorre à medida em que clientes customizam os seus produtos de acordo com suas necessidades (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010).

A essência da LPS é a satisfação do usuário, reusabilidade e manutenabilidade. Como exemplo, o code smell *Duplicated Code* diminui a peformace, introduz erros e ainda prejudica a manutenabilidade de LPSs (DEVI U.; KESSWANI, 2016). Assim, para suportar o reuso em larga escala, os componentes de uma LPS devem ser fáceis de manter, e por isso, desenvolvedores deveriam identificar essas estruturas anômalas, como *code smells* que podem prejudicar a manutenibilidade de uma LPS. Caso contrário, os problemas relacionados a manutenibilidade podem prejudicar vários produtos da LPS (FERNANDES et al., 2017). *Code Smells* podem indicar problemas relacionados com aspectos da qualidade do código tal como legilibidade e a modificabilidade. Tendo isso em vista, *code smells* podem causar problemas para os desenvolvedores em atividades na fase de manutenção do software (FOWLER, 1999).

Yamashita e Moonen (2013a) afirmam em seu trabalho que *code smells* são apenas indicadores prejudiciais de problemas no código. Nem todos os *code smells* são prejudiciais, e alguns, dependendo do contexto, não são prejudiciais. Os autores citam que a maioria dos estudos da literatura têm se levado em consideração apenas a ocorrência individual de *code smells* e que os trabalhos na academia devem estudar mais as relações entre os *code smells* e seus problemas para a manutenibilidade do código.

Pietrzak e Walter (2006) foram os primeiros a investigar o relacionamento entre *code smells*, que são chamados de relações *Inter-Smell*. Os autores definem seis tipos de relações *Inter-Smell* e afirmam que o estudo dessas associações e dependências entre *code smells* podem resultar

em uma melhor compreensão sobre ponteciais problemas que essas anomalias podem causar para a manutenabilidade. Relações *Inter-smell* ocorrem quando existem relações e dependências entre *code smells*. Um exemplo de relação entre *code smells*: uma *Large Class* pode conter *Duplicated Code* (PIETRZAK; WALTER, 2006).

Em Yamashita et al. (2015) os autores afirmam, baseados em estudos anteriores que levam em consideração relações entre *code smells*, que as atuais abordagens para análise e detecção focadas apenas na ocorrência de *smells* individuais e não nos efeitos das relações entre *code smells* podem estar limitando a capacidade de explicar e prever problemas de manutenibilidade de software.

Fernandes et al. (2017) afirmam que ocorrências individuais de *code smells* não são suficientes para caracterizar problemas de manutenibilidade em LPSs. Asssim, os autores avaliam o impacto de aglomerações de *code smells* para o problema de manutenção conhecido como, instabilidade. Os autores definem aglomeração como sendo dois ou mais code smells que afetam o mesmo elemento estrutural de uma LPS, como por exemplo, uma classe. Os autores utilizam quatro LPSs implementadas usando programação orientada a *feature*. Os resultados encontrados sugerem que aglomerações são mais efetivas do que ocorrências individuais de *code smells* para identificar instabilidade em LPSs.

Em Vale et al. (2014) é realizada uma revisão sistemática da literatura na qual os autores encontram 70 diferentes tipos de *bad smells* e 95 métodos de refatoração relacionados ao contexto de LPS. Entretanto, nenhum dos trabalhos encontrados, investigam questões de relações *Inter-smell* no contexto de LPSs implementadadas utilizando o paradigma de orientação a objetos.

Tendo estabelecida e fundamentada a importância de estudar o impacto de relações *Inter-smell* na manutenibilidade e qualidade do código no contexto de LPS orientada a objetos, este trabalho tem como objetivo geral realizar um estudo de caso para investigar as ocorrências de *Inter-smell* na versão orientada a objetos das LPSs MobileMedia e Health Watcher, o impacto dos *Inter-smell* para a qualidade interna do código e manutenibilidade dessas LPSs. Este trabalho destina-se a engenheiros de aplicação, pesquisadores e quaisquer profissionais que trabalhem com o desenvolvimento ou manutenção de LPSs orientada a objetos.

Este trabalho está organizado em outras cinco seções, além desta seção de Introdução. A Seção 2 apresenta os os conceitos que fundamentam o desenvolvimento deste trabalho. A Seção 3 apresenta os trabalhos relacionados que estudam co-ocorrências para verificar problemas

em software. Na Seção 4 são apresentados os passos executados para a realização deste trabalho. A Seção 5 apresenta o Estudo de Caso que foi realizado neste trabalho e suas limitações. A Seção 6 apresenta algumas conclusões, além das possibilidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentados os conceitos que fundamentam o desenvolvimento deste trabalho. Na Seção 2.1 são introduzidos os conceitos sobre Linha de Produto de Software, seu ciclo de vida e exemplos de LPSs. Na Seção 2.2 os conceitos referentes a *code smells*, estratégias para a detecção dos *code smells*, relações *Inter-smell* e métricas para detecção de relações entre os code smells são definidas. Por fim, na Seção 2.3 é apresentada a ferramenta e a suíte de métricas que foram utilizadas para medir a qualidade das LPSs MobileMedia e Health Watcher.

2.1 Linha de Produto de Software

O software está cada vez mais inserido no contexto de muitos produtos competitivos modernos, não importando o seu tamanho, ou complexidade. Portanto, a competitividade no desenvolvimento de software tem se tornado, cada vez mais, uma preocupação para empresas de todos os tamanhos e segmentos de mercado. Particularmente, a abordagem de linhas de produto de software ganhou considerável atenção nos últimos anos por seus benefícios voltados ao custo, tempo de mercado e também outros objetivos de negócio (LINDEN; SCHMID; ROMMES, 2007).

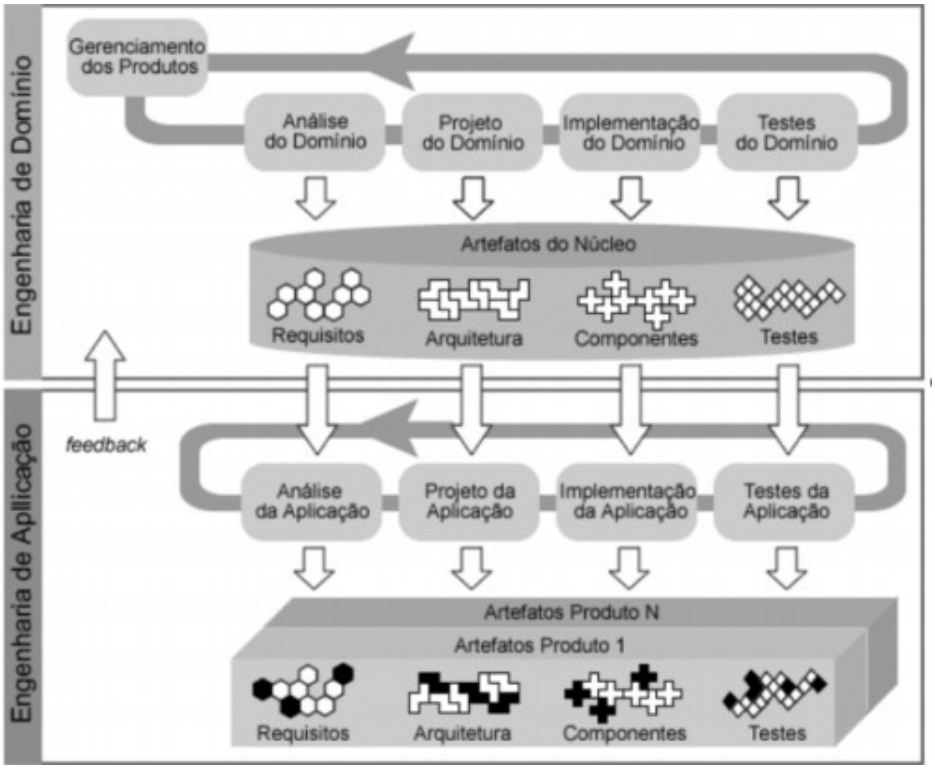
Uma Linha de Produto de Software (LPS) é um conjunto de aplicações com componentes compartilhados, sendo que cada aplicação possui suas próprias restrições e requisitos (SOMMERVILLE, 2011). Linhas de Produto de Software prometem muitos benefícios, como por exemplo: i) produtos feito sob medida de acordo com as necessidades dos clientes; ii) redução dos custos; iii) melhoria na qualidade; iv) tempo de lançamento do produto é reduzido (APEL et al., 2013).

O paradigma de engenharia de LPS é separado em dois processos: Engenharia de Domínio e Engenharia de Aplicação. A engenharia de domínio é o processo responsável pela definição dos aspectos comuns e variáveis da LPS, além de definir o escopo da LPS. A engenharia de aplicação é o processo que se concentra em derivar e construir as aplicações a partir da reutilização dos artefatos do domínio e exploração da variabilidade da linha de produto (POHL; BÖCKLE; LINDEN, 2005).

A Engenharia de Domínio e a Engenharia de Aplicação possuem subprocessos durante o ciclo de vida do desenvolvimento. A Figura 1 mostra cada um destes processos, com

seus respectivos subprocessos.

Figura 1 – Engenharia de Linha de Produto



Fonte – Adaptado de Linden, Schmid e Rommes (2007)

Na Engenharia de Domínio, pode-se observar seus subprocessos (LINDEN; SCHMID; ROMMES, 2007): i) Gerenciamento dos Produtos; ii) Análise do Domínio; iii) Projeto do Domínio; iv) Implementação do Domínio; e iv) Testes do Domínio. O Gerenciamento dos Produtos lida com os aspectos econômicos e estratégia de mercado da linha de produto de software, é nesse subprocesso que é definido o que é incluído no escopo, ou seja, determinando quais serão as features comuns e variáveis dos produtos gerados bem como o cronograma com as datas estimadas para lançar os produtos. A Análise do Domínio se preocupa com todas as atividades de documentação e elicitação dos requisitos comuns e variáveis da linha de produto em geral, isto é, não foca apenas em uma aplicação particular gerada a partir da linha do produto, mas em todas as aplicações que serão geradas. O Projeto do Domínio engloba todas as atividades para definição da arquitetura base da linha de produto, assim, a arquitetura base fornece uma estrutura de alto nível comum para todas as aplicações. A Implementação do Domínio recebe, como entrada, a arquitetura base e a lista dos artefatos que serão reusados e, no final desse subprocesso, gera o projeto detalhado e a implementação dos componentes de software reusáveis. Por fim, os Testes do Domínio são responsáveis pela verificação e validação

dos componentes reusáveis e para diminuir o esforço para teste de aplicação. Esse subprocesso também gera artefatos de testes reusáveis (POHL; BÖCKLE; LINDEN, 2005).

A Engenharia de Aplicação possui (LINDEN; SCHMID; ROMMES, 2007): i) Análise da Aplicação; ii) Projeto da Aplicação; iii) Implementação da Aplicação; e iv) Testes da Aplicação. A Análise da Aplicação preocupa-se com o desenvolvimento da especificação dos requisitos de aplicação, e gera a especificação dos requisitos de aplicações particulares da linha de produto. O Projeto da Aplicação engloba as atividades que produzem a arquitetura de aplicação; ele recebe, como entrada, a arquitetura base, e assim, consegue visualizar a estrutura geral das aplicações para levar em consideração a variabilidade das aplicações da LPS como, por exemplo, um dispositivo de hardware que está somente disponível para uma determinada aplicação. A Implementação da Aplicação é o subprocesso que lida com a criação de uma determinada aplicação, e suas preocupações são a seleção e a configuração dos componentes reusáveis, assim como a implementação desses componentes para uma determinada aplicação; a saída desse subprocesso é a aplicação em execução juntamente com os artefatos do projeto detalhado. Os Testes da Aplicação compreendem as atividades necessárias para validar e verificar se uma aplicação funciona de acordo com suas especificações; esse subprocesso recebe como entrada a aplicação implementada e os artefatos de testes reusáveis para que seja concebido, como saída, o relatório com os resultados de todos os testes realizados, assim como o relatório de problemas com todos os defeitos detectados (POHL; BÖCKLE; LINDEN, 2005).

Este trabalho se concentra na Engenharia de Aplicação, mais precisamente na implementação da aplicação, pois é nesse subprocesso que a aplicação é criada e têm seus componentes implementados. Com a detecção de *code smells* fica mais claro quais partes do código devem ser modificadas ou melhoradas. É no subprocesso da implementação da aplicação que os desenvolvedores deveriam se preocupar com as possíveis anomalias que possam prejudicar a manutenibilidade da LPS, e caso ocorra esse prejuízo, a reusabilidade também sofrerá um impacto negativo ocasionando, dessa maneira, um problema grave no contexto de LPS, já que a reusabilidade é um dos principais pilares da Engenharia de Linha de Produto em relação aos sistemas tradicionais.

2.1.1 Exemplos de LPSs

Na literatura, trabalhos estudam LPSs desenvolvidas com diferentes paradigmas e técnicas de programação como: programação orientada a *features*, programação orientada a

aspectos e programação orientada a objetos. As seguintes LPSs foram encontradas utilizando esses paradigmas:

- Programação orientada a *features*: Notepad (KIM et al., 2010), TankWar (SCHULZE; APEL; KÄSTNER, 2010) e WebStore (GAIA et al., 2014).
- Programação orientada a aspectos: BestLap e MobileMedia (FIGUEIREDO et al., 2008).
- Programação orientadas a objetos: MobileMedia (FIGUEIREDO et al., 2008) e Drawing Shapes (ARAAR; SERIDI, 2016)

Neste trabalho são utilizadas as versões orientada a objetos das LPSs MobileMedia e Health Watcher.

MobileMedia é uma LPS de aplicações que manipulam foto, música e vídeo em dispositivos móveis, como celulares (FIGUEIREDO et al., 2008). Essa LPS foi escolhida para ser estudada neste trabalho por ser uma das poucas LPS implementadas utlizando o paradigma orientado a objetos, pela disponibilidade do código de suas oito *releases* e de seus artefatos, além de ter sido utilizada em vários trabalhos (FIGUEIREDO et al., 2008), (ABDELMOEZ; KHATER; EL-SHOAFY, 2012),(DYER; RAJAN; CAI, 2012a). A Figura 2 apresenta o modelo de *features* da MobileMedia. Essa figura apresenta uma visão simplificada do modelo de *features* de MobileMedia. As *features* obrigatórias são: *create/delete*, *Label Media* e *ViewPlay media*. As *features* alternativas são os tipos de mídia: *foto*, *music* e *video*. Além disso, as features opcionais são: *SMS transfer*, *favourites* e *copy media*.

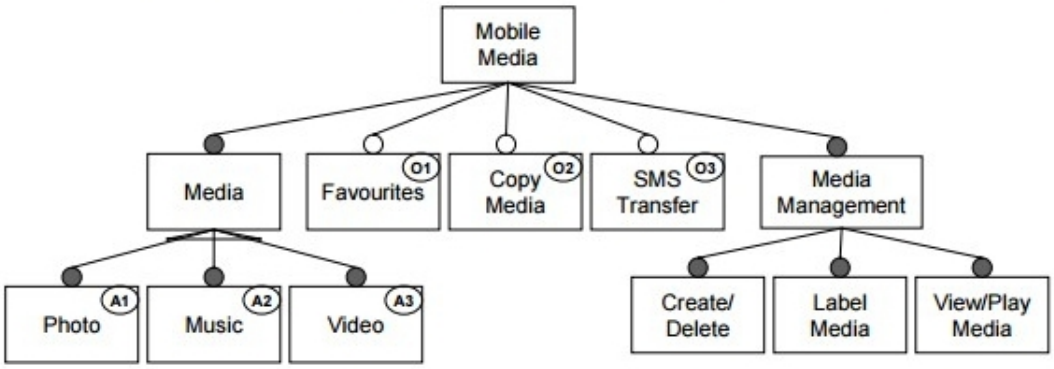
Uma *feature* pode ser utilizada para descrever funcionalidade ou características de qualidade de um sistema específico. A abordagem de modelagem de *features*, permite decomposição hierárquica das *features* em *sub-features*. Uma *feature* pode ser decomposta em *sub-features* obrigatórias, opcionais ou alternativas(POHL; BÖCKLE; LINDEN, 2005).

Riebisch (2003) definem o conceito de *feature* obrigatória, opcional e alternativa da seguinte maneira:

- **Obrigatória** - Uma *feature* é considerada obrigatória quando ela é comum e está presente em todos as produtos da linha.
- **Opcional** - Uma *feature* é considerada opcional quando sua inclusão é optativa.
- **Alternativa** - Uma *feature* é alternativa quando existe um relacionamento entre duas ou mais features vizinhas na hierarquia e apenas uma dessas *features* pode ser escolhida.

O modelo de *features* representa a informação de todos os possíveis produtos em termos de *features* e relações entre eles. O modelo de *features* tem sido amplamente usado na engenharia de software (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010).

Figura 2 – Modelo de *Features* de MobileMedia



Fonte – Adaptado de Figueiredo et al. (2008)

Features estão situadas na solução de domínio, assim a partir da derivação de um novo produto o modelo de features pode ser gerado. Para a construção do modelo de *features*, é necessário ter conhecimento sobre os requisitos e arquitetura da linha de produto, pois esse artefato é utilizado para refletir a variabilidade do sistema em termos de funcionalidade e arquitetura (LINDEN; SCHMID; ROMMES, 2007).

Os cenários compreendem os tipos de mudanças envolvendo *features* obrigatórias, opcionais e alternativas, assim como preocupações em aspectos não-funcionais. A Tabela 1 mostra as diferenças de cada um dos oito produtos e as mudanças de cenários.

Tabela 1 – Cenários em MobileMedia

Release	Descrição	Tipo de Mudança
R1	MobileMedia core	
R2	Tratamento de exceção	Inclusão de preocupação não-funcional
R3	Nova *feature* adicionada para contar o número de vezes que uma foto é vista e ordenar as fotos pela maior frequência de visualização Nova feature adicionada para editar o rótulo da foto.	Inclusão de feature opcional e obrigatória
R4	Nova *feature* para permitir que os usuários especifiquem e visualizem suas fotos preferidas	Inclusão de *feature* opcional
R5	Nova *feature* adicionada para permitir que os usuários mantenham múltiplas cópias de fotos	Inclusão de *feature* opcional
R6	Nova *feature* adicionada para enviar foto para outros usuários via SMS	Inclusão de *feature* opcional
R7	Nova *feature* adicionada para salvar,tocar e organizar música. O gerenciamento de foto foi transformada em uma *feature* alternativa.	Mudança de um *feature* obrigatória para duas alternativas
R8	Nova *feature* adicionada para gerenciar vídeos	Inclusão de *feature* alternativa

Fonte – Adptado de Figueiredo et al. (2008)

Na LPS MobileMedia, um total de sete cenários foram incorporados, gerando oito *releases*. Os cenários compreendem diferentes tipos de mudanças envolvendo *feature* obrigatórias, opcionais e alternativas bem como preocupações com propriedades não-funcionais.

Health Watcher é uma LPS de sistemas WEB, e o principal propósito dessa LPS é permitir que pessoas registrem suas queixas relativas a problemas de saúde. Essa LPS foi inicialmente desenvolvida em 2001, e atualmente conta com nove *releases* (DYER; RAJAN; CAI, 2012b). Essa LPS foi escolhida pela sua disponibilidade, por ser um sistema não trivial com mais de 4000 linhas código implementadas em Java, e por já ter sido utilizada em diversos trabalhos na literatura (GREENWOOD et al., 2007), (SOARES; LAUREANO; BORBA, 2002), (KULESZA et al., 2006).

Os cenários compreendem os tipos de mudanças adição de novas *features*. A Tabela 2 mostra as diferenças de cada um dos três produtos da LPS Health Watcher.

Tabela 2 – Cenários em Health Watcher

Release	Descrição
R1	Health Watcher base
R2	Aplicação do padrão *Command* para remover dependência nos Servlets
R3	Aplicação do padrão *State* para continuar atualizando as queixas depois de elas estarem fechadas.

Fonte – Adptado de Leavens et al. (2012)

Esse trabalho vai estudar e analisar o impacto de relações entre *code smells* conhecidas como *Inter-smell* para manutenibilidade e qualidade do código de oito *releases* de MobileMedia e em três *releases* de Health Watcher.

2.2 *Code Smells*

Code Smells podem indicar problemas relacionados com aspectos de qualidade do código tal como legilibidade e a modificabilidade. Como consequência, podem causar problemas para os desenvolvedores em atividades na fase manutenção do software (FOWLER, 1999).

Fowler (1999) define um catálago de 22 *Code Smells*. Alguns *Code Smells* podem representar problemas no contexto de métodos, tais como *Long Method* e *Feature Envy*. Outros podem representar problemas no contexto de classes, como *Lazy Class*, *Large Class* e *Data Class*. Este trabalho vai utilizar a ferramenta JDeodorant que está apta para detectar os 4 *code smells* que estão definidos abaixo:

Long Method: é o *code smell* que é definido como um método que possui muitos parâmetros e muitas variáveis temporárias. Comentários podem indicar a ocorrência desse *smell*

Feature Envy: ocorre quando um método faz muitas chamadas para outras classes para obter dados ou alguma funcionalidade.

God Class: esse *code smell* é definido como uma classe que controla muitos objetos e possui muita lógica de negócio tornando-a uma classe que "faz tudo".

Type Cheking: esse tipo de *code smell* ocorre quando ao invés de separar um tipo de dado específico, você tem um conjunto de números e strings que representam esse tipo de dado.

Este trabalho também irá utilizar a ferramenta PMD para detectar um *code smell* específico:

Duplicated Code: é o *code smell* mais prejudicial. Ele tende a ser explícito ou sutil. O exemplo mais simples desse *code smell* é quando uma mesma expressão ou códido repetido

está em dois métodos de uma mesma classe.

Antes de detectar as relações *Inter-smell*, deve-se primeiramente fazer a detecção de ocorrências individuais de *code smells*, isso é necessário para verificar e mapear as co-ocorrências dos *code smells*, ou seja a aparição de mais de um *code smell* em um mesmo método, ou a aparição dos *code smells Long Method*, *Feature Envy*, *Type Checking* e *Duplicated Code* em uma *God Class*.

Dessa forma, este trabalho vai verificar a ocorrência de cinco *code smells* nas oito *releases* orientada a objetos da LPS MobileMedia e nas três *releases* de Health Watcher utilizando as ferramentas JDeodorant e PMD.

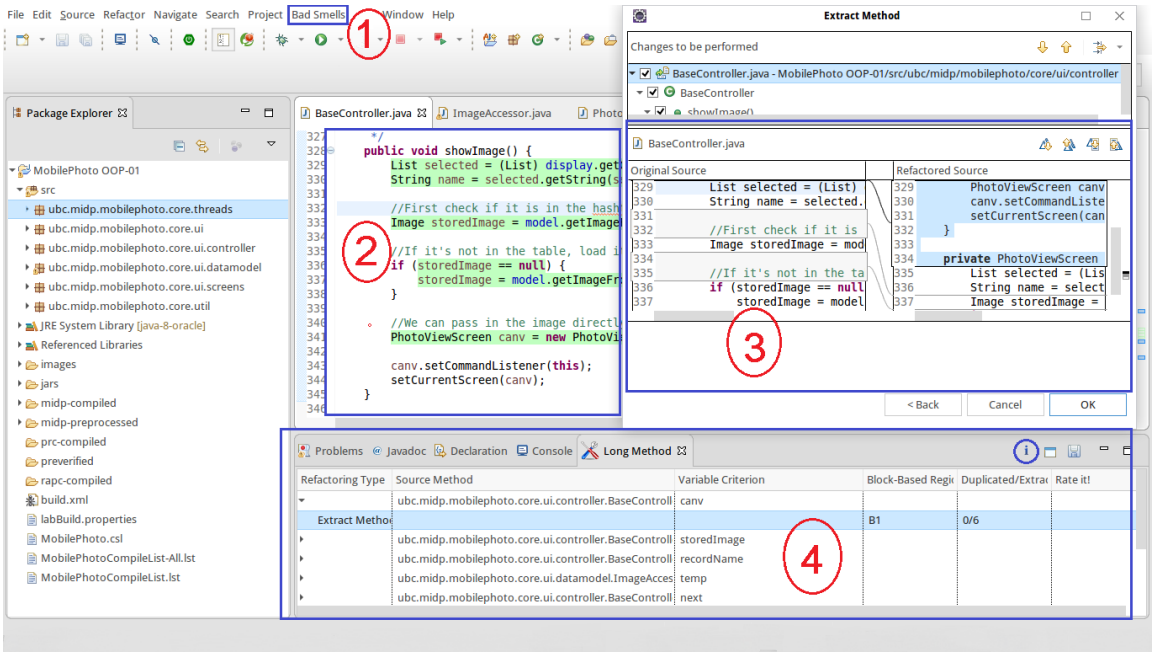
2.2.1 Detecção de Code Smells

Fernandes et al. (2016) realizam um revisão na literatura para avaliar as ferramentas existentes de detecção de *code smells*. No total foram encontradas 84 ferramentas, porém apenas 29 estavam disponíveis para download e instalação. Os resultados indicam que JDeodorante é a ferramenta que indica mais instâncias de *code smells* em sua configuração padrão.

JDeodorant é um *plugin* do Eclipse que detecta problemas de projeto em softwares escritos em Java, esse problemas são conhecidos como code smells. Atualmente quatro *code smells* são suportados pela ferramenta: *Long Method*, *Feature Envy*, *God Class* e *Type/State Cheking* (KAUR; SINGH, 2016). Essa ferramenta foi escolhida pela sua disponibilidade e documentação online, além de já ter sido utilizada e avaliada em diversos trabalhos: (FERNANDES et al., 2016), (MAZINANIAN et al., 2016), (KAUR; SINGH, 2016).

A Figura 3 mostra as principais funcionalidades da ferramenta JDeodorant que serão utilizadas neste trabalho. O Elemento 1 mostra o menu de *Bad Smells*, ou seja, todos os *code smells* que a ferramenta pode detecar. O Elemento 2 mostra o código marcado com a cor verde indicando a presença de *code smell* nesse trecho. O Elemento 3 mostra a sugestão de refatoração do código para eliminação do *code smell* a ferramenta JDeodorant fornece o código atual e uma visão prévia do código refatorado bem como as partes serão modificadas. O Elemento 4 mostra qual *code smell* que está sendo analisado, qual tipo de refatoração pode ser aplicada e as entidades afetadas pelo *code smell*. Assim, a ferramenta JDeodorant vai ser utilizada para detecção de *code smells* nas oito *releases* da LPS MobileMedia e nas três *releases* da LPS Health Watcher.

Figura 3 – Funcionalidades de JDeodorant

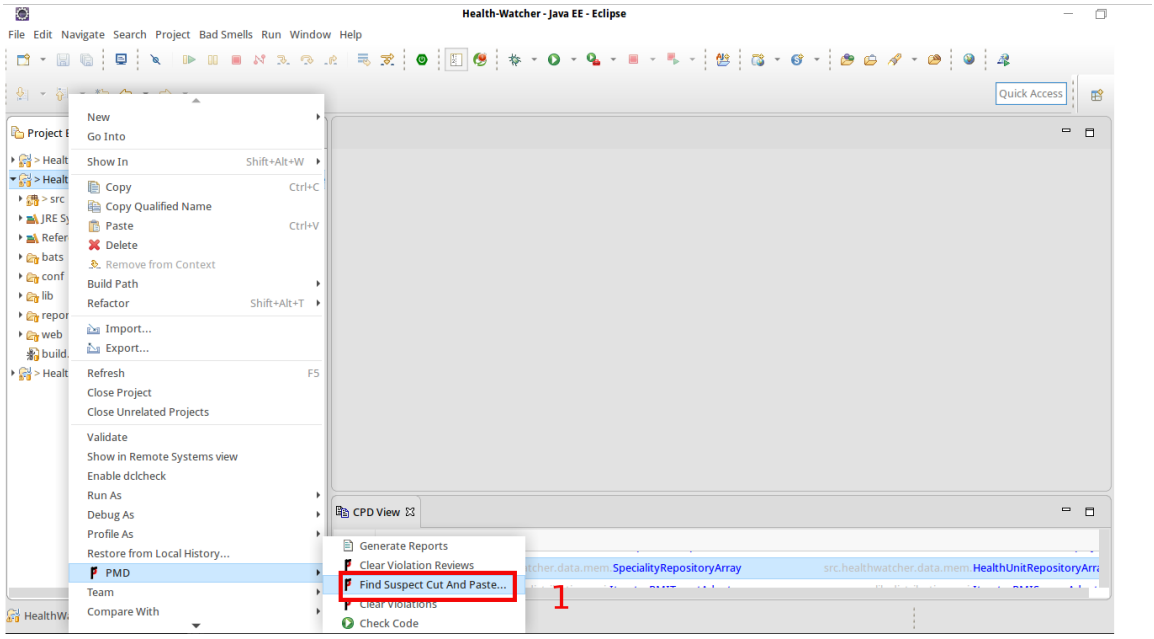


Fonte – Elaborada pelo Autor

Para identificar as ocorrências de *Duplicated Code* foi utilizada a ferramenta PMD. Essa ferramenta foi utilizada por ser *open source* e ter comunidade ativa, pela sua disponibilidade para ser instalada como um plugin no eclipse e segundo Li e Cui (2010), a ferramenta PMD é altamente eficiente. Outra justificativa é que ela tem sido utilizada em trabalhos na literatura: (SHEN; RUAN, 2008),(FONTANA et al., 2013),(CHAHAR; CHAUHAN; DAS, 2012).

A Figura 4 mostra a funcionalidade de PMD que foi utilizada para detectar as ocorrências de *Duplicated Code*. Maiores detalhes da detecção das ocorrências de *Duplicated Code* serão apresentados na seção 4.

Figura 4 – Ferramenta PMD



Fonte – Elaborada pelo Autor

2.2.2 Relações Inter-Smell

Relações *Inter-smell* ocorrem quando existem relações e dependências entre *code smells*. Um exemplo de relação entre *code smells*: uma *Large Class* pode conter *Duplicated Code* (PIETRZAK; WALTER, 2006).

Existem vários trabalhos na literatura que estudam apenas ocorrências individuais de *code smells* (KHOMH; PENTA; GUEHENEUC, 2009; OLBRICH; CRUZES; SJØBERG, 2010; OLBRICH et al., 2009; SJØBERG et al., 2013). No entanto, existem poucos trabalhos que abordam ou analisam as relações *Inter-Smell*. Dessa forma, são necessários mais estudos empíricos considerando essas relações (YAMASHITA et al., 2015).

Pietrzak e Walter (2006) 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 para os software:

- ***Plain Support*** é 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*** é o contrário 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": $(A \not\Rightarrow B)$.
- ***Inclusion*** é 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: $(A \Rightarrow B)$.

Yamashita et al. (2015) afirmam que vários estudos têm discutido o impacto de *code smells* individuais sob diferentes aspectos de manutenibilidade em sistemas tradicionais, porém, existem poucos trabalhos que analisam os relacionamentos entre *code smells* e suas implicações na manutenibilidade de software

Este trabalho leva em consideração os tipos de relações *Inter-smell* descritos nesta Seção, para analisar qual o efeito dessas relações para a qualidade do código e manutenibilidade das LPSs MobileMedia e Health Watcher.

2.2.3 Detecção de Relações Inter-Smell

Para identificar o número total e quais as relações *Inter-smell* em cada *release* de MobileMedia e de Health Watcher são utilizadas as seguintes métricas definidas por (LOZANO; MENS; PORTUGAL, 2015):

- *exist(CS)*: número de entidades (Classes ou Métodos) onde o *code smell* CS foi encontrado.
- *co-exist(CS1, CS2)*: números de classes e métodos na qual ambos *code smells* foram encontrados.

Usando as métricas base acima, são derivadas as seguintes métricas indiretas:

- $exist - overlap(CS1, CS2) = \dfrac{co - exist(CS1, CS2)}{exist(CS1)}$

O processo para identificação de relações *Inter-smell* é totalmente manual e são utilizadas as métricas descritas acima, para de fato, conseguir identificar essas anomalias. As métricas *exist(CS)* e *co-exist(CS1,CS2)* são as primeiras à serem calculadas. Após o cálculo dos valores dessas métricas, a métrica $exist - overlap(CS1, CS2)$ usa os valores dessas duas métricas e 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 $(A \not\Rightarrow B)$: se $co - exist(A,B) \approx 0$
- Inclusion $(A \Rightarrow B)$: se $exist - overlap(A,B) \approx 1$

Este trabalho utiliza essas métricas no processo de identificações das relações *Inter-smell* definidas na seção 2.2.2.

2.3 Medição e Manutenibilidade de Código

Segundo a ISO (2005) a manutenibilidade é um aspecto da qualidade de software que está relacionada ao grau de eficácia e eficiência com a qual um produto ou sistema tem para ser modificado a fim de corrigir seus defeitos, melhorar a performace, se adaptar à novos requisitos e ás mudanças no ambiente.

Medição de software é uma avaliação essencial para uma boa engenharia de software. Para saber se o projeto de software está em conformidade com os requisitos, se é de alta qualidade e se o código já está pronto para ser lançado, muitos desenvolvedores utilizam os resultados das métricas como resposta. Consequentemente, desenvolvedores devem ser capaz de avaliar a manutenibiliade de um produto para verificar o que pode ser melhorado (FENTON; BIEMAN, 2014).

As métricas propostas por Chidamber e Kemerer (1994) são pioneiras na área de métricas orientadas a objeto e apresentam uma base teórica para medir código orientado a objeto. Atributos como: tamanho, complexidade, herança, acoplamento e coesão são levados em consideração pelas *CK metrics*. Essa suíte de métricas tem sido a mais utilizada na comunidade científica e suas definições têm sido usadas como base para a construção de novas métricas (NUÑEZ-VARELA et al., 2017).

Neste trabalho a suíte *CK metrics* (CHIDAMBER; KEMERER, 1994) foi escolhida para ser utilizada na verificação da qualidade interna das Linhas de Produto MobileMedia e Health Watcher devido essas métricas já terem sido utilizadas diversos trabalhos (MALHOTRA; CHUG, 2016; DYER; RAJAN; CAI, 2012a; ELISH; ELISH, 2009).

- *Weighted Methods per Class (WMC)* é definida como o somatório da complexidade ciclomática de todos os métodos locais definidos em uma classe. É aconselhável que o valor de WMC seja o menor possível.
- *Depth of Inheritance Tree (DIT)* mede o caminho máximo de herança de qualquer

classe para sua superclasse.

- *Number of Children (NOC)* é usada para mensurar a profundidade da hierarquia de classes. Essa profundidade é contada à partir do número de filho de uma superclasse.
- *Coupling between Objects (CBO)* é usada para medir o acoplamento entre duas classes. Duas classes são consideradas acopladas se os métodos declarados em uma classe usa variáveis ou métodos definidos em outra classe.
- *Response for a Class (RFC)* é 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. Classes coesas são consideradas mais fáceis de manter. Apesar do nome, quanto maior for o valor dessa métrica, menor será a coesão de uma classe.

Para calcular as métricas descritas na seção 2.3, foi utilizada a ferramenta ckjm. Essa ferramenta calcula as métricas propostas por Chidamber e Kemerer (1994) processando o bytecode de arquivos Java compilados. A métricas são extraídas classe por classe (SPINELLIS, 2005).

Em Singh e Malhotra (2012), os autores afirmam que as métricas que foram definidas na seção 2.3 são negativamente correlacionadas com a manutenibilidade de software, ou seja, quanto menor o valor das métricas, maior a manutenibilidade do software.

Para avaliar a manutenibilidade das LPSs a partir dos valores das métricas, este trabalho utiliza a mesma abordagem de Tarwani e Chug (2016), na qual os autores utilizam o somatório de todas as métricas para medir e comparar a manutenibilidade,quanto menor o valor do somatório, maior a manutenibilidade. As LPSs MobileMedia e Health Watcher são medidas com seus respectivos códigos originais, ou seja, antes do processo de remoção dos *Inter-smell*, para que após o processo de remoção dessas co-ocorrências, 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 TRABALHOS RELACIONADOS

Esta Seção apresenta os estudos relacionados encontrados na revisão bibliográfica que se relacionam a este trabalho.

Yamashita e Moonen (2013a) analisam o impacto das relações *Inter-Smell* na manutenibilidade de quatro sistemas industriais de tamanho médio escritos em Java. Os autores analisam doze *code smells* e se utilizam da ajuda de 6 engenheiros de software que são contratados para manter os quatro sistemas por um período de quatro semanas. Durante quatro semanas os engenheiros implementam requisições de mudança e os autores registram diariamente quais os problemas encontrados e quais artefatos eram relacionados a esses problemas de manutenção. Usando análise de componentes principal, os autores conseguem indicar se os problemas são ocasionados pelos *code smells* individuais ou pelas relações *Inter-Smell*. 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. No entanto, os autores fazem um estudo apenas com sistemas tradicionais e eles não levam em consideração o contexto de Linhas de Produto de Software. Neste trabalho, também será estudado o impacto das relações *Inter-Smell* na manutenibilidade, mas no contexto de Linhas de Produto de Software e não em sistemas tradicionais.

Macia et al. (2012) realizam um estudo para saber se a detecção automática de *code smells* é relevante para ajudar a descobrir problemas de modularidade na arquitetura do software. Os autores analisam 38 versões de 5 aplicações de domínios diferentes. Os *Code Smells* são detectados utilizando estratégias de detecção automática já definidas na literatura. Os autores consultam os arquitetos originais dos sistemas para identificar problemas arquiteturais para correlacionar os problemas encontrados com *Code Smells*. Das 5 aplicações utilizadas neste estudo, 2 são Linhas de Produto de Software: (*Mobile Media* e *Aspectual Media*), implementadas utilizando Java e AspectJ respectivamente. Os resultados encontrados pelos autores indicam que: i) atuais estratégias não são precisas para identificar *code smells* que prejudicam a modularidade da aquitetura; ii) co-ocorrências de *code smells* parecem ser melhores indicadores de problemas na modularidade da arquitetura do que ocorrências individuais de *code smells*. Assim, os autores reforçam que os desenvolvedores devem ser avisados sobre os impactos e a existência co-ocorrências de *code smells* para que, façam refatorações e remoções dessas anomalias em estágios iniciais da implementação. No entanto, os autores verificam questões relacionadas à problemas na modularidade da arquitetura e não levam em consideração a manutenibilidade

dos sistemas. De forma semelhante, neste trabalho, as co-ocorrências de *code smells*(mais precisamente relações *Inter-smell*) serão estudadas e analisadas para verificar o grau de impacto dessas anomalias para a manutenibilidade e qualidade do código nas LPSs MobileMedia e Health Watcher.

Fernandes et al. (2017) afirmam que ocorrências individuais de *code smells* não são suficientes para caracterizar problemas de manutenibilidade em Linha de Produto de Software (LPS). Asssim, os autores avaliam o impacto de aglomerações de *code smells* para o problema de manutenção conhecido como instabilidade. Os autores definem aglomeração como sendo dois ou mais *code smells* que afetam o mesmo elemento estrutural de uma LPS, como por exemplo, uma classe. Os autores utilizam quatro LPSs implementadas usando programação orientada a *feature*. Os resultados encontrados sugerem que *code smells* geralmente estão interconectados com outros *code smells* em elementos críticos de LPS, tal como, classes e 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 cosideração LPSs implementadas utilizando programação orientada a *feature* e apesar de levar em cosideração aglomerações, não levam em consideração as relações *Inter-smell*. O presente trabalho também avalia o impacto de relações de *code smells* para a manutenibilidade de uma LPS, porém os tipos de relações avaliadas serão relações *Inter-smell* e as LPSs utilizadas neste trabalho são as versões orientada a objetos de MobileMedia e Health Watcher.

4 METODOLOGIA

Nesta seção é apresentada a metodologia do trabalho que será guiada pelas seguintes passos: i) Identificar LPS orientada a objetos; ii) Identificar e coletar os *code smells*; iii) Identificar relações *Inter-smell*; iv) Medir e refatorar a qualidade interna do código; e v) Analisar o impacto das relações *Inter-smell* na manutenibilidade da LPS. Os passos da metodologia embasam o planejamento e execução do estudo de caso com o objetivo de analisar o impacto dos *inter-smells* na manutenibilidade de LPSs orientadas a objetos.

4.1 Identificar LPS orientada a objetos

Foi realizada uma busca na literatura para identificar LPSs orientada a objetos para serem utilizadas neste trabalho. Assim, foi escolhida a versão orientada a objetos da LPS MobileMedia pela disponibilidade de código de suas oito *releases* e de seus artefatos e por sua ampla aceitação no meio acadêmico, visto que, já foi utilizada em diversos estudos (FIGUEIREDO et al., 2008; ABDELMOEZ; KHATER; EL-SHOAFY, 2012; DYER; RAJAN; CAI, 2012a). A outra LPS encontrada foi *Health Watcher*, neste trabalho foram analisadas três *releases* da mesma, essa LPS foi escolhida por ter código fonte disponível, por apresentar um número elevado de linhas de código por *release* e por já ter sido utilizada em vários estudos da literatura (GREENWOOD et al., 2007), (SOARES; LAUREANO; BORBA, 2002), (KULESZA et al., 2006).

4.2 Identificar e coletar os *code smells*

Os *code smells* das LPSs MobileMedia e *Health Watcher* foram identificados utilizando as ferramenta JDeodorant e PMD, essas ferramentas foram escolhidas pela disponibilidade e pela utilização em trabalhos anteriores para detecção de *code smells* (PAIVA[1] et al., 2015),(SHEN; RUAN, 2008). Atualmente 4 *code smells* são suportados pela ferramenta JDeodorant: *Long Method*, *Feature Envy*, *God Class* e *Type Cheking* (KAUR; SINGH, 2016). A ferramenta PMD foi utilizada para detectar o *code smell Duplicated Code*.

4.3 Identificar as relações *Inter-smell*

Após passo de identificação de ocorrências individuais de *code smell* na LPS MobileMedia, são detectadas as relações *Inter-smell* dos 5 *code smells*. Os tipos de relacionamento são descritos na Seção 2.2.2. Para identificar as relações *Inter-smells* serão utilizadas as métricas definidas na Seção 2.2.3.

4.4 Medir e refatorar a qualidade interna do código

Nesse procedimento é mensurada primeiramente a manutenibilidade das LPSs MobileMedia e *Health Watcher* contendo as relações *Inter-smell*. Para a medição são utilizadas métricas já conhecidas na literatura e que são descritas na Seção 2.3, as métricas são coletadas utilizando ferramenta ckjm (SPINELLIS, 2005). Um catálogo de métodos para refatoração de vários tipos de *code smells* foram definidos por Fowler (1999). Dessa forma, eliminando a ocorrência de alguns *code smells* pode-se descaracterizar algumas co-ocorrências de *Inter-smell*. Para eliminar as relações *Inter-smell* serão realizadas refatorações nas oito *releases* da LPS MobileMedia e nas três *releases* da *Health Watcher* utilizando o catálogo de Fowler (1999) que a ferramenta JDeodorant implementa. Após esse passo, é realizada novamente a medição das métricas das *releases* sem as coocorrências de *Inter-smell* para que seja feita a comparação entre *releases*.

4.5 Analisar o impacto das relações *Inter-smell* na manutenibilidade da LPS

Após a medição da manutenibilidade das *releases* refatoradas das LPSs MobileMedia e *Health Watcher* é realizada a comparação de manutenibilidade com as *releases* originais contendo as relações *Inter-smell*. As comparações serão verificadas *release* por *release*, por exemplo, a *release* 1 original do MobileMedia é comparada com a *release* 1 refatorada do MobileMedia e o mesmo ocorre com a LPS *Health Watcher*.

Após a comparação de todas as *releases* originais com as refatoradas, espera-se verificar o impacto das co-ocorrências de *Inter-smell* para a manutenibilidade.

5 ESTUDO DE CASO

Este capítulo apresenta um estudo de caso sobre o impacto de *Inter-smell* na manutenibilidade de LPSs. A Seção 5.1 apresenta o planejamento do estudo de caso, detalhando as questões de pesquisa, os objetos do caso de estudo e os procedimentos de coleta de dados. Os resultados deste estudo de caso são apresentados na seção 5.2. Na seção 5.3 é realizada um discussão dos resultados obtidos neste estudo de caso. A seção 5.4 apresenta as ameaças à validade do estudo.

5.1 Planejamento do Estudo de Caso

Para avaliar o impacto de ocorrências de *Inter-smell* para a manutenibilidade de LPSs orientada a objetos, esse trabalho utiliza a abordagem de estudo de caso. O objetivo de utilizar este método de pesquisa é investigar algum fenômeno dentro do seu contexto real, descrevendo ou analisando um sistema delimitado (ou caso) (YIN, 2003). Um estudo de caso é caracterizado pela busca de compreensão, onde o pesquisador é o principal instrumento de coleta e análise dos dados, é uma estratégia de investigação indutiva e o produto final é bem descritivo (MERRIAM; TISDELL, 2015).

Para atingir o objetivo do estudo, ocorrências de *code smells* foram coletadas, relações *Inter-smell* identificadas, medição da qualidade de todas as *releases* das LPSs MobileMedia e *Health Watcher*, remoção de ocorrências de *Inter-smell* nas *releases* e comparação das *releases* com e sem ocorrências de *Inter-smell*.

5.1.1 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 (YAMASHITA et al., 2015),(PIETRZAK; WALTER, 2006), (YAMASHITA; MOONEN, 2013a), (FERNANDES et al., 2017). No entanto, nenhum desses trabalhos estudou o impacto de *co-ocorrências* ou relações entre *code smells* para a qualidade e manutenibilidade do código em LPSs orientada a objetos. Portanto, o principal objetivo deste estudo de caso é investigar qual o impacto da presença dos *Inter-smell* para a manutenibilidade e qualidade do código de duas LPSs orientada a objetos.

Para atingir esse objetivo, foram definidas três questões de pesquisa:

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?

5.1.2 Objetos do Estudo de Caso

Um dos estudos de caso deste trabalho é a LPS MobileMedia. MobileMedia é uma LPS de aplicações que manipulam foto, música e vídeo em dispositivos móveis, como celulares (FIGUEIREDO et al., 2008). Essa LPS é implementada utilizando o padrão arquitetural Model-View-Controller (MVC), e ela foi escolhida pela disponibilidade de código de suas oito *releases*, seus artefatos e por sua ampla aceitação no meio acadêmico, visto que, já foi utilizada em diversos estudos (FIGUEIREDO et al., 2008; ABDELMOEZ; KHATER; EL-SHOAFY, 2012; DYER; RAJAN; CAI, 2012a).

O outro estudo de caso é a LPS *Health Watcher*. Neste trabalho foram analisadas três *releases* dessa LPS, *Health Watcher* é uma LPS de sistemas WEB, e o principal propósito dessa LPS é permitir que pessoas registrem suas queixas relativas a problemas de saúde. Essa LPS foi inicialmente desenvolvida em 2001, e atualmente conta com nove *releases* (DYER; RAJAN; CAI, 2012b). Ela foi escolhida por ter código fonte disponível, por apresentar um número elevado de linhas de código por *release* e por já ter sido utilizada em vários estudos da literatura (GREENWOOD et al., 2007), (SOARES; LAUREANO; BORBA, 2002), (KULESZA et al., 2006).

5.1.3 Procedimentos para a Coleta dos Dados

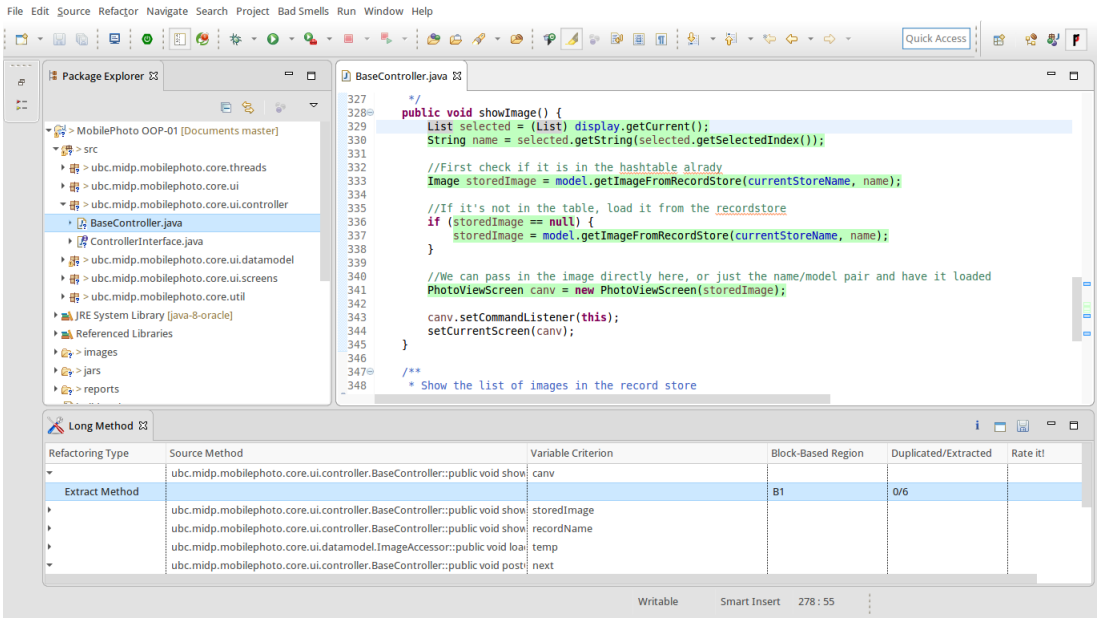
Para responder as questões de pesquisa, um conjunto de dados foi coletados das LPSs MobileMedia e *Health Watcher*. Os passos para a coleta foram os seguintes: i) Identificar e coletar *code smells*; ii) Identificar relações *Inter-smell*; iii) Medir e remover as ocorrências de *Inter-smell*; e iv) Analisar o impacto de ocorrências de *Inter-smell* na manutenibilidade da LPS.

i) Identificar e coletar *code smells*: os *code smells* das LPSs MobileMedia e *Health Watcher* foram identificados utilizando as ferramenta JDeodorant e PMD, essas ferramentas foram escolhidas pela disponibilidade e pela utilização em trabalhos anteriores para detecção de *code smells* (PAIVA[1] et al., 2015),(SHEN; RUAN, 2008). Atualmente 4 *code smells* são

suportados pela ferramenta JDeodorant: *Long Method*, *Feature Envy*, *God Class* e *Type Cheking* (KAUR; SINGH, 2016). A ferramenta PMD foi utilizada para detectar o *code smell Duplicated Code*.

Foi realizado o levantamento e armazenamento do número total de *code smells* presentes em cada *release* das LPSs MobileMedia e *Healt Watcher*.

Figura 5 – Detecção de *code smells* na ferramenta JDeodorant

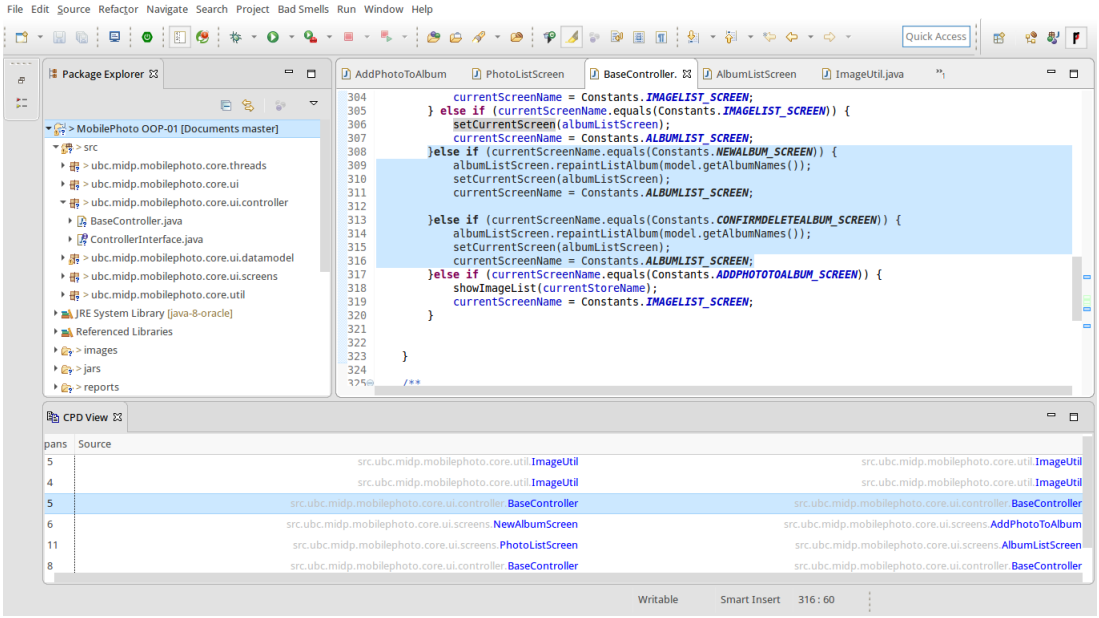


Fonte – Elaborado pelo Autor

A Figura 5 mostra o processo de detecção do *code smell Long Method* utilizando a ferramenta JDeodorant. Nesta Figura pode ser observado o critério de seleção da ferramenta para identificar esse *code smell*, que está dermarcado com a cor verde.

Para a identificação do *code smell Duplicated Code* foi utilizada a ferramenta PMD como um *plugin* do eclipse. A Figura 6 mostra a lista de classes com esse *smell*. Nesse caso foi selecionada a classe *BaseController* para verificação e como pode ser observado, a ferramenta seleciona o trecho de código que está duplicado.

Figura 6 – Detecção de *Duplicated Code* com a ferramenta PMD



Fonte – Elaborado pelo Autor

Após identificação dos *code smells*, deu-se o início do armazenamento dessas anomalias em planilhas. A Tabela 3 mostra o padrão ou representação utilizada para armazenamento das ocorrências individuais de *code smells*.

Tabela 3 – Padrão de armazenamento das ocorrências de *code smell*

Pacote	Classe	Método	LM	FE	TC	GC	DC
pacote1	Classe1	metodo1()	X				

Após a detecção de ocorrências individuais de *code smells*, foi percebido que alguns *code smells* estavam "agrupados", ou seja, em um primeiro caso foi encontrado que em um mesmo método existia a presença de mais de um *code smell*, e que em um segundo caso foi detectado a presença de outros *code smells* em uma *God Class*. A Tabela 4 mostra as duas situações na Classe2 e Classe3, respectivamente.

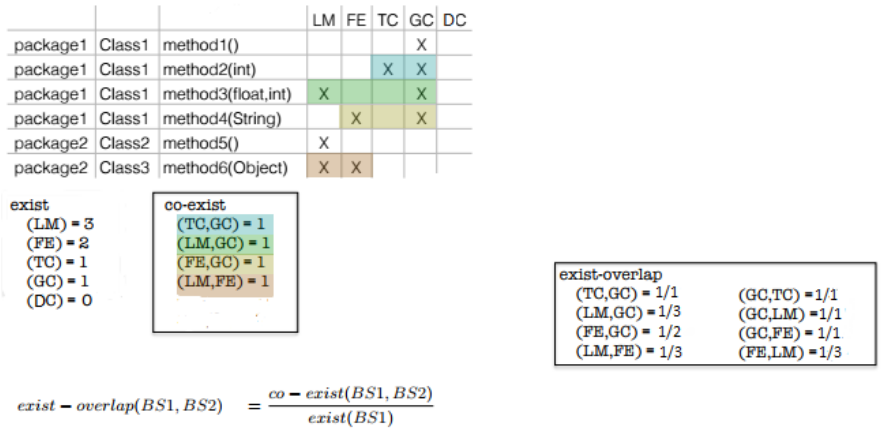
Tabela 4 – Padrão de armazenamento das co-ocorrências de *code smell*

Pacote	Classe	Método	LM	FE	TC	GC	DC
pacote1	Classe1	metodo1()	X				
pacote1	Classe2	método2()	X				X
pacote1	Classe3	método3()	X			X	

Desta forma, com os *code smells* e as co-ocorrências (*Inter-smell*) já detectadas, o processo para Identificação das relações *Inter-smell* pôde ter início.

ii) Identificar relações *Inter-smell*: após o procedimento de identificação de ocorrências individuais de *code smell* e das co-ocorrências desses *code smells* nas LPSs MobileMedia e *Health Watcher*, foram detectadas as relações *Inter-smell* dos 5 *code smells*, *Duplicated Code* (DC), *God Class* (GC), *Long Method* (LM), *Feature Envy* (FE) e *Type Checking* (TC). Os tipos de relacionamento são descritos na Seção 2.2.2. Para identificar as relações *Inter-smells* foram utilizadas as métricas definidas na Seção 2.2.3.

Figura 7 – Processo para identificar relações *Inter-smell*



Fonte – Elaborado pelo Autor

A Figura 7 mostra como foi o procedimento para identificação das relações *Inter-smell*. Esse processo foi feito totalmente de maneira manual, os cálculos e resultados obtidos foram armazenados em planilhas. Note que as co-ocorrências de *code smells* foram marcadas com cores para facilitar no processo de identificação, após isso, as métricas indiretas *exist* que é o número de ocorrências de um *code smell* e *co-exist* que é o número de co-ocorrências de um *code smell* (CS1) juntamente com outro *code smell* (CS2), foram armazenadas. Após o cálculo das métricas indiretas, foi utilizada a métrica $exist - overlap(CS1, CS2)$ que é equivalente a métrica *co-exist* de um CS1 e CS2 dividido pelo *exist* do CS1. Como exemplo, note que o $exist - overlap(TC, GC)$ é equivalente a 1/1 pois o *co-exist(TC,GC)* = 1 e o número de ocorrências de TC = 1. Perceba que o inverso também acontece, o $exist - overlap(GC, TC)$ é equivalente a 1/1 pois o *co-exist(GC,TC)* = 1 e o número de ocorrências de GC = 1.

Após calcular o $exist - overlap(CS1, CS2)$, as relações *Inter-smell* já podem ser identificadas a partir das seguintes regras:

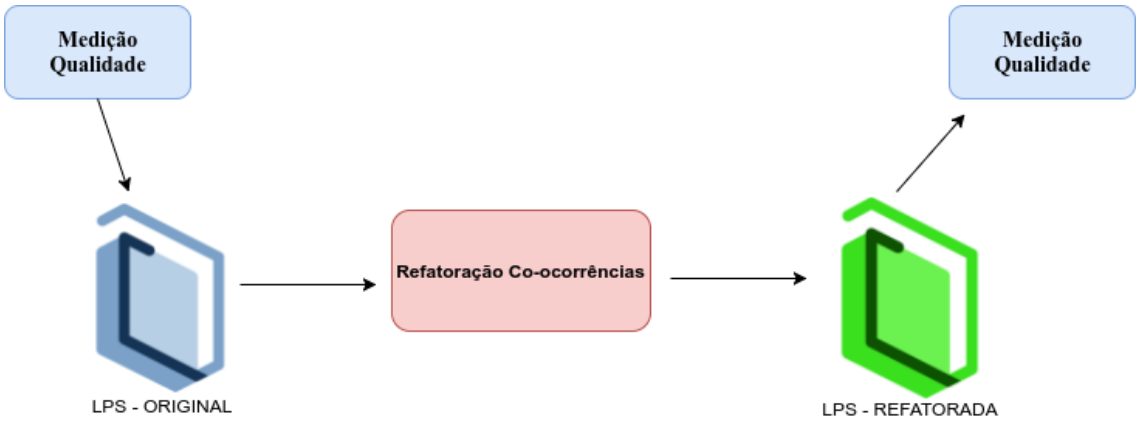
- Plain Support $(A \rightarrow B)$: se $exist - overlap(A, B) > exist - overlap(B, A)$

- Mutual Support $(A \leftrightarrow B)$: se $exist - overlap(A,B) \approx exist - overlap(B,A)$
- Rejection $(A \not\Rightarrow B)$: se $co - exist(A,B) \approx 0$
- Inclusion $(A \Rightarrow B)$: se $exist - overlap(A,B) \approx 1$

A relação *Rejection* é a mais fácil de ser identificada, como exemplo, na Figura 7 o *code smell* DC não possui nenhuma ocorrência individual, portanto nesse exemplo, ele também não terá co-ocorrências com outros *code smells*, e nesse caso, assume-se que o *co-exist* de (DC,GC) = 0, (DC,LM) = 0, (DC,TC) = 0 e (DC,FE) = 0.

iii) Medir e remover as ocorrências de *Inter-smell*: nesse procedimento foi mensurada a qualidade interna do código de cada classe das LPSs MobileMedia e *Health Watcher* contendo as relações *Inter-smell*, apenas essas classes tiveram as métricas coletadas. Para o processo de medição foi utilizada a ferramenta ckjm que implementa as métricas já conhecidas na literatura e que são descritas na Seção 2.3.

Figura 8 – Processo de medição e refatoração de *Inter-smell*



Fonte – Elaborado pelo Autor

A Figura 8 mostra o processo de medição e refatoração neste trabalho. Primeiramente foi utilizada a ferramenta ckjm para a medição das métricas descritas na seção 2.3, nessa primeira parte foram medidas todas as classes com co-ocorrências de *code smells* nas oito *releases* de MobileMedia e nas três *releases* de Health Watcher. Ou seja, a medição antes do processo de refatoração.

A Tabela 5 mostra como foram armazenadas as métricas de cada classe com co-ocorrências de *code smells*.

Tabela 5 – Exemplo de armazenamento das métricas

Pacote	Classe	WMC	DIT	NOC	CBO	RFC	LCOM
pacote1	Classe1	20	1	0	21	142	0
pacote1	Classe2	2	0	0	14	24	1
pacote1	Classe3	6	1	0	7	25	0

Após esse processo inicial de medição, deu-se início ao processo de refatoração das co-ocorrências, ou seja, todas as classes e métodos que possuíam co-ocorrências foram refatoradas de modo que apenas um *code smell* individual continuasse. Na Tabela 6 pode ser observado um exemplo hipotético de co-ocorrências de *code smells* antes do processo de refatoração.

Tabela 6 – Co-ocorrências de *code smells* antes da refatoração

Pacote	Classe	Método	LM	FE	TC	GC	DC
pacote1	Classe1	metodo1()	X				X
pacote1	Classe2	método2()		X		X	
pacote1	Classe3	método3()	X		X		

O resultado do processo de refatoração das co-ocorrências pode ser visualizado na Tabela 7.

Tabela 7 – Resultado após o processo de refatoração

Pacote	Classe	Método	LM	FE	TC	GC	DC
pacote1	Classe1	metodo1()	X				
pacote1	Classe2	método2()				X	
pacote1	Classe3	método3()			X		

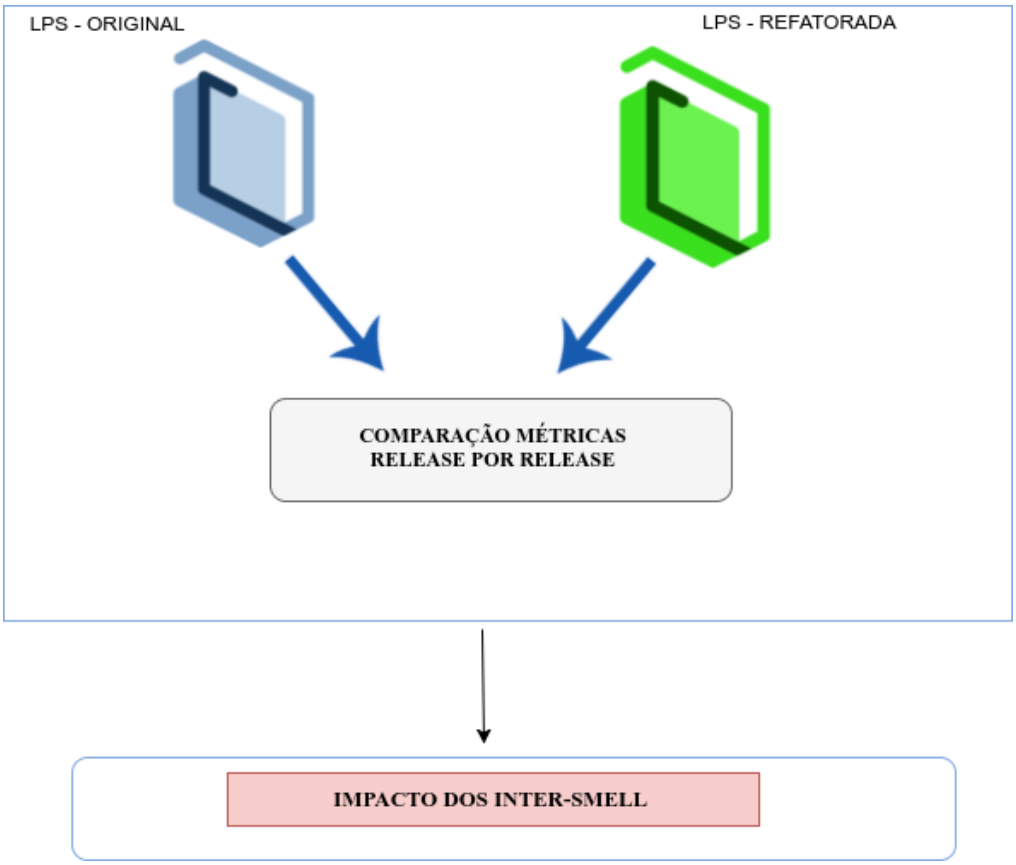
Como pode ser visualizado na Tabela 7, foi eliminado pelo menos um *code smell* para descaracterizar a co-ocorrência. Não existiu critério na escolha do *code smell* a ser removido, o objetivo neste passo era apenas descaracterizar a co-ocorrência na classe ou no método.

Como pode ser observado na Figura 8, após a refatoração das co-ocorrências, a etapa de medição foi realizada novamente nas classes que tiveram co-ocorrências removidas seguindo o mesmo padrão da Tabela 5.

iv) Analisar o impacto de ocorrências de *Inter-smell* na manutenibilidade da LPS: a Figura 9 mostra o processo de comparação após o processo de refatoração. Todas as *releases* foram comparadas, por exemplo, a *release* 1 original de MobileMedia foi comparada com a *release* 1 refatorada de MobileMedia e esse processo continuou para as sete *releases*

restantes de MobileMedia. O processo foi igual para Health Watcher, as três *releases* originais foram comparadas com as três refatoradas.

Figura 9 – Processo de Comparação da releases das LPSs



Fonte – Elaborado pelo Autor

A comparação foi realizada através dos resultados das métricas de qualidade, e foi feita a comparação apenas nas classes que tinham co-ocorrências de *code smells*. Para avaliar a manutenibilidade das LPSs a partir dos valores das métricas, este trabalho utiliza a mesma abordagem de Tarwani e Chug (2016), na qual os autores utilizam o somatório de todas as métricas para medir e comparar a manutenibilidade, quanto menor o valor do somatório, maior a manutenibilidade. Assim, este trabalho conseguiu avaliar se a presença dessas co-ocorrências ou *Inter-smell* prejudicaram a qualidade e a manutenibilidade do código dessas duas LPSs orientada a objetos.

5.2 Resultados

Nesta seção são apresentados o resultados que respondem as questões de pesquisa deste trabalho.

5.2.1 QP1:Quais são os code smells mais frequentes em LPSs orientada a objetos?

O *code smell* mais comum entre todas as oito *releases* de MobileMedia, foi o *code smell Duplicated Code* (DC) jutamente com *Long Method* (LM). A Tabela 8 mostra o número de ocorrências de *code smells* em cada *release* de MobileMedia. Como pode ser observado, na maioria das *releases*, à medida em que o número de LM aumenta, o número de ocorrências de DC também obtêm um crescimento significante. Nas *releases* 6, 7 e 8 o número de ocorrências de *God Class* diminuiram, mas em compensação, ocorrências de DC aumentaram de maneira mais elevada em relação as outras *releases*.

Tabela 8 – Ocorrências de *code smells* em MobileMedia

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

Como na LPS MobileMedia, o code smell com mais ocorrências em *Health Watcher* foi *Duplicated Code* juntamente com *Long Method*. A Tabela 9 mostra o número de ocorrências de *code smells* em cada *release* de HealthWatcher. E como também ocorreu em MobileMedia, à medida em que o número de LM aumenta, o número de ocorrências de DC também obtêm um crescimento significante. Nas *releases* 2 e 3 o o número de ocorrências de DC deram um salto significativo em relação a *release* 1.

Tabela 9 – Ocorrências de *code smells* em Health Watcher

LPS	GC	FE	LM	DC	TC
Health Watcher - 1	0	4	73	378	3
Health Watcher - 2	0	4	87	783	3
Health Watcher - 3	0	0	88	804	4

Como pode ser observado, apesar do elevado número de ocorrências de LM, não foi detectada nenhuma ocorrência de GC. Um indício que pode explicar esse fato é que cada *release* de *Health Watcher* possui um número elevado de classes, ocasionando uma maior separação de responsabilidades em cada uma delas.

5.2.2 QP2:Qual o padrão que pode ser observado nos code smells a partir das relações Inter-smell de LPSs orientada a objetos?

Nesta seção 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*.

Tabela 10 – Relações *Inter-smell* em MobileMedia

Release	Plain Support	Rejection	Inclusion
R1	$(GC \rightarrow LM)$, $(LM \rightarrow DC)$,$(FE \rightarrow GC)$, $(GC \rightarrow DC)$.	$(TC \not\Rightarrow LM)$, $(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(TC \not\Rightarrow DC)$, $(LM \not\Rightarrow FE)$, $(DC \not\Rightarrow FE)$.	0
R2	$(FE \rightarrow LM)$, $(GC \rightarrow LM)$, $(LM \rightarrow DC)$, $(FE \rightarrow GC)$, $(FE \rightarrow DC)$, $(GC \rightarrow DC)$.	$(TC \not\Rightarrow LM)$, $(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(TC \not\Rightarrow DC)$.	$(GC \Rightarrow DC)$
R3	$(FE \rightarrow LM)$, $(GC \rightarrow LM)$, $(LM \rightarrow DC)$, $(FE \rightarrow GC)$, $(GC \rightarrow DC)$.	$(TC \not\Rightarrow LM)$, $(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(TC \not\Rightarrow DC)$, $(DC \not\Rightarrow FE)$	0
R4	$(FE \rightarrow LM)$, $(GC \rightarrow LM)$, $(LM \rightarrow DC)$, $(FE \rightarrow GC)$, $(GC \rightarrow DC)$.	$(TC \not\Rightarrow LM)$, $(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(TC \not\Rightarrow DC)$, $(DC \not\Rightarrow FE)$.	0
R5	$(FE \rightarrow LM)$, $(GC \rightarrow LM)$, $(LM \rightarrow DC)$, $(FE \rightarrow GC)$, $(GC \rightarrow DC)$.	$(TC \not\Rightarrow LM)$, $(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(TC \not\Rightarrow DC)$, $(DC \not\Rightarrow FE)$.	$(GM \Rightarrow LM)$
R6	$(GC \rightarrow LM)$, $(LM \rightarrow DC)$, $(GC \rightarrow DC)$.	$(TC \not\Rightarrow LM)$, $(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(TC \not\Rightarrow DC)$, $(DC \not\Rightarrow FE)$, $(LM \not\Rightarrow FE)$, $(FE \not\Rightarrow GC)$	0
R7	$(LM \rightarrow DC)$.	$(TC \not\Rightarrow LM)$, $(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(TC \not\Rightarrow DC)$, $(DC \not\Rightarrow FE)$, $(LM \not\Rightarrow FE)$, $(FE \not\Rightarrow GC)$, $(LM \not\Rightarrow GC)$, $(GC \not\Rightarrow DC)$.	0
R8	$(GC \rightarrow LM)$, $(LM \rightarrow DC)$, $(GC \rightarrow DC)$.	$(TC \not\Rightarrow LM)$, $(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(TC \not\Rightarrow DC)$, $(DC \not\Rightarrow FE)$, $(LM \not\Rightarrow FE)$, $(FE \not\Rightarrow GC)$.	$(GC \Rightarrow DC)$.

Fonte – Elaborado pelo Autor

A Tabela 10 mostra as relações *Inter-smell* encontradas na LPS MobileMedia após o processo manual de identificação dessas anomalias. E com isso, foram feitas algumas observações em cada relação *Inter-smell*:

- ***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* ($LM \rightarrow DC$). Isso chamou atenção, pois como pode ser visto pela Tabela 8, foi nessa *release* que todos os *code smells* tiveram pelo menos uma ocorrência.

- ***Mutual Support***: Nenhuma ocorrência de *Inter-smell* foi detectada para essa relação e por isso essa relação não foi colocada na Tabela 10.
- ***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. A *release* 2 teve o menor número de ocorrências para a relação *Rejection*, ou seja, nessa *release* os *code smells* estão mais acoplados formando várias ocorrências de *Inter-smell*. Por outro lado, as *releases* 1, 6, 7, 8 foram as que tiveram o maior número de ocorrências da relação *Rejection*, ou seja, o número de ocorrências de *Inter-smell* ou de co-ocorrências de *code smells* foram menores para essas *releases*. A *release* 7 de maneira mais visível, obteve nove ocorrências da relação *Rejection* tendo apenas LM e DC como os únicos *code smells* que se relacionaram, assim, é de se esperar que nessa *release* exista um menor acoplamento entre *code smells*. Outro resultado encontrado foi que nas *releases* 6,7 e 8 foi encontrada a ocorrência: $FE \not\Rightarrow GC$, isso pode ser explicado observando a Tabela 8, o número de ocorrências de GC e FE diminuiram em relação as outras *releases*.
- ***Inclusion***: Essa relação é mais forte que 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.

Tabela 11 – Relações *Inter-smell* em Health Watcher

Release	Plain Support	Rejection	Inclusion
R1	$(FE \rightarrow LM)$, $(LM \rightarrow DC)$, $(FE \rightarrow DC)$	$(TC \not\Rightarrow LM)$, $(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(TC \not\Rightarrow DC)$, $(LM \not\Rightarrow GC)$, $(FE \not\Rightarrow GC)$, $(GC \not\Rightarrow DC)$	$(FE \Rightarrow LM)$, $(FE \Rightarrow DC)$
R2	$(FE \rightarrow LM)$, $(LM \rightarrow DC)$, $(FE \rightarrow DC)$.	$(TC \not\Rightarrow LM)$, $(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(TC \not\Rightarrow DC)$, $(LM \not\Rightarrow GC)$, $(FE \not\Rightarrow GC)$, $(GC \not\Rightarrow DC)$	$(FE \Rightarrow LM)$, $(FE \Rightarrow DC)$
R3	$(TC \rightarrow LM)$, $(TC \rightarrow DC)$, $(LM \rightarrow DC)$.	$(TC \not\Rightarrow FE)$, $(TC \not\Rightarrow GC)$, $(LM \not\Rightarrow FE)$, $(LM \not\Rightarrow GC)$, $(FE \not\Rightarrow GC)$, $(DC \not\Rightarrow FE)$, $(GC \not\Rightarrow DC)$	0

Fonte – Elaborado pelo Autor

A Tabela 11 mostra as relações encontradas na LPS MobileMedia após o processo manual de identificação dessas anomalias *Inter-smell* na LPS *Health Watcher* . Dessa forma, foram feitas algumas observações em cada relação *Inter-smell*:

- ***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. Isso chamou atenção, pois como pode ser visto pela Tabela 9, a *release* 3 tem apenas uma ocorrências a mais do *code smell* TC em relação as *releases* 1 e 2.
- ***Mutual Support***: Nenhuma ocorrência de *Inter-smell* foi detectada para essa relação e por isso essa relação não foi colocada na Tabela 11.
- ***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 9.
  Todas as *releases* tiveram o mesmo número de ocorrências para a relação *Rejection*, ou seja, em todas essas *releases* os *code smells* estão menos acoplados entre si, formando menos ocorrências de *Inter-smell*. O que mais chamou atenção, foi a rejeição do *code smell* GC para os outros *code smells*, que como dito anteriormente, a ausência de *code smell* pode ser explicada pela boa separação de responsabilidades entre as classes da LPS H*ealth Watche*r.
- ***Inclusion***: Essa relação é mais forte que a *Plain Support* e ela foi detectada nas *releases* 1 e 2. Em ambas as *releases* a 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.

5.2.3 QP3: Qual o impacto das relações Inter-smell para a qualidade e manutenibilidade do código de LPSs orientada a objetos?

Para avaliar a manutenibilidade das LPS com e sem os *Inter-smell*, ou seja, antes e depois do processo de refatoração, foram utilizadas métricas definidas por (CHIDAMBER;

KEMERER, 1994). O conjunto de métricas utilizadas neste trabalho foram definidas na Seção 2.3.

Em Singh e Malhotra (2012), os autores afirmam que as métricas que foram definidas na seção 2.3 são negativamente correlacionadas com a manutenibilidade de software, ou seja, quanto menor o valor das métricas, maior a manutenibilidade do software.

Métricas de software são utilizadas para medir diferentes aspectos de software, como herança, tamanho, acoplamento, coesão, entre outros. Contudo, manutenibilidade é a facilidade na qual software pode ser modificado, corrigido e atualizado. A relação entre métricas e manutenibilidade de software é sempre o inverso, enquanto o valor da métrica aumenta, o índice de manutenibilidade diminui e vice-versa (DUBEY; RANA, 2011).

Para avaliar a manutenibilidade das LPSs a partir dos valores das métricas, foi utilizada a mesma abordagem de Tarwani e Chug (2016), na qual os autores utilizam o somatório de todas as métricas para medir e comparar a manutenibilidade.

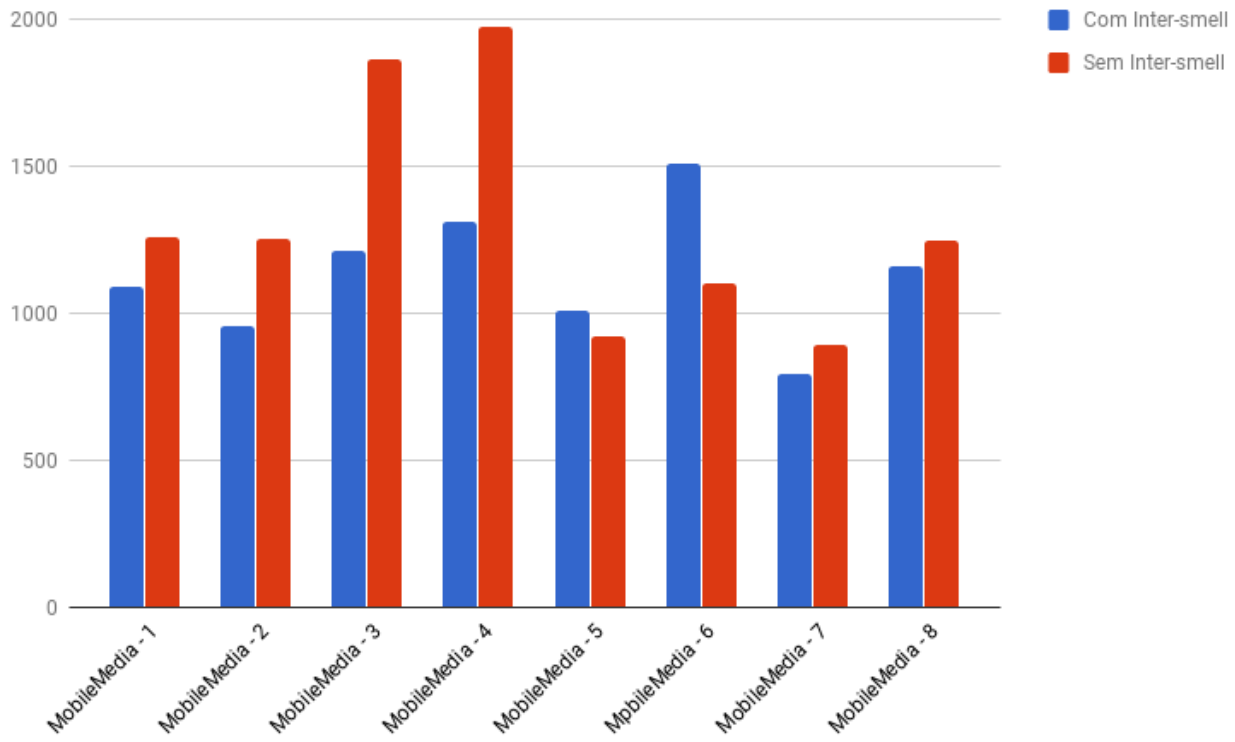
Como foi mencionado na seção 5.1.3, o processo de refatoração, medição e comparação foram realizados apenas nas classes que possuiam co-ocorrências de *code smells*.

A Figura 10 mostra os valores de manutenibilidade que foram medidos com a presença de *Inter-smell* e sem a presença de *Inter-smell*, a Tabela 12 apresenta em detalhes sobre como a manutenibilidade aumentou ou diminuiu nas *releases* da LPS MobileMedia. 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.

Como pode ser observado na Figura 10, 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.

Figura 10 – Manutenibilidade com *Inter-smell* e sem *Inter-smell* da LPS MobileMedia



Fonte – Elaborado pelo Autor

Tabela 12 – Manutenibilidade da LPS MobileMedia com e sem *Inter-smell*

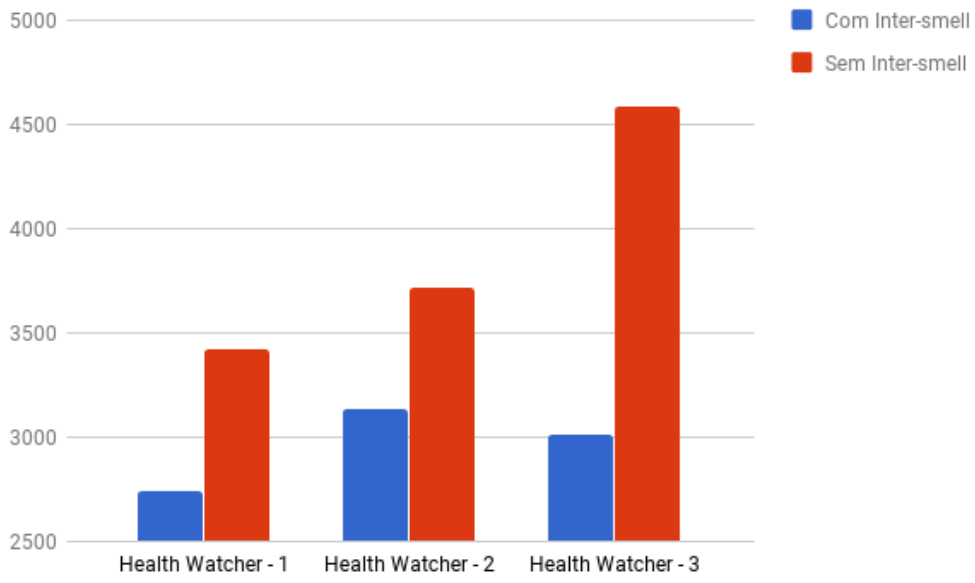
LPS - Release	Com Inter-smell	Sem Inter-smell
MobileMedia - 1	1092	1262
MobileMedia - 2	955	1254
MobileMedia - 3	1213	1867
MobileMedia - 4	1311	1978
MobileMedia - 5	1008	924
MobileMedia - 6	1509	1101
MobileMedia - 7	793	891
MobileMedia - 8	1163	1250

A Figura 11 mostra os valores de manutenibilidade que foram medidos com a presença de *Inter-smell* e sem a presença de *Inter-smell*, a Tabela 13 apresenta em detalhes sobre como a manutenibilidade aumentou ou diminuiu nas *releases* da LPS *Health Watcher*. Nesse caso, o valor da manutenibilidade que é o somatório das métricas, está entre 2500 e 5000, na qual 2500 descreve uma *release* que é mais fácil de manter e 5000 indica uma *release* com a mais alta dificuldade para realizar tarefas de manutenção. Lembrando que quanto mais o somatório de cada *release* se aproximar de 0, maior será a capacidade de manutenção.

Como pode ser observado na Figura 11, em todas as *releases* a remoção de *Inter-smell* trouxe um impacto negativo para a manutenibilidade. Principalmente para a *release* 3 a remoção dos *Inter-smell* trouxe um impacto negativo na manutenibilidade muito maior comparado com os resultados encontrados para as outras *releases*.

Em nenhuma *release* da LPS *Health Watcher* foi encontrado que a remoção de *Inter-smell* trouxe impacto positivo para a manutenibilidade.

Figura 11 – Manutenibilidade com *Inter-smell* e sem *Inter-smell* da LPS *Health Watcher*



Fonte – Elaborado pelo Autor

Tabela 13 – Manutenibilidade da LPS *Health Watcher* com e sem *Inter-smell*

LPS - Release	Com Inter-smell	Sem Inter-smell
Health Watcher - 1	2741	3421
Health Watcher - 2	3130	3715
Health Watcher - 3	3009	4579

A Tabela 14 mostra o somatório de cada métrica da *release* 1 da LPS MobileMedia com os *Inter-smell* e sem *Inter-smell*. Como pode ser observado, as métricas WMC, CBO, RFC e LCOM na *release* refatorada tiveram um aumento em relação a *release* original, tornando a *release* refatorada com métodos mais complexos, acoplados e com menos coesão, principalmente com menos coesão, pois o valor da métrica LCOM teve um aumento mais elevado.

Tabela 14 – Somatório por métrica para a *release* 1 da LPS MobileMedia

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
MobileMediaOriginal - 1	142	8	0	54	524	363	1092
MobileMediaRefatorada - 1	156	8	0	60	568	470	1262

A Tabela 15 mostra o somatório de cada métrica da *release* 2 de MobileMedia com os *Inter-smell* e sem *Inter-smell*. Como pode ser observado, também houve um aumento das métricas WMC, CBO, RFC e LCOM da *release* refatorada para a *release* original, tornando a *release* refatorada com métodos mais complexos, acoplados e com menos coesão. O que chamou atenção foi que na *release* refatorada a coesão piorou quase duas vezes mais em relação a *release* original.

Tabela 15 – Somatório por métrica para a *release* 2 da LPS MobileMedia

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
MobileMediaOriginal - 2	129	8	0	88	493	237	955
MobileMediaRefatorada - 2	150	8	0	94	496	506	1254

A Tabela 16 mostra o somatório de cada métrica da *release* 3 da LPS MobileMedia com os *Inter-smell* e sem *Inter-smell*. Como pode ser observado, também houve um aumento das métricas WMC, CBO, RFC e LCOM da *release* refatorada para a *release* original, tornando a *release* refatorada com métodos mais complexos, acoplados e com menos coesão. A complexidade dos métodos aumentou de maneira significativa com a métrica WMC e a coesão piorou muito, a métrica LCOM foi a principal responsável pela diferença de manutenibilidade entre as *releases*.

Tabela 16 – Somatório por métrica para a *release* 3 d LPS MobileMedia

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
MobileMediaOriginal - 3	143	8	0	104	537	421	1213
MobileMediaRefatorada - 3	176	8	0	108	585	990	1867

A Tabela 17 mostra o somatório de cada métrica da *release* 4 da LPS MobileMedia com os *Inter-smell* e sem *Inter-smell*. Como pode ser observado, novamente houve um aumento dos valores das métricas WMC, CBO, RFC e LCOM da *release* refatorada para a *release* original, tornando a *release* refatorada com métodos mais complexos, acoplados e com menos coesão. Como nos casos anteriores, as métricas WMC, RFC e principalmente LCOM tiveram um aumento bem significativo na *release* refatorada em comparação com a *release* original.

Tabela 17 – Somatório por métrica para a *release* 4 da LPS MobileMedia

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
MobileMediaOriginal - 4	147	8	0	109	619	427	1311
MobileMediaRefatorada - 4	182	8	0	113	669	1005	1978

A Tabela 18 mostra o somatório de cada métrica da *release* 5 da LPS MobileMedia com os *Inter-smell* e sem *Inter-smell*. Como pode ser observado, diferentemente das *releases* anteriores, houve um aumento das métricas WMC, CBO, RFC, porém, o valor da métrica LCOM na *release* refatorada diminuiu em relação a *release* original tornando-a mais coesa e fazendo com que a *release* refatorada tivesse um maior índice de manutenibilidade. O que chama atenção é que a métrica LCOM possui um peso significativo para a manutenibilidade.

Tabela 18 – Somatório por métrica para a *release* 5 da LPS MobileMedia

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
MobileMediaOriginal - 5	107	7	0	95	490	309	1008
MobileMediaRefatorada - 5	113	7	0	99	496	209	924

A Tabela 19 mostra o somatório de cada métrica da *release* 6 da LPS MobileMedia com os *Inter-smell* e sem *Inter-smell*. Como pode ser observado, a métrica WMC e CBO na *release* refatorada obteve um aumento em relação a *release* original. Porém, as métricas RFC e LCOM tiveram uma diminuição mais significativa impactando diretamente na manutenibilidade e tornando a *release* refatorada bem mais manutenível do que a *release* original. Mais uma vez, apesar da diminuição do valor RFC, a métrica LCOM foi determinante para esse resultado.

Tabela 19 – Somatório por métrica para a *release* 6 da LPS MobileMedia

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
MobileMediaOriginal - 6	130	8	0	120	661	590	1509
MobileMediaRefatorada - 6	144	8	0	126	561	262	1101

A Tabela 20 mostra o somatório de cada métrica da *release* 7 da LPS MobileMedia com os *Inter-smell* e sem *Inter-smell*. Como pode ser observado, novamente houve um aumento das métricas WMC, CBO, RFC e LCOM da *release* refatorada para a *release* original, tornando a *release* refatorada com métodos mais complexos, acoplados e com menos coesão. Porém, não houve um aumento muito significativo das métricas, assim, o índice de manutenibilidade não ficou tão discrepante, mesmo assim, a *release* original conseguiu ser mais manutenível do que a refatorada.

Tabela 20 – Somatório por métrica para a *release* 7 da LPS MobileMedia

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
MobileMediaOriginal - 7	66	3	0	113	418	193	793
MobileMediaRefatorada - 7	86	3	0	113	438	251	891

A Tabela 21 mostra o somatório de cada métrica da *release* 8 da LPS MobileMedia com os *Inter-smell* e sem *Inter-smell*. Como pode ser observado, também houve um aumento das métricas WMC, CBO, RFC e LCOM da *release* refatorada para a *release* original, tornando a *release* refatorada com métodos mais complexos, acoplados e com menos coesão. E como pode ser percebido, mais uma vez a métrica LCOM foi determinante e definiu qual *release* foi a que teve maior índice de manutenibilidade, no caso, a *release* refatorada.

Tabela 21 – Somatório por métrica para a *release* 8 da LPS MobileMedia

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
MobileMediaOriginal - 8	103	2	0	162	652	244	1163
MobileMediaRefatorada - 8	115	2	0	162	664	307	1250

A Tabela 22 mostra o somatório de cada métrica da *release* 1 de Heath Watcher com os *Inter-smell* e sem *Inter-smell*. A métricas WMC, CBO e LCOM sofreram um aumento na *release* refatorada, enquanto que a métrica RFC sofreu uma diminuição, porém, como pode ser observado, a *release* refatorada possui três vezes menos coesão do que a *release* original o que foi determinante para o resultado final no índice de manutenibilidade.

Tabela 22 – Somatório por métrica para a *release* 1 de Health Watcher

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
Health Watcher Original - 1	313	20	0	418	1677	313	2741
Heath Watcher Refatorada - 1	388	20	0	419	1664	930	3421

A Tabela 23 mostra o somatório de cada métrica da *release* 2 da *Heath Watcher* com os *Inter-smell* e sem *Inter-smell*. Também foi observado uma relativa diminuição dos valores da métrica RFC na *release* refatorada, porém, as métricas WMC, CBO e principalmente LCOM aumentaram deixando a *release* original com maior índice de manutenibilidade em relação a *release* refatorada.

Tabela 23 – Somatório por métrica para a *release* 2 de Health Watcher

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
Health Watcher Original - 2	328	19	0	501	2028	254	3130
Heath Watcher Refatorada - 2	415	19	0	516	2019	746	3715

A Tabela 24 mostra o somatório de cada métrica da *release* 3 de Heath Watcher com os *Inter-smell* e sem *Inter-smell*. Nessa caso, as métricas WMC, CBO, RFC e LCOM tiveram um aumento na *release* refatorada em relação a *release* original. Pricipalmente as métricas WMC, RFC e LCOM. LCOM por exemplo teve um aumento cinco vezes maior na *releases* refatorada, o que provocou de maneira mais discrepante a diferença de manutenibilidade entre as duas *releases*.

Tabela 24 – Somatório por métrica para a *release* 3 de Health Watcher

LPS - Release	WMC	DIT	NOC	CBO	RFC	LCOM	TOTAL
Health Watcher Original - 3	333	20	0	481	1924	251	3009
Heath Watcher Refatorada - 3	472	20	0	505	2185	1397	4579

Este trabalho analisou oito *releases* da LPS MobileMedia e três *releases* de Health Watcher. Em cada uma das *releases* foram detectadas co-ocorrências 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. Neste trabalho foi encontrado em seis *releases* de MobileMedia (1, 2, 3, 4, 7 e 8) e nas três *releases* de Health Watcher que a presença de ocorrências de *Inter-smell* não provocou impacto negativo para a manutenibilidade e qualidade do código.

5.3 Discussão

Com a análise feita por esse trabalho, foi possível constatar indícios de que que as co-ocorrências de *code smells* conhecidas como *Inter-smell* não tiveram impacto negativo para a manutenibilidade ou a qualidade do código em *releases* de MobileMedia e Health Watcher, duas LPSs orientada a objetos implementadas com Java.

Como pode ser obeservado nas Tabelas 8 e 9, à medida em que o número de ocorrências do *code smell Long Method* aumentava, o número de ocorrências de *Duplicated Code* também aumentava, esse fenômeno foi observado em ambas as LPSs. O que chama atenção é que na Tabela 8, as *releases* 6, 7 e 8 de MobileMedia possuem menos ocorrências dos *code smells God Class* e *Feature Envy* e isso fica mais evidente ainda quando a Tabela 10 é analisada, o número de ocorrências de *Inter-smell* na relação *Rejection* aumenta, diminuindo as ocorrências na relação *Plain Support*. O mesmo pode ser observado na *release* 3 de *Health Watcher* na Tabela

9 e na Tabela 11, o número de ocorrências de *Feature Envy* diminui em relação as *releases 1 e 2*, mostrando o resultado na relação *inclusion* na qual a *release* 3 não possui nenhuma ocorrência. Nesse contexto, as relações *Inter-smell* ajudam a identificar certos padrões relacionados a *code smells*, porém, nesse trabalho foi não detectado o impacto que o número de ocorrências de relações *Inter-smell* trazem para a manutenibilidade das *releases*.

Para determinar a manutenibilidade das LPS com e sem os *Inter-smell*, foram utilizadas métricas definidas por (CHIDAMBER; KEMERER, 1994). O conjunto de métricas utilizadas neste trabalho foram definidas na Seção 2.3. As Figuras 10 e 11 mostram os índices de manutenibilidade das *releases* com a presença de *Inter-smell* e sem *Inter-smell* para cada LPS. Na Figura 10 pode ser observado que das oitos *releases* de MobileMedia, seis mostraram que a presença de ocorrências de *Inter-smell* não causaram impacto negativo para a manutenibilidade. O que chama atenção é que a diferença de manutenibilidade entre as *releases* com *Inter-smell* e sem *Inter-smell* não foi tão grande na maioria das *releases* de MobileMedia, apenas as *releases* 3,4 e 6 apresentaram valores mais discrepantes. 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* quando para a *release* 6 que mostrou impacto negativo da presença de *Inter-smell*. Como pode ser visualizado na Figura 11, todas as três *releases* com *Inter-smell* tiveram um índice de manutenilidade maior do que aquelas que não tinha ocorrências de *Inter-smell*, e diferentemente de maioria das *releases* de MobileMedia, os resultados na comparação das *releases* da *Health Watcher* foram bem mais discrepantes. Mais uma vez, como pode ser observado da Tabela 14 até a Tabela 24, a métrica LCOM fez grande diferença nos resultados, indicando que essa pode ser uma importante métrica para a manutenibilidade de LPSs orientada a objeto.

Fernandes et al. (2017) foram os primeiros a trabalharem com co-ocorrências de *code smells* no contexto de Linha de Produto de Software (LPS), no trabalho os autores chamam essas co-ocorrências de aglomerações e verificam o impacto dessas anomalias para o problema de manutenção conhecido como instabilidade. Os resultados encontrados sugerem que *code smells* geralmente estão interconectados com outros *code smells* em elementos críticos de LPS, tal como, classes e que aglomerações são mais efetivas do que ocorrências individuais de *code smells* para identificar instabilidade ou estruturas que geralmente prejudicam a manutenibilidade da LPS. Porém, no presente trabalho foi encontrado que as co-ocorrências de *code smells* conhecidas como *Inter-smell* não tiveram impacto negativo para a manutenibilidade ou a qualidade do código

em *releases* da MobileMedia e da *Health Watcher*. Um ponto a destacar é que em Fernandes et al. (2017) os autores trabalharam com LPSs implementadas utilizando o paradigma orientado a *feature*, enquanto que neste trabalho foram utilizadas LPSs orientada a objetos.

5.4 Ameaças à validade

Esta seção discute as ameaças que podem ter afetado os resultados deste trabalho. A validade de um estudo denota a confiabilidade dos resultados e até que ponto os resultados são verdadeiros e não tendenciosos pelo ponto de vista subjetivo dos pesquisadores (WOHLIN et al., 2012).

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* tornando os resultados encontrados mais sólidos. A detecção de relações *Inter-smell* foi realizada de forma manual, porém, as métricas utlizadas foram validadas anteriormente em trabalho na literatura. 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 por ferramentas, diminuindo a chance de erros na detecção. Mesmo assim, as estratégias implementados por essas ferramentas podem ser um potencial fator de ameaça à validade, pois 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 consequetemente 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 em outras linguagens.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho realizou um estudo de caso para avaliar o impacto de co-ocorrências de *code smells* conhecidas como relações *Inter-smell* para a manutenibilidade de duas LPSs orientada a objetos implementadas utilizando a linguagem Java. Foram identificadas duas LPSs orientada a objetos chamadas 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.

Yamashita e Moonen (2013a) afirmam em seu trabalho que *code smells* são apenas indicadores prejudiciais de problemas no código. Os autores citam que os trabalhos na academia devem estudar mais as relações entre os *code smells* e seus problemas para a manutenibilidade do código.

Alguns trabalhos na literatura estudam o impacto de *code smells* para algum aspecto de qualidade em sistemas tradicionais (YAMASHITA; MOONEN, 2013a), (YAMASHITA et al., 2015), (MACIA et al., 2012). E nesse contexto, existem algumas controvérsias sobre o impacto de *code smells* para o código. Alguns trabalhos apontam que *code smells* tem baixo impacto ou não impactam projetos de software: (YAMASHITA; COUNSELL, 2013), (HALL et al., 2014), (YAMASHITA; MOONEN, 2013b). Enquanto que outros trabalhos apotam que *code smells* tem um impacto significativo para projetos de software: (OUNI et al., 2015), (KHOMH et al., 2012), (ABBES et al., 2011). Da mesma forma que esses trabalhos tiveram resultados diferentes, no estudo apresentado foi encontrado que essas relações entre *code smells* não trouxeram impacto negativo para a manutenibilidade das *releases* de MobileMedia e Health Watcher. Diferente do que foi encontrado em Fernandes et al. (2017), na qual os autores identificaram que aglomerações de *code smells* trazem impacto negativo para à manutenibilidade. Porém, 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. Além disso, as descobertas apontam que os engenheiros de aplicação devem se preocupar mais com a coesão da classes na implementação da aplicação, preservando assim a manutenibilidade dos produtos e beneficiando de forma direta a reusabilidade, que é um dos maiores pilares na Engenharia de Linha Produto de Software.

Trabalhos futuros identificados a partir das descobertas deste trabalho, sugerem:

i) analisar mais LPSs implementadas em diferentes paradigmas; ii) reproduzir o estudo com ferramentas que detectam outros *code smells*; iii) construir uma ferramenta para detectar relações *Inter-smell*; e, iv) replicar o estudo utilizando mais métricas para avaliar a manutenibilidade jutamente com desenvolvedores especialistas no domínio.

REFERÊNCIAS

ABBES, M.; KHOMH, F.; GUEHENEUC, Y.-G.; ANTONIOL, G. An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. In: IEEE. **Software maintenance and reengineering (CSMR), 2011 15th European conference on**. [S.l.], 2011. p. 181–190.

ABDELMOEZ, W.; KHATER, H.; EL-SHOAFY, N. Comparing maintainability evolution of object-oriented and aspect-oriented software product lines. In: IEEE. **Informatics and Systems (INFOS), 2012 8th International Conference on**. [S.l.], 2012. p. SE–53.

APEL, S.; BATORY, D.; KÄSTNER, C.; SAAKE, G. **Feature-Oriented Software Product Lines**. [S.l.]: Springer, 2013.

ARAAR, I. E.; SERIDI, H. Software features extraction from object-oriented source code using an overlapping clustering approach. **Informatica**, Slovenian Society Informatika/Slovensko drustvo Informatika, v. 40, n. 2, p. 245, 2016.

BENAVIDES, D.; SEGURA, S.; RUIZ-CORTÉS, A. Automated analysis of feature models 20 years later: A literature review. **Information Systems**, Elsevier, v. 35, n. 6, p. 615–636, 2010.

CHAHAR, C.; CHAUHAN, V. S.; DAS, M. L. Code analysis for software and system security using open source tools. **Information Security Journal: A Global Perspective**, Taylor & Francis, v. 21, n. 6, p. 346–352, 2012.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. **IEEE Transactions on software engineering**, IEEE, v. 20, n. 6, p. 476–493, 1994.

DEVI U., A. S.; KESSWANI, N. A review on quality models to analyse the impact of refactored code on maintainability with reference to software product line. 2016.

DUBEY, S. K.; RANA, A. Assessment of maintainability metrics for object-oriented software system. **ACM SIGSOFT Software Engineering Notes**, ACM, v. 36, n. 5, p. 1–7, 2011.

DYER, R.; RAJAN, H.; CAI, Y. An exploratory study of the design impact of language features for aspect-oriented interfaces. In: ACM. **Proceedings of the 11th annual international conference on Aspect-oriented Software Development**. [S.l.], 2012. p. 143–154.

DYER, R.; RAJAN, H.; CAI, Y. An exploratory study of the design impact of language features for aspect-oriented interfaces. In: ACM. **Proceedings of the 11th annual international conference on Aspect-oriented Software Development**. [S.l.], 2012. p. 143–154.

ELISH, M. O.; ELISH, K. O. Application of treenet in predicting object-oriented software maintainability: A comparative study. In: IEEE. **Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on**. [S.l.], 2009. p. 69–78.

FENTON, N.; BIEMAN, J. **Software metrics: a rigorous and practical approach**. [S.l.]: CRC Press, 2014.

FERNANDES, E.; OLIVEIRA, J.; VALE, G.; PAIVA, T.; FIGUEIREDO, E. A review-based comparative study of bad smell detection tools. In: ACM. **Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering**. [S.l.], 2016. p. 18.

FERNANDES, E.; VALE, G.; SOUSA, L.; FIGUEIREDO, E.; GARCIA, A.; LEE, J. No code anomaly is an island. In: SPRINGER. **International Conference on Software Reuse**. [S.l.], 2017. p. 48–64.

FIGUEIREDO, E.; CACHO, N.; SANT'ANNA, C.; MONTEIRO, M.; KULESZA, U.; GARCIA, A.; SOARES, S.; FERRARI, F.; KHAN, S.; FILHO, F. C. et al. Evolving software product lines with aspects: an empirical study on design stability. In: ACM. **Proceedings of the 30th international conference on Software engineering**. [S.l.], 2008. p. 261–270.

FONTANA, F. A.; ZANONI, M.; RANCHETTI, A.; RANCHETTI, D. Software clone detection and refactoring. **ISRN Software Engineering**, Hindawi Publishing Corporation, v. 2013, 2013.

FOWLER, M. **Refactoring:** Improving the Design of Existing Code. [S.l.]: Addison-Wesley, 1999.

GAIA, F. N.; FERREIRA, G. C. S.; FIGUEIREDO, E.; MAIA, M. de A. A quantitative and qualitative assessment of aspectual feature modules for evolving software product lines. **Science of Computer Programming**, Elsevier, v. 96, p. 230–253, 2014.

GREENWOOD, P.; BARTOLOMEI, T.; FIGUEIREDO, E.; DOSEA, M.; GARCIA, A.; CACHO, N.; SANT'ANNA, C.; SOARES, S.; BORBA, P.; KULESZA, U. et al. On the impact of aspectual decompositions on design stability: An empirical study. In: SPRINGER. **European Conference on Object-Oriented Programming**. [S.l.], 2007. p. 176–200.

HALL, T.; ZHANG, M.; BOWES, D.; SUN, Y. Some code smells have a significant but small effect on faults. **ACM Transactions on Software Engineering and Methodology (TOSEM)**, ACM, v. 23, n. 4, p. 33, 2014.

ISO, I. Iec 25000 software and system engineering–software product quality requirements and evaluation (square)–guide to square. **International Organization for Standarization**, 2005.

KAUR, S.; SINGH, S. Spotting & eliminating type checking code smells using eclipse plug-in: Jdeodorant. **International Journal of Computer Science and CommunicationEngineering**, v. 5, n. 1, 2016.

KHOMH, F.; PENTA, M. D.; GUEHENEUC, Y.-G. An exploratory study of the impact of code smells on software change-proneness. In: IEEE. **Reverse Engineering, 2009. WCRE'09. 16th Working Conference on**. [S.l.], 2009. p. 75–84.

KHOMH, F.; PENTA, M. D.; GUÉHÉNEUC, Y.-G.; ANTONIOL, G. An exploratory study of the impact of antipatterns on class change-and fault-proneness. **Empirical Software Engineering**, Springer, v. 17, n. 3, p. 243–275, 2012.

KIM, C.; BODDEN, E.; BATORY, D.; KHURSHID, S. Reducing configurations to monitor in a software product line. In: SPRINGER. **Runtime Verification**. [S.l.], 2010. p. 285–299.

KULESZA, U.; SANT'ANNA, C.; GARCIA, A.; COELHO, R.; STAA, A. V.; LUCENA, C. Quantifying the effects of aspect-oriented programming: A maintenance study. In: IEEE. **Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on**. [S.l.], 2006. p. 223–233.

LEAVENS, G. T.; CHIBA, S.; HAUPT, M.; OSTERMANN, K.; WOHLSTADTER, E. **Transactions on Aspect-Oriented Software Development IX**. [S.l.]: Springer, 2012. v. 7271.

LI, P.; CUI, B. A comparative study on software vulnerability static analysis techniques and tools. In: IEEE. **Information Theory and Information Security (ICITIS), 2010 IEEE International Conference on**. [S.l.], 2010. p. 521–524.

LINDEN, F. J. Van der; SCHMID, K.; ROMMES, E. **Software product lines in action:** the best industrial practice in product line engineering. [S.l.]: Springer Science & Business Media, 2007.

LOZANO, A.; MENS, K.; PORTUGAL, J. Analyzing code evolution to uncover relations. In: IEEE. **Patterns Promotion and Anti-patterns Prevention (PPAP), 2015 IEEE 2nd Workshop on**. [S.l.], 2015. p. 1–4.

MACIA, I.; GARCIA, J.; POPESCU, D.; GARCIA, A.; MEDVIDOVIC, N.; STAA, A. von. Are automatically-detected code anomalies relevant to architectural modularity?: an exploratory analysis of evolving systems. In: ACM. **Proceedings of the 11th annual international conference on Aspect-oriented Software Development**. [S.l.], 2012. p. 167–178.

MALHOTRA, R.; CHUG, A. An empirical study to assess the effects of refactoring on software maintainability. In: IEEE. **Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on**. [S.l.], 2016. p. 110–117.

MAZINANIAN, D.; TSANTALIS, N.; STEIN, R.; VALENTA, Z. Jdeodorant: clone refactoring. In: ACM. **Proceedings of the 38th International Conference on Software Engineering Companion**. [S.l.], 2016. p. 613–616.

MERRIAM, S. B.; TISDELL, E. J. **Qualitative research:** A guide to design and implementation. [S.l.]: John Wiley & Sons, 2015.

NUÑEZ-VARELA, A. S.; PÉREZ-GONZALEZ, H. G.; MARTÍNEZ-PEREZ, F. E.; SOUBERVIELLE-MONTALVO, C. Source code metrics: A systematic mapping study. **Journal of Systems and Software**, Elsevier, v. 128, p. 164–197, 2017.

OLBRICH, S.; CRUZES, D. S.; BASILI, V.; ZAZWORKA, N. The evolution and impact of code smells: A case study of two open source systems. In: IEEE COMPUTER SOCIETY. **Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement**. [S.l.], 2009. p. 390–400.

OLBRICH, S. M.; CRUZES, D. S.; SJØBERG, D. I. Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems. In: IEEE. **Software Maintenance (ICSM), 2010 IEEE International Conference on**. [S.l.], 2010. p. 1–10.

OUNI, A.; KESSENTINI, M.; BECHIKH, S.; SAHRAOUI, H. Prioritizing code-smells correction tasks using chemical reaction optimization. **Software Quality Journal**, Springer, v. 23, n. 2, p. 323–361, 2015.

PAIVA[1], T.; DAMASCENO[1], A.; PADILHA[1], J.; FIGUEIREDO[1], E.; SANT'ANNA, C. Experimental evaluation of code smell detection tools. In: ACM. **Workshop on Software Visualization, Evolution and Maintenance**. [S.l.], 2015. p. 15.

PIETRZAK, B.; WALTER, B. Leveraging code smell detection with inter-smell relations. **Extreme Programming and Agile Processes in Software Engineering**, Springer, p. 75–84, 2006.

POHL, K.; BÖCKLE, G.; LINDEN, F. J. van D. **Software product line engineering:** foundations, principles and techniques. [S.l.]: Springer Science & Business Media, 2005.

RIEBISCH, M. Towards a more precise definition of feature models. **Modelling variability for object-oriented product lines**, BookOnDemand Publ. Co, p. 64–76, 2003.

SCHULZE, S.; APEL, S.; KÄSTNER, C. Code clones in feature-oriented software product lines. In: ACM. **ACM SIGPLAN Notices**. [S.l.], 2010. v. 46, n. 2, p. 103–112.

SHEN, B.; RUAN, T. A case study of software process improvement in a chinese small company. In: IEEE. **Computer science and software engineering, 2008 international conference on**. [S.l.], 2008. v. 2, p. 609–612.

SINGH, Y.; MALHOTRA, R. **Object-oriented software engineering**. [S.l.]: PHI Learning Pvt. Ltd., 2012.

SJØBERG, D. I.; YAMASHITA, A.; ANDA, B. C.; MOCKUS, A.; DYBÅ, T. Quantifying the effect of code smells on maintenance effort. **IEEE Transactions on Software Engineering**, IEEE, v. 39, n. 8, p. 1144–1156, 2013.

SOARES, S.; LAUREANO, E.; BORBA, P. Implementing distribution and persistence aspects with aspectj. In: ACM. **ACM Sigplan Notices**. [S.l.], 2002. v. 37, n. 11, p. 174–190.

SOMMERVILLE, I. **SOFTWARE ENGINEERING**. [S.l.]: Addison-Wesley, 2011.

SPINELLIS, D. Tool writing: a forgotten art?(software tools). **IEEE Software**, IEEE, v. 22, n. 4, p. 9–11, 2005.

TARWANI, S.; CHUG, A. Sequencing of refactoring techniques by greedy algorithm for maximizing maintainability. In: IEEE. **Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on**. [S.l.], 2016. p. 1397–1403.

VALE, G.; FIGUEIREDO, E.; ABÍLIO, R.; COSTA, H. **Bad smells in software product lines**: A systematic review. In: IEEE. **Software Components, Architectures and Reuse (SBCARS), 2014 Eighth Brazilian Symposium on**. [S.l.], 2014. p. 84–94.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012.

YAMASHITA, A.; COUNSELL, S. Code smells as system-level indicators of maintainability: An empirical study. **Journal of Systems and Software**, Elsevier, v. 86, n. 10, p. 2639–2653, 2013.

YAMASHITA, A.; MOONEN, L. **Exploring the impact of inter-smell relations on software maintainability**: An empirical study. In: IEEE. **Software Engineering (ICSE), 2013 35th International Conference on**. [S.l.], 2013. p. 682–691.

YAMASHITA, A.; MOONEN, L. To what extent can maintenance problems be predicted by code smell detection?–an empirical study. **Information and Software Technology**, Elsevier, v. 55, n. 12, p. 2223–2242, 2013.

YAMASHITA, A.; ZANONI, M.; FONTANA, F. A.; WALTER, B. **Inter-smell relations in industrial and open source systems**: A replication and comparative analysis. In: IEEE. **Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on**. [S.l.], 2015. p. 121–130.

YIN, R. K. Case study research: design and methods, applied social research methods series. **Thousand Oaks, CA: Sage Publications, Inc. Afacan, Y., & Erbug, C.(2009). An interdisciplinary heuristic evaluation method for universal building design. Journal of Applied Ergonomics**, v. 40, p. 731–744, 2003.