UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

BRUNO CARREIRO DA SILVA

Um Método de Refatoração para Modularização de Interesses Transversais

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Daltro José Nunes
Orientador

Porto Alegre, Outubro de 2009

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

da Silva, Bruno Carreiro

Um Método de Refatoração para Modularização de Interesses Transversais / Bruno Carreiro da Silva. – Porto Alegre: PPGC da UFRGS, 2009.

135 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2009. Orientador: Daltro José Nunes.

1. Refatoração. 2. Moduralização de Interesses Transversais. 3. Desenvolvimento de Software Orientado a Aspectos. I. Nunes, Daltro José. II. Título.

SUMÁRIO

ÍNDICE DE LISTAGENS

LISTA DE ABREVIATURAS E SIGLAS

ACD	*Advice Crosscutting Degree*
API	*Application Programming Interface*
AOSD	*Aspect Oriented Software Development*
CDA	*Crosscutting Degree of an Aspect*
CDC	*Concern Diffusion over Components*
CDLOC	*Concern Diffusion over Lines of Code*
CDO	*Concern Diffusion over Operations*
DOF	*Degree of Focus*
DOS	*Degree of Scattering*
DSOA	*Desenvolvimento de Software Orientado a Aspectos*
FCD	*Feature Crosscutting Degree*
GoF	*Gang of Four*
Java ME	*Java Micro Edition*
LOC	*Lines of Code*
NOF	*Number of Features*
OA	Orientação a Aspectos ou Orientado(a) a Aspectos
OO	Orientação a Objetos ou Orientado(a) a Objetos
POA	Programação Orientada a Aspectos
SMS	*Short Message Service*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Sistemas de software bem modularizados podem trazer diversos benefícios como reusabilidade, compreensão, adaptabilidade, manutenibilidade, entre outros. O conceito de separação de interesses está diretamente ligado à ideia de modularização e consiste na capacidade de manter cada interesse em sua própria unidade modular. Um interesse pode estar relacionado tanto a requisitos funcionais como não-funcionais e em diferentes níveis de abstração. Algumas das técnicas que têm sido utilizadas para modularização de interesses são a Programação Orientada a Aspectos (POA) e Refatoração. Entretanto, a maioria das propostas de refatoração que envolvem a POA possui limitações para a modularização de interesses transversais: muitas delas são de granularidade fina; algumas são definidas imprecisamente e possuem sobreposição de intenções. A seleção e composição de tais refatorações para a modularização de interesses é uma tarefa difícil e não-trivial, além de variar em cada contexto específico, o que dificulta o reuso. Algumas das propostas de refatorações são voltadas especialmente para interesses transversais, no entanto possuem um nível de abstração pouco elevado e encontram-se acopladas a mecanismos específicos de linguagens de programação OO e OA. Adicionalmente, a aplicação de refatorações deve ser planejada e acompanhada sistematicamente durante o desenvolvimento e manutenção de um software, pois envolve alocação de recursos e avaliação de custo/benefício. O objetivo deste trabalho é apresentar um método de refatoração para modularização de interesses transversais, através de refatorações de granularidade alta, baseado em padrões recorrentes de estruturas transversais (chamados de sintomas). Além disso, como parte do método, propõem-se algoritmos para análise de impacto a fim de apoiar desenvolvedores no processo decisório de aplicação de refatorações candidatas. Inicialmente, dois estudos bibliográficos foram conduzidos: o primeiro sobre propostas de refatorações que envolvem aspectos e o segundo sobre trabalhos de medição de interesses. Dois estudos de caso foram realizados totalizando 22 interesses de dois sistemas alvos. Este trabalho de avaliação possibilitou uma análise quantitativa e qualitativa dos resultados onde foi possível verificar a aplicabilidade do proposta.

Palavras-chave: Refatoração, Moduralização de Interesses Transversais, Desenvolvimento de Software Orientado a Aspectos.

A Refactoring Method for Crosscutting Concerns Modularisation

ABSTRACT

Well-modularized software systems can bring several benefits such as reuse, comprehension, adaptability, maintainability, among others. The concept of separation of concerns refers to the idea of modularisation, which consists on the ability to keep every concern in its own modular unit. A concern can refer to functional and non-functional requirements and can also be in different abstraction levels. Some of the techniques which have been applied for crosscutting concerns modularization are Aspect-Oriented Programming (AOP) and Refactoring. However, most of the aspect-oriented refactorings have limitations regarding the modularisation of crosscutting concerns. A number of them presents fine-grained transformations. While some of them are well-documented catalogues, a number of them are defined imprecisely, addressing the same situation and having overlapping intentions. They do not allow the designer to holistically reason about the elements involved in a crosscutting concern. It becomes difficult and non-trivial to choose a set of fine-grained refactorings and organize them in a feasible order to achieve the concern modularisation in a specific context. Some of the refactoring techniques are particularly focused on crosscutting concerns, however they are not placed in a sufficient abstraction level. Moreover they are coupled to specific OO and AO language mechanisms. Additionally, the application of refactorings should be planned and realized systematically during software development and maintenance since it involves resource allocation and tradeoff analysis. The goal of this work is to present a refactoring method for crosscutting concerns modularisation, through coarse-grained refactorings based on recurring patterns of crosscutting shapes (called symptoms). Also, as part of the method, algorithms for change-impact analysis are proposed in order to support developers during the decision process of the application of refactoring candidates. Initially, two bibliographic studies were made: the first one about refactoring approaches which involve aspects, and the second one about concern measurement techniques. Two case studies were carried out totalizing 22 concerns of two target systems. This evaluation work allowed a quantitative and qualitative analysis of the results. Thus it was possible to verify the applicability of our approach.

Keywords: Refactoring, Crosscutting Concerns Modularisation, Aspect-Oriented Software Development.

1 INTRODUÇÃO

Durante a década de 70, David L. Parnas (1972) e Edsger Dijkstra[1] (1976) já apresentavam contribuições para os conceitos atuais de separação de interesses, coesão e acoplamento através de suas noções de modularização e *information hiding*. Segundo o IEEE (1990), a modularização em sistemas de software refere-se ao grau de modificações necessárias em componentes de um determinado sistema que são provocadas em decorrência de mudanças em um componente. Assim, sistemas bem modularizados podem trazer diversos benefícios como reusabilidade, compreensão, adaptabilidade, manutenibilidade entre outros (PARNAS, 1972; BOOCH, 1994; MEYER, 1997). Por esse motivo, engenheiros de software consideram a modularização como um princípio chave ao se tomar decisões arquiteturais e de desenho[2] (SANT'ANNA, 2008).

Segundo Tarr (1999), as ideias de modularização, que foram inicialmente introduzidas por Parnas e modernamente conhecidas como separação de interesses, ajudam na obtenção de grande parte dos objetivos principais da Engenharia de Software: *"nossa capacidade em atingir os objetivos da Engenharia de Software depende fundamentalmente da habilidade de manter separados todos os interesses de importância dos sistemas de software"*.

Utilizando o conceito de modularização apresentado por Parnas, Monteiro (2005) define a separação de interesses como *"a capacidade de manter cada interesse em sua própria unidade modular"*. Um interesse é portanto uma propriedade importante que deve ser tratada separadamente (ELRAD; FILMAN; BADER, 2001). Interesses podem estar relacionados tanto a requisitos funcionais como não funcionais e em diferentes níveis de abstração, como por exemplo (ELRAD; FILMAN; BADER, 2001; HANNEMANN; KICZALES, 2002; ROBILLARD; MURPHY, 2007): regras de negócio, auditoria, qualidade de serviço, segurança, *caching*, *buffering*, distribuição, persistência, realização de padrões de projeto[3], entre outros.

Tendo portanto, como uma das metas a separação de interesses, a comunidade da Engenharia de Software vivenciou o surgimento da Programação Orientada a Aspectos (POA) e, de maneira geral, o DSOA - Desenvolvimento de Software Orientado a Aspectos (do inglês AOSD - *Aspect Oriented Software Development*). Como destaca Baniassad *et al.* (2006), o DSOA emergiu a partir de um amadurecimento das pesquisas relacionadas à modularização e à separação de interesses.

[1] Apesar do artigo de Parnas em 1972, quem primeiro utilizou o termo separação de interesses foi Dijkstra em 1976.

[2] No contexto desse trabalho, desenho de software é a tradução considerada para o termo, em inglês, *software design*. Um desenho pode ser representado através de modelos (diagramas) ou código.

[3] Embora o termo *design* esteja traduzido como desenho neste texto, optou-se por utilizar a tradução "padrões de projeto" para o termo em inglês *design patterns*, pois é como ele é mais conhecido no Brasil.

A abordagem de DSOA tem sido aplicada juntamente com técnicas de refatoração tais como (HANENBERG; OBERSCHULTE; UNLAND, 2003), (GARCIA et al., 2004) (HANNEMANN; MURPHY; KICZALES, 2005), (MARIN; MOONEN; DEURSEN, 2005a), (MONTEIRO; FERNANDES, 2006), entre outros, a fim de oferecer um mecanismo auxiliar na utilização do desenvolvimento orientado a aspectos (OA) em sistemas de software orientados a objetos (OO). A refatoração é uma das técnicas essenciais utilizadas para mitigar problemas relacionados à evolução de software (ROBERTS, 1999). Com o objetivo de aumentar atributos de qualidade dos sistemas, as práticas de refatoração surgiram através do emprego de reestruturações sobre unidades de código preservando o seu comportamento externo (OPDYKE, 1992; FOWLER et al., 1999).

Entretanto, a aplicação de refatorações deve ser acompanhada sistematicamente como uma disciplina durante o desenvolvimento e manutenção do software (MENS; TOURWÉ, 2004; PIVETA, 2009). Portanto, para separar interesses transversais e melhor modularizar os sistemas OO através de refatorações, é necessário um acompanhamento sistemático de algumas atividades: primeiramente, é preciso identificar interesses transversais, que são os potenciais candidatos a serem refatorados; em seguida, deve-se selecionar as devidas refatorações (seja para códigos ou modelos); e, se possível, avaliar o impacto da aplicação das refatorações (seja objetivamente, através de métricas, ou subjetivamente); por fim, aplicar as refatorações a fim de separar os interesses transversais e melhorar a modularização do software (seja manualmente ou auxiliada por ferramentas).

O reconhecimento de que a identificação e a refatoração de interesses transversais são atividades importantes para a manutenção de software não é novo. Diversos trabalhos têm sido direcionados para a identificação de interesses transversais tais como (ROBILLARD; MURPHY, 2007), (FIGUEIREDO et al., 2008), (FIGUEIREDO et al., 2009). Além destes, há também uma variedade de propostas sobre refatoração orientada a aspectos que podem ser aplicadas na modularização de interesses transversais: (HANENBERG; OBERSCHULTE; UNLAND, 2003), (GARCIA et al., 2004) (HANNEMANN; MURPHY; KICZALES, 2005), (MARIN; MOONEN; DEURSEN, 2005a), (MONTEIRO; FERNANDES, 2006), entre outros. No entanto, a maioria dessas abordagens possui um nível de granularidade baixo, pois concentram-se na reestruturação específica de partes do código, sem voltarem-se para a modularização de um interesse transversal como um todo.

As refatorações de granularidade baixa disponíveis na literatura (IWAMOTO; ZHAO, 2003; HANENBERG; OBERSCHULTE; UNLAND, 2003; BINKLEY et al., 2006) são definidas sem uma terminologia consistente e padronizada. A maioria delas (GARCIA et al., 2004; MONTEIRO; FERNANDES, 2006; LADDAD, 2006) está direcionada para a mesma situação possuindo objetivos similares. Enquanto algumas das propostas possuem catálogos bem documentados, outras encontram-se definidas imprecisamente. Além disso, frequentemente necessita-se de uma lista de transformações disassociadas para modularizar interesses transversais típicos, como a realização do padrão de projeto *Observer* (GAMMA et al., 1995). Assim, torna-se difícil e não-trivial a formação de um conjunto de refatorações de granularidade baixa, bem como a sua organização de maneira efetiva, a fim de atingir a modularização de interesses. Adicionalmente, conforme Murphy-Hill, Parnin e Black (2009), aproximadamente 40% das refatorações iniciadas por ferramenta ocorrem em lote [4]. E, ainda, os autores destacam que a maioria delas são aplicadas manualmente, revelando uma prática suscetível a erros. Tal situação pode ser observada explicitamente em um cenário de modularização de interesses, quando o desenvolvedor

[4]Diversas refatorações aplicadas em sequência em um período curto de tempo.

necessita realizar diversas refatorações de granularidade baixa (SILVA et al., 2009). Portanto, não é recomendável que tal trabalho seja realizado de forma *ad hoc*, tornando-se aparente a necessidade de um guia para apoiar tal atividade.

Apenas os trabalhos de Hannemman *et al.* (2005) e Marin *et al.* (2005a) propõem refatorações focadas na modularização de interesses transversais. Entretanto, estas duas propostas ainda trazem limitações, pois estão em um nível de abstração pouco elevado para modularização de interesses diante da diversidade e heterogeneidade de interesses transversais encontrados em sistemas de software, conforme é discutido detalhadamente no capítulo 2.

O objetivo principal do presente trabalho é, portanto, propor um método de refatoração para apoiar a modularização de interesses transversais. Este método inclui a integração de uma abordagem existente para detecção e classificação de interesses transversais a um catálogo de refatorações de granularidade alta a ser utilizado sobre os diferentes tipos de interesses, assim como um conjunto de algoritmos para análise de impacto de refatorações candidatas.

As contribuições desta dissertação concentram-se em:

- apresentar um estudo sobre as propostas de refatorações OA existentes na literatura e estabelecer uma análise crítica sobre as mesmas;
- apresentar um estudo sobre métricas orientadas a interesses e destacar suas limitações;
- apresentar o método de refatoração que inclui um catálogo de refatorações integrado à técnica de identificação e classificação de interesses baseada em sintomas. Tais refatorações encontram-se voltadas especialmente para a modularização de interesses transversais, utilizando-se de refatorações de menor granularidade já existentes;
- algoritmos que apoiam a análise de impacto das refatorações candidatas;
- e, por fim, oferecer uma avaliação quantitativa de tal abordagem através de estudos de caso.

Esta dissertação foi realizada através de uma pesquisa descritiva, passando pelo estudo bibliográfico das principais fontes publicadas na área. Para o delineamento deste trabalho, estudou-se inicialmente os conceitos relacionados ao DSOA, bem como as propostas relacionadas à análise e medição de interesses transversais, a fim de levantar um conhecimento sobre o estado da arte das técnicas de identificação e medição de interesses. Além disso, pesquisou-se também os trabalhos sobre refatorações OA existentes na literatura. Tal estudo possibilitou um conhecimento amplo sobre essas abordagens a fim de levantar suas limitações para embasar a composição das novas refatorações encontradas neste texto. Portanto, a criação e apresentação das refatorações apresentadas neste trabalho baseiam-se principalmente nos estudos sobre (i) análise quantitativa através de métricas, (ii) identificação e classificação de interesses, e (iii) propostas de refatoração OA existentes na literatura. Posteriormente, foram realizados estudos de caso com o objetivo de avaliar a proposta. Por fim, destinou-se um tempo específico para a análise dos resultados e a conclusão do trabalho.

O restante do texto encontra-se organizado na seguinte forma: o capítulo 2 apresenta o estudo realizado sobre as propostas de refatorações que envolvem aspectos encontradas na literatura; em seguida, o capítulo 3 explica as métricas orientadas a interesses encontradas

no estudo bibliográfico realizado e discute suas limitações; o capítulo 4 reúne e descreve um conjunto de sintomas orientados a interesses identificado em trabalho conjunto com Figueiredo *et al.* (2009); posteriormente, o capítulo 5 apresenta o catálogo de refatorações voltadas para modularização de interesses; o capítulo 7 compreende a apresentação dos estudos de caso realizados, bem como a análise dos resultados obtidos; e, finalmente, o capítulo 8 faz as considerações finais e discute os trabalhos futuros.

2 UM ESTUDO SOBRE PROPOSTAS DE REFATORAÇÕES OA

Este capítulo tem como objetivo apresentar uma revisão bibliográfica realizada sobre as propostas de refatorações orientadas a aspectos (OA). Assim, é possível obter tanto uma visão geral sobre o que se tem pesquisado na área de refatorações, considerando o paradigma de programação orientada a aspectos (POA), como uma visão mais detalhada sobre alguns dos trabalhos estudados.

Hannemann (2006) apresentou uma classificação para as refatorações dividindo-as em três classes, como mostra a tabela 2.1: primeiro, a classe das refatorações tradicionais OO que são estendidas de forma a assegurar que elas não alteram o comportamento das construções OA; segundo, as refatorações que envolvem explicitamente e principalmente construções da POA; e, por último, as refatorações que capturam interesses tranversais entrelaçados e espalhados em código OO e os transformam em aspectos de acordo com a POA.

Tabela 2.1: Classificação das propostas de refatorações que envolvem POA

Abordagem/Técnica	**Alvo**	**Foco/Motivação**	**Exemplos**
Refatorações OO cientes de Aspectos	Construções OO	Assegurar que as reestruturações OO atualizam os relacionamentos existentes com os aspectos.	Renomear método, Extrair método etc.
Refatorações OA	Construções OA e OO	Prover novas refatorações que envolvam principalmente construções OA.	Trocar membro por declaração intertipo, Renomear aspecto etc.
Refatorações para Interesses Transversais	Implementações de Interesses Transversais	Modularização de interesses transversais através de aspectos.	Refatoração para o padrão de projeto *Observer*.

O restante deste capítulo está dividido da seguinte forma: a seção 2.1 apresenta os trabalhos sobre refatorações OO cientes de aspectos; em seguida, a seção 2.2 expõe as propostas de refatorações OA; posteriormente, a seção 2.3 apresenta os trabalhos de refatorações voltadas para interesses transversais; na seção 2.4 são discutidas limitações acerca dos trabalhos mencionados neste capítulo; e, por fim, a seção 2.5 resume brevemente esta parte do trabalho.

2.1 Refatorações OO Cientes de Aspectos

A primeira classe de refatorações citada por Hannemann (2006) é também conhecida como *aspect-aware refactorings*, isto é, refatorações OO cientes também da programa-

ção OA. Estas refatorações lidam principalmente com elementos da orientação a objetos, porém devem considerar que aspectos podem estar ligados a tais elementos e que estes podem ser refatorados. A partir do momento em que tais elementos OO são modificados, os apectos podem passar para um estado inconsistente ou inesperado, possivelmente alterando o comportamento externo do sistema. Por isso, há a necessidade de verificar-se a existência de aspectos relacionados e refatorá-los também. Tal situação foi inicialmente identificada e apresentada em (IWAMOTO; ZHAO, 2003) e (HANENBERG; OBERSCHULTE; UNLAND, 2003).

Em (IWAMOTO; ZHAO, 2003), Iwamoto e Zhao investigaram como e quais refatorações orientadas a objetos, como as propostas por Fowler (FOWLER et al., 1999), poderiam ser aplicadas também em código orientado a aspecto. Neste caso, o objetivo dos autores era estudar o impacto que as refatorações OO tinham sobre possíveis construções orientadas a aspectos relacionadas aos elementos refatorados. Os autores tomaram como base as refatorações OO propostas por Fowler e verificaram quais poderiam impactar em aspectos relacionados aos elementos refatorados.

A tabela 2.2 apresenta um subconjunto das refatorações de Fowler (FOWLER et al., 1999) indicando: em círculo, as refatorações que podem realizar impacto em aspectos, isto é, aquelas que possivelmente alteram a semântica dos aspectos relacionados, caso existam; e em triângulo, as refatorações que não impactam na possível existência de aspectos relacionados, ou seja, tais refatorações podem ser aplicadas sem se preocupar com a existências de construções da POA.

Como pode-se notar, a aplicação da maioria das refatorações da tabela 2.2 influencia possíveis aspectos existentes. Apenas 3 das 31 refatorações OO listadas não possuem impacto sobre construções da programação OA, isto é, podem ser aplicadas independentes da existência de aspectos no sistema. As demais 28 refatorações apresentadas no estudo indicam a necessidade de estendê-las para apoiar suas aplicações em sistemas que possuem aspectos. Tais refatorações OO que precisam estar cientes da existência de aspectos estão no primeiro grupo de refatorações propostas em (HANNEMANN, 2006), como apresentadas e explicadas na tabela 2.1.

Como exemplo, apresenta-se o código da listagem 2.1 (adaptado de (IWAMOTO; ZHAO, 2003)). Este trecho de código possui uma classe chamada *Sample* e um aspecto de nome *AspectSample*. A classe possui o método *pm()* que imprime a frase *print method* na tela, e o método *main* para executá-la. Já o aspecto possui a definição de adendo que captura chamadas ao método *pm()* da classe *Sample* e, antes da execução do método, imprime a frase *pm ok* na tela.

Listagem 2.1: Um exemplo simplificado relacionando uma classe e um aspecto

```
1  public aspect AspectSample {
2          before() : call ( * Sample.pm()) {
3           System.out.println("pm ok");
4          }
5  }
6
7  public class Sample {
8          public static void main(String args[]){
9                  new Sample.pm();
10         }
11         public void pm() {
12                 System.out.println("print method");
13         }
14 }
```

Tabela 2.2: Refatorações OO e o impacto em aspectos

Refatorações OO	Impacto em Aspectos
Add Parameter	◯
Extract Class	◯
Extract Method	◯
Extract Subclass	◯
Hide Method	◯
Inline Method	◯
Introduce Explaining Variable	◯
Move Method	◯
Parameterize Method	◯
Pull Up Field	◯
Push Down Field	◯
Rename Method	◯
Replace Conditional with Polymorphism	◯
Replace Magic Number with Symbolic Constant	△
Replace Parameter with Explicit Methods	◯
Remove Parameter	◯
Encapsulate Downcast	◯
Extract Interface	◯
Extract Superclass	◯
Inline Class	◯
Inline Temp	△
Move Field	◯
Move Setting Method	◯
Pull Up Constructor	◯
Pull Up Method	◯
Push Down Method	◯
Replace Array with Object	◯
Replace Exception with Test	◯
Replace Nested Conditional with Guarddd Clauses	△
Replace Temp with Query	◯
Set Encapsulate Field	◯

◯: Modificações podem ser necessárias em aspectos. △: Aplicável sem impacto na existência de aspectos.

Caso seja aplicada a refatoração *Rename Method* (FOWLER et al., 1999) a fim de modificar o nome do método *pm()* para, por exemplo, *print_method()*, atribuindo-o assim um nome mais significativo, o aspecto e seu adendo não serão mais úteis e, neste caso, não se comportarão como esperado. Em casos como este, deve-se levar em consideração não somente as construções da POO, mas também as possíveis construções da POA que podem estar relacionadas aos elementos OO refatorados. A listagem 2.2 demonstra o aspecto refatorado em decorrência da modificação do nome do método *pm()*.

Listagem 2.2: Classe e aspecto após refatoração

```
1  public aspect AspectSample {
2          before() : call ( * Sample.print_method()) {
3           System.out.println("pm ok");
4          }
5  }
6
7  public class Sample {
8          public static void main(String args[]){
```

```
9                   new Sample.print_method();
10          }
11
12          public void print_method() {
13                  System.out.println("print method");
14          }
15  }
```

Em (HANENBERG; OBERSCHULTE; UNLAND, 2003) os autores foram além da análise de Iwamoto e Zhao (2003) mostrada na tabela 2.2. Neste trabalho, foram propostas três condições necessárias para se garantir a preservação do comportamento externo (ou comportamento observável) adicionais às condições inicialmente propostas no trabalho seminal de Opdyke em (OPDYKE, 1992). As condições de Opdyke foram apresentadas a fim de orientar a verificação da preservação do comportamento observável de um sistema após refatorações. Em (HANENBERG; OBERSCHULTE; UNLAND, 2003), os autores apresentaram três novas condições a fim de considerar a possível presença de aspectos relacionados a elementos OO refatorados, que são:

1. A quantidade de pontos de junção que são capturados por um dado conjunto de pontos de junção não deve ser modificado após a refatoração.

2. Os pontos de junção que são capturados por um dado conjunto de junção devem ter uma posição equivalente dentro do fluxo de controle do sistema se comparado ao estado anterior à refatoração.

3. A informação dos pontos de junção oferecida por cada conjunto de junção não deve ser diminuída.

A primeira condição é necessária para garantir que os pontos de junção capturados por aspectos antes e após a refatoração são os mesmos. Adicionalmente, a segunda condição garante que, além da quantidade, os pontos de junção capturados pelos conjuntos de junção após a refatoração devem ter uma posição equivalente no fluxo de controle do sistema antes da refatoração. Por fim, a terceira condição tem o objetivo de assegurar que a informação relacionada aos pontos de junção de cada conjunto de junção não deve ser enfraquecida, isto é, diminuída, em relação à informação antes da refatoração, caso haja alguma alteração nestes conjuntos. Neste trabalho os autores apesentaram também a mecânica de aplicação de três refatorações OO, considerando nesse caso a presença de aspectos relacionados: *Rename Method*, *Extract Method* e *Move Method*.

Embora importantes, não há registros nas publicações pesquisadas de que tais condições propostas em (HANENBERG; OBERSCHULTE; UNLAND, 2003) estejam sendo implementadas por ferramentas de auxílio a refatoração. Parte deve-se ao fato de que as condições foram definidas de maneira informal, isto é, imprecisa e pouco rigorosa, deixando-as suscetíveis a dúbia interpretação.

2.2 Refatorações OA

O segundo grupo da classificação apresentada por Hannemman (tabela 2.1) inclui refatorações que reestruturam primordialmente as construções orientadas a aspectos, podendo também modificar elementos OO que estão relacionados. Elas são denominadas também como refatorações OA, pois estão direcionadas explicitamente para as construções da programação orientada a aspectos.

Iwamoto e Zhao (2003) apresentaram também exemplos de refatorações OA: *Extract Pointcut* e *Extract Advice*. As listagens 2.3 e 2.4 mostram um exemplo da refatoração *Extract Pointcut*. A listagem 2.3 contém uma classe chamada *Impressao* com dois métodos: um método para imprimir uma determinado texto na tela, e o método *main* para executá-la. Além disso, há também um aspecto chamado *Registro* com dois adendos: ambos capturam a chamada ao método *imprime*, porém adicionam um texto antes e depois da execução do método, respectivamente.

Listagem 2.3: Classe e aspecto antes da refatoração OA

```
1  public class Impressao {
2
3          public void imprime (String texto) {
4             System.out.println(texto);
5          }
6
7          public static void main (String args[]) {
8             Impressao imp = new Impressao();
9             imp.imprime("Ola!");
10         }
11  }
12
13 public aspect Registro {
14
15         before() : call (void Impressao.imprime(String)) {
16            System.out.println("Imprimindo␣texto:");
17         }
18
19         after() : call(void Impressao.imprime(String)) {
20            System.out.println("impressao␣concluida!") ;
21         }
22 }
```

É possível notar que os adendos do aspecto *Registro* capturam o mesmo ponto de junção (chamadas ao método *imprime* da classe *Impressao*), com a única diferença de que um adendo é executado antes e o outro depois que o método *imprime* é executado. Diante disto, a refatoração *Extract Pointcut* pode ser aplicada para extrair um conjunto de junção a fim de capturar o ponto junção utilizado nos dois adendos. Além disso, o conjunto de junção criado pode ser reutilizado em demais adendos que possam vir a ser criados. A listagem 2.4 exemplifica o resultado dessa refatoração OA aplicada na listagem 2.3. Um conjunto de junção denominado *chamadaImpressao* foi extraído e criado a partir dos adendos para ser reutilizado.

Listagem 2.4: Classe e aspecto após refatoração OA

```
1  public class Impressao {
2
3          public void imprime (String texto) {
4             System.out.println(texto);
5          }
6
7          public static void main (String args[]) {
8             Impressao imp = new Impressao();
9             imp.imprime("Ola!");
10         }
11  }
12
13 public aspect Registro {
```

```
14
15          pointcut chamadaImpressao() : call (void Impressao.imprime(String));
16
17          before() : chamadaImpressao() {
18              System.out.println("Imprimindo texto:");
19          }
20
21          after() : chamadaImpressao() {
22              System.out.println("impressao concluida!") ;
23          }
24  }
```

Como é possível observar através das listagens 2.3 e 2.4, as refatorações OA são motivadas em primeira instância na modificação de construções orientadas a aspectos, podendo ou não modificar elementos da programação orientada a objetos.

Adicionalmente, o trabalho (HANENBERG; OBERSCHULTE; UNLAND, 2003) descreveu os passos de aplicação de três refatorações desse tipo: *Extract Advice*, *Extract Introduction* e *Separate Pointcut*.

Em (GARCIA et al., 2004) foi apresentado um conjunto de dez refatorações OA, inspiradas nas já propostas em (FOWLER et al., 1999) para a POO: Extrair campo para aspecto, Extrair método para aspecto, Extrair código para aspecto, Extrair definição de conjunto de junção, Suprimir definição de conjunto de junção, Renomear conjunto de junção, Renomear aspecto, Puxar adendo/conjunto de junção, Suprimir hierarquia de aspectos, Alinhar definição de conjunto de junção. Cada refatoração proposta inclui: um exemplo ilustrativo, a motivação para aplicação, e os passos de realização.

Binkley *et al.* (2006) apresentam uma abordagem de refatorações apoiada por ferramenta. Essa proposta inclui a ferramenta *AOP-Migrator* e ainda seis refatorações OA: *Extract Begining/End of Method/Handler*, *Extract Before/After call*, *Extract Conditional*, *Pre Return*, *Extract Wrapper* e *Extract Exception Handling*.

Rura (2003) também apresenta contribuições para a área de refatorações OA. Em seu trabalho, ele revisita as refatorações OO propostas inicialmente em (OPDYKE, 1992) considerando a presença de aspectos. Além disso, Rura (2003) propôs pequenas refatorações (com passos muito simples e quase atômicos) a fim de dar suporte a refatorações que necessitam de um conjunto maior de passos. Estas refatorações maiores, o autor chama de refatorações OA de alto nível e são quatro: *Move Static Introduction*, *Extract Common Members to Aspect*, *Extract Interface Support to Aspect* e *Extract Disjoint State into Aspect*.

Laddad (2006) reune também diversas refatorações para criação e reestruturação de aspectos. Tais refatorações OA encontram-se voltadas para situações em que se utiliza estruturas e técnicas recorrentes da programação orientada a objeto, vistas como oportunidades de refatoração: *Extract Method Calls*, *Extract Exception Handling*, *Extract Concurrency Control*, *Extract Worker Object Creation*, *Replace Argument Trickle by Wormhole*, *Extract Interface Implementation*, *Replace Override with Advice*, *Extract Lazy Initialization*, *Extract Contract Enforcement*.

Por fim, o catálogo de Monteiro, proposto em (MONTEIRO, 2005), também inclui refatorações dentro dessa classificação, porém é discutido exclusivamente na seção 2.2.1, pois engloba outros tipos de refatorações e, no geral, possui um escopo mais amplo merecendo assim um destaque maior.

2.2.1 Catálogo de Monteiro

Monteiro propôs em (MONTEIRO, 2005) (MONTEIRO; FERNANDES, 2006) um catálogo de refatorações para lidar com construções orientadas a aspectos. Tais refatorações foram derivadas de experiências prévias, especialmente as vivenciadas a partir de dois estudos de caso relatados em sua tese.

As refatorações do catálogo encontram-se divididas em três grupos: o primeiro grupo agrega as refatorações para extração de elementos da programação OO para a programação OA; o segundo grupo inclui as refatorações para reestruturação interna de aspectos já existentes; e, por fim, o terceiro grupo de refatorações foi proposto especialmente para lidar com a generalização entre aspectos. A tabela 2.3 lista as 27 refatorações do catálogo organizadas em seus respectivos grupos.

Em (MONTEIRO, 2005) e (MONTEIRO; FERNANDES, 2006) são detalhadas as descrições de cada refatoração num formato semelhante ao catálogo de (FOWLER et al., 1999). Para cada refatoração são apresentados os seguintes itens:

- nome, identificando a refatoração;
- a situação típica, explicando as situações passíveis de aplicar a refatoração;
- a ação recomendada, resumindo a reestruturação realizada pela refatoração;
- a motivação, enfatizando e focando na necessidade de aplicação;
- um exemplo, para facilitar o entendimento e ilustrar uma possível aplicação;
- e, por fim, a mecânica, que reune uma sequência de passos para guiar a realização da refatoração.

Embora não esteja baseado em definições formais, a estrutura do catálogo de Monteiro possui uma facilidade de compreensão maior que as demais refatorações discutidas nesse capítulo. Tal fato possibilita o aumento do grau de aplicabilidade das refatorações em trabalhos posteriores, como em (PAGLIARI; NUNES, 2007), principalmente na realização de estudos experimentais.

2.3 Refatorações para Interesses Transversais

O terceiro e último grupo de refatorações de acordo com a classificação proposta em Hannemann (2006) (e apresentada na tabela 2.1) refere-se às reestruturações necessárias para refatorar interesses transversais através de construções OA. Com fins ilustrativos, a figura 2.1 mostra graficamente o resultado da aplicação de uma refatoração para modularização de um interesse transversal hipotético. Considere que as barras verticais da figura são os componentes do sistema e as partes sombreadas são os trechos da implementação que estão diretamente envolvidos na realização do interesse transversal. Logo, uma implementação não-modularizada é representada pelas barras da esquerda na figura 2.1. Componentes, e seus respectivos membros, que participam do interesse transversal, possuem trechos de código, relacionados ao interesse, espalhados em todo o programa. Uma possível refatoração para extração e modularização deste interesse transversal hipotético, utilizando aspectos, poderia obter a implementação do sistema representada pelas barras da parte direita da figura 2.1, onde a realização do interesse aparece completamente concentrada em um componente (neste caso, um aspecto).

Tabela 2.3: Refatorações do catálogo de Monteiro

Grupo	Nome da Refatoração	Tradução
Refatorações para Extração de Aspectos	*Change abstract class to interface.*	Trocar classe abstrata por interface.
	Extract feature into aspect.	Extrair funcionalidade para aspecto.
	Extract fragment into. advice.	Extrair fragmento para adendo.
	Extract inner class to stand-alone.	Extrair classe interna para autônoma.
	Inline class within aspect.	Internalizar classe em aspecto.
	Inline interface within aspect.	Internalizar interface em aspecto.
	Move field from class to intertype.	Mover atributo de classe para intertipo.
	Move method from class to intertype.	Mover método de classe para intertipo
	Replace implements with declare parents.	Trocar *implements* por *declare parents*.
	Split abstract class into aspect and interface.	Dividir classe abstrata em aspecto e interface.
Refatorações para Reestruturação Interna de Aspectos	*Extend marker interface with signature.*	Estender interface marcadora por assinatura.
	Generalise target type with marker interface.	Generalizar tipo alvo com interface marcadora.
	Introduce aspect protection.	Introduzir proteção de aspecto.
	Replace intertype field with aspect map.	Trocar campo intertipo por mapemanto no aspecto.
	Replace intertype method with aspect method.	Trocar método intertipo por método no aspecto.
	Tidy up internal aspect structure.	Limpar estrutura interna de aspectos.
Refatorações para Lidar com Generalização em Aspectos	*Extract superaspect.*	Extrair superaspecto.
	Pull up advice.	Generalizar adendo.
	Pull up declare parents.	Generalizar *declare parents*.
	Pull up intertype declaration.	Generalizar declaração intertipo.
	Pull up marker interface.	Generalizar interface marcadora.
	Pull up pointcut.	Generalizar conjunto de junção.
	Push down advice.	Especializar adendo.
	Push down declare parents.	Especializar *declare parents*.
	Push down intertype declaration.	Especializar declaração intertipo.
	Push down marker interface.	Especializar interface marcadora.
	Push down pointcut.	Especializar conjunto de junção.

Uma refatoração deste grupo deve levar em consideração todos os elementos e relacionamentos que compõem e participam do interesse transversal, podendo assim envolver diversos componentes (classes, aspectos, interfaces etc.) e seus respectivos membros. As reestruturações que envolvem a refatoração de interesses transversais devem, portanto, considerar os elementos envolvidos como um todo. Tais refatorações possuem um nível de granularidade maior, pois, em apenas uma refatoração, transformações são realizadas em um número maior de elementos. Isto deve-se ao fato de que um interesse transversal geralmente está manifestado em diversos componentes. Dessa maneira, tais refatorações se aproveitam daquelas de granularidade mais fina, isto é, refatorações “menores”, para

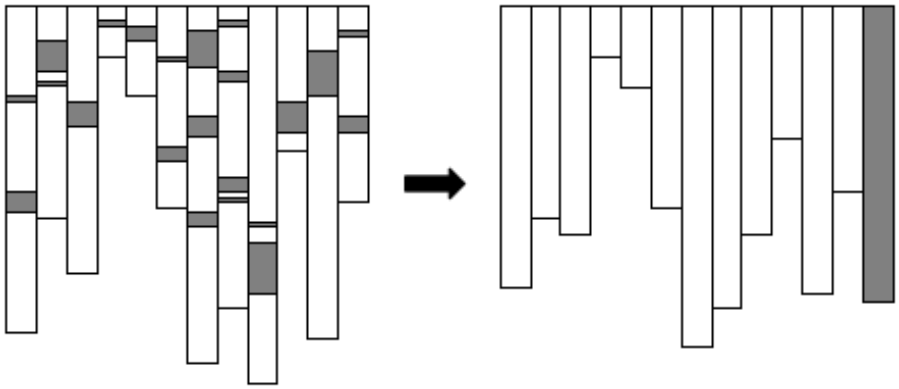


Figura 2.1: Um interesse transversal não-modularizado extraído para um aspecto

auxiliar no processo de reestruturação do interesse transversal como um todo.

Duas propostas que apresentam refatorações deste grupo foram encontradas na literatura: primeiro, o trabalho de Hannemann *et al.* (2005), que apresenta refatorações baseadas em papéis; e segundo, o trabalho de Marin *et al.* (2005a) que propõe refatorações baseadas em tipos de interesses transversais.

Refatoração Baseada em Papéis

Hannemann *et al.* (2005) apresentou a proposta de *Refatoração Baseada em Papéis* para modularização de interesses transversais. A abordagem de Hannemann *et al.* descreve as refatorações em termos de papéis e seus elementos, em um nível independente da implementação, como ilustra a figura 2.2 (adaptada de (HANNEMANN; MURPHY; KICZALES, 2005)). A idéia principal da proposta sustenta que as instruções de refatoração são as mesmas para qualquer elemento concreto do programa que realiza um determinado papel do interesse tranversal. Para tanto, um mapeamento deve ocorrer entre os elementos abstratos dos papéis e os elementos concretos da implementação de um sistema. Os passos de refatoração, que encontram-se descritos sobre os elementos abstratos dos papéis, devem então ser aplicados sobre os elementos concretos que foram mapeados.

Hannemann *et al.* apresentou um conjunto de quatro atividades que devem ser seguidas para a aplicação da abordagem:

1. **Seleção da refatoração para interesse transversal:** o desenvolvedor deve selecionar, a partir de uma lista, a refatoração apropriada para o interesse transversal a ser moduralizado. A refatoração selecionada deve incluir uma descrição abstrata do interesse transversal alvo e um conjunto de instruções para efetuar as reestruturações necessárias.

2. **Realização do mapeamento:** nesta atividade, uma ferramenta deve auxiliar o desenvolvedor no mapeamento dos elementos abstratos para elementos concretos do sistema.

3. **Planejamento da refatoração:** nesta etapa, o desenvolvedor deve verificar em quais situações deve-se ou não aplicar refatorações dependendo do impacto a ser obtido como resultado. Esta atividade também deve ser auxiliada por uma ferramenta. Por exemplo, a ferramenta pode alertar o desenvolvedor se um novo aspecto a ser criado possuirá um nome que colide com o de uma outra entidade do sistema.

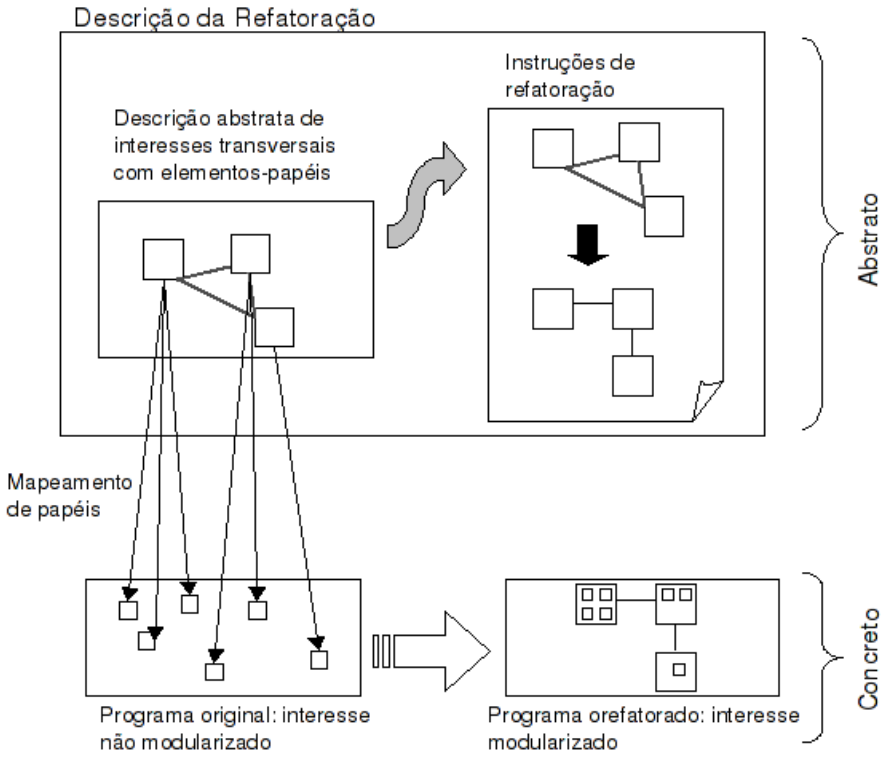


Figura 2.2: Esboço gráfico das refatorações baseadas em papéis

4. **Execução:** após o planejamento da refatoração, uma ferramenta poderá auxiliar o desenvolvedor na transformação dos elementos do código para obter o resultado desejado, seguindo as definições realizadas nas etapas anteriores.

Refatoração Baseada em Tipos de Interesses Transversais

Também direcionada à modularização de interesses transversais, Marin *et al.* (2005a) apresentaram a proposta de *Refatoração Baseada em Tipos de Interesses Transversais*. De acordo com os autores, um tipo de interesse transversal é constituído de três propriedades: uma intenção, um idioma de implementação e um mecanismo de linguagem orientada a aspectos para representá-lo concretamente. Interesses transversais concretos encontrados nos sistemas são instâncias dos tipos definidos pelos autores.

Em (MARIN; MOONEN; DEURSEN, 2005b) um conjunto de treze tipos de interesses transversais são definidos e apresentados. A tabela 2.4 demonstra, como exemplo, três desses tipos.

2.4 Discussão sobre as Propostas de Refatorações

A diversidade de refatorações OA levantadas no estudo está resumida na tabela 2.5. Esta tabela reune as refatorações OA citadas na seção 2.2.

Diante da variedade de trabalhos e principalmente de refatorações (tabela 2.5) é possível realizar algumas observações. As refatorações são apresentadas utilizando terminologias distintas com diferentes formas de descrição. Algumas apresentam apenas uma breve definição textual (pouco precisa) enquanto outras apresentam um passo-a-passo detalhado para aplicação incluindo exemplos. A falta de uma terminologia consistente e

Tabela 2.4: Tipos de interesses transversais

Tipo	Intenção	Idioma OO	Mecanismo de Aspectos	Instâncias
Propagação de Exceção	Propagar uma exceção quando não há uma captura apropriada programada.	Exceção propagada para os componentes que chamam.	Usar mecanismo de *declare soft* (risco: identidade da exceção pode ser perdida)	SQLException lançada a partir de métodos da biblioteca JDBC.
Cláusula declarativa *throws*	Adicionar exceção específica para a cláusula *throws* de todos os métodos dentro de um contexto formalizado.	Adicionar diretamente a nova cláusula *throws*	Não suportado. Desejável mecanismo *declare throws*	Exceção específica de RMI (SOARES et al., 2002) Exceção para transações (KIENZLE; GUERRAOUI, 2002).
Aplicação de Política	Impor uma política em um grupo de elementos do sistema.	Não suportado por linguagem, mas por uso de documentação interna/comentários.	Mecanismos de *declate warning/error*.	Limitar o acesso para uma determinada funcionalidade. Ex.: Acesso a funcionalidades da biblioteca AWT a partir de componentes EJB (LADDAD, 2003)

uma descrição padronizada pode prejudicar o entendimento na aplicação das refatorações e até mesmo a proposição de novas refatorações com intenções semelhantes ou iguais às já existentes. Algumas das refatorações apresentadas na tabela 2.5 possuem intenções similares e causam efeitos parecidos ou idênticos quando aplicadas. Por exemplo: *Extract Pointcut* (linha 01 da tabela) é equivalente a *Extract Pointcut Definition* (linha 09); *Extract Begining/End of Method/Handler* (linha 16 da tabela) pode ser englobado pela definição de *Extract Code to Advice* (linha 08). Além disso, é possível encontrar refatorações com o mesmo nome: *Extract Advice* (linhas 02 e 05 da tabela) e *Extract Exception Handling* (linhas 21 e 27).

Já em relação às propostas de refatorações para interesses transversais é possível observar o seguinte. Ambas as propostas levantadas no estudo (HANNEMANN; MURPHY; KICZALES, 2005) e (MARIN; MOONEN; DEURSEN, 2005a) preocupam-se em agrupar interesses com estruturas similares, a fim de criar uma categorização. Entretanto, a proposta de Hannemann *et al.* (2005), mesmo envolvendo interesses transversais com papéis abstratos, continua tendo um nível de abstração pouco elevado. Por exemplo, o interesse do padrão de projeto *Observer* (bem como seus papéis *Subject/Observer*) são abstratos no sentido de existirem diversas implementações concretas possíveis para eles, isto é, podemos ter diversas instâncias do padrão *Observer* em diversos contextos. No entanto, a estrutura do interesse transversal formada pelos componentes participantes do padrão *Observer* se configuram de uma maneira em que é possível encontrar outros interesses que não são o padrão *Observer* mas que possuem a mesma configuração estrutural.

O próprio padrão *Mediator* (GAMMA et al., 1995) pode ter estrutura igual ao do *Observer*. A figura 2.3 ilustra duas situações distintas em que os interesses dos padrões *Mediator* (parte superior, letra a) e *Observer* (parte inferior, letra b) encontram-se configurados de maneira semelhante. Como é possível notar, em ambas as situações existem interfaces completamente dedicadas ao interesse (seja ele do padrão *Mediator* ou *Observer*), isto é, interfaces que existem especialmente para a realização do padrão. Além disso, há classes que implementam tais interfaces e possuem uma pequena porção de sua estrutura dedicada à participação do interesse. Portanto, a configuração estrutural do interesse transversal é a mesma para ambas situações e assim seria possível aplicar o mesmo conjunto e sequência de refatorações para modularizar os respectivos interesses. Entretanto, estes dois exemplos da figura 2.3 estariam classificados como dois tipos distintos

Tabela 2.5: Sumário das refatorações OA levantadas no estudo

#	Nome	Referência
01	*Extract Pointcut*	(IWAMOTO; ZHAO, 2003)
02	*Extract Advice*	
03	*Extract Introduction*	(HANENBERG; OBERSCHULTE; UNLAND, 2003)
04	*Separate PointCut*	
05	*Extract Advice*	
06	*Extract Field to Aspect*	(GARCIA et al., 2004)
07	*Extract Method to Aspect*	
08	*Extract Code to Advice*	
09	*Extract Pointcut Definition*	
10	*Collapse Pointcut Definition*	
11	*Rename Pointcut*	
12	*Rename Aspect*	
13	*Pullup Advice/Pointcut*	
14	*Collapse Aspect Hierarchy*	
15	*Inline Pointcut Definition*	
16	*Extract Begining/End of Method/Handler*	(BINKLEY et al., 2006)
17	*Extract Before/After call*	
18	*Extract Conditional*	
19	*Pre Return*	
20	*Extract Wrapper*	
21	*Extract Exception Handling*	
22	*Move Static Introduction*	(RURA, 2003)
23	*Extract Common Members to Aspect*	
24	*Extract Interface Support to Aspect*	
25	*Extract Disjoint State into Aspect*	
26	*Extract Method Calls*	(LADDAD, 2006)
27	*Extract Exception Handling*	
28	*Extract Concurrency Control*	
29	*Extract Worker Object Creation*	
30	*Replace Argument Trickle by Wormhole*	
31	*Extract Interface Implementation*	
32	*Replace Override with Advice*	
33	*Extract Lazy Initialization*	
34	*Extract Contract Enforcement*	
35	Refatorações do Catálogo de Monteiro (vide Tabela 2.3)	(MONTEIRO, 2005)

de interesses de acordo com a proposta de Hannemann *et al.* (2005).

Além da questão ilustrada e exemplificada na figura 2.3, Hannemann *et al.* (2005) não apresentam a aplicação da proposta para outros tipos de interesses que não sejam os padrões de projeto. Portanto, a aplicabilidade da abordagem não foi avaliada em outros contextos.

Quanto à segunda proposta para refatoração de interesses transversais (Marin *et al.* (2005a)), os autores não conseguem atingir, em sua classificação baseada em tipos, uma independência do idioma de implementação. Como pode ser observado na tabela 2.4, a classificação dos interesses estão ligados a um idioma de implementacão OO relacionado a mecanismos específicos da programação orientada a aspectos. Por exemplo, o tipo de interesse *Cláusula Declarativa Throws* (segunda linha da tabela 2.4) é intrinsecamente dependente do idioma de implementação, pois a própria definição depende da existência de

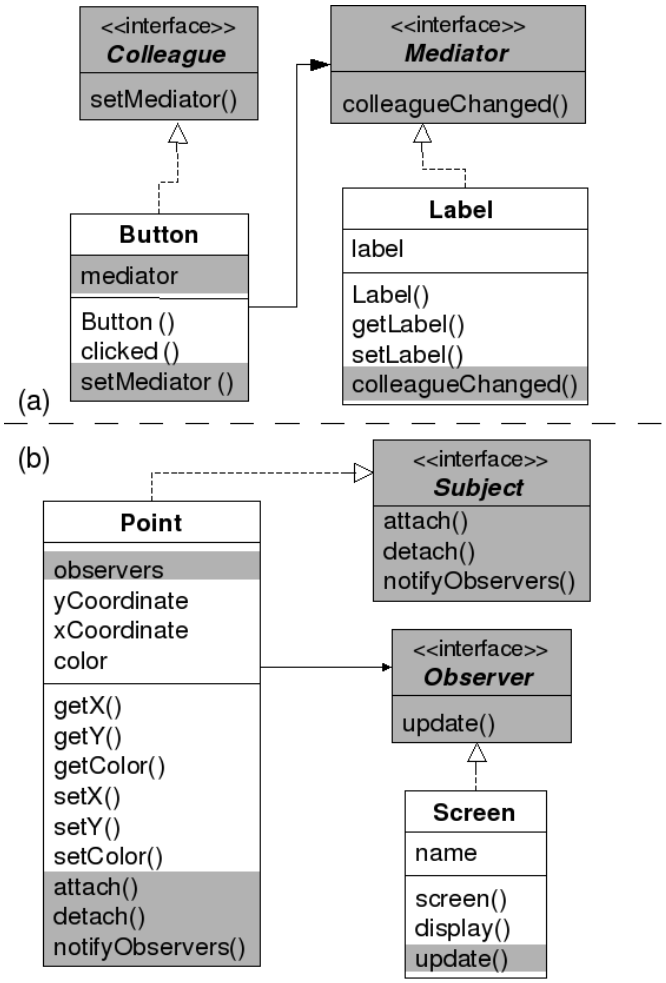


Figura 2.3: Instâncias dos padrões *Mediator* e *Observer*

cláusulas *throws* na linguagem de programação OO. A mesma observação pode ser feita para o tipo *Propagação de Exceção* (primeira linha da tabela 2.4), o qual está diretamente ligado ao mecanismo de propagação de exceção de linguagens OO e OA. Portanto, nessa proposta o nível de abstração atingido pela categorização em tipos também encontra-se pouco elevado, pois os tipos de interesses ainda dependem de idiomas de implementação. Para essa proposta também vale a análise feita para a anterior de Hannemann *et al.* (2005) – é possível encontrarmos tipos diferentes de acordo com a classificação de Marin *et al.* (2005a) mas que sua configuração estrutural pode se encontrar igual.

2.5 Considerações Finais do Capítulo

Este capítulo apresentou trabalhos relacionados a propostas de refatorações que envolvem a POA das diversas maneiras possíveis: refatorações OO cientes de aspectos, as próprias refatorações OA e as refatorações para interesses tranversais. Discutiu-se também nesta parte do trabalho algumas limitações levantadas ao longo do estudo bibliográfico que conduziu essa etapa da dissertação. Tal estudo revelou-se importante no conhecimento das abordagens disponíveis na literatura para aplicação de refatorações que envolvem aspectos. Este conhecimento é fundamental para a proposta do método de refatoração que é objetivo do presente trabalho. No próximo capítulo, apresenta-se métricas

voltadas para medição de interesses. Tais métricas são fundamentais para análises quantitativas e qualitativas sobre a separação de interesses e modularização de sistemas de software.

3 UM ESTUDO SOBRE MÉTRICAS PARA INTERESSES

Este capítulo apresenta métricas para interesses, levantadas a partir de um estudo bibliográfico realizado sobre os principais trabalhos relacionados à análise e medição de interesses, bem como uma breve discussão sobre as propostas estudadas. Tal estudo é importante para o entendimento de como é possível avaliar objetivamente interesses em sistemas de software, percorrendo o estado-da-arte referente a este campo.

O capítulo encontra-se organizado da seguinte forma: as seções de 3.1 a 3.6 dedicam-se à explicação das métricas para interesses encontradas na pesquisa; posteriormente, a seção 3.7 discute e sumariza o estudo; por fim, a seção 3.8 conclui o capítulo.

3.1 Concentração, Dedicação e Disparidade

Em (WONG; GOKHALE; HORGAN, 2000) foram apresentadas três métricas para medição de interesses: Concentração, Dedicação e Disparidade. Tais métricas tem como base os conceitos de *feature* e blocos. De acordo com os autores, uma *feature* representa a descrição de uma funcionalidade realizada pelo sistema a partir de uma determinada entrada. Enquanto que um bloco é uma sequência de comandos consecutivos ou expressões que não realizem a transferência de controle para outro ponto do sistema.

As três métricas apresentadas por Wong *et al.* (2000) são complementares entre si. Seja *F* uma *feature* do sistema e *C* um componente:

- a Disparidade ($DISP_{CF}$) mede a quantidade de blocos relacionados a *F* que estão localizados em *C*. Portanto, esta métrica informa quantitativamente o grau de aproximação de uma *feature* para um componente. Quanto menor for este grau de aproximação, maior será a disparidade;

- a Concentração ($CONC_{FC}$) mede a proporção em que os blocos de *C* coincidem com os blocos que realizam *F*.

- a Dedicação ($DEDI_{CF}$) mede a proporção em que os blocos de *C* estão dedicados à implementação de *F* em relação ao conjunto de todos os blocos de C.

A figura 3.1 (adaptada de (WONG; GOKHALE; HORGAN, 2000)) traz exemplos considerando um componente *C* e uma *feature F* hipotéticos. Os círculos representam conjuntos de blocos, sendo B_C o conjunto dos blocos do componente *C*, e B_F o conjunto de blocos envolvidos na implementação da *feature F*. Os cinco casos ilustrados na figura demonstram as possíveis situações de relacionamento entre as métricas descritas.

No caso I da figura 3.1, a interseção entre os conjuntos B_C e B_F é nula. Portanto, a disparidade entre *C* e *F* é máxima, consequentemente a concetração de *F* em *C* é zero, bem como a dedicação de *C* para *F*.

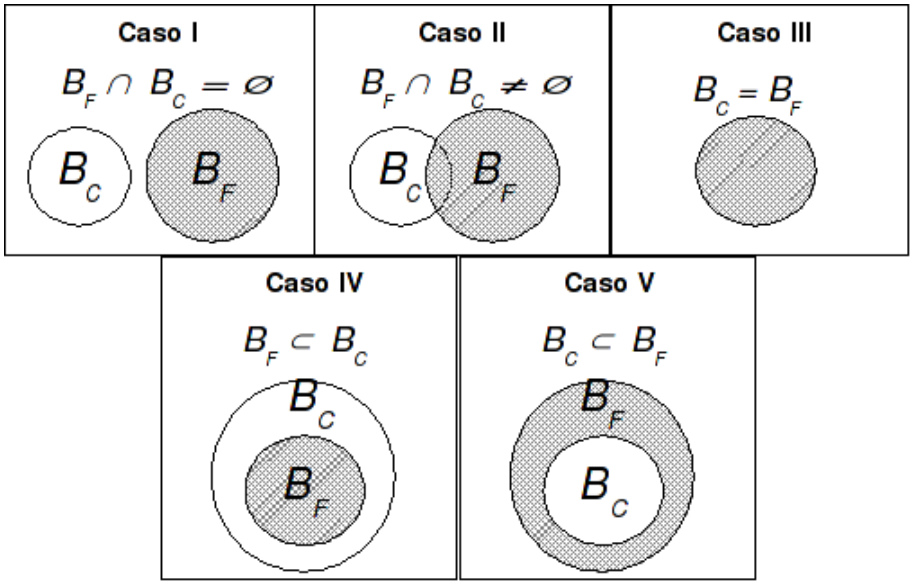


Figura 3.1: Possíveis relacionamenteos entre blocos de um componente e uma *feature*

Já no caso II, a interseção entre os blocos do componente e da *feature* não é nula. Isto significa que as métricas assumem valores intermediários entre 0 e o máximo (a depender da escala, por exemplo 0 a 1, 0 a 10, 0 a 100 etc). Este segundo caso permite observar que há um certo grau de disparidade que depende da interseção entre B_C e B_F. Da mesma maneira, percebe-se que *C* dedica uma parte de seus blocos à implementação de *F* (Dedicação), assim como alguns dos blocos que realizam *F* encontram-se localizados em *C* (Concentração).

No caso III, ocorre a igualdade dos conjuntos B_C e B_F, ou seja, *C* está completamente dedicado à realização de *F*, assim como, *F* é totalmente implementado por *C*. Diante disso, a disparidade entre *C* e *F* ($DISP_{CF}$) é zero.

No quarto caso, tem-se que B_F é um subconjunto de B_C e $B_F \neq B_C$ (isto é, $B_F \subset B_C$). Diante disso, observa-se que a concentração de *F* em *C* é máxima, porém a disparidade e a dedicação assumem valores intermediários entre 0 e o máximo. De fato, *C* está parcialmente dedicado a *F*.

Por fim, o quinto caso exemplifica uma situação semelhante, porém com a diferença de que B_C é um subconjunto de B_F (isto é, $B_F \subset B_C$). Portanto tem-se que $DEDI_{CF}$ é máxima, enquanto que a disparidade e a concentração assumem valores intermediários entre zero e o máximo. De fato, nota-se que *C* está completamente dedicado à implementação de *F*, porém existem blocos de *F* que não encontram-se localizados em C.

3.2 DOS e DOF

Eaddy *et al.* (EADDY; AHO; MURPHY, 2007) apresentaram uma extensão das métricas de Dedicação e Concentração previamente propostas em (WONG; GOKHALE; HORGAN, 2000). Enquanto a definição original das métricas envolve as noções de blocos e *features*, Eaddy *et al.* as substituem por linhas de código (LOC - *Lines of Code*) e interesses, respectivamente. Portanto, a Concentração de um interesse *s* em um componente *t*, assim como a Dedicação de um componente *t* para um interesse *s*, passam a ser definidas, respectivamente, como segue:

$$CONC(s,t) = \frac{LOC\ no\ componente\ t\ relacionado\ ao\ interesse\ s}{LOC\ relacionadas\ ao\ interesse\ s}$$

$$DEDI(t,s) = \frac{LOC\ no\ componente\ t\ relacionado\ ao\ interesse\ s}{LOC\ no\ componente\ t}$$

Adicionalmente, os autores propõem duas novas métricas, derivadas a partir dessas novas definições, que são: DOS (*Degree of Scattering*) e DOF (*Degree of Focus*). DOS mede o grau de espalhamento de um interesse entre os componentes do sistema, enquanto DOF mede o grau de proximidade de um componente em relação aos interesses do sistema, isto é, quão bem separados em componentes estão os interesses em questão. Tais métricas são representadas, respectivamente, pelas seguintes equações:

$$DOS(s) = 1 - \frac{|T|\ \sum_{t}^{T}(CONC(s,t) - \frac{1}{|T|})^2}{|T|-1}$$

$$DOF(t) = \frac{|S|\ \Sigma_{s}^{S}(DEDI(t,s) - \frac{1}{|S|})^2}{|S|-1}$$

Onde: T é o conjunto de componentes, sendo $|T| > 1$; S é o conjunto de interesses, sendo $|S| > 1$; t é um componente do sistema ($t \epsilon T$); e s é um interesse do sistema ($s \epsilon S$).

O grau de espalhamento de um interesse, $DOS(s)$, tem as seguintes propriedades:

- é normalizado entre 0 (interesse completamente localizado) e 1 (interesse uniformemente espalhado nos componentes);
- é diretamente proporcional ao número de componentes relacionados ao interesse;
- é inversamente proporcional à concentração, isto é, quanto menos concentrado é o interesse, maior será o seu espalhamento.

O grau de proximidade de um componente em relação aos interesses, $DOF(t)$, tem as seguintes propriedades:

- é normalizado entre 0 (completamente desfocado, isto é, componente com múltiplos interesses) e 1 (completamente focado, isto é, componente totalmente dedicado ao interesse que encontra-se totalmente concentrado no mesmo);
- é inversamente proporcional ao número de interesses relacionados ao componente;
- é diretamente proporcional à dedicação, isto é, quanto mais uniformemente dividida está a implementação do componente entre os interesses do sistema (menor dedicação), menor será o seu foco.

Eaddy *et al.* (2007) concluem que a média dos resultados de DOS de todos os interesses do sistema é inversamente proporcional ao grau de modularização do mesmo, enquanto que a média dos resultados de DOF de todos os componentes do sistema é diretamente proporcional. Dessa maneira, é desejável um baixo valor na média de DOS's e um alto valor na média de DOF's. Por fim, deve-se observar o nível de granularidade dos interesses e seus componentes. Valores DOS, assim como DOF, são comparáveis considerando-se componentes e interesses em um mesmo nível de granularidade.

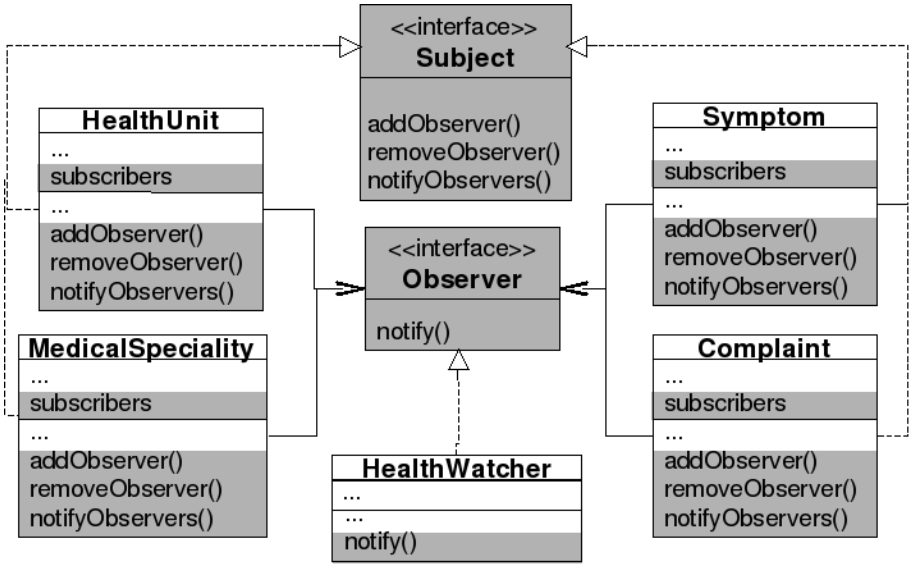


Figura 3.2: Padrão de projeto *Observer* como um interesse

3.3 CDC, CDO e CDLOC

Sant'Anna *et al.* (2003) definiram um conjunto de métricas para avaliar quantitativamente a coesão, acoplamento, tamanho e separação de interesses em sistemas. Desse conjunto, três métricas foram definidas a fim de verificar o grau de separação de um determinado interesse. Tais métricas quantificam a difusão de um interesse sobre componentes, operações e linhas de código.

CDC (*Concern Diffusion over Components*) e CDO (*Concern Diffusion over Operations*) medem o grau de espalhamento em diferentes níveis de granularidade, podendo ser aplicadas tanto em modelos quanto em código. CDC conta o número de componentes que estão envolvidos na participação de um interesse. Enquanto o CDO conta o número de métodos (no caso de classes e interfaces) e adendos (no caso de aspectos) relacionados à realização de um interesse.

O diagrama de classes da figura 3.2 demonstra uma instância do padrão de projeto *Observer* como exemplo de um interesse transversal retirado de um sistema real chamado *Health Watcher* utilizado previamente em estudos de caso como em (GREENWOOD et al., 2007). As regiões sombreadas indicam as partes dos componentes que estão envolvidas na realização do interesse, que encontram-se misturadas com as demais intenções de cada componente (retiradas por questões de simplicidade do exemplo).

De acordo com o exemplo da figura 3.2, nota-se que o comportamento relacionado ao padrão Observer encontra-se espalhado sobre sete componentes (CDC = 7) e dezessete operações (CDO = 17). Tais métricas (CDC e CDO) também foram aplicadas em especificações de arquiteturas deste software (SANT'ANNA et al., 2007).

Já a métrica CDLOC (*Concern Diffusion over Lines of Code*) restringe-se ao nível de código, pois mede o grau de entrelaçamento de um interesse em determinado componente, contando a alternância em que trechos de código se dedicam à realização de um interesse. A figura 3.3 demonstra partes do código da classe *HealthUnit* encontrada no exemplo da figura 3.2. Nesta classe há oito alternâncias (CDLOC = 8) entre trechos de código dedicados à implementação do interesse *Observer* e demais trechos dedicados a outras intenções.

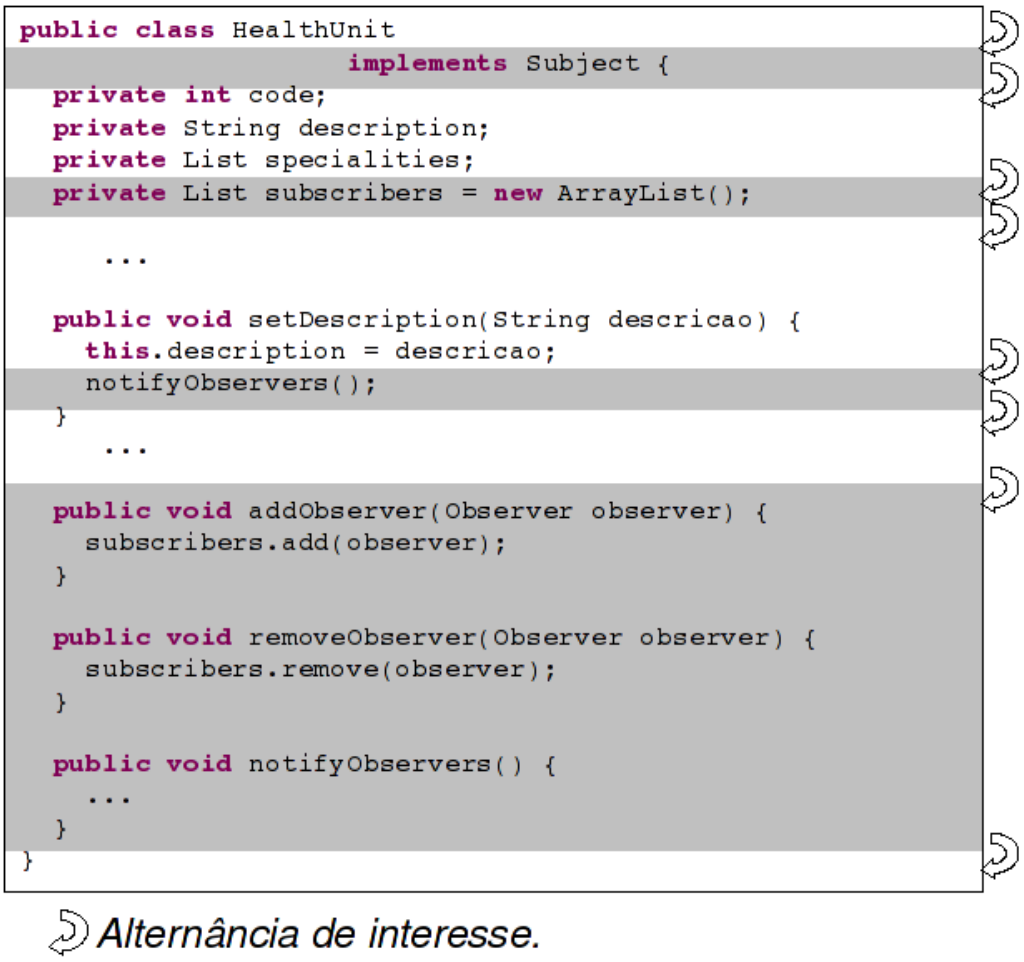


Figura 3.3: Alternância de interesse na classe *HealthUnit*

3.4 CDA

Ceccato e Tonella (2004) definiram um conjunto de métricas para aplicação em sistemas de software orientado a aspectos, sendo CDA (*Crosscutting Degree of an Aspect*) considerada como orientada a interesses neste documento. A métrica CDA (*Crosscutting Degree of an Aspect*) conta o número de componentes afetados por conjuntos de pontos de junção e por introduções de um determinado aspecto. O objetivo desta métrica é medir o impacto que um aspecto produz sobre outros componentes do sistema.

A figura 3.4 representa uma possível solução do exemplo do padrão de projeto *Observer* utilizando aspecto (versão OO inicialmente apresentada na figura 3.2). Nesta ilustração pode-se observar que as interfaces puderam ser movidas para o aspecto criado (*HealthWatcherObserver*). Adendos, conjuntos de pontos de junção e declarações intertipos dentro do aspecto ficam responsáveis por adicionar membros e comportamentos relacionados à participação do interesse, isto é, a realização do padrão *Observer*. Tais interferências são representadas graficamente pelo relacionamento com o estereótipo *Crosscuts*. Sendo assim, o aspecto *HealthWatcherObserver*, criado para realização do padrão *Observer* (na versão em aspectos), afeta cinco classes. Portanto, a métrica CDA aplicada a este exemplo possui valor 5.

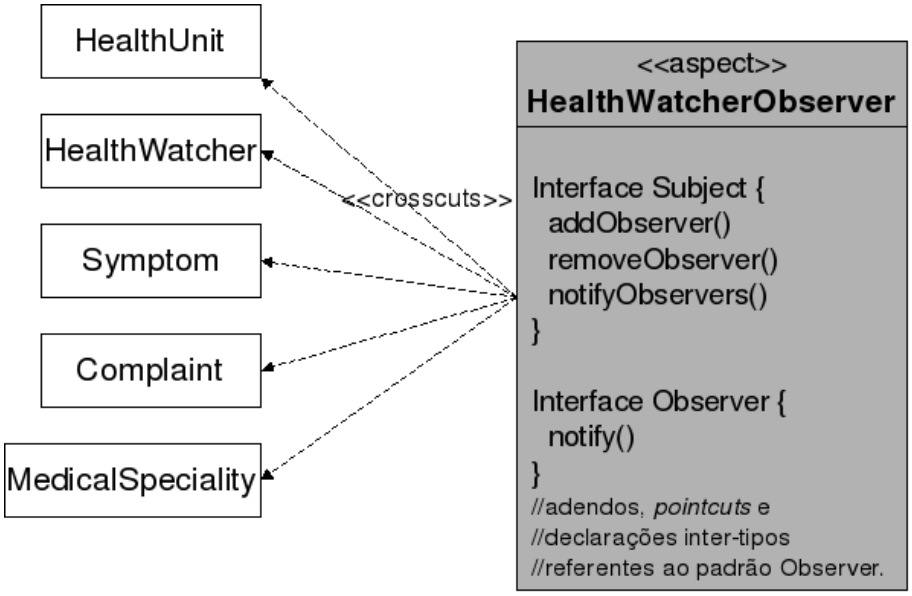


Figura 3.4: Representação do Padrão *Observer* como aspecto

3.5 NOF, FCD e ACD

Lopez *et al.* (2007) definiram, de maneira semi-formal, um conjunto de métricas para aplicação tanto em sistemas orientados as aspectos quanto orientados a objetos. Deste conjunto, as métricas orientadas a interesses são: NOF (*Number of Features*), FCD (*Feature Crosscutting Degree*) e ACD (*Advice Crosscutting Degree*). Apesar de os autores usarem a palavra *feature* – termo comum da área de Linha de Produto de Software – é possível interpretá-la também como interesse (FIGUEIREDO; CACHO, 2008).

A métrica NOF mede simplesmente o número de *features* encontrados em um sistema. FCD mede o número de classes que são afetadas por todos os adendos de uma determinada *feature*, bem como por suas declarações intertipos. Por fim, ACD mede o número de classes afetadas exclusivamente por um adendo de uma determinada *feature*.

De acordo com o exemplo da figura 3.4, a métrica FCD teria valor 5, pois os adendos do aspecto afetam cinco classes. Neste caso as métricas FCD e CDA (seção 3.4) possuem interpretações equivalentes, pois o interesse é implementado em apenas um aspecto. Logo, tanto o aspecto quanto o interesse afetam cinco classes (*HealthUnit*, *Symptom*, *Complaint*, *MedicalSpeciality* e *HealthWatcher*).

Embora as definições das métricas CDA (seção 3.4) e FCD sejam voltadas para aspectos, é possível também que ambas sejam estendidas para sistemas orientados a objetos. Caso os termos “aspecto” da definição da métrica CDA e “feature” da métrica FCD sejam interpretados como interesse transversal em sistemas OO, tais métricas podem ser equivalentes. Por exemplo, considerando-se o padrão *Observer* na versão OO como apresentado de início na figura 3.2, ao aplicar tal interpretação, teria-se o valor 7 em ambas métricas CDA e FCD. Adicionalmente, é possível observar também que a métrica CDC (seção 3.3) é equivalente a CDA e FCD quando aplicada a sistemas OO.

Já em relação à métrica ACD, deve-se observar a quantidade de classes afetadas por um adendo. Voltando ao exemplo do *Observer*, a figura 3.5 destaca um trecho de código relacionado a um adendo retirado do aspecto *HealthWatcherObserver* (ilustrado inicialmente na figura 3.4). Esta porção de código possui quatro conjuntos de junção responsáveis por capturar os pontos em que ocorrem mudança de estado nos objetos observados e

```
public aspect HealthWatcherObserver {
  ...
  pointcut HealthUnitDescriptionChange(Subject subject):
     call(void HealthUnit.setDescription(String)) &&
   target(subject);

  pointcut SymptomDescriptionChange(Subject subject):
     call(void Symptom.setDescription(String)) &&
target(subject);

  pointcut ComplaintChange(Subject subject):
     ( call(void Complaint.setComplaintState(ComplaintState)) ||
       call(void Complaint.setAttendant(Employee))||
       call(void Complaint.setObservation(String))||
       call(void Complaint.setStatus(int)) ) &&
       target(subject);

  pointcut MedicalSpcialityDescriptionChange(Subject subject):
     call(void HealthSpeciality.setDescription(String)) &&
         target(subject);

  after(Subject subject):HealthUnitDescriptionChange(subject) ||
     SymptomDescriptionChange(subject) ||
     ComplaintChange(subject)          ||
     MedicalSpcialityDescriptionChange(subject)
     {
         subject.notifyObservers();
     }
}
```

Figura 3.5: Adendo e conjuntos de junção no aspecto *HealthWatcherObserver*

que então devem notificar os observadores por isso. O comportamento para notificar os observadores é adicionado pelo adendo em questão através da chamada do método *notifyObservers()*. Portanto, de acordo com este exemplo, a métrica ACD possui valor 4, pois o adendo mostrado na figura afeta quatro classes (*HealthUnit*, *Symptom*, *Complaint* e *MedicalSpeciality*).

3.6 *Size*, *Touch*, *Spread* e *Focus*

Ducasse *et al.* (2006) apresentaram uma técnica chamada *Distribution Map* para analisar propriedades de sistemas. Esta técnica é genérica na medida em que pode ser instanciada e aplicada em diversas situações, especialmente na medição de interesses transversais em sistemas de software.

A proposta dos autores analisa abstratamente propriedades de sistemas definindo dois conceitos importantes: *Reference Partition* (ou Partição de Referência) e *Comparison Partition* (ou Partição de Comparação). Partição de Referência consiste em uma parte da estrutura intrínseca do sistema (por exemplo, pacotes, classes, interfaces etc.), enquanto que a Partição de Comparação representa o resultado de uma análise, que na maioria das vezes são propriedades mutuamente exclusivas associadas a elementos do sistema. A partir desses conceitos, os autores apresentam quatro métricas orientadas a interesses: *Size*, que conta o número de subpartes em partições de referência associadas a uma partição de comparação; *Touch*, que define a proporção entre o valor da métrica *Size* e o número de subpartes encontradas nas partições de referência do sistema; *Spread*, conta o número

de partições de referência que é tocada por uma determinada propriedade; e, por fim, *Focus*, que mede a proximidade entre a partição de referência e a partição de comparação, isto é, quanto maior o número, maior serão as partes tocadas pela propriedade que são inteiramente tocadas por ela.

Como os autores apresentam os conceitos de partições de maneira abstrata, torna-se possível instanciá-los da seguinte forma: partições de referências podem ser consideradas como componentes de desenho, classes e interfaces, onde suas partes correspondem a operações e atributos; enquanto a partição de comparação pode ser interpretada como interesses do sistema alvo. A figura 3.6 representa o exemplo previamente apresentado na figura 3.2 de acordo com a abordagem *Distribution Map*. A partição de comparação é o padrão de projeto *Observer*. As áreas dos retângulos maiores representam as partições de referência envolvidas na realização do padrão, sendo duas interfaces (de nomes *Subject* e *Observer*) e cinco classes (de nomes *Complaint*, *HealthUnit*, *HealthWatcher*, *MedicalSpeciality* e *Symptom*). Os quadrados menores representam subpartes de cada partição de referência (métodos e atributos), sendo os preenchidos aqueles que fazem parte da realização do padrão.

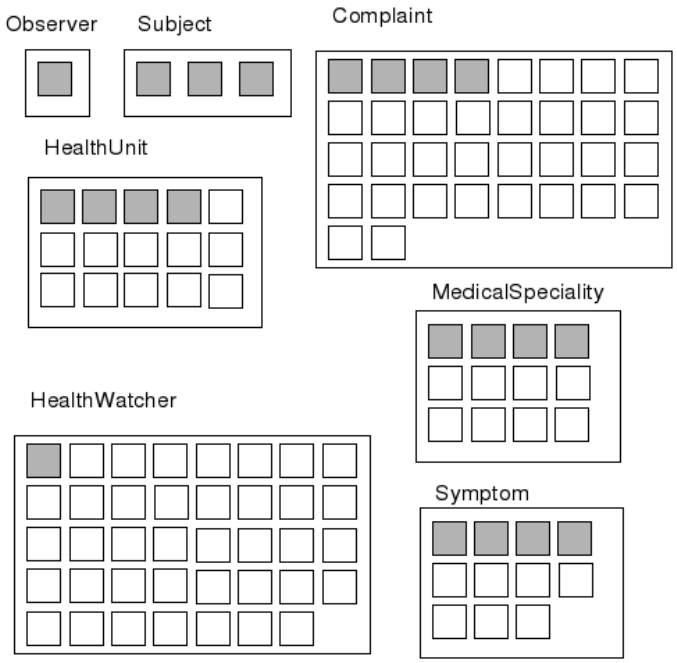


Figura 3.6: Visualizando a propriedade *Observer* através do *Distribution Map*

De acordo com a definição da métrica *Size*, tem-se no exemplo (figura 3.6) que o valor desta medição é 21, pois existem vinte e um membros (métodos e atributos), ou seja quadrados preenchidos, distribuídos entre os componentes envolvidos na realização do padrão *Observer*. Pela métrica *Touch*, tem-se como resultado o valor 0.22 como sendo o quociente entre o valor do *Size* (21) e o número total de subpartes ilustradas nas partições de referência da figura (94). Já de acordo com a métrica *Spread*, obtem-se como resultado o valor 7, pois a realização do padrão *Observer*, neste exemplo, está espalhado em sete componentes como partições de referência. Observa-se que a métrica *Spread* é equivalente à métrica CDC (seção 3.3) proposta por (SANT'ANNA et al., 2003), tendo também valor 7.

A métrica *Focus* está fundamentada em dois cálculos, descritos a seguir:

- a relação entre a quantidade de membros de um componente dedicados a um interesse com a quantidade total de membros do componente;

- a relação entre a quantidade de membros de um componente dedicados a um interesse com a quantidade total de membros (de todos os componentes) que participam da realização do interesse.

A métrica *Focus* consiste num somatório desses dois cálculos considerando cada componente envolvido na realização do interesse. De acordo com o exemplo, tem-se o valor 0.39 para o interesse *Observer*. A demonstração do seu cálculo não é trivial como as demais métricas e, portanto, vai além do escopo do presente trabalho, embora possa ser obtida em (DUCASSE; GIRBA; KUHN, 2006).

Por fim, vale observar que interesses bem encapsulados tem um valor alto de *Focus* enquanto interesses transversais possuem um baixo valor. No exemplo considerado, o interesse *Observer* está mais próximo de um baixo valor, como é possível notar o interesse não está bem encapsulado. Entretanto, a interpretação de tais valores como sendo alto ou baixo é subjetiva e depende da consideração e experiência de cada projetista. Por exemplo, em determinado projeto o valor 0.39 da métrica *Focus* poderia ser concebido como um nível aceitável de encapsulamento, enquanto em outros projetos isto seria considerado como um interesse altamente transversalizado.

3.7 Discussão sobre as Métricas para Medição de Interesses

A partir do levantamento e estudo bibliográfico sobre as propostas de medição de interessses foi possível obter um conhecimento abrangente e aprofundado sobre o estado-da-arte relacionado à utilização de métricas para este tipo de medição. A tabela 3.1 organiza de maneira resumida informações relevantes observadas através do estudo e está disposta da seguinte forma: as duas primeiras colunas, respectivamente, identificam a métrica a partir do seu nome e sua sigla (caso exista); a terceira coluna refere-se à seção onde ela foi apresentada neste capítulo; a quarta e a quinta colunas dizem respeito às diferentes terminologias utilizadas pelos autores das propostas respectivamente para os termos unidade modular e interesse; a penúltima coluna informa o alvo da medição, isto é, que tipo de elemento do sistema é mensurado; por fim, a última coluna diz respeito ao nível de abstração em que é possível aplicar a métrica.

Tomando como exemplo as métricas de Sant'anna *et al.* (2003) apresentadas na seção 3.3, é possível ter a seguinte leitura da tabela: a métrica CDC tem como alvo de medição componentes, pois mede a difusão do interesse sobre os componentes do sistema, enquanto a métrica CDO mede a difusão do interesse sobre operações, tendo assim um alvo diferente de medição. Entretanto, ambas as métricas podem ser aplicadas tanto no nível de abstração de implementação quanto em modelos que contenham informações sobre operações de seus componentes. Já a métrica CDLOC, tem como alvo de medição linhas de código, isto é, um nível de abstração muito baixo, podendo ser aplicada apenas em código de implementação.

As métricas de Wong *et al.* (2000) (seção 3.1), dispostas nas três primeiras linhas da tabela 3.1, podem ser aplicadas apenas no nível de implementação, pois os alvos de medição considerados pelos autores são os blocos de codificação dos componentes.

As métricas *Size* e *Touch* (seção 3.6), consideram como alvo de medição os membros das partições, isto é, atributos e operações dos componentes. Neste caso, tanto faz se é um atributo ou uma operação que participa da realização do interesse.

De maneira geral, as métricas que não necessitam entrar no detalhe sobre linhas de código e blocos de implementação de componentes podem ser aplicadas tanto no nível de implementação quanto no nível de modelos.

Entretanto, é possível observar algumas limitações relacionadas à diversidade de propostas consideradas nesse estudo. Primeiramente, nota-se que diferentes termos são utilizados pelos autores das propostas para denotar elementos equivalentes como alvo de medição. Por exemplo, o termo *bloco de componente* considerado nas métricas de Concentração, Dedicação e Disparidade (seção 3.1) é equivalente ao termo *linhas de código* considerado nas métricas DOF e DOS (seção 3.2) e CDLOC (seção 3.3). O termo *classes* considerado nas métricas FCD e ACD (seção 3.5) é equivalente ao termo *componentes* utilizado na métrica CDC (seção 3.3) e ao termo *partição* utilizado nas métricas *Spread* e *Focus*. Tal confusão entre os termos dificulta o entendimento sobre como diferentes métricas se relacionam.

Além disso, Figueiredo *et al.* (2008) destaca a inconsistência entre a terminologia utilizada para os conceitos de *interesse* e *unidade modular*. Por exemplo, voltando-se para a quarta coluna da tabela 3.1, é possível observar que a terminologia para unidade modular é chamada de *componente*, *módulo*, *partição*, *classe*, *interface* e *aspecto* pelas diferentes métricas. E ainda, de acordo com a coluna 5, percebe-se que a terminologia para o conceito de interesse é chamado de maneiras distintas como sendo: *feature*, o próprio termo *interesse* e *propriedade*. A maneira não-uniforme em que as métricas são definidas resulta em uma falta de padronização de tais terminologias, reforçando a dificuldade sobre o entendimento de como métricas distintas relacionam-se. Muitas métricas são expressas de forma ambígua, limitando assim a sua utilização.

Em (FIGUEIREDO et al., 2008) são destacados mais dois problemas relacionados às propostas apresentadas neste capítulo. O primeiro refere-se à inexistência de métricas que cubram todas as propriedades de modularização associadas a um ou mais interesses. As métricas apresentadas neste capítulo, de maneira geral, têm como objetivo quantificar quatro propriedades referentes à modularização de interesses: espalhamento, entrelaçamento, tamanho e proximidade. Por exemplo: as métricas da seção 3.1 medem a proximidade que um elemento de desenho está em relação à participação na realização de um interesse; enquanto que a métrica NOF (seção 3.5) quantifica o tamanho do conjunto de interesses do sistema. Entretanto, apesar do reconhecimento de que coesão e acoplamento desempenham papel fundamental na manutenibilidade de um sistema (BRIAND; DALY; WüST, 1998) (BRIAND; DALY; WüST, 1999) (CHIDAMBER; KEMERER, 1994) (GREENWOOD et al., 2007), as métricas levantadas neste estudo não quantificam tais propriedades quanto à modularização de interesses. Os graus de coesão e acoplamento impactados por um interesse foram identificados como fatores que influenciam no sucesso da modularização de interesses transversais através do uso de aspectos em determinados sistemas (CACHO et al., 2006) (FILHO et al., 2006) (GARCIA et al., 2006) (GREENWOOD et al., 2007).

Por fim, os autores destacam que há, sobre as métricas existentes para interesses, uma sobreposição de objetivos de medição. Diversas decisões devem ser tomadas na definição de uma métrica, dentre elas (e principalmente) o objetivo do que se pretende medir. No entanto, percebe-se que tais decisões não encontram-se documentadas (ou pelo menos disponíveis na literatura). Consequentemente, em alguns casos não fica claro qual o potecial uso da métrica e como diferentes métricas podem ser utilizadas de maneira complementar. Por exemplo, as definições das métricas CDC (seção 3.3), CDA (seção 3.4), FCD (seção 3.5) e *Spread* (seção 3.6) são similares (no exemplo apresentado do padrão

Observer elas possuem valor 7). Contudo, sem um conjunto de critérios para comparar tais métricas, objetivos similares de medição tornam-se difíceis de serem identificados. E ainda, esta característica limita a capacidade de replicar, de maneira confiável, estudos empíricos que utilizam tais métricas.

No mesmo trabalho, os autores propõem um *framework* conceitual para medição de interesses motivado pelas limitações acerca do estado-da-arte relacionado à medição de interesses discutidos nesta seção. Tal proposta inclui a definição de uma estrutura criteriosa e de uma terminologia padrão que servem como base para abordagens mais precisas na definição de métricas orientadas a interesses.

3.8 Considerações Finais do Capítulo

Neste capítulo foram apresentadas métricas para avaliação quantitativa levantadas a partir de trabalhos relacionados à medição de interesses. Através deste estudo bibliográfico foi possível não só apresentar mas discutir também algumas características importantes dos trabalhos. O conhecimento destas métricas é relevante para uma análise objetiva e quantitativa sobre o grau de separação de interesses e modularização de sistemas de software. Além disso, o uso de métricas voltadas para interesses permite também a reunião de dados importantes para uma análise qualitativa. No capítulo que segue, apresenta-se os sintomas sensíveis a interesses, que representam algumas das diversas formas que interesses transversais podem se encontrar nos sistemas, tomados como oportunidades para refatoração.

Tabela 3.1: Sumário das métricas para interesses

Métrica	**Sigla**	**Seção**	**Terminologia para Unidade Modular**	**Terminologia para Interesse**	**Alvo da Medição**	**Nível de Aplicabilidade**
Concentração	-	3.1	Componente	Feature	Blocos de componentes	Implementação
Dedicação	-	3.1	Componente	Feature	Blocos de componentes	Implementação
Disparidade	-	3.1	Componente	Feature	Blocos de componentes	Implementação
Degree of Focus	DOF	3.2	Componente	Interesse	Linhas de código e componentes	Implementação
Degree of Scattered	DOS	3.2	Componente	Interesse	Linhas de código e componentes	Implementação
Crosscutting Degree over Components	CDC	3.3	Componente	Interesse	Componentes	Modelos e implementação
Crosscutting Degree over Operations	CDO	3.3	Componente	Interesse	Operações	Modelos e implementação
Crosscutting Degree over Lines of Code	CDLOC	3.3	Componente	Interesse	Linhas de código	Implementação
Crosscutting Degree of an Aspect	CDA	3.4	Módulo	Interesse	Aspectos (Conjuntos de junçao e introduções)	Modelos e Implementação
Number of Features	NOF	3.5	Classe, interface e aspecto	Feature	Features	Modelos e Implementação
Feature Crosscutting Degree	FCD	3.5	Classe, interface e aspecto	Feature	Features	Modelos e Implementação
Advice Crosscutting Degree	ACD	3.5	Classe, interface e aspecto	Feature	Features	Modelos e Implementação
Size	-	3.6	Partição	Propriedade	Membros de partições	Modelos e implementação
Touch	-	3.6	Partição	Propriedade	Membros de partições	Modelos e implementação
Spread	-	3.6	Partição	Propriedade	Membros de partições	Modelos e implementação
Focus	-	3.6	Partição	Propriedade	Membros de partições	Modelos e implementação

4 INTERESSES TRANSVERSAIS COMO SINTOMAS

Este capítulo tem como objetivo apresentar sintomas relacionados a interesses transversais vistos como oportunidades de melhoria na modularização do desenho do software. Tais sintomas têm sido definidos e explorados através de estudos empíricos em cooperação com pesquisadores da Universidade de Lancaster (Reino Unido) (SILVA et al., 2009) (FIGUEIREDO et al., 2009) e são considerados como motivadores para as refatorações propostas no capítulo 5.

Embora a proposição de um catálogo de sintomas sensíveis ao interesse não seja objetivo do presente trabalho, optou-se por explorar com detalhes neste capítulo a explicação dos sintomas, pois isto é parte integrante do método de refatoração proposto no contexto desta dissertação. Portanto, isto é requisito essencial no entendimento das refatorações para modularização de interesses transversais no capítulo 5, deixando assim o texto autocontido.

Inicialmente, uma breve discussão é realizada na seção 4.1 sobre os sintomas convencionais que não são sensíveis ao interesse, isto é, são sintomas que envolvem as construções programáticas sem focar na configuração estrutural do interesse transversal como um todo. Já a seção 4.2 concentra-se na apresentação dos sintomas sensíveis ao interesse inicialmente definidos em (FIGUEIREDO et al., 2009), enquanto que a seção 4.3 inclui as heurísticas de identificação para os mesmos; por fim, a seção 4.4 realiza o fechamento deste capítulo.

4.1 Sintomas Convencionais

A identificação de alguns sintomas oferece indicativos de que algo indesejável ocorre no desenho do software. Em um trabalho seminal (FOWLER et al., 1999), Kent Beck e Fowler propuseram um catálogo que inclui a descrição de 22 sintomas prejudiciais para desenho de software OO. Muitos desses sintomas são universais e podem também ser relevantes para identificação de problemas em desenhos de sistemas OA. Baseado no livro de Fowler, Monteiro e Fernandes (2006) apresentaram 3 sintomas tanto para desenhos OO quanto para OA, que são: Dupla Personalidade (*Double Personality*), Classes Abstratas (*Abstract Classes*) e Preguiça de Aspecto (*Aspect Laziness*).

Adicionalmente, Piveta *et al.* (2006) apresentam sintomas exclusivos para programas em AspectJ, tais como: Definição Anônima de Conjunto de Junção (*Anonymous Pointcut Definition*), Aspecto Extenso (*Large Aspect*), Aspecto Preguiçoso (*Lazy Aspect*), Introdução de Método Abstrato (*Abstract Method Introduction*) e *Feature Envy*. Os autores desenvolveram também uma ferramenta para detecção automática de tais sintomas. Posteriormente (PIVETA et al., 2007), Piveta e colegas propuseram um guia para evitar a ocorrência destes sintomas prejudiciais durante o desenvolvimento de software OA.

No contexto deste trabalho, tais sintomas são chamados de convencionais, pois, independente de serem OO ou OA, lidam diretamente e exclusivamente com construções programáticas específicas como, por exemplo, relacionados à estrutura geral de componentes (classe/aspecto preguiçoso, classe/aspecto extenso etc.), ou relacionados a membros internos de componentes (duplicação de código, método longo, *switch statements*, definição anônima de conjunto de junção, entre outros). Dessa maneira, os sintomas convencionais não encontram-se voltados para a manifestação ou configuração de um interesse transversal como um todo. Eles estão focados apenas em características peculiares de código independente do contexto de um interesse transversal em que possam eventualmente estar inseridos.

Ao utilizar a abordagem de Estratégias de Detecção propostas por Marinescu (2004) é possível verificar algumas limitações relacionadas aos sintomas convencionais. Como define Marinescu (2004), uma Estratégia de Detecção *"é a expressão quantificável de uma regra pela qual fragmentos de desenho que estão em conformidade com a regra podem ser detectados em código fonte"*. As estratégias de detecção estão centradas no uso de métricas tradicionais de modularização para quantificar os atributos de qualidade do desenho. Por exemplo, uma de suas estratégias tem como intenção a deteção do sintoma *Shotgun Surgery* (FOWLER et al., 1999). Este sintoma ocorre quando uma mudança em alguma característica do sistema implica modificações em outros lugares também. Quando tais mudanças se encontram distribuídas em diversos lugares do sistema, elas se tornam difíceis de achar e também fáceis de omitir alguma modificação importante. A regra definida abaixo é uma estratégia para detectar se um determinado componente do sistema possui o sintoma *Shotgun Surgery*.

$$ShotgunSurgery := ((CM,\ TopValues(20\%))\ \textbf{and}\ (CC, HigherThan(5)))$$

A regra de Marinescu definida acima é baseada em duas métricas de acoplamento chamadas de CM (do inglês *Changing Method*) e CC (do inglês *Changing Classes*). CM (LANZA; MARINESCU; DUCASSE, 2005) conta o número de métodos distintos que acessam um atributo ou chamam um método de uma determinada classe, enquanto CC (LANZA; MARINESCU; DUCASSE, 2005) conta o número de classes que acessam um atributo ou chamam um método também para uma determinada classe. *TopValues* e *HigherThan* são mecanismos de filtro que podem ser parametrizados com um valor representando um limite. Por exemplo, a regra acima declara que, para ser detectado como *Shotgun Surgery*, um componente do desenho deve estar entre os 20% que possuem os maiores valores na medição de CM e o valor da medição CC deve ser maior que 5. A escolha do *Shotgun Surgery* como exemplo ilustrativo desta seção deve-se ao fato de este sintoma estar associado a componentes que possuem modularizações incorretas de interesses transversais, como destaca Monteiro e Fernandes (2006).

A figura 4.1 demonstra um recorte do desenho de um sistema *web* chamado *Health Watcher* (SOARES; LAUREANO; BORBA, 2002) (explicado com mais detalhes no capítulo 7). A figura destaca em cinza os elementos responsáveis pela implementação do padrão de projeto *Observer* (GAMMA et al., 1995). Aplicando a métrica CC para esta instância do *Observer*, obtem-se $CC = 0$ para a interface *Subject*. De acordo com esse valor e computando a regra de *Shotgun Surgery* definida anteriormente, verifica-se apenas usando a medição CC que esta interface não indicaria tal sintoma. Isto ocorre porque CC é 0, já que nenhuma classe do sistema acessa diretamente a interface *Subject*. Entretanto, o sintoma *Shotgun Surgery* poderia ser claramente identificado, pois mudanças nos métodos da interface *Subject* demandariam diversas outras mudanças nos elementos que

realizam o interesse *Observer*. Por exemplo, ao renomear o método *addObserver()* na interface *Subject*, necessitam-se atualizações nas classes *Complaint*, *Employee*, *HealthUnit* e em diversas outras classes que eventualmente chamam o método *addObserver()*.

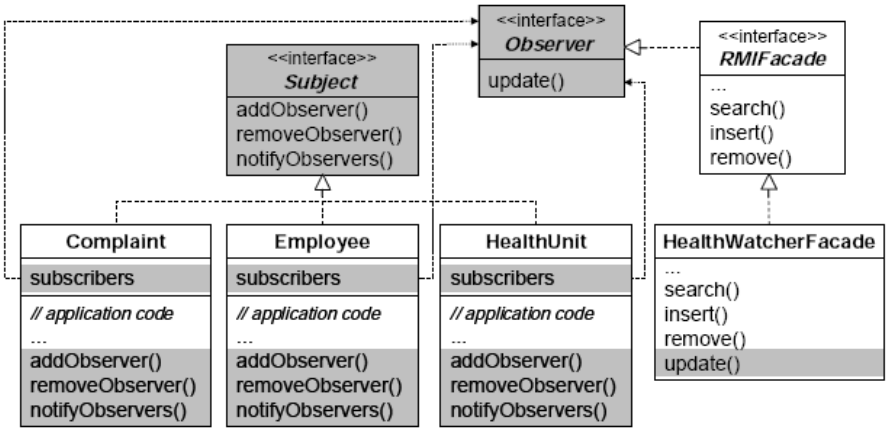


Figura 4.1: Um exemplo de desenho mostrando o padrão *Observer*

Tal situação não aponta que a regra de Marinescu está incorreta. Entretanto, ela não foi capaz de detectar que diversas classes estão relacionadas ao interesse do padrão *Observer*, pois de fato este sintoma é convencional e não é sensível ao interesse. Portanto, a regra não deve capturar um problema que está relacionado à manutenção de um interesse transversal. Isto é, a regra não consegue verificar que diversas classes seriam afetadas por mudanças relacionadas a este interesse. As limitações no uso de estratégias de detecção de sintomas convencionais vão além deste exemplo. Todas elas falham na captura de dependências de elementos do desenho causadas pela realização de interesses, pois os sintomas são, por definição, independentes do contexto de interesses no qual podem estar inseridos.

Nenhum dos sintomas encontrados em (FOWLER et al., 1999), (MONTEIRO; FERNANDES, 2006) e (PIVETA et al., 2006) está relacionado a interesses transversais. A detecção de sintomas sensíveis ao interesse é importante na medida em que indica oportunidades de refatoração que podem melhorar a modularização do sistema. Estudos empíricos recentes têm destacado que a presença de interesses transversais é um fator determinante para a ocorrência de sintomas prejudiciais no desenho do sistema e, consequentemente, em sua implementação (EADDY; AHO; MURPHY, 2007) (FILHO et al., 2006) (FIGUEIREDO et al., 2008) (EADDY et al., 2008). É nesse sentido que foi proposto em (FIGUEIREDO et al., 2009) um conjunto de sintomas sensíveis ao interesse, levando-se em consideração diferentes formas (ou manifestações) em que se pode configurar interesses transversais.

4.2 Sintomas Sensíveis ao Interesse

A iniciativa de identificação e classificação de interesses transversais não é nova. Alguns trabalhos propõem a categorização de interesses a partir das diversas possibilidades de configuração, como em (MARIN; MOONEN; DEURSEN, 2005b) e (DUCASSE; GIRBA; KUHN, 2006).

A classificação baseada em tipos de interesses transversais proposta por Marin *et al.* (2005b) está diretamente ligada a mecanismos específicos de linguagens de programação

OO ou OA. Enquanto que a proposta de Ducasse *et al.* (2006) revela-se mais genérica e independente de linguagens OO ou OA. Em seu trabalho, Ducasse *et al.* (2006) propôs a classificação de estruturas transversais utilizando metáforas (*Octopus* e *Black Sheep*). Adicionalmente, os autores destacam como lidar com tais estruturas pode ajudar nas atividades de manutenção como, por exemplo, a refatoração.

Esta seção dedica-se à apresentação de treze sintomas sensíveis ao interesse, nomeados através de metáforas. Dos treze sintomas, dois foram inicialmente propostos em (DUCASSE; GIRBA; KUHN, 2006) e os demais em (SILVA et al., 2009) e (FIGUEIREDO et al., 2009). O uso de metáforas traz uma representação simbólica através de um vocabulário intuitivo (KENDALL; KENDALL, 2003). Isto facilita o entendimento e ajuda na compreensão dos sintomas de acordo com as diferentes formas em que os interesses transversais podem se configurar.

Os sintomas são agrupados em quatro categorias. Tais categorias foram reunidas de acordo com diferentes maneiras que interesses podem se organizar: sintomas com formas transversais, sintomas relativos a herança, sintomas relativos a acoplamento e sintomas com outros problemas de modularidade. Vale ressaltar que os treze sintomas apresentados nesta seção remetem às estruturas transversais (estáticas), sem considerar, portanto, o comportamento dinâmico dos componentes participantes.

Este conjunto de sintomas visa à cobertura das diversas possibilidades que interesses podem se manifestar no desenho do software, no entanto não impede que existam outros tipos de manifestações. Em cada subseção, os sintomas são apresentados seguindo um formato contendo os seguintes itens:

- nome, representando mnemonicamente o sintoma através de uma metáfora;
- descrição, explicando textualmente a configuração do sintoma incluindo uma ilustração com uma estrutura simbólica e abstrata do mesmo;
- formalismo, apresentando os sintomas através de teoria dos conjuntos e lógica de primeira ordem;
- exemplo, ilustrando o sintoma para facilitar o seu entendimento.

Vale destacar que os sintomas sensíveis ao interesse identificam formas e o classificam de maneira metafórica a partir de suas configurações estruturais. Dessa maneira, conclui-se que, como os sintomas são relativos ao interesse, tanto faz usar as metáforas para os sintomas quanto para os interesses. Portanto, ao mencionar um interesse como "interesse dominador" é equivalente dizer que o mesmo interesse tem o "sintoma dominador".

4.2.1 Formalismo Base

O formalismo desta seção e ao longo do capítulo é complementar ao apresentado originalmente em (FIGUEIREDO et al., 2009). Inicialmente, definições básicas são apresentadas a fim de apoiar a descrição dos sintomas sensíveis ao interesse nas próximas seções.

Seja S o conjunto de sistemas, $Comps$ o conjunto de componentes[1], $Interesses$ o conjunto de interesses, $C(S)$ o conjunto dos conjuntos de componentes dos sistemas, e $Interesses(S)$ o conjunto dos conjuntos de interesses dos sistemas, define-se $s \in S$ como sendo um sistema, $Comps(s) \subseteq C(S)$ como sendo o conjunto de componentes do

[1] Vale lembrar que a noção de componente considerada aqui refere-se a classes, interfaces e aspectos

sistema s, e $Ints(s) \subseteq Interesses(S)$ como sendo o conjunto dos interesses do sistema s.

Em seguida, define-se a relação Ci, que mapeia os componentes participantes de um interesse em um determinado sistema s, como uma relação que está contida no produto cartesiano entre os conjuntos dos interesses de s e o conjunto dos componentes de s.

$$Ci \subseteq Comps(s) \times Ints(s)$$

Portanto, seja $i \in Ints(s)$ um interesse do sistema s, $Ci(i)$ é um conjunto de componentes do sistema s que implementa o interesse i, ou seja, $Ci(i) = \{c \mid (c, i) \in Ci\}$.

4.2.2 Formas Transversais

Esta seção reune três sintomas relacionados a interesses identificados pelas seguintes metáforas: Ovelha Negra, Polvo e Dominador. Estes sintomas referem-se a interesses realizados por componentes parcialmente ou totalmente dedicados ao mesmo. Contudo, não há obrigatoriedade de tais componentes possuírem acoplamentos e relacionamentos de herança entre si.

Para apoiar a formalização dos sintomas desta categoria, adiciona-se ao formalismo base a seguinte função parcial:

$$\textit{dedicação: } Comps(s) \times Ints(s) \longrightarrow \mathbb{N}$$

Seja $c \in Comps(s)$ um componente pertencente ao sistema s, e $i \in Ints(s)$ um interesse do sistema s, *dedicação*(c, i) indica o quanto do componente c está dedicado na realização do interesse i. Esta função é parcial, pois podem existir interesses que não são implementados por determinados componentes, assim como podem existir componentes que não implementam todos os interesses do sistema. Na prática, esta função é obtida experimentalmente através de coleta de dados usando métricas.

4.2.2.1 Ovelha Negra

Descrição: Ovelha Negra (traduzido do inglês *Black Sheep*) é um interesse transversalizado no sistema mas que encontra-se apenas em pequenas partes espalhadas por componentes distintos. Este sintoma ocorre quando somente pequenas quantidades de operações e atributos dedicados à realização do interesse encontram-se espalhados no desenho do sistema. Dessa maneira, um interesse Ovelha Negra não possui componentes que estejam em sua maior parte dedicados à implementação do mesmo. A figura 4.2 ilustra uma representação abstrata e simbólica deste sintoma. As caixas representam componentes e as regiões preenchidas de cinza indicam as partes dos componentes que realizam o interesse. Conforme descrito e indicado na figura, apenas pequenos trechos de alguns componentes dedicam-se à realização de um interesse com este sintoma.

Formalização: $OvelhaNegra(i) \Leftrightarrow \forall\, c \in Ci(i) \cdot \textit{dedicação}(c, i) < 33\%$

Esta expressão indica que para um interesse i ser Ovelha Negra é preciso que todos os componentes participantes tenham dedicado menos que 33% de seus dados e comportamentos ao interesse. A escolha do limiar dedicação (nesse caso, 33%) é derivada a partir da consideração de boas práticas e experiência do projetista e/ou desenvolvedor, e podem variar também a depender do contexto, como discute Lanza e Marinescu (2005).

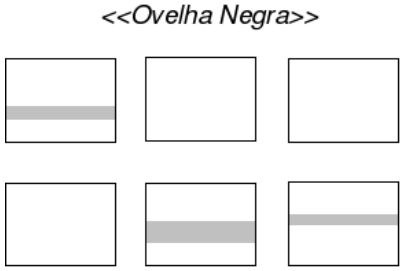


Figura 4.2: Configuração simbólica do sintoma Ovelha Negra

Exemplo: A figura 4.3 apresenta as cinco classes participantes do interesse *Favourites* no sistema *Mobile Media*[2] como uma instância do sintoma Ovelha Negra. Ao lado de cada classe descreve-se brevemente a sua estrutura interna (quantidade total de métodos e atributos) e as partes dedicadas à implementação do interesse. Três classes (*MediaController*, *MediaListController* e *MediaUtil*) dedicam apenas alguns trechos de código dentro de alguns de seus métodos. As duas classes restantes (*MediaListScreen* e *MediaData*) implementam o interesse com apenas parte de seus métodos e/ou atributos, representando, respectivamente, apenas 11% e 19% de sua estrutura interna. As demais partes das classes encontram-se relacionadas a outras intenções. Dessa maneira, o interesse *Favourites* encontra-se espalhado por 5 classes que possuem baixa dedicação, configurando assim o sintoma Ovelha Negra.

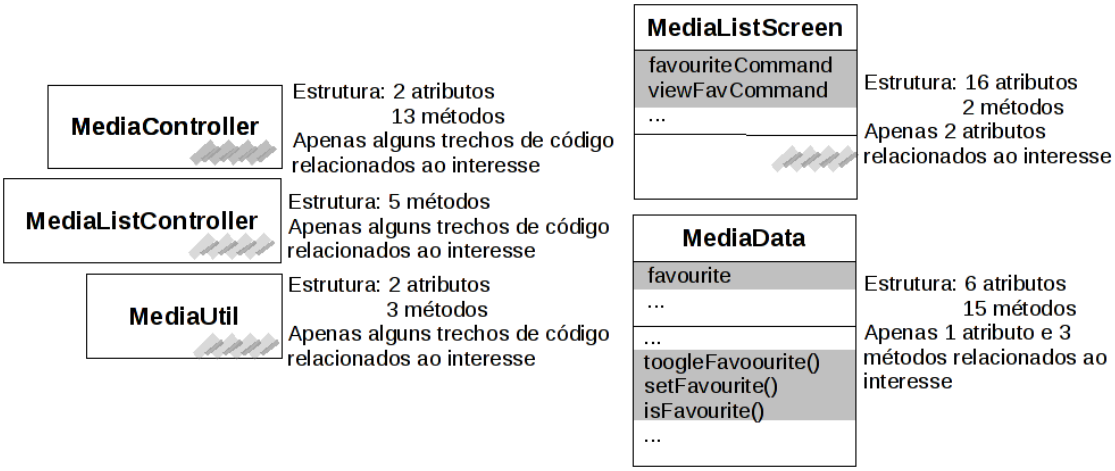


Figura 4.3: Classes participantes do interesse *Favourites* no *Mobile Media*

4.2.2.2 Polvo

Descrição: O sintoma Polvo (do inglês *Octopus*) é definido sobre um interesse que mantém-se bem modularizado em um ou mais componentes, porém encontra-se espalhado sobre outros componentes também. Os componentes que encontram-se bem modularizados são considerados o corpo do Polvo e as demais partes os tentáculos. Como pode ser observado na representação abstrata da figura 4.4, há uma caixa totalmente preenchida representando um componente como o corpo do sintoma Polvo e demais caixas (parcialmente preenchidas) indicando os tentáculos.

[2]http://sourceforge.net/projects/mobilemedia/

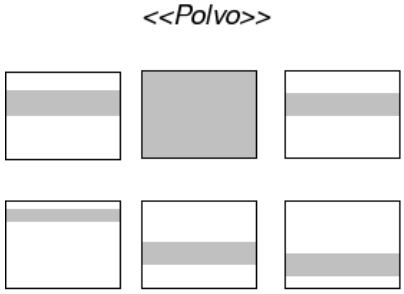


Figura 4.4: Configuração simbólica do sintoma Polvo

Formalização: $Polvo(i) \Leftrightarrow \exists\, c\, \exists\, d \in Ci(i) \cdot$ *dedicação*$(c,i) < 33\% \wedge$ *dedicação*$(d,i) > 67\%$
No caso do sintoma Polvo, a expressão indica que deve haver componentes participantes com baixa (33%) e alta dedicação (67%), isto é, deve existir ao menos um tentáculo e um corpo. A quantidade de tentáculos e corpos necessária para a configuração do sintoma pode ser ajustável de acordo com a necessidade do desenvolvedor e do contexto.

Exemplo: O diagrama da figura 4.5 ilustra quatro classes que participam da realização do padrão de projeto *Observer* (GAMMA et al., 1995). O sintoma *Polvo* configura-se neste exemplo pela existência das interfaces *Subject* e *Observer* que encontram-se totalmente dedicadas ao interesse (corpos do Polvo), e pelas classes *Point* e *Screen* que estão parcialmente dedicadas (tentáculos).

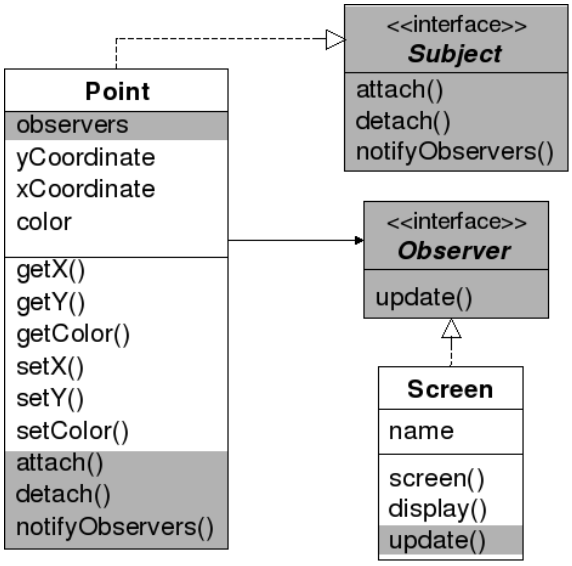


Figura 4.5: Diagrama de classes com o padrão *Observer*

4.2.2.3 Dominador

Descrição: O sintoma Dominador (do inglês *God Concern*) refere-se ao interesse que engloba diversas funcionalidades do sistema. Em outras palavras, os componentes que realizam tal interesse encontram-se espalhados e dominam grande parte do sistema. A figura 4.6 ilustra simbolicamente tal situação. Dessa maneira, um interesse Dominador manifesta-se justamente contra as ideias básicas de modularização.

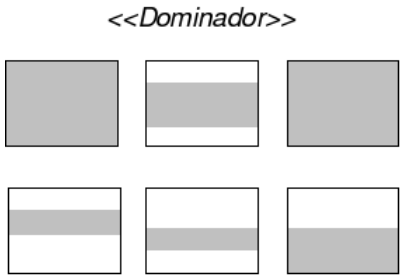


Figura 4.6: Configuração simbólica do sintoma Dominador

Formalização: $Dominador(i) \Leftrightarrow \exists\, c \in Ci(i) \cdot$ *dedicação*$(c, i) < 50\% \wedge$ *dedicação'*$(Comps(s), i) > 33\%$

Onde, *dedicação'*$(Comps(s), i) = \frac{\sum_{c \in Comps(s)} \textit{dedicação}(c, i)}{|Comps(s)|}$

A expressão indica que um interesse é Dominador quando a dedicação absoluta (definida por *dedicação'* acima) é maior que 33%. A dedicação absoluta é calculada em função da dedicação de todos os componentes do sistema sobre um interesse i. É feito um somatório de cada dedicação e divide-se pela quantidade de componentes de um sistema s em questão. Além disso, deve haver pelo menos um componente com baixa dedicação (nesse caso, menor que 50%). Esta última restrição é necessária para eliminar a possibilidade de classificar um interesse como Dominador tendo componentes participantes bem modularizados, isto é, com dedicação alta (acima de 50%). Dessa forma, a existência de pelo menos um componente com baixa dedicação indica que a dedicação absoluta foi computada considerando ao menos um componente com baixa dedicação e não somente componentes com alta dedicação.

Exemplo: O interesse *Photo* do sistema *Mobile Media* nas versões de 1 a 6 é um exemplo de um interesse Dominador (FIGUEIREDO et al., 2009). Este interesse foi classificado como Dominador no estudo de caso do *Mobile Media* apresentado em (FIGUEIREDO et al., 2009). Além desse interesse, no estudo de caso do *Health Watcher* (capítulo 7) foi identificado o interesse *Command* como sendo também um Dominador. Em ambos os casos, tais interesses encontram-se espalhados em diversos componentes ao longo do sistema, sendo que alguns deles possuem a maior parte dos seus membros dedicados à realização do respectivo interesse.

4.2.3 Sintomas Relativos a Herança

Esta seção reune dois sintomas relacionados a interesses que envolvem estruturas hierárquicas, seja através da própria relação de herança ou pela relação de implementação de interfaces. As manifestações de tais sintomas nos interesses são representadas através das seguintes metáforas: Planta Trepadeira e Doença Hereditária.

Para apoiar a formalização dos sintomas relativos a herança, adiciona-se ao formalismo base apresentado na seção 4.2.1 a seguinte função:

$$descendente:\ Comps(s),\ Comps(s) \longrightarrow Boolean$$

Sejam $c1 \in Comps(s)$ e $c2 \in Comps(s)$ dois componentes do sistema s, *descendente*$(c1, c2)$ mapeia para o valor booleano $Verdadeiro$ caso $c1$ seja descendente direto

ou indireto de $c2$ ou para $Falso$, caso não seja. Na prática, esta função é obtida experimentalmente verificando se há relação de herança ou implementação de interfaces (direta ou indiretamente) entre os componentes. Vale destacar que esta função considera descendência direta e indireta.

Além disso, define-se formalmente o componente *raiz* e os *galhos* da árvore de herança como segue:

$$raiz \ = \ r \ \in \ Ci(i) \mid \nexists \, d \ \in \ Ci(i) \ \cdot \ descendente(r, d) \ \wedge \ d \ \neq \ r$$

$$Galhos(i) \ = \ \{c \ \in \ Ci(i) \mid descendente(c, raiz)\}$$

$Galhos(i)$ é o conjunto de componentes do interesse que são descendentes (diretos ou indiretos) da raiz. Dessa forma, a raiz é um elemento r do conjunto de componentes que implementam o interesse, de modo que todos os componentes que são galhos são descendentes deste elemento r e, além disso, r não deve ser descendente de mais nenhum componente do interesse.

Por fim, define-se o conjunto $DescendentesLivres(i) \ = \ \{c \ \in \ Comps(s) |$ ***descendente***$(c, raiz) \ \wedge \ c \ \notin \ Ci(i)\}$, como sendo o conjunto dos componentes que fazem parte da árvore de herança (isto é, são descendentes da raiz) mas não implementam o interesse (não fazem parte do conjunto de componentes que implementam o interesse).

4.2.3.1 Planta Trepadeira

Descrição: Planta Trepadeira (tradução para *Climbing Plant*) é um sintoma que afeta a raiz de uma árvore de herança e propaga-se pela sua estrutura ao longo dos componentes descendentes, como ilustra abstratamente a figura 4.7. Tais componentes descendentes, chamados de galhos da árvore, devem estar totalmente ou parcialmente dedicados à realização do interesse. Portanto, todos os galhos devem se encontrar de alguma forma afetados, isto é, de alguma maneira participam da realização do interesse.

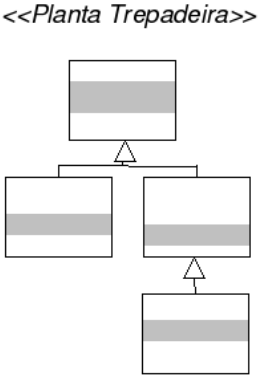


Figura 4.7: Configuração simbólica do sintoma Planta Trepadeira

Formalização: $PlantaTrepadeira(i) \ \Leftrightarrow \ Galhos(i) \ \neq \ \varnothing \ \wedge$
$DescendentesLivres(i) \ = \ \varnothing$

Assim, um interesse i tem o sintoma Planta Trepadeira se, e somente se, houver componentes galhos (consequentemente uma raiz) e se todos os componentes da árvore de herança implementam o interesse, isto é, não há descendentes "livres" do interesse.

Exemplo: A figura 4.1, discutida na seção 4.1, pode ser vista também como exemplo ilustrativo de sintomas sensíveis ao interesse relativos a herança. Vale notar duas estruturas hierárquicas partindo de duas implementações de interfaces - interface *Subject* e interface *Obsever*. O padrão de projeto *Observer* pode ser dividido em dois interesses de acordo com os papéis *Subject* (observado) e o próprio *Observer* (observador). O interesse referente ao papel de *Subject* na figura 4.1 configura-se com o sintoma Planta Trepadeira (uma das estruturas hierárquicas da figura). Neste caso, a raiz é a interface *Subject* completamente dedicada ao interesse, pois é um componente exclusivamente com a intenção de definir um tipo para o papel de *Subject* do padrão de projeto, com seus respectivos métodos as serem implementados pelas classes concretas. Ainda no exemplo da figura, as classes *Complaint*, *Employee* e *HealthUnit* são os galhos do sintoma Planta (os sujeitos observados concretos do padrão *Observer*). A outra parte do padrão encontrada na figura referente ao papel *Observer* é discutido na próxima seção.

4.2.3.2 Doença Hereditária

Descrição: Como na Planta Trepadeira, a Doença Hereditária (do inglês *Hereditary Desease*) afeta também a raiz de uma árvore de herança, no entanto propaga sua estrutura transversal apenas para alguns dos descendentes. Portanto, este sintoma não se manifesta em todos os componentes da estrutura hierárquica (conforme figura 4.8). Logo, haverá sempre um componente chamado de origem da doença, outros chamados de descendentes sadios e outros de descendentes doentes.

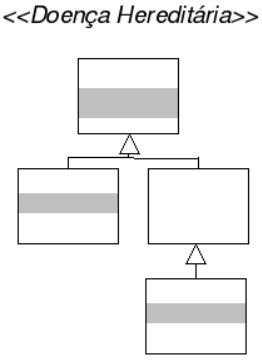


Figura 4.8: Configuração simbólica do sintoma Doença Hereditária

Formalização: $\textit{DoençaHereditária}(i) \Leftrightarrow Galhos(i) \neq \varnothing \wedge DescendentesLivres(i) \neq \varnothing$

Neste caso, para um interesse ter o sintoma Doença Hereditária, necessariamente deverá existir ao menos um componente "livre" do interesse, isto é, pelo menos um descendente sadio. Seguindo a metáfora deste sintoma, os demais seriam: a raiz, como a origem da doença; e os galhos, como os descendentes doentes.

Exemplo: Continuando com o exemplo da figura 4.1, retomado também na seção anterior (4.2.3.1), é possível notar a existência de uma segunda estrutura hierárquica referente ao papel *Observer* (observador) do próprio padrão de projeto *Observer*. Neste exemplo, é possível visualizar o sintoma Doença Hereditária para o interesse do papel *Observer*. Além da raiz (interface também de nome *Observer*) que representa a origem da doença, há ainda a interface *RMIFacade* na hierarquia, porém não participando da realização do interesse, isto é, ela não implementa o método

da interface *Observer* e, portanto, não é uma observadora. Neste caso, ela representa um descendente sadio. Por fim, logo abaixo, ligada na estrutura, vem a classe *HealthWatcherFacade* que está dedicada ao interesse através da implementação do método *update()*, sendo, portanto, observador concreto e um descendente doente de acordo com o sintoma Doença Hereditária.

4.2.4 Sintomas Relativos a Acoplamento

Esta seção apresenta quatro sintomas representados pelas metáforas: Interesse Enraizado, Tsunami, Serpente e Rede Neural. A característica em comum relacionada aos interesses que possuem os sintomas desta categoria é a existência de acoplamento entre os respectivos componentes participantes. As diferentes maneiras pelas quais os componentes se encontram acoplados definem os quatro sintomas apresentados.

Para apoiar a formalização dos sintomas desta seção, define-se a seguinte função:

$$Conectado : Comps(s), Comps(s) \longrightarrow Boolean$$

Sejam $c1 \in Comps(s)$ e $c2 \in Comps(s)$ dois componentes do sistema s, *conectado*$(c1, c2)$ mapeia para o valor booleano $Verdadeiro$ se o componente $c1$ está conectado diretamente ou indiretamente ao componente $c2$. Caso contrário a função mapeia para o valor $Falso$. Como a função considera também a conexão indireta entre componentes, vale notar, portanto, que a relação é transitiva: dado $c1$, $c2$ e $c3 \in Comps(s)$; se $Conectado(c1, c2)$ e $Conectado(c2, c3)$ então $Conectado(c1, c3)$. Além disso, a função é total, pois sempre é possível termos um valor booleano indicando se quaisquer pares de componentes estão conectados, porém ela não é uma função reflexiva. Na prática, esta função é obtida experimentalmente verificando se na implementação dos componentes há acoplamento entre eles.

4.2.4.1 Interesse Enraizado

Descrição: O sintoma Interesse Enraizado (do inglês *Tree Root*) é formado por componentes de dois tipos: tronco e ramo. Interesses desse tipo possuem apenas um troco e um ou mais ramos. O tronco é o componente que recebe conexões de todos os demais componentes do intereresse (ramos). Isto significa que os ramos estão diretamente ou indiretamente acoplados ao tronco, como ilustra a figura 4.9.

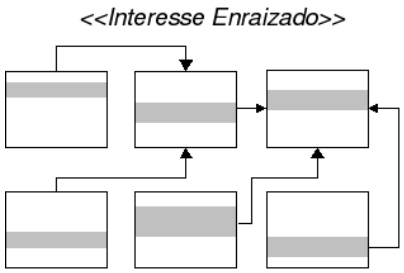


Figura 4.9: Configuração simbólica do sintoma Interesse Enraizado

Formalização: $InteresseEnraizado(i) \Leftrightarrow (|Ramos(i)| > |tronco(i)|) \wedge Ramos(i) \neq \varnothing$

Onde, $tronco(i) = \{t \in Ci(i) \mid \nexists\, c \in Ci(i) \cdot Conectado(t, c)\}$
e $Ramos(i) = \{c \in Ci(i) \mid \exists\, t \in tronco(i) \cdot Conectado(c, t) \wedge c \neq t\}$

Assim, um interesse i possui o sintoma Enraizado se, e somente se, o conjunto de ramos não for vazio e se este for maior que o conjunto de troncos. Por sua vez, o conjunto de ramos é o conjunto de componentes que estão conectados (diretamente ou indiretamente) com componentes tronco. Por fim, o conjunto de troncos reune componentes do interesse que não estão conectados com qualquer outro componente.

Exemplo: O diagrama de classes da figura 4.10 ilustra os componentes envolvidos na realização do interesse referente ao padrão de projeto *State* (GAMMA et al., 1995) extraídos do sistema *Health Watcher* (SOARES; LAUREANO; BORBA, 2002). Como é possível notar pelas setas de relacionamento entre os componentes, todas as classes da borda da figura dedicam-se parcialmente ou inteiramente ao interesse e convergem conexões de acoplamento para uma única classe, por isso são consideradas ramos. A classe *Situation*, no centro do diagrama, recebe, portanto, as conexões e não está acoplada a mais nenhum outro componente que participa do interesse, sendo portanto o tronco.

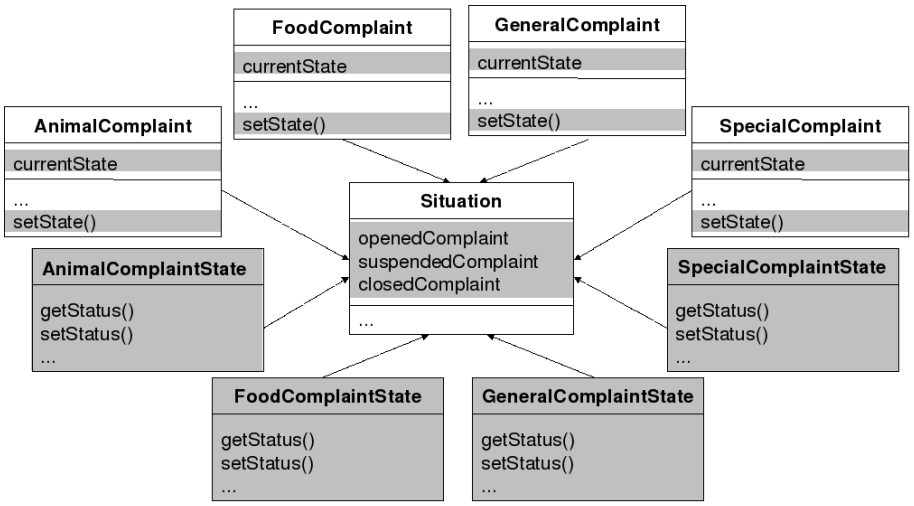


Figura 4.10: Diagrama de classes com o padrão *State* extraído do *Health Watcher*

4.2.4.2 Tsunami

Descrição: O sintoma Tsunami é formado por um componente núcleo chamado de origem da onda (ou epicentro). Este componente está diretamente ou indiretamente acoplado aos demais componentes participantes do interesse. Um interesse Tsunami configura-se como uma propagação de onda, onde um componente (origem da onda) possui conexões diretas e indiretas com os outros (demais ondas). Diferentemente do Interesse Enraizado, as conexões entre componentes no Tsunami não convergem para um participante. Pelo contrário, a configuração de um interesse Tsunami é inversa em relação ao Interesse Enraizado. Enquanto um tem ramos conectados ao tronco, no outro tem-se a origem da onda que distribui conexões para os demais (conforme figura 4.11). As ondas de extremidade do Tsunami são componentes que apenas recebem conexões e não estão acoplados com outros participantes do interesse.

Formalização: $Tsunami(i) \Leftrightarrow (|Ondas(i)| > |epicentro(i)|) \wedge Ondas(i) \neq \varnothing$

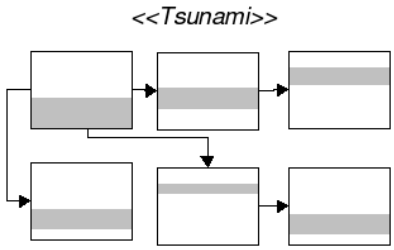


Figura 4.11: Configuração simbólica do sintoma Tsunami

Onde, $epicentro(i) = \{e \in Ci(i) \mid \nexists\, c \in Ci(i) \cdot Conectado(c, e)\}$
e $Ondas(i) = \{c \in Ci(i) \mid e \in epicentro(i) \cdot Conectado(e, c) \wedge e \neq c\}$

Assim, um interesse i possui o sintoma Tsunami se, e somente se, o conjunto de ondas não for vazio e se este for maior que o conjunto de epicentros. O conjunto de ondas é o conjunto de componentes que tem conexões (diretas ou indiras) a partir de um componente epicentro, onde o conjunto epicentro reune componentes do interesse que estão conectados com qualquer outro componente participante (ondas).

Exemplo: A figura 4.12 ilustra os componentes envolvidos na realização do interesse de Persistência extraído do *Health Watcher*. As setas indicam as conexões de acoplamento existentes entre eles. O sintoma Tsunami se configura pela existência da classe *HealthWatcherFacade* (como sendo a origem da onda) que possui conexões com outros cinco componentes incluindo a classe *RDBRepositoryFactory*, que por sua vez está conectada com outras cinco.

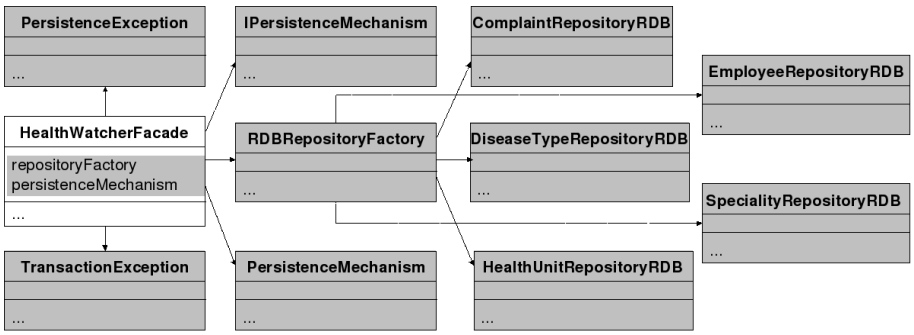


Figura 4.12: Componentes do in teresse de Persistência extraído do *Health Watcher*

4.2.4.3 Serpente

Descrição: O sintoma Serpente (tradução do inglês *King Snake*) é caracterizado pela maior cadeia acíclica de conexões entre componentes que participam do interesse, como representado na figura 4.13. O primeiro componente desta cadeia é chamadado de Cabeça da Serpente e o último, localizado no fim, é chamado de cauda.

Formalização: $Serpente(i) \Leftrightarrow Cadeia(i) \neq \varnothing$

Onde, $Cadeia(i) = \{c \in Ci(i) \mid \exists\, h \in$ ***cabeça(i)*** $\cdot\, Conectado(h, c) \wedge$
$\exists\, t \in cauda(i) \cdot Conectado(c, t)\}$, ***cabeça(i)*** $= \{c \in Ci(i) \mid \nexists\, d \in Ci(i) \cdot Conectado(d, c)\}$,
ou seja, ***cabeça(i)*** $= epicentro(i)$ e $cauda(i) = \{c \in Ci(i) \mid \nexists\, d \in Ci(i) \cdot Conectado(c, d)\}$,
isto é, $cauda(i) = tronco(i)$.

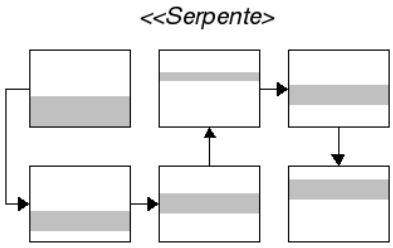


Figura 4.13: Configuração simbólica do sintoma Serpente

Sendo assim, um interesse possui o sintoma Serpente se, e somente se, houver uma cadeia de conexões entre componentes participantes do interesse. Vale destacar que esta cadeia não deve conter ciclos, embora esta restrição não esteja representada na formalização. Metaforicamente, o conjunto $Cadeia(i)$ reune elementos que são componentes do interesse e que tem um componente cabeça da serpente conectado a eles, assim como eles devem estar conectados a um componente cauda. O conjunto *cabeça(i)* segue a mesma definição de *epicentro(i)* no sintoma Tisunami, assim como o conjunto *cauda(i)* segue a mesma definição de *tronco(i)* no sintoma Interesse Enraizado.

Exemplo: O diagrama de classes da figura 4.14 ilustra os componentes envolvidos no padrão de projeto *Abstract Factory* (GAMMA et al., 1995) extraídos do sistema *Health Watcher*. O interesse referente à realização do padrão configura-se como Serpente pela cadeia de conexões (acoplamentos) existentes entre os componentes participantes, que começa pela classe *HealthWatcherFacade* (cabeça) e termina na classe *AbstractRepositoryFactory* (cauda).

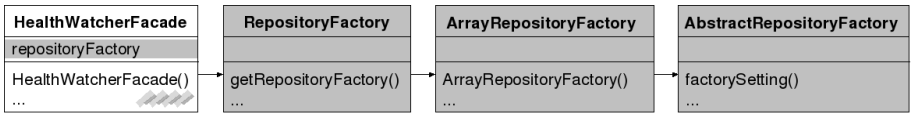


Figura 4.14: Padrão *Abstract Factory* como Serpente extraído do *Health Watcher*

4.2.4.4 Rede Neural

Descrição: O sintoma Rede Neural (do inglês *Neural Network*) consiste em uma rede acíclica de componentes interconectados. Um interesse com este sintoma possui uma estrutura que remete às redes neurais artificiais que inclui: componentes da camada de entrada, isto é, componentes que não recebem conexões mas possuem acoplamentos com outros; componentes da camada de saída, reunindo aqueles que não são responsáveis por acoplamento e sim apenas recebem conexões de outros; e, opcionalmente, componentes de camada intermediária, que envolvem aqueles componentes que se localizam entre as camadas de entrada e saída. A figura 4.15 representa abstratamente tal situação.

Formalização: $RedeNeural(i) \Leftrightarrow \forall\, c \in CamadaEntrada(i)\, \exists\, d \in$ ***CamadaSaída(i)*** $. Conectado(c,d) \wedge \forall\, f \in$ ***CamadaSaída(i)*** $\exists\, g \in CamadaEntrada(i) \cdot Conectado(g, f) \wedge CamadaEntrada(i) \neq \varnothing \wedge$ ***CamdaSaída***$(i) \neq \varnothing$

Onde, $CamadaEntrada(i) = \{e \in Ci(i) \mid \exists\, c \in Ci(i) \cdot Conectado(e, c) \wedge$

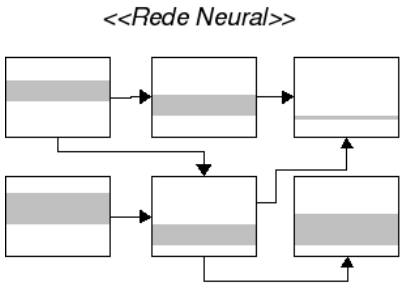


Figura 4.15: Configuração simbólica do sintoma Rede Neural

$\nexists\, d \in Ci(i) \cdot Conectado(d, e)\}$ e

CamdaSaída$(i) = \{c \in Ci(i) \mid \exists\, d \in Ci(i) \cdot Conectado(d, c) \wedge$

$\nexists\, f \in Ci(i) \cdot Conectado(c, f)\}$

Um interesse possui o sintoma Rede Neural se, e somente se, todos os componentes da camada de entrada estão conectados com pelo menos um componente da camada de saída. Assim como todos os componentes da camada de saída devem receber conexões de pelo menos um componente da camada de entrada. Esta camada é formada por componentes que estão conectados com pelo menos um outro componente do interesse e não recebem conexões de qualquer outro componente. Já a camada de saída é composta de componentes que recebem conexões de pelo menos um outro componente do interesse e não estão conectados com qualquer outro.

Opcionalmente, pode-se definir componentes formando a camada intermediária:

CamadaIntermediária$(i) = \{h \in Ci(i) \mid \exists\, c1 \in CamadaEntrada(i) \cdot$ ***Conectado***$(c1, h) \wedge \exists\, c2 \in$ ***CamadaSaída***$(i) \cdot Conectado(h, c2)\}$.

Exemplo: A figura 4.16 ilustra um diagrama de classes com os componentes que realizam o interesse *LabelMedia* extraído do sistema *Mobile Media* (YOUNG; MURPHY, 2005). As setas representam as relações de acoplamento existentes entre os participantes, formando um emaranhado (rede). As classes *ImageUtil*, *PhotoController* e *PhotoViewScreen* não recebem conexões de acoplamente e, assim, representam a camada de entrada da Rede Neural. As classes *ImageData*, *AddPhotoToAlbum*, *NewLabelScreen* e *AlbumData* apenas recebem conexões de acoplamento e não possuem relacionamento com outros componentes que realizam o interesse, fazendo portanto o papel da camada de saída da rede.

4.2.5 Outros Problemas de Modularidade

Este agrupamento de sintomas refere-se a outros problemas de modularidade representados por quatro metáforas: Cópia Carbono, Ovelha Dolly, Interesse Exclusivo de Dados e Interesse Comportamental. Tais sintomas sinalizam que de alguma maneira o uso de dados e comportamentos dos componentes pode ocasionar problemas de modularidade, como por exemplo, a replicação de atributos e operações ou até mesmo a falta de alguns deles.

Para a formalização dos sintomas deste grupo, fazem-se necessárias as seguintes definições. Primeiramente, seja $s \in S$ um sistema, define-se três novos conjuntos: $Atributos(s)$, como o conjunto de todos os atributos de s; ***Operações***(s), com o conjunto de todas as operações de s; e ***Declarações***(s), como sendo o conjunto de todas as

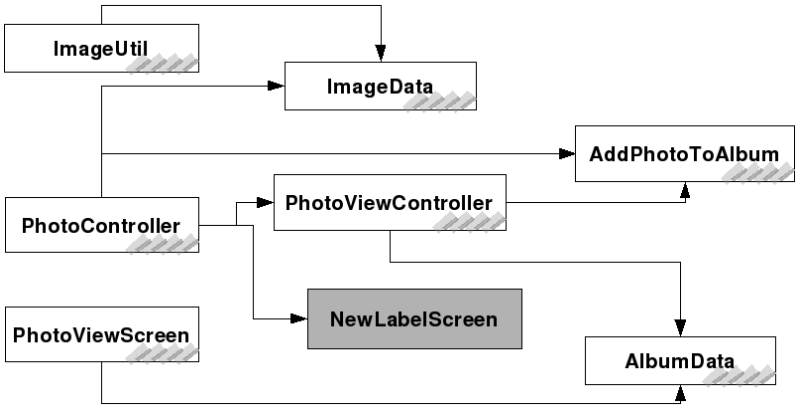


Figura 4.16: Componentes participantes do interesse *LabelMedia* extraído do *Mobile Media*

declarações de s. Em seguida, define-se o conjunto $Membros(s) = Atributos(s) \cup$ *Operações*(s) $\cup$ *Declarações*(s), como sendo o conjunto de todos os membros do sistema s, que é formado pela união de seus atributos, operações e declarações.

Adicionalmente, seja $i \in Ints(s)$ um interesse do conjunto de interesses de um sistema s, define-se $M(i) \subseteq Membros(s)$, como o conjunto de membros que implementam o interesse i do sistema s.

Por fim, define-se a função *Cópia* da seguinte maneira:

$$\textit{Cópia} : Membros(s),\ Membros(s) \longrightarrow Boolean$$

Sejam $m1 \in M(i)$ e $m2 \in M(i)$ dois membros que implementam o interesse i do sistema s, *Cópia*$(m1, m2)$ mapeia para o valor booleano $Verdadeiro$ se o membro $m1$ é cópia de $m2$. Caso contrário a função mapeia para o valor $Falso$. Vale observar a simetria: se $m1$ é cópia de $m2$, $m2$ é cópia de $m1$, logo *Cópia*$(m1, m2) =$ *Cópia*$(m2, m1)$. Na prática, esta função é obtida experimentalmente verificando se na implementação dos componentes os referidos membros são iguais.

4.2.5.1 Cópia Carbono

Descrição: A Cópia Carbono (tradução do inglês *Copy Cat*) é um sintoma que representa a replicação de dados e/ou comportamento que fazem parte de um mesmo interesse. Em outras palavras, dado um interesse, a sua ocorrência em componentes é verificada em trechos copiados (replicados) em diferentes componentes ao longo do sistema, como ilustra a figura 4.17.

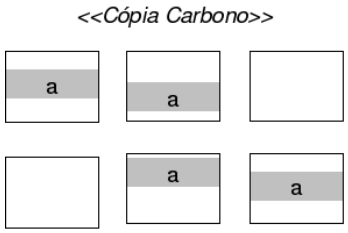


Figura 4.17: Configuração simbólica do sintoma Cópia Carbono

Formalização: *CópiaCarbono(i)* $\Leftrightarrow \exists\, m,\ n \in Membros(i) \cdot$ *Cópia*(m, n)

Este sintoma é identificado em um interesse se, e somente se, houver pelo menos dois membros iguais que fazem parte da implementação do interesse.

Exemplo: A figura 4.18 destaca dois fragmentos de código retirados dos componentes que implementam o interesse de distribuição no sistema *Health Watcher*. Os trechos são semelhantes e referem-se a declarações de tratamento de exceção que podem ocorrer durante a comunicação distribuída utilizada no *Health Watcher*, que é uma aplicação *web*. O primeiro fragmento foi encontrado 28 vezes ao longo dos componentes do interesse *Distribution*, enquanto que o segundo foi detectado 7 vezes. Caso houvesse a necessidade, por exemplo, de modificar o tratamento de exceção do quadro 1 da figura, as atividades de manutenção seriam prejudicadas pelo alto grau de espalhamento e entrelaçamento existente na cópia (28 locais a serem modificados). Se estes fragmentos (cópias) estivessem modularizados em aspectos, um único ponto seria utilizado tanto para compreender o objetivo ou intenção do código, assim como para realizar atividades de manutenção sobre ele.

28x
```
try{
...
} catch (RemoteException e)
{
  throw new CommunicationException(e.getMessage());
}
```

7x
```
try{
...
} catch (RemoteException e)
{
  e.printStackTrace();
}
```

Figura 4.18: Cópias ao longo do interesse de distribuição no sistema *Health Watcher*

4.2.5.2 Ovelha Dolly

Descrição: O sintoma Ovelha Dolly (do inglês *Dolly Sheep*) refere-se a um tipo especial de problema de modularidade que ocorre quando há os sintomas Cópia Carbono e Ovelha Negra. Nesse caso, o sintoma é chamado de Ovelha Dolly, representando uma "ovelha replicada" ou clonada. Em outras palavras, este sintoma se manifesta em interesses espalhados em pequenas partes do sistema (Ovelha Negra) sendo que estes trechos espalhados são cópias um do outro (Cópia Carbono). A figura 4.19 representa tal situação.

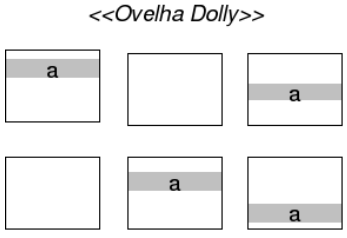


Figura 4.19: Configuração simbólica do sintoma Ovelha Dolly

Formalização: $OvelhaDolly(i) \Leftrightarrow$ ***CópiaCarbono(i)*** $\wedge\ OvelhaNegra(i)$

Um interesse possui o sintoma Ovelha Dolly se, e somente se, ele possuir também os sintomas Cópia Carbono e Ovelha Negra.

Exemplo: A figura 4.20 ilustra duas classes hipotéticas que implementam o padrão de projeto *Prototype* (GAMMA et al., 1995). A primeira, classe Produto, possui alguns atributos, sendo que um deles é um objeto da classe Preço. A classe Preço tem alguns atributos e métodos acessores, assim como na classe Produto. O que ambas têm em comum, como destacado em cinza na figura, é a implementação do método *clone()*. Este método retorna uma cópia do objeto (isto é, um clone exatamente), como mostra o trecho de código destacado. Na implementação deste padrão de projeto, ocorre a necessidade de implementar o método clone nas diversas classes que se tem a necessidade de copiá-las (ou cloná-las). Essa intenção fica, portanto, transversalizada com os demais interesses que as classes podem eventualmente possuir. Este exemplo configura-se como Ovelha Dolly por dois motivos: (i) pelo fato de ser implementado apenas por um método em cada componente que se deseja requisitar cópias de objetos - assim ele é um Ovelha Negra, pois ocupa pequenas partes (menos que um terço) do componente, neste caso apenas um método; (ii) e ainda é Cópia Carbono, pois a implementação do método *clone()* está igual nos componentes, sendo assim cópias.

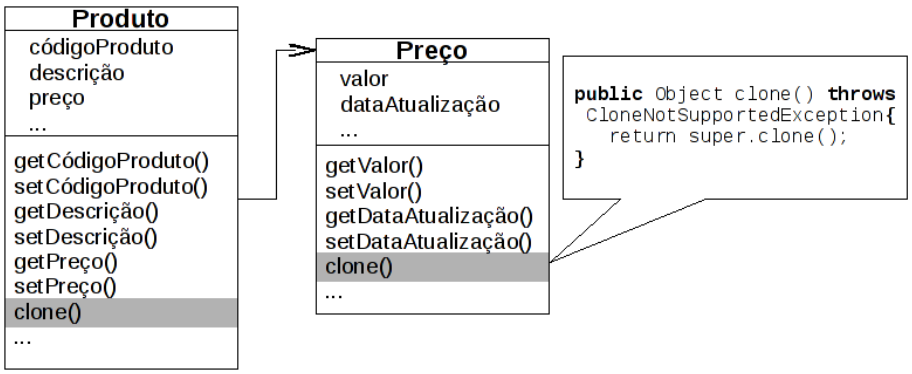


Figura 4.20: Classes implementando o padrão de projeto *Prototype*

4.2.5.3 Interesse Exclusivo de Dados

Descrição: O sintoma para Interesse Exclusivo de Dados (do inglês *Data Concern*) ocorre na existência de componentes que realizam o interesse apenas com atributos, sem haver a existência de comportamento associado, isto é, as únicas operações referem-se a métodos acessores de dados encapsulados. Como ilustrado na figura 4.21, as partes dedicadas ao interesse referem-se apenas a dados.

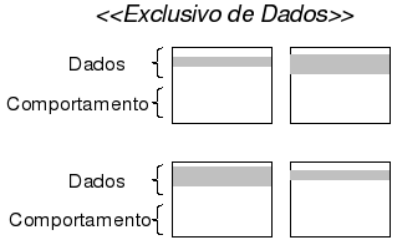


Figura 4.21: Configuração simbólica do sintoma de Interesse Exclusivo de Dados

Formalização: $InteresseExclusivoDados(i) \Leftrightarrow \forall\, m \in M(i) \cdot Atr(m) \vee$ *Acessor-DeAtributo*(m)

Onde: $Atr(m)$ é um predicado unário que indica que o membro m é um atributo; e $AcessorDeAtributo(m)$ também é um predicado unário que indica que o membro m é um método acessor de atributo. Assim, um interesse é exclusivo de dados se, e somente se, todos os membros que implementam o interesse são atributos ou métodos acessores dos atributos.

Exemplo: Este tipo de sintoma é bem específico e assim certamente difícil de encontrar. Um exemplo de interesse Exclusivo de Dados pode ser obtido na implementação do papel *Context* no padrão de projeto *Strategy* (GAMMA et al., 1995). Os componentes que implementam este padrão devem possuir um dos dois papéis: *context* ou *strategy*. Componentes com o papel *context* devem possuir um atributo que faz referência ao objeto *strategy*, enquanto que componentes com o papel *strategy* devem possuir uma variação de implementação (estratégia) para um determinado fim. Desse modo, componentes com o papel *strategy* possuem apenas comportamento e componentes com o papel *context* apenas um atributo. Logo, o papel *context* considerado como um interesse caberia no sintoma de interesse Exclusivo de Dados. Complementarmente, o papel *strategy* considerado como um interesse teria o sintoma interesse Comportamental (próxima seção).

A figura 4.22 ilustra um exemplo de implementação do padrão de projeto *Strategy* retirado da biblioteca de classes *Swing*[3] da API Java. Esta biblioteca de classes permite a criação e manipulação de objetos gráficos e eventos de interface gráfica com o usuário. A classe *JComponent* representa componentes gráficos da API *Swing* (por exemplo, caixas de texto, rótulos, botões etc.). Um *JComponent* possui o interesse *context* na implementação do padrão, pois faz referência a um objeto *border* que, por sua vez, contém a estratégia de pintura da borda através do método *paintBorder()*, tendo assim o interesse do papel *strategy* no padrão. Diversas variações de borda podem existir desde que implementem uma estratégia de pintura. Invariavelmente, componentes com o papel *context*, no caso da figura a classe *JComponent*, deverão possuir apenas uma referência ao objeto que contém a estratégia. Este objeto é um atributo da classe no exemplo. Além disso, eventualmente outros componentes que necessitem da estratégia de pintura de borda precisarão ter um atributo fazendo referência a um objeto que contém a estratégia. Dessa maneira, o interesse *context* do padrão *Strategy* configura-se como um interesse Exclusivo de Dados.

4.2.5.4 Interesse Comportamental

Descrição: Diferentemente do Interesse Exclusivo de Dados, o sintoma para Interesse Comportamental (*Behavioural Concern*) ocorre quando os componentes que realizam o interesse são compostos apenas por comportamentos, isto é, possuem apenas operações sem atributos associados à implementação do interesse.

Formalização: $InteresseComportamental(i) \Leftrightarrow \nexists\, m \in M(i) \cdot Atr(m)$

[3] A documentação da biblioteca *Swing* e de toda API Java encontra-se disponível eletronicamente em http://java.sun.com/j2se/1.5.0/docs/api/

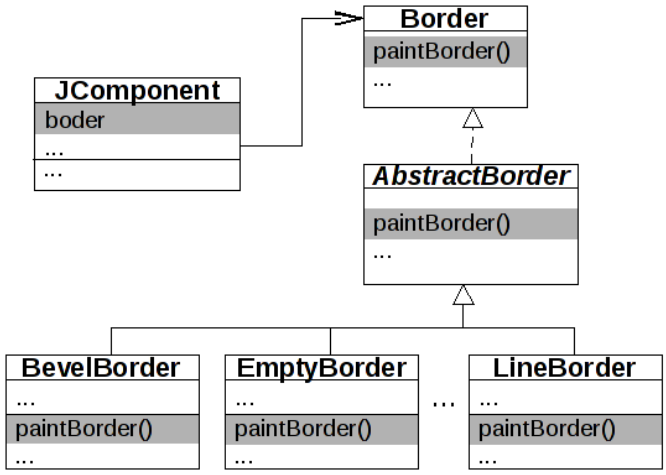


Figura 4.22: Classes implementando o padrão de projeto *Strategy*

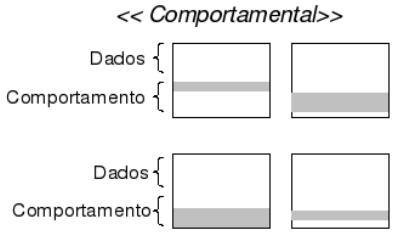


Figura 4.23: Configuração simbólica do sintoma de Interesse Comportamental

Onde, $Atr(m)$ é um predicado unário que indica que o membro m é um atributo. Logo, um interesse é comportamental se, e somente se, não existir membro quer participe da sua implementação que seja um atributo.

Exemplo: O exemplo apresentado para Ovelha Dolly na figura 4.20 pode ser retomado aqui como exemplo também de interesse Comportamental. Como é possível notar na região sombreada da figura, existe apenas a implementação do método *clone()*, não havendo dados associados. Neste caso particular da implementação do padrão de projeto *Prototype* (figura 4.20), tem-se a necessidade de apenas implementar o método *clone()*. Vale ressaltar que, mesmo que o corpo do método *clone()* seja diferente ao longo dos componentes clonáveis, o interesse do padrão de projeto não seria um Ovelha Dolly, pois não seria mais um Cópia Carbono, mas seria sim um interesse Comportamental.

Adicionalmente, é possível verificar um interesse Comportamental também no exemplo da figura 4.22 onde é ilustrada a implementação do padrão de projeto *Strategy* na biblioteca de classes *Swing* da API Java. O interesse do papel *strategy* consiste apenas na implementação do método que contém a estratégia de pintura. As diversas classes de borda que podem existir devem implementar o método *paintBorder()* fazendo o papel *strategy* no padrão. Este papel pode ser considerado como um interesse Comportamental, pois é implementado apenas por métodos.

4.3 Heurísticas de Detecção

Esta seção apresenta as heurísticas para detecção dos sintomas sensíveis ao interesse disponíveis eletronicamente como um apêndice[4] do artigo (FIGUEIREDO et al., 2009). Tais heurísticas reunem informações de uma série de técnicas complementares entre si para análise de interesses que incluem: métricas convencionais (CHIDAMBER; KEMERER, 1994) (LANZA; MARINESCU; DUCASSE, 2005) e orientadas a interesses (EADDY et al., 2008) (FIGUEIREDO et al., 2008); detecção de clones baseada em interesses (BRUNTINK et al., 2004); e dependência entre componentes baseada em interesses (FIGUEIREDO et al., 2008).

Vale ressaltar que as heurísticas de detecção não esgotam as possibilidades de identificar os sintomas, não impedindo assim que outras heurísticas possam ser propostas para detecção dos mesmos. O restante desta seção discute as técnicas empregadas para cada agrupamento de sintomas previamente definidos na seção anterior.

4.3.1 Regras Heurísticas Baseadas em Métricas

As regras heurísticas baseadas em métricas são inspiradas nas estratégias de detecção propostas por Marinescu (2004). Em seu trabalho, Marinescu utiliza métricas convencionais voltadas para a detecção de um conjunto de problemas convencionais de modularização, como *Shotgun Surgery* (FOWLER et al., 1999), por exemplo. Entretanto, Figueiredo *et al.* (2009) estende o mecanismo de estratégias de detecção com métricas recentes voltadas para medição de interesses (EADDY et al., 2008) (FIGUEIREDO et al., 2008). A tabela 4.1 apresenta uma breve descrição de seis métricas convencionais e orientadas a interesse usadas na elaboração das regras heurísticas que compõem a estratégia de detecção dos sintomas baseados em formas transversais (seção 4.2.2): Ovelha Negra, Polvo e Dominador. Mais informações sobre as métricas da tabela 4.1 podem ser encontradas em suas respectivas referências.

Tabela 4.1: Métricas convencionais e orientadas a interesse usadas nas regras heurísticas

Métricas	**Definições**
Número de Componentes (NC) (LANZA; MARINESCU; DUCASSE, 2005)	Conta o número de classes, interfaces e aspectos.
Número de Atributos (NOA) (CHIDAMBER; KEMERER, 1994)	Conta o número de atributos de cada classe, interface ou apecto.
Número de Operações (NOO) (CHIDAMBER; KEMERER, 1994) (SANT'ANNA et al., 2003)	Conta o número de métodos e adendos de cada, classe, interface ou aspecto.
Difusão do Interesse sobre Componentes (CDC) (FIGUEIREDO et al., 2008)	Conta o número de componentes que realizam parte de um determinado interesse.
Número de Atributos do Interesse (NOCA) (FIGUEIREDO et al., 2008)	Conta o número de atributos que contribuem para a realização de um interesse.
Número de Operações do Interesse (NOCO) (FIGUEIREDO et al., 2008)	Conta o número de métodos e adendos que participam da realização do interesse.

A figura 4.24 mostra três regras heurísticas R01, R02 e R03, baseadas nas métricas apresentadas na tabela 4.1, que compõem a estratégia de detecção dos sintomas com formas transversais (seção 4.2.2), respectivamente: Ovelha Negra, Polvo e Dominador. Na figura 4.24 define-se também as condições A (Baixa Dedicação) e B (Alta Dedicação)

[4]http://www.lancs.ac.uk/postgrad/figueire/concern/patterns/

usadas pelas regras. Tais condições encontram-se separadas para poderem ser reutilizadas em mais de uma regra e para facilitar a leitura e o entendimento.

A primeira regra (R01) identifica um interesse com o sintoma Ovelha Negra se todos os componentes participantes dedicam um percentual baixo de seus atributos e operações para a realização do mesmo (menos que 33%). Já a segunda regra (R02) verifica se um interesse, não classificado como Ovelha Negra, é um Polvo. De acordo com essa regra, este sintoma é identificado se todo componente que realiza o interesse atende à condição A ou B, isto é, dedica-se pouco ou muito ao interesse. E, além disso, pelo menos dois dos componentes participantes devem se dedicar pouco e ao menos um deve se dedicar muito, isto é, o Polvo deve possuir ao menos um corpo e dois tentáculos. Um componente tem alta dedicação caso 67% de seus atributos e operações participam da realização do interesse (condição B). Por fim, a regra R03 tem como propósito identificar o sintoma Dominador. Um interesse pode ser detectado como Dominador caso duas condições sejam satisfeitas: (i) se o interesse afeta mais de 33% dos componentes de todo o sistema e (ii) se a maioria dos componentes afetados (não todos) possuem alta dedicação ao interesse (condição B).

Condição A – *Baixa Dedicação*:
(NOCA / NOA < 0.33) **e** (NOCO / NOO < 0.33)

Condição B – *Alta Dedicação*:
(NOCA / NOA ≥ 0.67) **e** (NOCO / NOO ≥ 0.67))

R01 – *Ovelha Negra*:
Se (*Baixa Dedicação*) **para todo componente com o** INTERESSE **então** INTERESSE TRANSVERSAL é OVELHA NEGRA

R02 - *Polvo*:
Se ((*Baixa Dedicação*) **ou** (*Alta Dedicação*) **para todo componente com o** INTERESSE)
e ((*Baixa Dedicação*) **para pelo menos 2 componentes com o** INTERESSE)
e ((*Alta Dedicação*) **para pelo menos 1 componente com o** INTERESSE) **então** INTERESSE TRANSVERSAL é POLVO

R03 - *Dominador*:
Se (CDC / NC) > 0.33 **e** (Alta Dedicação) **para a maioria dos componentes com o** INTERESSE **então** INTERESSE TRANSVERSAL é DOMINADOR

Figura 4.24: Regras heurísticas para detecção de formas transversais

Vale notar que um interesse pode assumir mais de um sintoma do mesmo agrupamento. Neste caso, não é possível identificar um interesse Ovelha Negra e Polvo ao mesmo tempo, entretanto, um interesse Dominador pode-se configurar também como um Polvo. Tal fato pode ser observado em parte dos resultados da avaliação apresentada no capítulo 7.

4.3.2 Algoritmos Baseados em Estruturas de Árvore

Os sintomas relativos a herança (Planta Trepadeira e Doença Hereditária) explicados na seção 4.2.3 são identificados mais facilmente. É pré-condição, para a identifiação de interesses com estes sintomas, a existência de estruturas de árvore entre os componentes participantes do interesse representando as relações de herança ou implementação de interfaces. Basicamente são necessários dois passos para detectá-los:

1. criar uma floresta, isto é, a união disjunta de árvores, incluindo todas as árvores de herança[5] que possuem ao menos um componente realizando o interesse em questão, considerando que toda raiz deve participar da realização do mesmo;

[5]Considera-se também a implementação de interfaces compondo a árvore de herança.

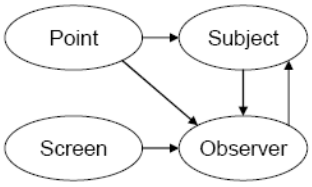
2. verificar (caso exista) quais árvores da floresta possuem ao menos um componente não participante do interesse.

Uma vez realizados os dois passos, considera-se as árvores com todos os componentes participantes do interesse como Planta Trepadeira. Por outro lado, as árvores que possuem pelo menos um componente não participante do interesse são consideradas com a estrutura do sintoma Doença Hereditária.

Como é possível a existência de múltiplas árvores de herança entre os componentes do mesmo interesse, é possível também que se identifique os dois sintomas deste agrupamento simultaneamente. Simplesmente, uma das árvores pode possuir todos os componentes participantes do interesse, e neste caso será uma Planta Trepadeira. Além disso, pode haver também uma árvore com pelo menos um componente não participando da realização do interesse - este será então classificado como Doença Hereditária.

4.3.3 Algoritmos de Pesquisa em Grafo

Alguns algoritmos de pesquisa em grafos compõem as heurísticas para identificação de interesses com os sintomas relativos a acoplamento (seção 4.2.4): Interesse Enraizado, Tsunami, Rede Neural e Serpente. Portanto, o primeiro passo para a identificação de tais interesses é a construção de um grafo dirigido, chamado de grafo de dependências do interesse, representando as conexões entre os componentes participantes de cada interesse em questão. Cada vértice do grafo representa um dos componentes que realizam o interesse, enquanto as arestas representam o acoplamento existente entre os componentes. Por exemplo, a figura 4.25 demonstra o grafo de dependências do interesse referente a uma implementação do padrão *Observer*. Esta figura também mostra um trecho de código da classe *Point* a fim de ilustrar as duas arestas que conectam este vértice ao *Subject* e ao *Observer*. Em outras palavras, as áreas sombreadas do código da classe *Point* referem-se ao acoplamento com as interfaces *Subject* e *Observer*, o que resulta nas duas arestas deixando o vértice *Point*.




```
public class Point implements Subject {
   private HashSet observers;
   // código específico da aplicação suprimido
   public void addObserver(Observer o) {
         this.observers.add(o);
     }
}
```

Figura 4.25: Construindo o grafo de dependências do interesse

Após a criação dos grafos para cada interesse, a heurística de identificação concentra-se em avaliar o grafo utilizando algumas variações de algoritmos de pesquisa em grafo como o de busca em abrangência (BFS - *Breadth First Search*) e o de busca em profundidade (DFS - *Depth First Search*) (KOZEN, 1991). O Interesse Enraizado e o Tsunami podem ser identificados a partir de uma adaptação no algoritmo de busca em abrangência. A variação do BFS proposta guarda o número de vértices que podem ser atingidos por cada um. A figura 4.26 ilustra a simulação desse algoritmo no grafo de exemplo da figura 4.25. Na figura 4.26, o número de vértices alcançados por um dado vértice é colocado entre colchetes quando visitado (tona-se escuro na figura). Ao fim da execução do BFS, a origem do Tsunami é o vértice com o maior valor assinalado. Já o Interesse Enraizado é identificado usando o mesmo algoritmo, porém utiliza-se como entrada o grafo com as arestas invertidas. Neste caso, o vértice com o maior valor indica o tronco.

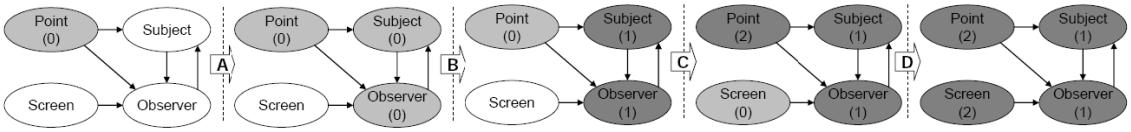


Figura 4.26: Execução ilustrada da busca em largura

Para os sintomas Rede Neural e Serpente, os algoritmos são executados sobre grafos dirigidos acíclicos (chamado de DAG - *Directed Acyclic Graph*). Figueiredo *et al.* (FIGUEIREDO et al., 2009)[6] propuseram uma implementação do DFS marcando os vértices que já foram percorridos para não refazerem a visita. O grafo de dependências do interesse, transformado em um DAG, inclui todos os componentes participantes do interesse com o sintoma Rede Neural. A rede de conexões (arestas) acíclicas entre os componentes (vértices) se assemelham a uma rede neural. Já para a identificação do Serpente, é necessário mais um passo. A variação do DFS deve marcar o vértice que possui o caminho mais longo (ou mais profundo) a partir de um outro vértice arbitrário. Em seguida, recupera-se os componentes do caminho mais longo entre dois vértices qualquer do DAG (cabeça e cauda da Serpente), sendo que todos os componentes desse caminho fazem parte do corpo da Serpente.

Como um interesse pode possuir mais de um sintoma do mesmo agrupamento, neste caso, é possível até mesmo, que um mesmo interesse possua os quatro sintomas relativos a acoplamento. A sobreposição de sintomas pode ser observada nos resultados da avaliação apresentada no capítulo 7.

4.3.4 Detecção de Clones e Análise Estrutural de Componentes

As técnicas de detecção de clone (BRUNTINK et al., 2004) são utilizadas em (FIGUEIREDO et al., 2009) como heurística para a identificação dos sintomas Cópia Carbono e Ovelha Dolly. Tal estratégia realiza uma procura por similaridades, relativas à realização do interesse, na estrutura dos componentes, que podem ser atributos, métodos, declarações ou trechos de código em geral, replicados. Esta estratégia, no entanto, ainda não foi automatizada, porém pode ser realizada "manualmente".

Para os outros dois sintomas agrupados como outros problemas de modularidade (Interesse Exclusivo de Dado e Interesse Comportamental), é proposta uma identificação baseada na análise estrutural dos componentes que realizam o interesse. Esta técnica consiste em verificar os atributos (dados) e operações (comportamentos) dos componentes participantes. Assim, um interesse cujos componentes possuem apenas atributos e eventualmente métodos acessores é classificado como Interesse Exclusivo de Dados. Similarmente, um interesse cujos componentes possuem apenas operações (por exemplo, métodos em caso de classes ou adendos em caso de aspectos) é considerado como Interesse Comportamental. Métricas orientadas a interesse, como NOCA e NOCO (vide tabela 4.1), podem ser utilizadas par apoiar a identificação destes sintomas. Entretanto, elas devem ser usadas com cuidado, pois elas não fazem distinção entre métodos acessores e demais operações.

Quanto à sobreclassificação de sintomas, vale as seguintes observações: um interesse Cópia Carbono pode ser identificado também como Ovelha Dolly. De fato, segundo a própria definição (seção 4.2.5.2), o sintoma Ovelha Dolly é uma combinação do Ovelha Negra com o Cópia Carbono. Porém, nem todo Cópia Carbono é um Ovelha Negra, e

[6]No apêndice em http://www.lancs.ac.uk/postgrad/figueire/concern/patterrns/

consequentemente, nem Ovelha Dolly. Os sintomas para Interesse Exclusivo de Dados e Comportamental são mutuamente exclusivos. Porém, é possível que eles sejam identificados também como Cópia Carbono.

4.4 Considerações Finais do Capítulo

Este capítulo apresentou um catálogo de sintomas relacionados a interesses transversais. Estes sintomas reúnem características e formas em que interesses podem se encontrar em sistemas de software. Tais sintomas foram propostos de maneira a se desprender o máximo possível de tecnologias e mecanismos específicos de programação OO e OA. Mostrou-se também heurísticas que podem ser adotadas para a detecção do sintomas descritos. Esta parte da dissertação é de fundamental importância para o entendimento das refatorações propostas no capítulo seguinte, pois a configuração destes sintomas são consideradas como oportunidades de refatoração. O capítulo a seguir apresenta, portanto, o catálogo de refatorações a fim de prover um mecanismo de apoio à modularização de interesses transversais.

5 REFATORAÇÕES DIRIGIDAS A SINTOMAS DE INTERESSES TRANSVERSAIS

Alguns trabalhos encontrados na literatura encontram-se voltados para a classificação de interesses com estruturas transversais (MARIN; MOONEN; DEURSEN, 2005b; HANNEMANN; MURPHY; KICZALES, 2005; DUCASSE; GIRBA; KUHN, 2006; FIGUEIREDO et al., 2009). Explorar como lidar com tais estruturas pode ajudar nas atividades de manutenção, como por exemplo a refatoração (SILVA et al., 2009). O suporte para detecção de sintomas relacionados a interesses é importante na medida em que indica oportunidades de refatoração. Estudos empíricos recentes têm destacado que a presença de interesses transversais é um fator determinante para a ocorrência de sintomas prejudiciais no desenho do sistema e, consequentemente, em sua implementação (FILHO et al., 2006; EADDY; AHO; MURPHY, 2007; FIGUEIREDO et al., 2008). A existência de tais sintomas recomenda que o desenho deva ser refatorado (FOWLER et al., 1999).

Este capítulo descreve as refatorações orientadas a aspectos propostas a partir da abordagem de identificação e classificação de interesses transversais baseados em sintomas (capítulo 4). Tais refatorações possuem o objetivo de apoiar a modularização de interesses transversais e encontram-se definidas em um nível de granularidade mais elevado em relação a outras refatorações de granularidade baixa, como aquelas apresentadas e discutidas no capítulo 2.

As manifestações de interesses transversais apresentadas no capítulo 4 são consideradas como sintomas que motivam a aplicação das refatorações apresentadas aqui. Vale destacar, no entanto, que não há a obrigatoriedade de aplicar as refatorações sempre que se encontrar a configuração de tais sintomas. As refatorações são propostas a fim de oferecer um apoio ao desenvolvedor ou projetista quando houver necessidade de melhor modularizar o desenho do software. A variedade das refatorações visa a cobrir a diversidade de manifestações transversais que pode-se encontrar nos interesses, ampliando assim a cobertura de necessidades anteriormente citadas. Adicionalmente, técnicas de análise de impacto podem auxiliar no processo de decisão entre refatorar ou não interesses transversais. O Capítulo 6 descreve a estratégia de análise de impacto baseada em medições proposta no contexto deste trabalho.

O restante do capítulo está organizado da seguinte maneira: a seção 5.1 discute a questão da preservação do comportamento externo em decorrência das refatorações em relação à preservação da intenção; em seguida, a seção 5.2 apresenta o formato e uma discussão preliminar das refatorações; posteriormente, as seções de 5.3 a 5.12 descrevem as refatorações; na seção 5.13 demonstra-se exemplos de aplicação das refatorações; e, por fim, a seção 4.4 conclui o presente capítulo.

5.1 Preservação do Comportamento Externo x Preservação da Intenção

Uma das propriedades discutidas na aplicação de refatorações é a questão da preservação do comportamento externo. Esta propriedade vem da própria definição de refatoração no trabalho seminal de Opdyke (1992), que diz que uma reestruturação (ou transformação) para se caracterizar como refatoração deve preservar o comportamento externo (ou observável). Trabalhos posteriores colocam a necessidade de testes durante o uso de refatorações como forma de verificar esta característica (FOWLER et al., 1999) (KERIEVSKY, 2004) (MONTEIRO; FERNANDES, 2006), isto é, avaliar se a semântica do software refatorado permanece a mesma, pelo menos do ponto de vista externo. Embora não seja uma abordagem precisa e rigorosa, a utilização de testes para verificar a preservação do comportamento externo ainda é um recurso amplamente disponível e praticado de acordo com o estudo feito sobre propostas de refatorações orientadas a aspectos no capítulo 2.

Adicionalmente, como discutido em (MONTEIRO, 2005) e (HANNEMANN; MURPHY; KICZALES, 2005) algumas reestruturações podem resultar em pequenas variações de comportamento. No entanto, isto não significa necessariamente que a intenção do desenvolvedor é comprometida. Em alguns casos, a intenção do programador pode ser substituída por variações de comportamento, revelando, inclusive, em alguns momentos, soluções melhores. Nesses casos, a preservação da intenção assume o papel de preservação do comportamento, no sentido de que as transformações realizadas preservam as funcionalidades do sistema, apesar de modificar a semântica do programa em pequenos trechos isolados. No entanto, os autores que mencionam essa questão da preservação da intenção não comentam sobre como verificá-la, sendo assim uma tarefa, a princípio, ainda subjetiva ou baseada em testes.

Recentemente a utilização de abordagens formais para provar a preservação do comportamento têm sido explorada em trabalhos sobre refatorações OO, como em (MASSONI; GHEYI; BORBA, 2005) (GHEYI; MASSONI, 2005) (SULTANA; THOMPSON, 2008). Estas iniciativas revelam-se importantes na garantia da preservação do comportamento e podem ser estendidas para aspectos. Cole e Borba (2004) definiram leis de programação OA para a derivação de refatorações na linguagem *AspectJ*. Tais leis permitem verificar se uma refatoração preserva ou não o comportamento. Entretanto, elas não estão completas para todos os recursos da linguagem *AspectJ* e ainda falta apoio ferramental.

Como acontece na maioria dos catálogos de refatoração (capítulo 2), esta fora do escopo deste trabalho propor uma técnica de preservação do comportamento. Como discutido no capítulo 8, este ponto foi definido como um dos trabalhos futuros. Portanto, espera-se que as refatorações propostas neste capítulo preservem ao menos a intenção. A preservação do comportamento é desejada e pode ser alcançada, entretanto, não foi provada no contexto das refatorações deste trabalho.

5.2 Formato e Discussão Preliminar das Refatorações

As refatorações apresentadas encontram-se organizadas em um formato padronizado baseado em trabalhos anteriores que propõem catálogos conhecidos como o de (FOWLER et al., 1999), (KERIEVSKY, 2004) e (MONTEIRO, 2005). Neste capítulo, foi reservada uma seção para cada refatoração que inclui os seguintes itens:

1. **Nome:** identifica a refatoração através da metáfora correspondente ao sintoma (os sintomas relacionados a interesses encontram-se descritos detalhadamente no capítulo 4).

2. **Situação Típica:** descreve brevemente a situação em que se configura o sintoma para se aplicar a refatoração em questão.

3. **Motivação:** apresenta problemas relacionados à presença do sintoma que motivam a aplicação da refatoração.

4. **Ação Recomendada:** resume os pasos de refatoração necessários para a eliminação ou diminuição do problema descrito na motivação.

5. **Mecânica:** guia a realização da refatoração.

Optou-se por apresentar exemplos das refatorações separadamente em uma seção, por questões de organização. A seção 5.13 apresenta alguns exemplos de refatorações para modularização de interesses transversais, como maneira de ilustrar possíveis aplicações das refatorações definidas neste capítulo, a fim de facilitar também o seu entendimento.

A mecânica de cada refatoração será sempre apresentada em função de refatorações de granularidade fina. Tais refatorações lidam com a reestruturação de elementos específicos como trechos de código, atributos, operações etc. Dessa maneira, quando aplicadas isoladamente não possuem a capacidade de modularização de interesses transversais. Entretanto, elas são essenciais no presente trabalho. Afinal, se aplicadas de forma combinada e organizada, podem compor as refatorações para modularização de interesses transversais. Inevitavelmente, recai-se sobre as refatorações de baixa granularidade para a reestruturação de elementos específicos de código, como os enumerados anteriormente. Assim, as chamadas refatorações de granularidade fina podem ser consideradas como "primitivas" das refatorações "maiores" para modularização de interesses.

Naturalmente, uma questão deve ser endereçada pelas refatorações de granularidade alta: Como usar as diversas refatorações OO e OA disponíveis na literatura combinadamente diante das diversas manifestações transversais a fim de obter uma melhor modularização dos interesses? Em quais momentos utilizar as refatorações primitivas dependerá do sintoma para o qual a refatoração está sendo motivada e, principalmente, pela manifestação do sintoma em um interesse concreto e num contexto específico. Por exemplo, não há como prever como componentes de um interesse com o sintoma Ovelha Negra estarão participando do mesmo. Em um determinado componente poderá ser através de um atributo dedicado ao interesse, em outro, através de alguma operação ou, em outro, através apenas de trechos de código isolados. Por esse motivo, as refatorações para modularização de interesses possuem passos condicionais, como pode ser observado nas seções subsequentes deste capítulo. Estes passos de refatoração são compostos por uma seleção de refatorações que podem ser vistas como primitivas que integram uma refatoração maior para a modularização de interesses transversais. Portanto, optou-se neste trabalho pelo uso de um subconjunto das refatorações de Monteiro (2005) sumarizadas na tabela 2.3 do capítulo 2. O subconjunto selecionado é formado por:

- Mover atributo de classe para intertipo;
- Mover método de classe para intertipo;
- Extrair fragmento para adendo;

- Trocar *implements* por *declare parents*;
- Trocar *extends* por *declare parents* (essa última não consta no catálogo do Monteiro, porém pode ser derivada a partir da anterior).

Em algumas situações pode-se utilizar também como apoio as refatorações do catálogo do de Fowler (1999) como Mover método, Generalizar método e Generalizar campo, por exemplo.

Por fim, a seleção das cinco refatorações menores do catálogo de Monteiro (2005) mais as refatorações de apoio do catálogo de Fowler (1999) deu-se pela relevância dos trabalhos e pelo nível de detalhe da documentação existente. Estas e outras propostas de refatorações fazem parte do estudo apresentado no capítulo 2. O catálogo do Monteiro é especificamente discutido na seção 2.2.1.

5.3 Refatoração para Ovelha Negra

Situação típica: O interesse encontra-se espalhado em pequenas porções do desenho do software, isto é, ocupa sempre a menor parte dos componentes envolvidos. Pequenas quantidades de operações e atributos dedicados à realização do interesse encontram-se espalhados entre os componentes participantes. Não há obrigatoriedade de haver herança e implementação de interfaces.

Motivação: Mesmo que pequenas porções dos componentes envolvidos estejam dedicadas ao interesse, o grau de espalhamento pode ser alto e pode crescer na medida em que o sistema evolui. É desejável, portanto, que se diminua ou elimine, se possível, o espalhamento centralizando as partes em uma única unidade modular. Assim, com o interesse melhor separado é possível obter uma implementação mais fácil de compreender e melhor preparada para mudanças e para o reuso.

Ação recomendada: Extrair para aspecto as partes espalhadas. A refatoração deve eliminar as partes envolvidas no interesse (atributos, operações, trechos de código) e assim diminuir também o grau de espalhamento do interesse sobre componentes.

Mecânica:

1. **Identificar as partes da Ovelha Negra:** identificar as partes dos componentes do interesse dedicadas à realização do mesmo.

2. **Passos de refatoração para separar as partes identificadas:**

   a) Se houver atributos relacionados ao interesse, aplicar “Mover atributo de classe para intertipo”.

   b) Se houver operações relacionadas ao interesse, aplicar “Mover método de classe para intertipo”.

   c) Se houver trechos de código relacionados ao interesse aplicar “Extrair fragmento para adendo”.

5.4 Refatoração para Polvo

Situação típica: Quando apenas alguns componentes participantes encontram-se predominantemente dedicados à realização do interesse, enquanto que os demais estão parcialmente (em sua menor parte) dedicados. Assim, o interesse revela-se bem modularizado

em um ou mais componentes (corpo do Polvo), porém espalhado em outros também (tentáculos). Não há obrigatoriedade da existência de herança ou implementação de interfaces para a manifestação do sintoma Polvo.

Motivação: Apesar de possuir partes bem modularizadas (como o corpo do Polvo), os tentáculos revelam que o interesse encontra-se espalhado por diversos componentes. Além disso, a tendência é que novos tentáculos apareçam na medida em que o sistema evolui (FIGUEIREDO et al., 2009). De acordo com estudos empíricos (FIGUEIREDO; CACHO, 2008) (SILVA et al., 2009), este sintoma pode ser responsável também pela ocorrência de outros sintomas ditos convencionais como, por exemplo, *Shotgun Surgery* e *Divergent Change* (FOWLER et al., 1999).

Ação recomendada: Separar o interesse com o sintoma Polvo e movê-lo para aspecto. Se o corpo estiver completamente dedicado ao interesse, opcionalmente movê-lo para aspecto. Caso contrário, extrair para aspecto as partes do corpo. Quanto aos tentáculos, utilizar refatorações para extrair suas partes dedicadas ao interesse modularizando-o em aspecto. Usar introduções para extrair atributos, operações, heranças e implementações de interface para aspectos. Criar conjuntos de junção e adendos para capturar pontos de junção e inserir comportamentos dinamicamente.

Mecânica:

1. **Identificar as partes do Polvo:** ao menos um corpo e dois tentáculos.

2. **Passos de refatoração para corpo:**

   a) Se o corpo estiver totalmente dedicado à realização do interesse, opcionalmente movê-lo para aspecto, pois ele já se encontra bem modularizado.

   b) Caso o corpo tenha um ou mais interesses além do Polvo, deve-se separar as partes que participam do interesse Polvo e movê-las para um novo aspecto:

      b.1) Para atributos relacionados ao interesse, aplicar "Mover atributo de classe para intertipo".

      b.2) Para operações relacionadas ao interesse, aplicar "Mover método de classe para intertipo".

      b.3) Para trechos de código relacionados ao interesse aplicar "Extrair fragmento para adendo".

      b.4) Para herança ou implementação de interfaces aplicar "Trocar *implements* por *declare parents*" ou "Trocar *extends* por *declare parents*".

3. **Passos de refatoração para tentáculos:**

   a) Se houver atributos relacionados ao interesse, aplicar "Mover atributo de classe para intertipo".

   b) Se houver operações relacionadas ao interesse, aplicar "Mover método de classe para intertipo".

   c) Se houver trechos de código relacionados ao interesse aplicar "Extrair fragmento para adendo".

   d) Se houver herança ou implementação de interfaces aplicar "Trocar *implements* por *declare parents*" ou "Trocar *extends* por *declare parents*".

5.5 Refatoração para Dominador

Situação típica: Quando o interesse é realizado por uma porção relativamente grande de componentes do sistema e que possuem uma sobrecarga de funcionalidades. Ocorre geralmente quando um interesse engloba múltiplas intenções e responsabilidades que poderiam ser subdividas em novos interesses.

Motivação: A existência de um interesse com o sintoma Dominador revela problemas de modularização no desenho. Além de estar distribuído em vários componentes, múltiplas funcionalidades estão também concentradas em um único interesse. A ocorrência de um interesse sobrecarregado de responsabilidades e intenções e espalhados sobre diversos componentes revela-se contra as ideias elementares de modularização (dividir para conquistar (KOZEN, 1991)).

Ação recomendada: Procurar subdividir o interese. Se possível, distribuir intenções correlatas sobre novos interesses, para facilitar a modularização de cada um deles. Minimizar a estrutura transversal e diminuir a quantidade de componentes envolvidos. Extrair para aspectos as partes envolvidas através do uso de introduções, conjuntos de junção e adendos.

Mecânica:

1. **Identificar possíveis "sub-interesses"**: identificar novos interesses que possam ser decompostos, a partir do Dominador, possuindo intenções bem definidas.

2. **Identificar as partes do Interesse Dominador:** "mestres" - componentes totalmente ou em sua maior parte dedicados ao interesse (alta dedicação); e os "escravos" - componentes com apenas pequenas partes dedicadas ao interesse (baixa dedicação).

3. **Passos de refatoração:**

   a) Caso tenham sido identificados novos interesses decompostos do Dominador, criar então pelo menos um aspecto para cada novo interesse.

   b) Se um componente estiver totalmente dedicado à realização do interesse, opcionalmente movê-lo para o aspecto correspondente, pois ele já se encontra bem modularizado.

   c) Caso um componente tenha outros interesses, deve-se separar as partes que participam do interesse Dominador e movê-las para o aspecto correspondente.

   d) Se houver herança ou implementação de interfaces aplicar "Trocar *implements* por *declare parents*" ou "Trocar *extends* por *declare parents*".

   e) Se houver atributos relacionados ao interesse, aplicar "Mover atributo de classe para intertipo".

   f) Se houver operações relacionadas ao interesse, aplicar "Mover método de classe para intertipo".

   g) Se houver trechos de código relacionados ao interesse aplicar "Extrair fragmento para adendo".

5.6 Refatoração para Planta Trepadeira

Situação típica: Os componentes participantes do interesse encontram-se relacionados em uma árvore de herança e/ou implementação de interfaces.

Motivação: O principal problema existente em um interesse Planta está relacionado a dependências implícitas envolvendo os componentes participantes (raiz e galhos da Planta). Estas dependências tornam-se evidentes quando uma mudança em um galho propaga-se através da raiz para outros galhos. Por exemplo, eventualmente a mudança na assinatura de um método em um dos galhos e que é herdado da raiz, ou de algum galho ancestral na hierarquia, pode provocar mudanças também nesse ancestral. Consequentemente, mudanças podem ocorrer também nos demais descendentes (galhos) envolvidos na estrutura. Tal efeito pode ser evitado caso o interesse esteja modularizado em aspectos.

Ação recomendada: Eliminar da estrutura hierárquica a implementação do interesse. Caso a hierarquia exista exclusivamente para a realização do interesse, deve-se removê-la e colocá-la em aspecto através de introduções.

Mecânica:

1. **Identificar as partes da planta:** ao menos uma raiz e dois galhos.

2. **Passos de refatoração para a raiz:**

   a) Se a raiz estiver totalmente dedicada à realização do interesse, opcionalmente movê-la para aspecto, pois ela já se encontra bem modularizada.

   b) Caso a raiz tenha um ou mais interesses, deve-se separar as partes que participam do interesse Planta e movê-las para um novo aspecto:

      b.1) Para atributos relacionados ao interesse, aplicar "Mover atributo de classe para intertipo".

      b.2) Para operações relacionadas ao interesse, aplicar "Mover método de classe para intertipo".

      b.3) Para trechos de código relacionados ao interesse aplicar "Extrair fragmento para adendo".

3. **Passos de refatoração para os componentes galhos:**

   a) Se houver atributos relacionados ao interesse, aplicar "Mover atributo de classe para intertipo".

   b) Se houver operações relacionadas ao interesse, aplicar "Mover método de classe para intertipo".

   c) Se houver trechos de código relacionados ao interesse aplicar "Extrair fragmento para adendo".

   d) Se houver herança ou implementação de interfaces **relacionada ao interesse** aplicar "Trocar *implements* por *declare parents*" ou "Trocar *extends* por *declare parents*". É importante observar aqui que, sendo o componente um galho, existirá obviamente uma declaração de herança ou implementação de interface com algum outro componente (que pode ser um outro galho ou a raiz da Planta). Entretanto, só é indicada a troca de *extends/implements* para *declare parents* em aspecto se a declaração de herança ou implementação de interface for relacionada ao interesse, isto é, se existir exclusivamente para a implementação do interesse.

5.7 Refatoração para Doença Hereditária

Situação típica: Neste caso, os componentes participantes da realização do interesse também fazem parte de uma árvore de herança ou implementação de interfaces, de maneira semelhante à seção anterior (5.6). Porém, a diferença está centrada na existência de componentes na estrutura hierárquica que não estão dedicados ao interesse.

Motivação: Apesar de neste caso existirem componentes na estrutura hierárquica que não fazem parte da realização do interesse, o mesmo problema discutido na motivação da refatoração para Planta Trepadeira (seção 5.6) pode ser observado aqui. Mudanças em algum descendente doente podem provocar também mudanças nos demais. Tal efeito pode ser evitado caso o interesse esteja modularizado em aspectos.

Ação recomendada: A iniciativa é a mesma da refatoração para Planta Trepadeira (seção 5.6). Eliminar da estrutura hierárquica a implementação do interesse. Caso a hierarquia exista exclusivamente para a realização do interesse, deve-se removê-la e colocá-la em aspecto através de introduções. A diferença está na existência de componentes que são descendentes sadios, por isso um cuidado adicional deve ser tomado para que estes não sejam afetados.

Mecânica:

1. **Identificar os participantes da Doença Hereditária:** origem da doença e descendentes divididos em sadios e doentes.

2. **Passos de refatoração para a origem da doença:**

   a) Se o componente origem da doença estiver totalmente dedicado à realização do interesse opcionalmente movê-lo para aspecto, pois ele já se encontra bem modularizado.

   b) Caso o componente tenha um ou mais interesses, deve-se separar as partes que participam do interesse Doença Hereditária e movê-las para um novo aspecto:

      b.1) Se houver atributos relacionados ao interesse, aplicar "Mover atributo de classe para intertipo".

      b.2) Se houver operações relacionadas ao interesse, aplicar "Mover método de classe para intertipo".

      b.3) Se houver trechos de código relacionados ao interesse aplicar "Extrair fragmento para adendo".

3. **Passo de refatoração para descendentes sadios:**

   a) Caso seu ancestral direto tenha sido movido completamente para aspecto, mover a declaração de herança (ou de implementação de interface) para aspecto como introdução. Aplicar "Trocar *implements* por *declare parents*" ou "Trocar *extends* por *declare parents*". Caso contrário, é recomendável manter a declaração de herança ou implementação de interface, pois se o componente ancestral não foi completamente movido, significa que apenas parte dele estava dedicado ao interesse e assim a relação hierárquica nesse ponto pode estar relacionada também a outro interesse. O descendente sadio, apesar de estar na hierarquia encontra-se dedicado a um outro interesse que não o Doença Heteditária juntamente com o seu ancestral (ou ancestrais).

4. **Passos de refatoração para descendentes doentes:**

a) Se houver atributos relacionados ao interesse, aplicar “Mover atributo de classe para intertipo”.

b) Se houver operações relacionadas ao interesse, aplicar “Mover método de classe para intertipo”.

c) Se houver trechos de código relacionados ao interesse aplicar “Extrair fragmento para adendo”.

d) Se houver herança ou implementação de interfaces aplicar “Trocar *implements* por *declare parents*” ou “Trocar *extends* por *declare parents*”.

5.8 Refatoração para Sintomas Relativos a Acoplamento

Diferentemente das demais, esta seção define uma mecânica de refatoração unificada para os quatro sintomas relativos a acoplamento. Portanto, primeiramente apresenta-se a situação típica, motivação e ação recomendada para refatoração de cada sintoma relativo a acoplamento e, por fim, descreve-se a mecânica de refatoração que é única para os quatro casos. Nesta categoria de sintomas, optou-se por esse formato pois os passos de refatoração para os quatro casos são exatamente os mesmos.

5.8.1 Refatoração para Interesse Enraizado

Situação típica: Componentes envolvidos na realização do interesse encontram-se relacionados entre si de maneira em que convergem conexões de acoplamento para um único componente, chamado de tronco, que não possui relação de acoplamento com nenhum outro participante. Os demais componentes são chamados de ramos. Não há obrigatoriedade de haver relacionamentos de herança.

Motivação: Este sintoma remete a problemas de acoplamento, uma vez que se tem diversos componentes relacionados entre si (ramos) e que convergem referências a um único (o tronco). Por isso, é possível observar que as atividades de manutenção relacionadas ao interesse podem ser prejudicadas pela existência de diversos pontos distribuídos envolvidos na realização do mesmo e pela existência de um componente central que converge as conexões de acoplamento. Mudanças no tronco, por exemplo, podem ocasionar propagação de problemas nos ramos.

Ação recomendada: Quando o tronco for um componente totalmente dedicado ao interesse, transformá-lo em aspecto é uma tarefa opcional. Entretanto, caso seja parcialmente dedicado, recomenda-se separar a parte do Interesse Enraizado em um novo aspecto. Quanto aos ramos, recomenda-se também extrair para aspecto as partes envolvidas na realização do interesse a fim de centralizá-las em uma única unidade modular.

5.8.2 Refatoração para Tsunami

Situação típica: Inversamente à situação do Interesse Enraizado, no Tsunami, um único componente (origem da onda ou epicentro) possui relações de acoplamento com outros componentes (ondas). Estes, por sua vez, podem estar acoplados com outros também e assim sucessivamente. Não há obrigatoriedade de haver relacionamentos de herança.

Motivação: Além de possuir um problema de acoplamento entre componentes, assim como no Interesse Enraizado, neste caso, o componente epicentro depende diretamente ou indiretamente de diversos outros componentes. Eventualmente, durante manutenções relacionadas ao interesse, modificações em componentes-ondas podem ocasionar também mudanças no epicentro. Adicionalmente, compreender o comportamento do componente

epicentro, pode significar compreender também as inúmeras relações de acoplamento que podem existir a partir dele.

Ação recomendada: De forma semelhante à ação recomendada para o Interesse Enraizado, caso a origem da onda seja totalmente dedicada ao interesse, opcionalmente transformá-la em aspecto. Entretanto, caso seja parcialmente dedicada, recomenda-se separar a parte referente ao interesse em um novo aspecto. Quanto aos demais componentes (ondas), recomenda-se também modularizar em aspecto as partes envolvidas na realização do interesse.

5.8.3 Refatoração para Serpente

Situação típica: Quando há um conjunto de componentes que formam uma cadeia acíclica de conexões de acoplamento. Como descrito na seção 4.2.4.3, o sintoma Serpente refere-se à maior cadeia existente. Na maioria das vezes esta situação ocorre em conjunto com outros sintomas como o Interesse Enraizado ou Tsunami.

Motivação: Assim como nas duas refatorações anteriores (seções 5.8.1 e 5.8.2), os problemas referentes a acoplamento são os mesmos. Neste caso específico, o problema pode ser resolvido pela eliminação desta cadeia de referências de acoplamento que existe especificamente para a realização do interesse através da utilização de aspectos.

Ação recomendada: Extrair para um aspecto a existência da maior cadeia acíclica de referências entre os componentes que configuram o sintoma Serpente. O aspecto deve concentrar e inserir as relações de acoplamento a fim de separar o interesse. Caso o sintoma Serpente exista em conjunto com o Interesse Enraizado ou Tsunami, cabe ao projetista ou desenvolvedor decidir qual a melhor refatoração a ser aplicada.

5.8.4 Refatoração para Rede Neural

Situação típica: Semelhantemente à refatoração para Serpente, devem existir relações de acoplamento entre os componentes envolvidos no interesse, porém organizados em forma de uma **rede** acíclica de conexões. Neste caso, os componentes participantes não se configuram como Tsunami, nem como Interesse Enraizado, pois não há um componente que faça o papel de epicentro nem de tronco. Não se configuram também como Serpente, pois os envolvidos não formam uma cadeia de relações de acoplamento, mas sim uma rede (emaranhado).

Motivação: Como nas situações anteriores para os sintomas relativos a acoplamentos (seções 5.8.1, 5.8.2 e 5.8.3), é desejável a diminuição do acoplamento entre componentes participantes de cada interesse em questão. Especificamente para este caso, pode-se evitar o emaranhado de referências de acoplamento existente, que por si só, traz prejuízos na modularização do sistema e, especialmente, afeta a capacidade de manutenção e compreensão do interesse.

Ação recomendada: A rede de componentes formada pelas suas relações de acoplamento deve ser eliminada ou diminuída pela utilização de aspectos. O interesse deve deixar de ser o responsável pelo emaranhado de conexões de acoplamento existente em parte do desenho do software.

5.8.5 Mecânica de Refatoração para Sintomas Relativos a Acoplamento

1. **Identificar os componentes envolvidos:** tronco e ramos para Interesse Enraizado; epicentro e ondas para Tsunami; componentes que fazem parte da maior cadeia (interconexões) acíclica de componentes para Serpente; e neurônios que compõem

a camada de entrada, a(s) intermediária(s) e a de saída para Rede Neural.

2. **Passos de refatoração:**

   a) Se o componente estiver totalmente dedicado à realização do interesse, opcionalmente movê-lo para aspecto, pois ele já se encontra bem modularizado.

   b) Caso o componente tenha outros interesses, deve-se separar as partes que participam do Interesse Enraizado e movê-las para um novo aspecto:

      b.1) Se houver atributos relacionados ao interesse, aplicar "Mover atributo de classe para intertipo".

      b.2) Se houver operações relacionadas ao interesse, aplicar "Mover método de classe para intertipo".

      b.3) Se houver trechos de código relacionados ao interesse aplicar "Extrair fragmento para adendo".

5.9 Refatoração para Cópia Carbono

Situação típica: Componentes possuem partes dedicadas à realização do interesse replicadas entre eles. Essas partes podem ser tanto dados (atributos) ou comportamentos (operações) ou até mesmo trechos de código dentro de operações. Esta situação se assemelha com o sintoma Duplicação de Código proposto por Fowler (1999). Entretanto, o Cópia Carbono refere-se a duplicações na implementação de um interesse, enquanto o sintoma de Fowler (1999) não necessariamente.

Motivação: Duplicações podem ocorrer como resultado de práticas de copiar-e-colar, por exemplo, e que aumentam os custos das atividades de manutenção (FOWLER et al., 1999). Fazer modificações simples em tais replicações como renomear um atributo implica em mudar também o nome das réplicas, por exemplo. Tal situação pode prejudicar a manutenibilidade do interesse no sistema. Exemplos como este poderiam ser eliminados concentrando uma única cópia em aspecto e introduzindo-a em diversas partes quando necessário através de conjuntos de pontos de junção e adendo.

Ação recomendada: Eliminar as cópias (partes que se encontram replicadas). Cópias estruturais (declaração de atributos e métodos completos) podem ficar concentradas em aspectos através de introduções sem a necessidade de pontos de junção. Para cópias comportamentais, deve-se utilizar conjuntos de pontos de junção para capturar os locais das cópias e introduzí-las através de adendo. Deve-se tentar criar conjuntos de junção que possam capturar o máximo possível de pontos, ao invés de um conjunto de junção para cada ponto a ser introduzida uma cópia.

Mecânica:

1. **Identificar as partes do Cópia Carbono:** as cópias podem ser comportamentais (fragmentos de código, chamadas para métodos ou outras declarações que indicam comportamento) ou estruturais (declaração de atributos e métodos completos).

2. **Passos de refatoração para as cópias comportamentais:**

   a) Se os componentes que possuem fragmentos copiados estão envolvidos em uma relação de herança, então indica-se aplicar "Extrair método" para a criação de um método com o fragmento de código replicado. Na sequência, deve-se

realizar “Generalizar método” no método extraído para mover a cópia transformada em método para a superclasse na relação de herança. A partir de então o fragmento que antes estava replicado deverá ser um método herdado e acessível em um único ponto pelos descendentes através da relação de herança. As demais cópias podem ser removidas, em seguida.

b) Caso contrário, extrair conjuntos de pontos de junção para capturar os locais das cópias comportamentais dinamicamente. Posteriormente, mover para aspecto as cópias criando um adendo para introduzí-las através dos conjuntos de junção criados. Para este passo, sugere-se a refatoração “Extrair fragmento para adendo”. Alternativamente, pode-se optar por extrair o trecho de código para um novo método. Neste caso, usa-se a refatoração “Extrair método” para expor o trecho copiado em forma de método e enquadrá-lo no passo 3.a) para cópias estruturais.

3. **Passos de refatoração para as cópias estruturais:**

   a) Se os componentes possuem métodos copiados e estiverem envolvidos em uma relação de herança, então indica-se aplicar “Generalizar método” em uma das cópias, para que o método replicado passe a estar em um único ponto e acessível em seus descendentes através da relação de herança. As demais cópias podem ser, em seguida, removidas.

   b) Se os componentes possuem atributos copiados e estiverem envolvidos em uma relação de herança, então indica-se aplicar “Generalizar campo” em uma das cópias, para que o atributo replicado passe a estar em um único ponto e acessível em seus descendentes através da relação de herança. As demais cópias podem ser, em seguida, removidas.

   c) Caso os componentes que possuem as cópias não estejam em uma relação de herança, deve-se aplicar “Mover atributo de classe para intertipo” para métodos copiados e aplicar “Mover método de classe para intertipo” para atributos copiados.

5.10 Refatoração para Ovelha Dolly

Situação típica: Refere-se a uma situação semelhante ao da refatoração anterior (seção 5.9), com a adição de que o interesse em questão deve ter também o sintoma Ovelha Negra. Assim, os componentes participantes terão que suas partes realizadoras do interesse encontram-se replicadas e espalhadas em pequenas proporções nos diversos componentes. Uma combinação da situação de refatoração para Ovelha Negra e Cópia Carbono, pois este seria justamente um caso particular de Cópia Carbono que também é Ovelha Negra.

Motivação: Da mesma forma que na refatoração para Cópia Carbono (seção 5.9), deve-se evitar as partes replicadas. Além disso, este sintoma remete também aos problemas discutidos na motivação para a refatoração da Ovelha Negra (seção 5.3).

Ação recomendada: Neste caso, a ação recomendada pode ser a mesma da refatoração para Cópia Carbono. A única diferença será que necessariamente tem-se cópias que ocupam pequenas partes dos componentes envolvidos, porém não há nenhuma restrição na refatoração para Cópia Carbono quanto a essa peculiaridade do Ovelha Dolly. Poderia-se aplicar também a refatoração para Ovelha Negra, afinal o sintoma Ovelha Dolly tam-

bém é um Ovelha Negra. Entretanto, a refatoração para Cópia Carbono tem um estratégia que melhor se aplica para a existência de cópias.

Mecânica: Assumir a mecânica de refatoração para Cópia Carbono.

5.11 Refatoração para Interesse Exclusivo de Dados

Situação típica: Os componentes que realizam o interesse possuem atributos e nenhum comportamento associado. É possível que existam métodos acessores para os atributos dos componentes, porém nenhum comportamento adicional. Isto pode acontecer em decorrência de estratégias incorretas de modularização ou é um indicativo de que o desenho do software não está ainda completo.

Motivação: A falta de comportamento nos componentes participantes do interesse revela o uso impróprio da modularização (FIGUEIREDO et al., 2009). De maneira similar aos objetos, um interesse deve possuir dados e comportamentos. A falta de operações "funcionais" pode indicar que dados e comportamentos correlacionados não encontram-se em um único lugar. Dessa forma, mudanças nos atributos dos componentes do interesse poderão afetar outros componentes que utilizam estes dados. Adicionalmente, deve-se levar em consideração que é mais fácil compreender um interesse e seus participantes quando se tem dados e comportamentos associados em poucas ou em uma única unidade modular.

Ação recomendada: Recomenda-se uma revisão na estratégia de modularização que pode culminar em duas alternativas: a primeira utilizando aspectos e a segunda utilizando apenas objetos. (i) Seguindo a primeira opção, deve-se então extrair para aspecto os atributos referentes ao Interesse Exclusivo de Dados nas classes, a fim de pelo menos mantê-los em uma única unidade modular (aspecto) através de introduções. Sugere-se verificar se as introduções não se encaixam em algum aspecto existente. Caso contrário, deve-se criar um novo aspecto que terá as novas introduções. Eventualmente, o aspecto criado poderá incorporar comportamentos. (ii) Pela segunda opção, verificar se os atributos dos componentes do Interesse Exclusivo de Dados não podem ser incorporados por uma ou mais classes existentes no desenho. Isto requer uma reavaliação estrutural e conceitual dos demais componentes. Esta opção é mais recomendada caso existam poucos atributos ou apenas um componente envolvido no interesse.

Mecânica:

1. **Identificar as partes do Interesse de Dados:** atributos e métodos acessores dos atributos.

2. **Passos de refatoração para os atributos (duas alternativas para este caso):**

   a) Mover atributos e métodos acessores correspondentes para aspecto através de introduções. Verificar se as introduções se encaixam em algum aspecto existente. Caso contrário cria-se um novo aspecto. Recomenda-se as refatorações "Mover atributo de classe para intertipo" para os atributos e "Mover método de classe para intertipo" para os métodos acessores, quando houver.

   b) Alternativamente, mover atributos e métodos acessores para outras classes. Verificar se há a possibilidade de outras classes incorporarem os atributos em questão. Decisões de projeto como estas dependem de julgamento do projetista ou desenvolvedor responsável. Neste caso, recomendada-se as refatorações "Mover atributo" e "Mover método" (FOWLER et al., 1999).

5.12 Refatoração para Interesse Comportamental

Situação típica: Neste caso, acontece o inverso da situação para a Refatoração de Interesse de Dados (seção 5.11). Os componentes que participam do interesse definem apenas comportamentos, isto é, operações para a realização do interesse sem possuir algum atributo associado.

Motivação: De maneira similar à seção anterior (5.11), o problema concentra-se na utilização imprópria de propriedades de modularização: um pela falta de comportamento, e este pela falta de dados. Um interesse deve ter associado a ele um ou mais componentes, parcialmente ou completamente dedicados ao mesmo, contendo dados e comportamentos relacionados à sua realização. Por exemplo, classes que implementam um interesse e apenas com métodos, utilizam sempre dados de outras classes. Desta maneira a manutenção nas demais classes poderá requerer mudanças nas classes exclusivamente comportamentais. Além disso, existe também a questão da compreensão, como comentado na seção anterior - é mais fácil compreender um interesse e seus componentes parcticipantes quando se tem os dados e comportamentos associados em poucas ou em uma única unidade modular.

Ação recomendada: Semelhantemente à seção 5.11, recomenda-se também uma revisão na estratégia de modularização que pode se resumir em duas alternativas. (i) A primeira consiste em extrair para aspecto os métodos dos componentes referentes ao Interesse Comportamental, a fim de pelo menos mantê-los em uma única unidade modular (aspecto) através de introduções. Deve-se verificar também se as introduções não se encaixam em algum aspecto existente. Caso contrário, deve-se criar um novo aspecto que terá as novas introduções. Eventualmente, o aspecto criado poderá incorporar dados. (ii) A segunda alternativa consiste em verificar se os métodos dos componentes do Interesse Comportamental não podem ser incorporados por uma ou mais classes existentes no desenho. Isto requer também uma reavaliação estrutural e conceitual dos demais componentes. Esta opção é mais recomendada caso existam poucos métodos ou apenas um componente envolvido no interesse.

Mecânica:

1. **Identificar as partes do Interesse Comportamental:** operações (métodos).
2. **Passos de refatoração para as operações (duas alternativas para este caso):**

   a) Mover operações para aspecto através de introduções. Verificar se as operações se encaixam em algum aspecto existente. Caso contrário cria-se um novo aspecto. Recomenda-se a refatoração "Mover método de classe para intertipo".

   b) Alternativamente, mover operações para outros componentes. Verificar se há a possibilidade de outros componentes incorporarem as operações em questão. Decisões de projeto como estas dependem de julgamento do projetista ou desenvolvedor responsável. Neste caso, recomenda-se a refatoração "Mover método" (FOWLER et al., 1999).

5.13 Exemplificação das Refatorações

Esta seção tem como objetivo reunir alguns exemplos de aplicação das refatorações apresentadas neste capítulo. Vale ressaltar que os exemplos referem-se isoladamente à aplicação das refatorações e não incluem outras etapas necessárias no método de refatoração como um todo (detecção de oportunidades e análise de impacto, por exemplo).

Pretende-se, portanto, demonstrar como os passos de refatoração para interesses transversais sintomáticos podem ser aplicados em algumas situações concretas. A aplicação de cada passo não é detalhada, pois consiste na realização de uma refatoração "primitiva" descrita em seu catálogo específico (catálogo de Fowler ou Monteiro, por exemplo).

5.13.1 Refatorando o Interesse *Favourites* no *Mobile Media*

A fim de exemplificar a refatoração de um interesse com o sintoma Ovelha Negra, toma-se como base o interesse *Favourites* do sistema *Mobile Media* apresentado na figura 4.3 (seção 4.2.2.1). Como ilustrado, a implementação do interesse *Favourites* encontra-se distribuída em pequenas partes de 5 componentes, isto é, ocupa menos de 33% de cada uma das 5 classes. A figura 5.1 ilustra a refatoração para Ovelha Negra aplicada à classe *MediaData*. Neste caso, os passos 2.a e 2.b da refatoração para Ovelha Negra foram realizados sobre o atributo e métodos destacados em cinza na figura.

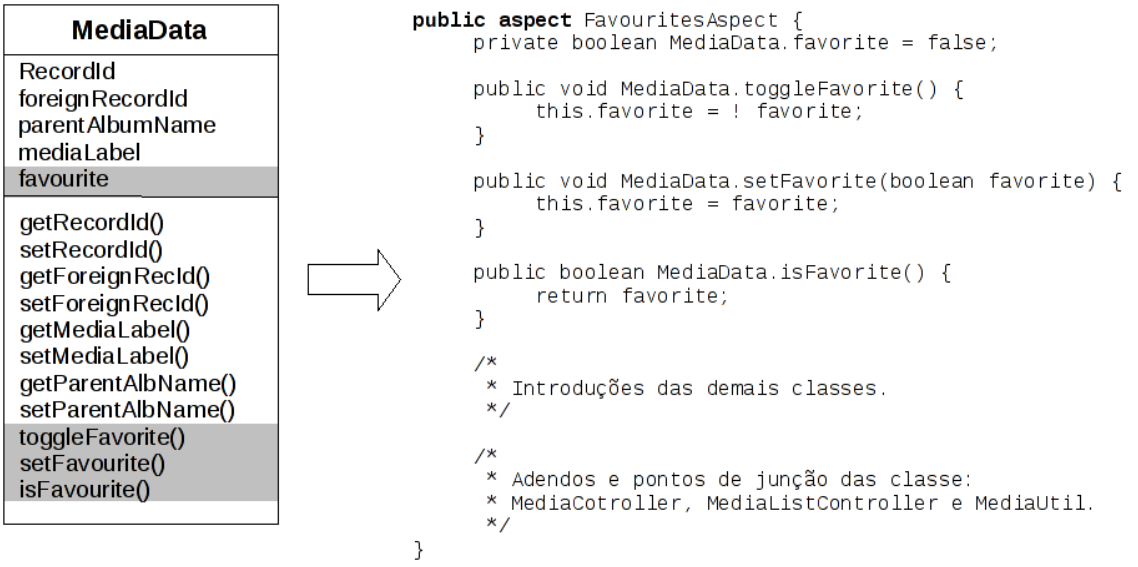


Figura 5.1: Refatoração da classe *MediaData* para o interesse *Favourites*

Os mesmos passos devem ser aplicados sobre as demais classes para modularizar no aspecto o interesse *Favourites*. O passo 2.a da refatoração deve ser aplicado somente sobre as classes *MediaController*, *MediaListController* e *MediaUtil*, pois são as classes que possuem fragmentos de código relacionados ao interesse Ovelha Negra (vide figura 5.2). Assim, o aspecto da figura 5.1 seria completado e as partes dedicadas ao interesse *Favourites* não estariam mais espalhadas pelos demais componentes e nem mesmo entrelaçadas com outras intenções e responsabilidades internas de cada classe participante, pois o interesse passa a estar modularizado em aspecto como mostra a figura 5.2. A existência ou não de relações de herança não influencia neste tipo de sintoma.

5.13.2 Refatorando o Interesse *Copy* no *Mobile Media*

O interesse *Copy* existente no sistema *Mobile Media* tem como objetivo realizar cópias dos arquivos de mídia existentes no dispositivo móvel. Para este exemplo, extraiu-se o interesse *Copy* na versão 4 do *Mobile Media*. Nesta versão, a cópia é realizada apenas em mídias do tipo foto, pois os demais tipos de mídia (como vídeo e música) somente foram introduzidos nas versões subsequentes. A figura 5.3 ilustra um diagrama com os componentes participantes do interesse. A classe *PhotoViewController* encontra-se desenhada

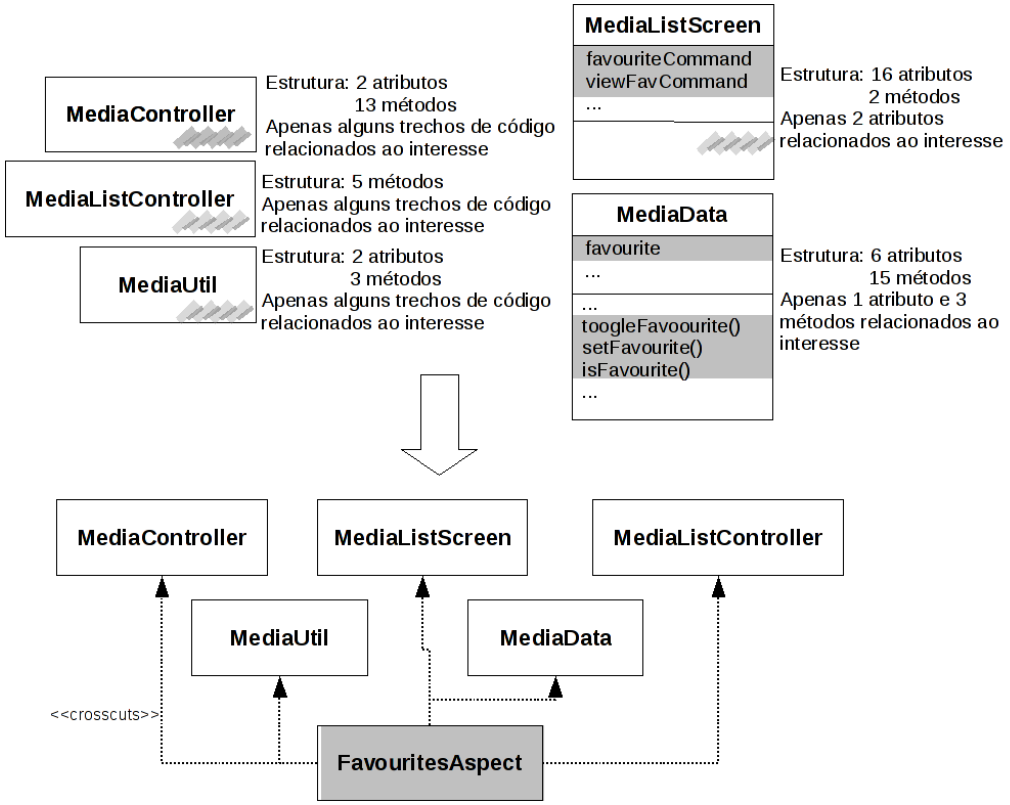


Figura 5.2: Componentes após refatoração do *Favourites*

na cor cinza para demonstrar que está totalmente sombreada, isto é, completamente dedicada ao interesse. Já as demais classes (*AlbumData*, *ImageAccessor*, *PhotoController*, *PhotoViewScreen*) estão parcialmente dedicadas ao interesse, e por isso estão marcadas mas não preenchidas completamente de cinza. A dedicação parcial nessas quatro classes classifica-se como baixa, pois cada uma possui menos que um terço de seus membros relacionados ao interesse. Dessa maneira, o *Copy* pode ser classificado como um interesse Polvo, sendo a classe *PhotoViewController* o corpo e as demais classes tentáculos.

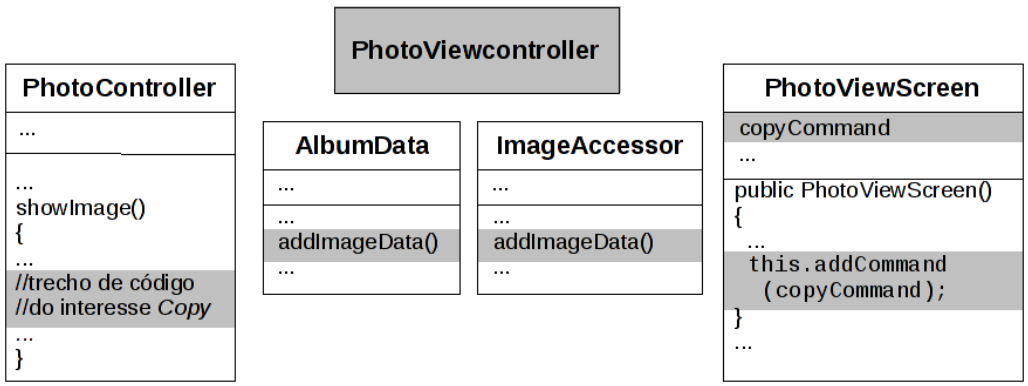


Figura 5.3: Classes participantes do interesse *Copy*

Neste exemplo, o corpo está completamente dedicado ao interesse e, portanto, é opcional movê-lo para aspecto (passo 2.a da refatoração para Polvo). Optou-se por não refatorar esta classe. Em seguida, voltando-se para os tentáculos, deve-se verificar se os passos 3.a a 3.d se aplicam a cada componente. A figura 5.4 demonstra a realização do

passo 3.c que se aplica à classe *PhotoController*. A região sombreada na classe corresponde ao fragmento de código dedicado à realização do interesse neste componente e que foi extraído para aspecto. No aspecto *CopyAspect*, primeiramente foi criado um conjunto de pontos de junção para capturar a chamada ao método *setPhotoScreen()*. Como o método *setPhotoScreen()* é o ponto seguinte ao trecho de código sombreado, criou-se, portanto, um adendo para adicionar o fragmento extraído antes da chamada ao método *setPhotoScreen()*.

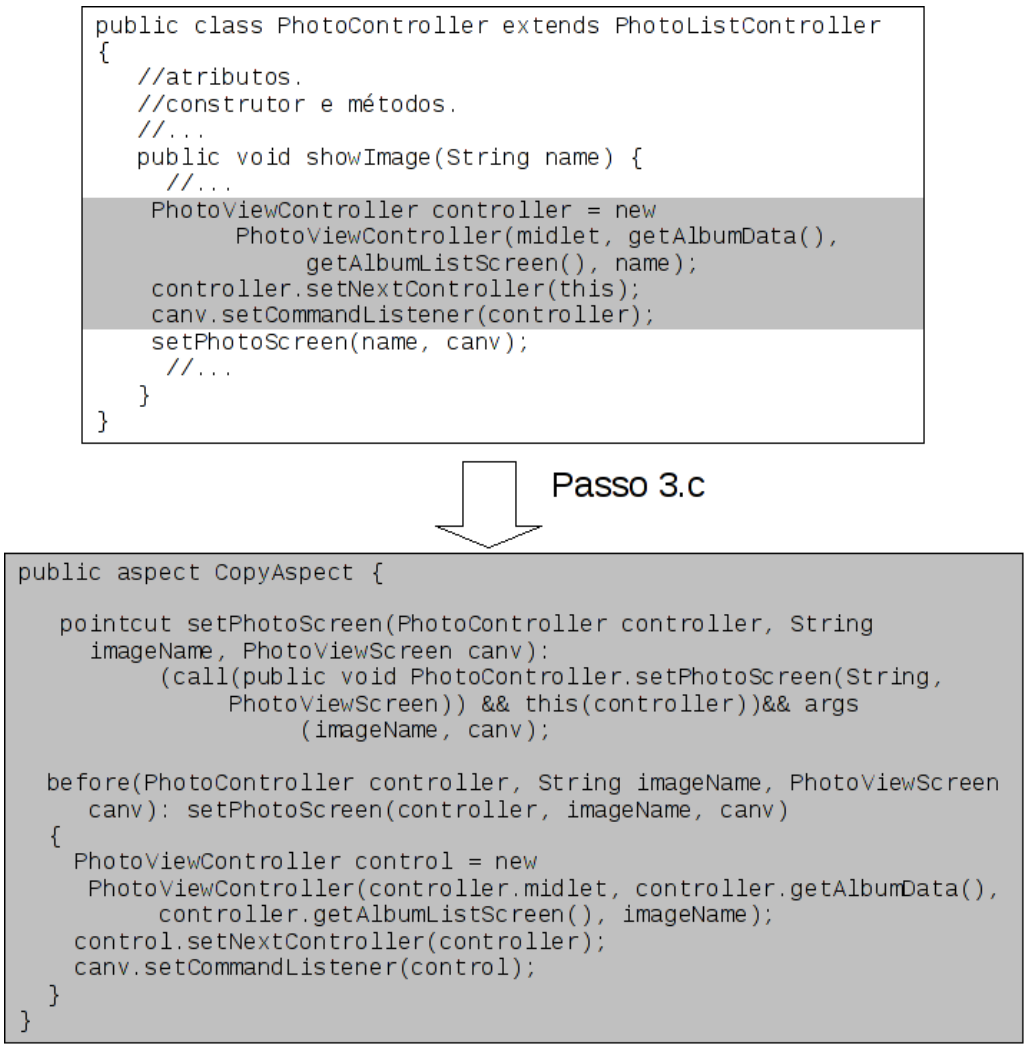


Figura 5.4: Refatorando o tentáculo *PhotoController*

Para a classe *PhotoViewScreen* realizam-se dois passos: o passo 3.a para mover o atributo *copyCommand* para aspecto como uma declaração intertipo; e, em seguida, o passo 3.c para extrair o fragmento de código relacionado ao interesse que consta no construtor da classe (figura 5.3). Na verdade, a aplicação da refatoração "Mover atributo de classe para intertipo" de Monteiro no passo 3.a já considera a possibilidade de haver trechos de código usando o atributo movido e sugere a refatoração "Extrair fragmento para adendo". Portanto, a própria aplicação do passo 3.a já deve indicar a realização do passo 3.c, pois o fragmento utiliza o atributo movido.

Já para as classes *AlbumData* e *ImageAccessor* deve-se aplicar o passo 3.b para mover os métodos relacionados ao interesse para aspecto. Como resultado, o aspecto criado para modularizar o interesse *Copy* é mostrado na figura 5.5. Nesse momento, além do aspecto

criado, apenas a classe *PhotoController* (ex-componente corpo) encontra-se dedicada ao interesse. As partes relacionadas ao interesse dos componentes tentáculos foram movidas para aspecto. Consequentemente, o *Copy* deixa ser um interesse Polvo.

```
public class CopyAspect {

   pointcut setPhotoScreen(PhotoController controller, String
     imageName, PhotoViewScreen canv):
          (call(public void PhotoController.setPhotoScreen(String,
               PhotoViewScreen)) && this(controller))&& args
                    (imageName, canv);

  before(PhotoController controller, String imageName, PhotoViewScreen
     canv): setPhotoScreen(controller, imageName, canv)
  {
    //techo de código extraído da classe PhotoController
  }

 public void ImageAccessor.addImageData(String photoname, ImageData
imageData, String albumname) throws InvalidImageDataException,
PersistenceMechanismException {
     //...
  }

 public void AlbumData.addImageData(String photoname, ImageData
imageData, String albumname)throws InvalidImageDataException,
PersistenceMechanismException{
    //...
  }

 public static final Command copyCommand = new Command("Copy",
Command.ITEM, 1);

 pointcut constructor(Image image) :
    call(PhotoViewScreen.new(Image)) && args(image);

  after(Image image) returning (PhotoViewScreen f): constructor(image)
  {
    f.addCommand(copyCommand);
  }
}
```

Figura 5.5: Aspecto resultante da refatoração

5.13.3 Refatorando *Observer* no *Health Watcher*

Nesta seção, retoma-se como exemplo a existência do padrão de projeto *Observer* no sistema *Health Watcher* (SOARES; LAUREANO; BORBA, 2002). A figura 5.6, já previamente apresentada na seção 4.1, refere-se a um recorte de desenho que possui a realização deste padrão. Os componentes que participam da implementação do padrão dividem-se em dois papéis: *subject*, para os componentes observados que devem manter uma coleção de observadores e os notifica na ocorrência de mudanças de estado; e *observer*, para os componentes observadores. Neste caso, a modularização do interesse referente à impementação do *Observer* será dividida em duas partes: primeiro, serão refatorados os componentes que possuem o interesse do papel *subject* e, em seguida, serão refatorados os componentes que possuem o interesse do papel *observer*.

5.13.3.1 Refatorando o Papel subject com o Sintoma Planta Trepadeira

Como visto na seção 4.2.3.1, o interesse do papel *Subject* configura-se com o sintoma Planta. Neste caso, a raiz é a interface de nome *Subject* e os galhos são as classes *Com-*

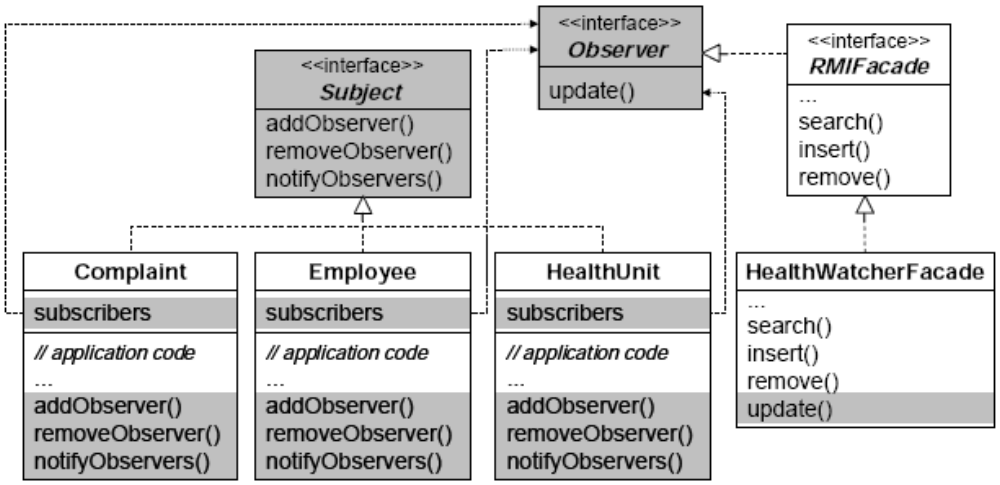


Figura 5.6: Um exemplo de desenho mostrando o padrão *Observer*

plaint, *Employee* e *HealthUnit*. Neste exemplo, a refatoração torna-se semelhante à do interesse Polvo, porém, no caso do interesse Planta, é necessário a existência da árvore de herança, enquanto que para o interesse Polvo não é obrigatório. Portanto, seguida a refatoração para interesse Planta, posterior a identificação dos componentes envolvidos, deve-se optar por refatorar ou não a raiz. Como a interface *Subject* não é utilizada por mais nenhum outro componente além dos galhos, optou-se por movê-la para aspecto de acordo com o passo 2.a.

A figura 5.7 ilustra trechos de código de um componente galho - classe *HealthUnit*. A região sombreada demonstra as partes referentes à implementação do interesse que se encontram entrelaçadas com demais interesses da classe. Seguindo a refatoração para os galhos, aplicou-se os passos: 3.a, para mover para aspecto o atibuto *subscribers* referente à lista de observadores mantida pelos sujeitos observados (os galhos); 3.b, para mover os métodos referentes à manutenção da coleção de observadores; e 3.d, para mover a implementação da interface *subject*.

Em seguida, aplicou-se o passo 3.c, para capturar os locais em que há mudança de estado dos sujeitos observados e a realização da notificação a partir do aspecto. No exemplo da classe *HealthUnit* a mudança de estado ocorre quando há mudança de descrição (chamada ao método *setDescription()*). Portanto, criou-se um conjunto de junção para capturar este ponto dinamicamente. A parte inferior da figura 5.8 ilustra este conjunto de junção e os demais para as outras classes. Além disso, o passo 3.c cria também um adendo necessário para chamar o método *notifyObservers()* a fim de introduzir o comportamento referente à notificação aos observadores. A parte superior da figura 5.8 demonstra graficamente que as classes não estão mais cientes do interesse *Subject*, que agora está modularizado no aspecto.

5.13.3.2 Refatorando o Papel observer com o Sintoma Doença Hereditária

Na figura 5.6, além de existir o sintoma Planta (pelo papel *Subject*), é possível visualizar também o sintoma Doença Hereditária sobre o interesse do papel *observer* relacionado ao padrão de projeto de mesmo nome. A árvore que se inicia pela interface *Observer* possui um componente que não implementa o interesse (a interface *RMIFacade*), configurando assim o interesse como Doença Hereditária. A interface *Observer*, portanto, seria a origem da doença, a interface *RMIFacade* um descendente sadio e a classe *HealthWat-*

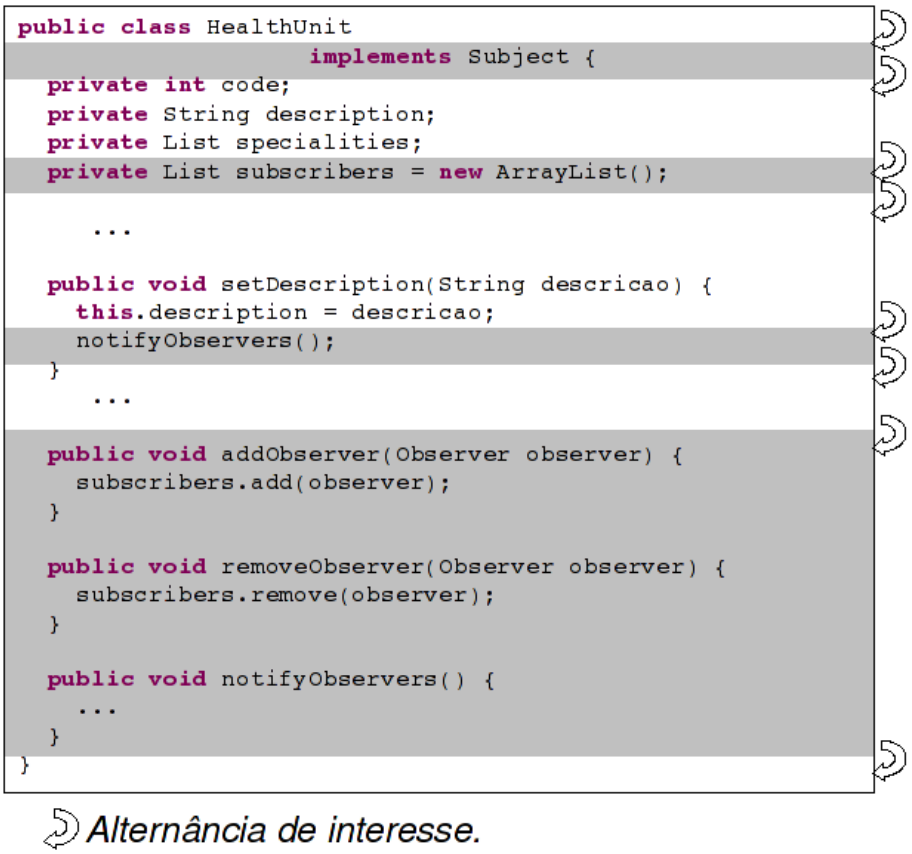


Figura 5.7: A classe *HealthUnit* e o interesse *Subject* sombreado

cherFacade um descendente doente.

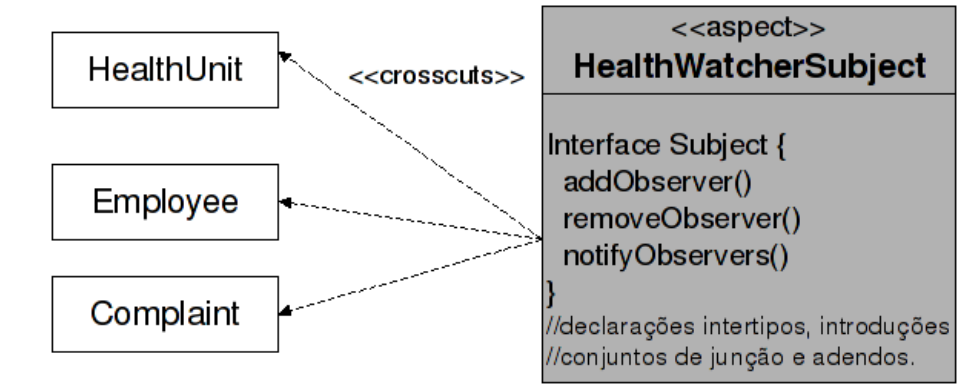
Primeiramente, a refatoração da interface *Observer* é opcional. Entretanto, neste caso, optou-se por mover para aspecto a interface, considerando a mesma estratégia adotada na refatoração do interesse relacionado ao papel *Subject* (seção 5.6).

Para o descendente sadio (interface *RMIFacade*) aplica-se a refatoração para trocar *implements* por *declare parents* em aspecto (passo 3.a). Em seguida, refatora-se o descedente doente (classe *HealthWatcherFacade*) movendo o método *update()* para aspecto através de introdução (passos 4.b). É possível visualizar o resultado sobre este exemplo após a aplicação das refatorações para Planta Trepadeira (seção 5.6) e para Doença Hereditária através da figura 5.9.

5.13.4 Refatorando *State* no *Health Watcher*

A implementação do padrão de projeto *State* no sistema *Health Watcher* é mostrado no capítulo 4 como um interesse Enraizado. A refatoração para este sintoma é exemplificada aqui conforme os passos definidos na seção 5.8.

A figura 4.10 (no capítulo 4) ilustra um recorte do desenho referente à implementação do padrão *State* no *Health Watcher*. As classes *GeneralComplaint*, *GeneralComplaintState*, *AnimalComplaint*, *AnimalComplaintState*, *FoodComplaint*, *FoodComplaintState*, *SpecialComplaint* e *SpecialComplaintState* ligadas à classe *Situation*. As relações na figura indicam acoplamento entre os componentes, sendo que as classes periféricas da figura são os ramos e a classe central (*Situation*) é o tronco. Apesar de as classes possuírem alta dedicação ao interesse, isto é, estão completamente ou tem sua maior parte dedicada à




```
public aspect HealthWatcherSubject {
  ...
  pointcut HealthUnitDescriptionChange(Subject subject):
     call(void HealthUnit.setDescription(String)) &&
   target(subject);

  pointcut ComplaintChange(Subject subject):
     ( call(void Complaint.setComplaintState(ComplaintState)) ||
       call(void Complaint.setAttendant(Employee))||
       call(void Complaint.setObservation(String))||
       call(void Complaint.setStatus(int)) ) &&
       target(subject);

  pointcut EmployeeChange(Subject subject):
     ( call(void Employee.setName(String)) ||
       call(void Employee.setLogin(String))||
       call(void Employee.setPassword(String)) ) &&
       target(subject);

  after(Subject subject): HealthUnitDescriptionChange(subject) ||
     ComplaintChange(subject)|| EmployeeChange(subject)
     {
         subject.notifyObservers();
     }
}
```

Figura 5.8: Classes refatoradas e o aspecto *HealthWatcherSubject*

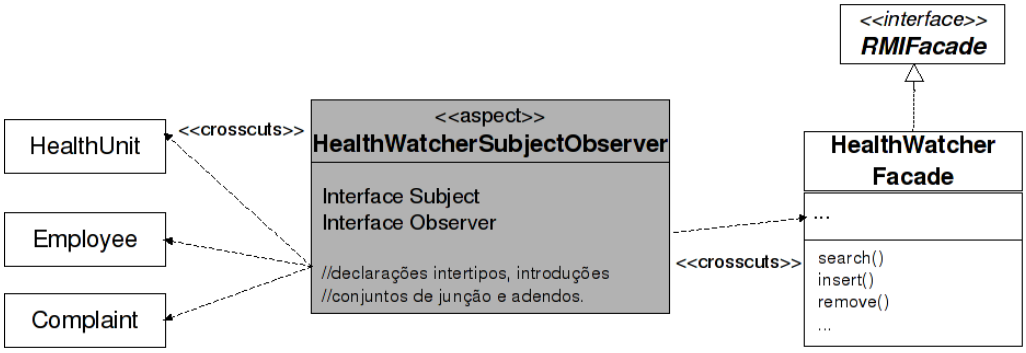


Figura 5.9: Classes refatoradas e o aspecto *HealthWatcherSubjectObserver* após a refatoração dos papéis do padrão *Observer*

realização do interesse, elas convergem conexões de acoplamento para uma única classe. Tal configuração identifica o interesse Enraizado.

A classe *Situation* (tronco) possui apenas três atributos relacionados ao interesse que podem ser movidos para aspecto. Neste caso, de acordo com a refatoração para interesse Enraizado, aplica-se o passo b.1 para cada atributo, que passará a fazer parte do aspecto. A figura 5.10 ilustra o aspecto criado após o passo de refatoração b.1 sobre a classe *Situation*.

```
public aspect ComplaintsAndStatesAspect {

  public static int Situation.QUEIXA_ABERTA = 1;

  public static int Situation.QUEIXA_SUSPENSA = 2;

  public static int Situation.QUEIXA_FECHADA = 3;

}
```

Figura 5.10: Aspecto criado após realização do passo b.1 no componente tronco

As relações mostradas na figura 4.10 representam trechos de código semelhantes nas classes ramos. A figura 5.11 demonstra na área sombreada um fragmento de código da classe *FoodComplaint* onde há o acoplamento com a classe *Situation*. Adicionalmente, esta situação se repete de maneira parecida nas seguintes classes ramos: *GeneralComplaint*, *SpecialComplaint* e *AnimalComplaint*.

```
public class FoodComplaint extends Complaint {

     //...

  public FoodComplaint(String solicitante, String descricao, String observacao,
String email,
      Employee atendente, int situacao, Date dataParecer, Date dataQueixa,
      Address enderecoSolicitante, int qtdeComensais, int qtdeDoentes, int
qtdeInternacoes,
      int qtdeObitos, String localAtendimento, String refeicaoSuspeita, Address
enderecoDoente) {

       super(solicitante, descricao, observacao,email, atendente, situacao,
dataParecer,dataQueixa, enderecoSolicitante,0);
      if(situacao==Situation.QUEIXA_ABERTA)
        state= new FoodComplaintStateOpen(qtdeComensais,qtdeDoentes,
qtdeInternacoes, qtdeObitos,localAtendimento,
            refeicaoSuspeita,enderecoDoente);
        else if(situacao==Situation.QUEIXA_FECHADA)
          state= new FoodComplaintStateClosed(qtdeComensais,qtdeDoentes,
qtdeInternacoes, qtdeObitos,localAtendimento,
              refeicaoSuspeita,enderecoDoente);
  }

     //...

}
```

Figura 5.11: Trecho de código da classe *FoodComplaint*

Já a figura 5.12 mostra na região sombreada o método *setStatus()* responsável também pela relação de acoplamento com a classe *Situation*. Este método se repete de maneira similar nas seguintes classes ramos: *GeneralComplaintState*, *SpecialComplaintState* e *AnimalComplaintState*.

Seguindo a refatoração para interesse Enraizado é possível aplicar o passo b.3 a fim de extrair o fragmento de código sombreado na figura 5.11 para um adendo em aspecto.

```
public abstract class FoodComplaintState implements Serializable{
     //...
  public void setStatus(int sit,FoodComplaint complaint) {
    if(sit!=complaint.getSituacao()){
      if(sit==Situation.QUEIXA_ABERTA){
        complaint.setComplaintState(new
FoodComplaintStateOpen(qtdeComensais,qtdeDoentes, qtdeInternacoes,
qtdeObitos,localAtendimento,
            refeicaoSuspeita,enderecoDoente));
      }else if(sit==Situation.QUEIXA_FECHADA){
        complaint.setComplaintState(new
FoodComplaintStateClosed(qtdeComensais,qtdeDoentes, qtdeInternacoes,
qtdeObitos,localAtendimento,
            refeicaoSuspeita,enderecoDoente));
      }else if(sit==Situation.QUEIXA_SUSPENSA){

      }
    }
  }
     //...
}
```

Figura 5.12: Trecho de código da classe *FoodComplaintState*

Além disso, o passo b.2 pode ser aplicado sobre o método sombreado na figura 5.12 movendo-o para aspecto. A figura 5.13 ilustra o aspecto criado anteriormente já atualizado. Este aspecto deverá contribuir também para melhor modularizar a implementação do interesse eliminando as conexões de acoplamento de cada classe ramo. Nesta figura, é possível observar o conjunto de junção criado para capturar o ponto onde foi extraído o fragmento e o adendo responsável por introduzir o mesmo (correspondem ao passo b.3). Além disso, há também o método movido a partir do passo b.2 na classe ramo *FoodComplaintState*.

```
public aspect ComplaintsAndStatesAspect {

  public static int Situation.QUEIXA_ABERTA = 1;
  public static int Situation.QUEIXA_SUSPENSA = 2;
  public static int Situation.QUEIXA_FECHADA = 3;

  after(FoodComplaint foodComplaint, String solicitante, String descricao, String
observacao, String email,
            Employee atendente, int situacao, Date dataParecer, Date dataQueixa,
            Address enderecoSolicitante, int qtdeComensais, int qtdeDoentes, int
qtdeInternacoes,
            int qtdeObitos, String localAtendimento,
            String refeicaoSuspeita, Address enderecoDoente) :
               initialization(FoodComplaint+.new(..)) && target(foodComplaint) &&
               args(solicitante, descricao, observacao, email, atendente,
                        situacao, dataParecer, dataQueixa, enderecoSolicitante,
qtdeComensais, qtdeDoentes, qtdeInternacoes,
                       qtdeObitos, localAtendimento, refeicaoSuspeita, enderecoDoente)
   {
     //código extraído para aspecto.
   }

   public void setStatus(int sit,FoodComplaint complaint) {
     //código do método movido para aspecto.
   }

}
```

Figura 5.13: Aspecto após realização dos passos b.2 e b.3 em duas classes

Os passos aplicados e ilustrados pelas figuras 5.11, 5.12 e 5.13 podem ser repetidos

para os demais ramos: realização do passo b.3 nas classes *GeneralComplaint*, *SpecialComplaint* e *AnimalComplaint*; e o passo b.2 para as classes *GeneralComplaintState*, *SpecialComplaintState* e *AnimalComplaintState*.

Como resultado, moveu-se do componente tronco as partes dedicadas ao interesse (três atributos pelo passo b.1) e ainda eliminou-se as relações de acoplamento que convergem para a classe *Situation*. Dessa maneira, mesmo que os demais componentes refatorados ainda possuam partes dedicadas ao interesse eles não configuram mais o sintoma de Interesse Enraizado. Opcionalmente, outros passo de refatoração podem ainda ser aplicados sobre demais partes dos componentes. Entretanto, a estrutura do sintoma foi eliminada.

5.13.5 Refatorando *Abstract Factory* no *Health Watcher*

O sistema *Health Watcher* possui duas implementações do padrão de projeto *Abstract Factory* (GAMMA et al., 1995). Uma delas é ilustrada na figura 5.14 como um interesse que possui o sintoma Rede Neural. Pela figura existem duas classes *HWServer* e *HWServlet* (componentes da camada de entrada da rede) que possuem acoplamento com a classe *FacadeFactory* (único componente da camada intermediária) que, por sua vez, tem conexões com as classes *AbstractFacadeFactory* e *RMIFacadeFactory* (ambas representando a camada de saída da rede).

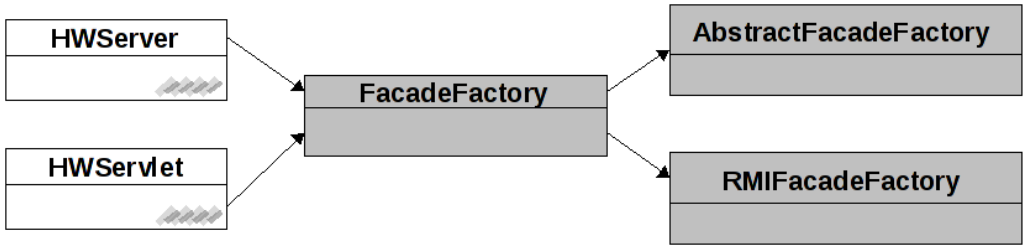


Figura 5.14: Realização do padrão *AbstractFactory* com o sintoma Rede Neural

As classes da camada de entrada encontram-se parcialmente dedicadas ao interesse, enquanto que as demais estão completamente dedicadas. Entretanto, os sintomas relativos a acoplamento, como o Rede Neural, referem-se às conexões existentes entre os componentes participantes do interesse, independente do grau de dedicação de cada um deles. Conforme o passo *a* da refatoração para Rede Neural (seção 5.8), como é opcional refatorar componentes totalmente dedicados ao interesse, decidiu-se mantê-los da mesma forma. Restam portanto os componentes da camada de entrada.

A figura 5.15 demonstra dois trechos de código (sombreados) relacionados ao interesse pertencentes às classes que compõem a camada de entrada da rede. Neste caso, aplica-se o passo b.3, em cada classe, para extrair o fragmento de código sombreado para adendo em um aspecto.

A figura 5.16 ilustra o aspecto criado em decorrência desta etapa. No aspecto existe um conjunto de junção e um adendo para cada situação, isto é, para cada fragmento extraído de cada classe da figura 5.15. Assim, a formação do sintoma Rede Neural foi desmontada. As classes *HWServer* e *HWServlet* podem ou não evoluir a partir de outros interesses mas não implementam mais o interesse relacionado ao padrão *Abstract Factory*. As demais classes totalmente dedicadas permanecem da mesma maneira. Eventualmente, qualquer necessidade de se utiliziar as classes *Factory* (*FacadeFactory*, *AbstractFactory* e *RMIFacadeFactory*), deve-se usá-las através do aspecto criado, pois ele está modula-

```
public class HWServlet extends HttpServlet
{
   //atributos da classe.

   public void init(ServletConfig config) throws ServletException {
     //...
     facade = FacadeFactory.getRepositoryFactory().createClientFacade;
     //...
   }

   //demais métodos da classe.
}
```

```
public class HWServer {

  public static void main(String[] args) {
     //...
      FacadeFactory.getRepositoryFactory().createServerFacade();
     //...
  }
}
```

Figura 5.15: Trechos de código da camada de entrada relacionados ao interesse

rizando esta utilização. Caso contrário, será criada uma nova configuração do sintoma Rede Neural.

```
public aspect AbstFactoryAspect {

     pointcut callingInitCommands(HWServlet hwServlet): call(void
HWServlet.initCommands()) && target(hwServlet);

     before (HWServlet hwServlet): callingInitCommands(hwServlet)
     {
          hwServlet.facade =
FacadeFactory.getRepositoryFactory().createClientFacade();
     }

     pointcut callingMain(): call(static void HWServer.main(..));

     after (): callingMain()
     {
          FacadeFactory.getRepositoryFactory().createServerFacade();
     }
}
```

Figura 5.16: Aspecto após realização do passo b.3 nas classes *HWServer* e *HWservlet*

Evidentemente, há outra possibilidade de refatoração para esta situação específica. Nesse caso, optou-se por não modificar as classes que estão inteiramente dedicadas ao interesse.

5.13.6 Refatorando o Padrão *Prototype*

O exemplo dado no capítulo 4 para o sintoma Ovelha Dolly, referente a uma implementação hipotética do padrão de projeto *Prototype* (GAMMA et al., 1995), é retomado nesta seção. A figura 4.20 ilustra duas classes (*Produto* e *Preço*) que realizam o padrão *Prototype* a partir da implementação de um método de clonagem (sombreado na figura). Este método realiza uma cópia do objeto e a retorna. Dessa forma, toda classe que rea-

liza este padrão de projeto deverá implementar o método *clone()*. A implementação do método poderia ser distinta em cada classe, entretanto, neste exemplo, as implementações são iguais.

Neste caso, apesar da simplicidade do exemplo, é possível indentificar 4 sintomas:

- primeiramente, o interesse referente ao padrão pode ser visto com o sintoma Ovelha Negra, pois o interesse encontra-se espalhado por componentes ocupando apenas uma pequena parte deles (baixa dedicação);
- além disso, o interesse também tem o sintoma Cópia Carbono, pois os métodos *clone()* são iguais (cópias) nas duas classes que implementam o padrão *Prototype*;
- consequentemente, sendo Ovelha Negra e Cópia Carbono, o interesse é também Ovelha Dolly. É justamente um caso particular onde se encontra as duas situações.
- por fim, identifica-se também o sintoma de interesse Comportamental, pois a realização do mesmo consiste apenas na existência de operações (métodos *clone()*) entre os participantes.

Portanto, para o exemplo da figura 4.20, há três possibilidades de refatoração (a refatoração para Ovelha Dolly é a mesma de Cópia Carbono):

- Aplicando a refatoração para Ovelha Negra: cabe apenas a realização do passo 2.b para as duas classes da figura (*Produto* e *Preço*). Os métodos de clonagem seriam introduzidos a partir de um aspecto.
- Aplicando a refatoração para Cópia Carbono: como as cópias referem-se à declaração completa do método, e não somente uma parte dele, elas são cópias estruturais. Cabe, portanto, a realização do passo 3. Como as duas classes não fazem parte de uma relação de herança e as cópias são apenas dos métodos, elimina-se os passos 3.a e 3.b. Logo, resta a aplicação do passo 3.c que move o método *clone()* em cada classe para aspecto.
- Aplicando a refatoração para Interesse Comportamental: o passo 2 possui duas alternativas. A segunda é descartada, pois não há a possibilidade de mover os métodos de clonagem para outras classes. Esses métodos copiam o objeto onde está implementado o próprio método, não fazendo sentido movê-los para outras classes. Portanto, a primeira alternativa é realizada, que também move os métodos para aspecto, assim como nas alternativas de refatoração anteriores.

Verifica-se portanto, que nas três refatorações realizou-se a mesma operação: mover os métodos *clone()* de cada classe para aspecto. A figura 5.17 ilustra o aspecto criado com os métodos movidos. Neste caso específico, o resultado seria o mesmo ao aplicar as três refatorações, entretanto isso não quer dizer que será sempre assim.

Adicionalmente, vale observar a seguinte situação: caso os métodos de clonagem fossem implementados de maneira diferente nas classes participantes, o interesse deixaria de ter o sintoma Cópia Carbono, pois os métodos deixariam de ser cópias um do outro. Como consequência, a implementação deixaria de ter também o sintoma Ovelha Dolly. Portanto, o interesse relacionado à implementação do padrão de projeto *Prototype* ficaria com os sintomas Ovelha Negra e Comportamental. A figura 5.18 exemplifica este caso. O método *clone()* na classe *Preço* permanece o mesmo, porém o método *clone()* na classe

```
public aspect PrototypeAspect {

      public Object Produto.clone() throws CloneNotSupportedException
      {
            return super.clone();
      }

      public Object Preço.clone() throws CloneNotSupportedException
      {
            return super.clone();
      }
}
```

Figura 5.17: Aspecto resultante da refatoração

Produto foi substituído por dois novos métodos para clonagem: o primeiro, obtém uma cópia "superficial", isto é, retorna um novo produto com os mesmos atributos do atual (incluindo o objeto *preço*); enquanto, que o segundo cria uma cópia "profunda", ou seja, retorna um novo objeto *produto* clonando também o objeto *preço* que ele possui, não utilizando o mesmo e sim uma cópia dele.

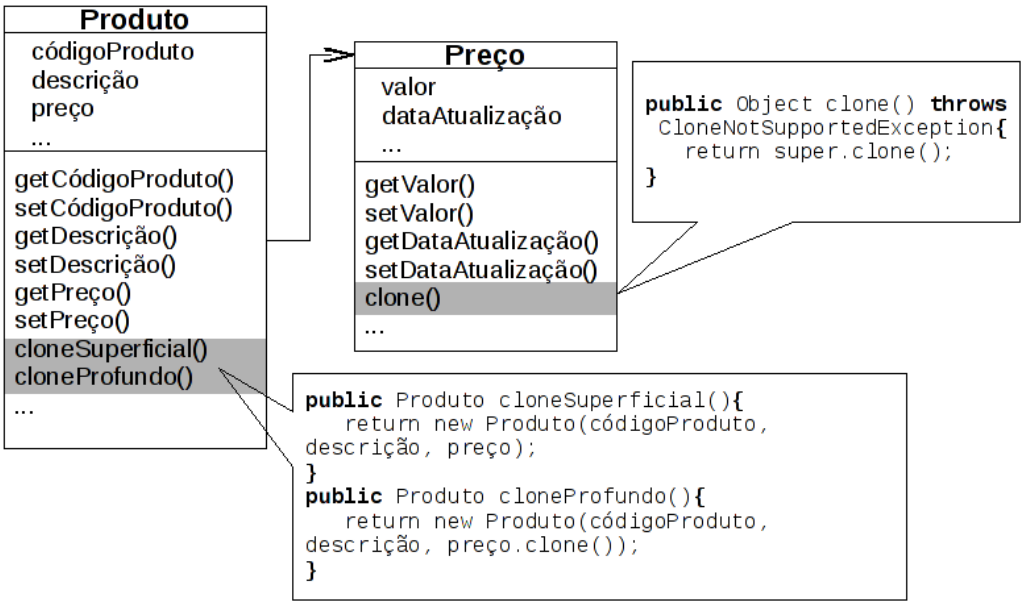


Figura 5.18: Variação do exemplo do padrão *Prototype*

A refatoração desta variação de implementação ilustrada na figura 5.18 tem resultado semelhante ao mostrado pelo aspecto da figura 5.17. O mesmo seria feito sobre os métodos de clonagem *cloneSuperficial()*, *cloneProfundo()* (na classe *Produto*) e *clone()* na classe *Preço*, isto é, estes métodos também seriam movidos para aspecto como declaração intertipo, seja pela refatoração para Ovelha Negra seja pela refatoração para Interesse Comportamental.

Ao final da refatoração, seja qual for a alternativa escolhida pela existência de diversos sintomas, as classes deixam de implementar o interesse em questão e ficam com outras intenções que possam ser atribuídas a elas. A responsabilidade de clonagem fica então modularizada em aspecto.

5.14 Considerações Finais do Capítulo

Este capítulo descreveu o catálogo de refatorações para os sintomas de interesses a fim de prover um mecanismo de apoio à modularização de interesses transversais. Primeiramente discutiu-se a questão da preservação do comportamento *versus* a preservação da intenção. Em seguida o catálogo foi apresentado e descrito detalhadamente. Para o bom entendimento das refatorações é necessário conhecer os sintomas apresentados no capítulo 4. Tais refatorações são aplicadas a partir da configuração específica de um sintoma em um interesse e são compostas por refatorações de granularidade mais fina. A aplicação de tais refatorações não visa à otimização de outros atributos de qualidade como legibilidade, organização interna, tamanho etc. O objetivo principal é a separação de interesses. Outras refatorações podem ser aplicadas a fim de oferecer uma reestruturação interna dos aspectos criados, conforme propõe Monteiro (MONTEIRO, 2005) (MONTEIRO; FERNANDES, 2006). O capítulo seguinte apresenta a proposta de análise de impacto das refatorações baseada em algoritmos sobre métricas de separação de interesses.

6 ALGORITMOS PARA ANÁLISE DE IMPACTO DAS REFATORAÇÕES

Este capítulo tem como objetivo apresentar uma estratégia para análise de impacto baseada em algoritmos que permitem avaliar o impacto da mudança relacionada a aplicação das refatorações apresentadas no capítulo 5. Estes algoritmos, juntamente com o uso da abordagem de detecção e classificação de interesses sintomáticos, e as refatorações, compõem o método de apoio à modularização de interesses transversais proposto neste trabalho.

Analisar o impacto de uma refatoração candidata significa avaliar sob algum ponto de vista os prós e contras em se realizar a refatoração. No escopo deste trabalho a análise de impacto é feita quantitativamente através de algoritmos que calculam a variação de medições de separação de interesses considerando três métricas (CDC, CDO e CDLOC) discutidas no capítulo 3. O restante do capítulo está dividido da seguinte forma: inicialmente, discute-se na seção 6.1 trabalhos relacionados à avaliação quantitativa de impacto das refatorações; em seguida, na seção 6.2 apresenta-se os algoritmos referentes às rotinas de impacto que são chamadas pelos algoritmos maiores da análise de impacto das refatorações, que são expostos na seção 6.3; por fim, a seção 6.4 conclui o presente capítulo.

6.1 Avaliação Quantitativa de Refatorações

Alguns trabalhos foram propostos a fim de avaliar quantitativamente atributos de qualidade do software diante do efeito de refatorações em código (BOIS; MENS, 2003; BOIS, 2006). Entretanto, tais propostas encontram-se voltadas para um subconjunto de refatorações orientadas a objetos. Dois trabalhos foram levantados possuindo um foco nas refatorações orientadas a aspectos e que são discutidos a seguir.

6.1.1 *Funções de Impacto por Piveta (2009)*

Em sua tese, Piveta (2009) propôs funções para análise de impacto com o objetivo de guiar o desenvolvedor na escolha de refatorações a serem aplicadas. O uso das funções de impacto auxilia os desenvolvedores na decisão sobre qual a melhor opção de refatoração a ser conduzida de acordo com métricas específicas consideradas. Neste trabalho, para uma determinada refatoração deve-se criar funções de impacto sobre uma determinada métrica para cada participante da refatoração. Assim, se houver χ participantes e α métricas, deverá existir $(\chi * \alpha)$ funções de impacto, para calcular cada variação de uma métrica pela refatoração de cada componente.

Piveta (2009) demonstrou o seu método na criação de funções de impacto para a refatoração OA Generalizar adendo, considerando as seguintes métricas: linhas de código

(LOC - *Lines of Code*), número de operações em um módulo (NOM - *Number of Operations in Module*), grau de transversalidade de um aspecto (CDA - *Crosscutting Degree of an Aspect*) e acoplamento na execução de um adendo (CAE - *Coupling on Advice Execution*). Em seguida, aplicou-se as funções em um estudo de caso. Adicionalmente, foi criada também uma API para permitir a criação de funções de impacto programaticamente.

6.1.2 *Processo de Avaliação Quantitativa por Pagliari (2007)*

Em (PAGLIARI; NUNES, 2007) e (PAGLIARI, 2007) é apresentado um processo de avaliação quantitativa de refatorações OA. O processo tem como objetivo estabelecer um guia sistemático sobre: (i) como desmembrar as refatorações sob análise em passos atômicos; (ii) como avaliar o impacto desses passos atômicos de acordo com as métricas escolhidas; (iii) e, por fim, como unir o impacto dos passos atômicos a fim de obter o impacto da refatoração como um todo, conforme métricas previamente estabelecidas. Assim, é possível obter evidências quantitativas dos benefícios trazidos pela refatoração avaliada e descobrir possíveis desvantagens de seu uso não previstas em sua definição, disponibilizando ao desenvolvedor informações que podem auxiliar, por exemplo, na tomada de decisão sobre a aplicação ou não da refatoração.

Pagliari (2007) utilizou este processo para um conjunto de 15 refatorações, incluindo 11 orientadas a aspectos retiradas do catálogo de Monteiro (2005) e 4 orientadas a objetos do catálogo de Fowler (1999). O impacto foi analisado de acordo com 10 métricas para medições de atributos de qualidade como separação de interesses, coesão, acoplamento e tamanho.

6.1.3 *Comparando as Duas Propostas*

Apesar de ambas as propostas (subseções anteriores) terem como base a avaliação quantitativa através de métricas, elas diferem em uma questão: enquanto as funções de impacto (PIVETA, 2009) se propõem à avaliação quantitativa sobre uma situação concreta, isto é, avaliam a refatoração sobre uma implementação específica, o processo de avaliação em Pagliari (2007) se propõe a determinar o impacto pela aplicação da refatoração independente do contexto, ou seja, para qualquer código a ser refatorado. Dessa maneira, o trabalho de Pagliari (2007), em muitos casos, não estabelece valores concretos no impacto, mas sim intervalos de variação, pois sua avaliação é sobre o impacto da refatoração independente dos componentes onde está sendo aplicada. Além disso, o trabalho de Pagliari propõe um processo, isto é, estabelece um fluxo de atividades a serem seguidas para avaliar quantitativamente o impacto de refatorações sobre atributos de qualidade através de métricas.

Para os algoritmos de análise de impacto das refatorações para modularização de interesses transversais (capítulo 5), decidiu-se assumir como base o trabalho de Pagliari (2007), pelos seguintes motivos:

- As refatorações propostas no capítulo 5 usam as refatorações OA do catálogo de Monteiro (2005), o mesmo catálogo de onde foram retiradas as 15 refatorações consideradas por Pagliari (2007).

- As 3 métricas de separação de interesse, consideradas nos algoritmos de análise de impacto propostos aqui (seções 6.2 e 6.3), estão incluídas também no estudo de impacto feito sobre as 15 refatorações do Monteiro. Assim, já é possível se basear na análise feita neste trabalho para a construção dos algoritmos.

- O processo proposto em (PAGLIARI; NUNES, 2007) e (PAGLIARI, 2007) estabelece um guia e demonstra como quebrar uma refatoração em passos atômicos e como calcular o impacto (mesmo que por intervalos) de tais passos. Dessa maneira, a facilidade de usar o processo para estender a análise apresentada é maior. Isso aumenta a capacidade de extensão e replicação do estudo feito para novos conjuntos de métricas e refatorações. Consequentemente, os algoritmos de impacto propostos aqui poderão ser estendidos mais facilmente, seguindo o processo proposto em (PAGLIARI, 2007).

6.2 Rotinas de Análise de Impacto

Foram criadas seis rotinas de impacto a fim de dar suporte aos algoritmos de análise de impacto das refatorações. Estas rotinas são apresentadas conforme listagens de 6.1 a 6.7 em forma de pseudocódigo seguindo o português estruturado (MANZANO; OLIVEIRA, 2008). Elas foram criadas separadamente dos algoritmos de análise de impacto das refatorações para permitir o reuso e facilitar a compreensão.

Três métricas de separação de interesses foram selecionadas para a análise de impacto:

- CDC (*Concern Diffusion over Components*) - Difusão de interesse sobre componentes;
- CDO (*Concern Diffusion over Operations*) - Difusão de interesse sobre operações;
- CDLOC (*Concern Diffusion over Lines of Code*) - Difusão de interesse sobre linhas de código.

Essas três métricas foram propostas em (SANT'ANNA et al., 2003) e são exploradas com mais detalhes no capítulo 3 desta dissertação.

Pode-se observar que as rotinas nas listagens de 6.1 a 6.7 lidam com o impacto nas medições de CDC, CDO e CDLOC considerando as seguintes refatorações de menor granularidade: Extrair método e Generalizar método de Fowler(1999); Mover método de classe para intertipo, Mover atributo de classe para intertipo, Extrair fragmento para adendo e Trocar *implements* por *declare parents*, todas estas de Monteiro (2005). Estas rotinas são chamadas pelos algoritmos das refatorações "maiores" (modularização de interesses transversais) apresentados na seção 6.3. A lógica de cálculo empregada nessas rotinas se baseia em análise feita no trabalho de Pagliari (2007), comentado na seção anterior.

Para todas as listagens desta seção, as rotinas são definidas como *Procedimentos* e as métricas que sofrem impacto são representadas como parâmetros de entrada por referência. Para a implementação tanto das rotinas de impacto, quanto dos algoritmos (seção 6.3), assume-se que o sintoma relacionado ao interesse transversal em questão seja conhecido e devidamente mapeado para os componentes e partes dos componentes que participam do mesmo. Portanto, tem-se como entrada para cada algoritmo: (i) o sintoma detectado em um determinado interesse; e (ii) as partes dos componentes que participam do sintoma na realização do interesse. Como saída, tem-se os valores referentes às variações das medições de CDC, CDO e CDLOC. Para deixar mais clara a apresentação das listagens neste capítulo, tais entradas e saídas não foram explicitamente descritas, embora devam ser consideradas.

A listagem 6.1 apresenta o procedimento *calculaImpactoExtrairMetodo* referente ao impacto da extração de método. Neste caso, das três métricas (CDC, CDO e CDLOC)

apenas duas sofrem impacto - CDO e CDLOC. Inicialmente, o procedimento verifica o impacto sobre a métrica CDO. Caso ainda permaneça trecho de código relacionado ao interesse no método onde foi extraído um fragmento para dar origem a um novo método, a métrica CDO aumenta, pois haverá um novo método relacionado ao interesse. Caso contrário, o método onde existia o fragmento extraído para novo método deixou de participar do interesse, porém um novo método é criado e assim a métrica não sofre alteração.

Em seguida, o procedimento verifica o impacto sobre a métrica CDLOC. Nesse caso, a variação dessa métrica depende do local onde foi extraído o trecho de código para método e do local onde foi criado o novo método. Se no local onde foi extraído o trecho não existia código adjacente relacionado ao interesse, então será eliminada duas alternâncias de código sobre o interesse, isto é, CDLOC diminui em duas unidades. Além disso, pode ser que o método criado seja colocado numa região onde há outros métodos relacionados ao interesse. Se for esse o caso, CDLOC não varia. Caso contrário, se não houver trechos de código relacionados ao interesse adjacente ao método criado, haverá mais duas alternâncias de código sobre o interesse.

Listagem 6.1: Rotina de impacto para Extrair Método

```
1
2  PROCEDIMENTO calculaImpactoExtrairMetodo(var CDO: inteiro, var CDLOC:
       inteiro)
3  INICIO
4          SE houver ainda parte do metodo relacionado ao interesse ENTAO
5                  CDO <- CDO + 1;
6          FIM_SE
7          SE nao houver trechos de codigo relacionados ao interesse
               imediatamente adjacentes ao fragmento extraido para metodo
               ENTAO
8                  CDLOC <- CDLOC - 2;
9                  SE nao houver trechos de codigo relacionados ao
                       interesse imediatamente adjacentes ao novo metodo
                       ENTAO
10                         CDLOC <- CDLOC + 2; //Pior caso: considerando
                               que nao vai ficar adjacente a nenhum outro
                               metodo sombreado e ainda permanece regiao
                               sombreada de onde foi extraido.
11                 FIM_SE
12         FIM_SE
13 FIM
```

A listagem 6.2 demonstra o procedimento referente ao impacto da refatoração OO Generalizar método. Nesse caso, das três métricas, apenas CDC e CDLOC podem variar. Primeiramente, na elevação do método para um superclasse, pode ser que a classe (ou interface) que receberá o método não seja um componente relacionado ao interesse. Portanto, este seria o caso de aumentar a medição do CDC, isto é, o interese estaria distribuído por mais um componente. Caso contrário, não haveria um novo componente participante do interesse e a medição do CDC não mudaria. O impacto na medição CDC também está sendo calculado nos próprios algoritmos das refatorações maiores (seção 6.3).

Em seguida, o procedimento verifica o local para onde foi "elevado" o método. Essa parte do procedimento é semelhante ao da listagem anterior. Se o método a ser elevado for retirado de uma região do componente onde não há código relacionado ao interesse, a medição CDLOC diminui em duas unidades. Além disso, caso o método seja elevado para um componente que não participa do interesse ou caso seja colocado sobre um trecho do componente onde não há outras partes relacionadas ao interesse, haverá um aumento

de CDLOC, pois criou-se duas alternâncias de código relacionado ao interesse.

Listagem 6.2: Rotina de impacto para Generalizar método

```
1
2  PROCEDIMENTO calculaImpactoGeneralizarMetodo(var CDC, var CDLOC)
3  INICIO
4          SE o componente para onde o metodo esta sendo elevado nao
               estiver relacionado ao interesse ENTAO
5                  CDC <- CDC + 1;
6          FIM-SE
7          SE nao houver codigo relacionado ao interesse imediatamente
               adjacente ao metodo a ser elevado ENTAO
8                  CDLOC <- CDLOC - 2;
9                  SE nao houver trechos de codigo relacionados ao
                       interesse imediatamente adjacentes ao novo metodo
                       ENTAO
10                         CDLOC <- CDLOC + 2;
11                 FIM_SE
12         FIM_SE
13 FIM
```

Já a listagem 6.3 mostra o procedimento referente ao impacto da refatoração Generalizar atributo. Este procedimento é similar ao da listagem anterior para Generalizar método, porém este impacta também na medição de CDO. De acordo com o procedimento, a medição CDC aumenta em 1 unidade se o componente que receberá o atributo não faz parte do interesse. A medição de CDLOC varia negativamente em duas unidades se o atributo a ser elevado não se encontra adjacente a outro trecho de código. E pode variar positivamente em duas unidades se o local para onde foi elevado não havia partes adjacentes relacionadas ao interesse. Por fim, a medição de CDO pode aumentar em duas unidades se for necessária a criação de dois métodos acessores (*get* e *set*).

Listagem 6.3: Rotina de impacto para Generalizar atributo

```
1
2  PROCEDIMENTO calculaImpactoGeneralizarAtributo(var CDC, var CDLOC, var
       CDO)
3  INICIO
4          SE o componente para onde o atributo esta sendo elevado nao
               estiver relacionado ao interesse ENTAO
5                  CDC <- CDC + 1;
6          FIM-SE
7          SE nao houver codigo relacionado ao interesse imediatamente
               adjacente ao atributo a ser elevado ENTAO
8                  CDLOC <- CDLOC - 2;
9                  SE o atributo for elevado para uma regiao nao
                       relacionada ao interesse ENTAO
10                         CDLOC <- CDLOC + 2;
11                 FIM_SE
12         FIM_SE
13         SE forem criados novos metodos acessores para o atributo ENTAO
14                 CDO <- CDO + 2;
15         FIM-SE
16 FIM
```

A listagem 6.4 refere-se ao procedimento que calcula o impacto da refatoração OA Mover Atributo de Classe para Intertipo. Nesse procedimento, alguns parâmetros são necessários além das métricas CDO e CDLOC que sofrem impacto:

- *n_op_modificadas* - o número de operações que utilizam o atributo e que por isso serão modificadas.
- *n_trechos* - o número de operações criadas pela extração de fragmentos para novos métodos.
- *n_metodos* - o número de operações criadas pela extração de fragmentos para para novos adendos.

A medição de CDO diminuirá tantas unidades quanto forem o número de métodos que deixarão de estar relacionados ao interesse por não estarem mais acessando o atributo movido. Entretanto, poderá aumentar pela quantidade de fragmentos extraídos para adendo ou extraídos para novos métodos pela necessidade de eliminar o uso do atributo movido em eventuais trechos de código. Além disso, o CDO pode aumentar também caso seja necessário a criação de métodos acessores no encapsulamento do atributo movido.

Por fim, o impacto sobre a medição de CDLOC é calculado, porém de forma imprecisa. Nesse caso, CDLOC pode diminuir em um valor indefinido de unidades não podendo precisar o quanto. Isso irá depender da maneira com que o componente usa o atributo e consequentemente da maneira com que o atributo irá ser movido para aspecto através da refatoração. Muitas situações são possíveis nesse casso. Entretanto, sabe-se que CDLOC pode somar unidades a depender das modificações realizadas nos métodos que usavam o atributo movido. Cada modificação no trecho de um método que usava o atributo pode, na pior das hipóteses, criar alternâncias de interesse no código. Por esse motivo, é atribuído um intervalo no impacto do CDLOC.

Listagem 6.4: Rotina de impacto para Mover Atributo de classe para intertipo em aspecto

```
1
2 PROCEDIMENTO calculaImpactoMoverAtributoParaAspecto(var
    n_op_modificadas, var n_trechos, var n_metodos, var CDO, var CDLOC)
3 INICIO
4         CDO <- CDO - n_op_modificadas + n_trechos + n_metodos;
5         SE o atributo nao estiver encapsulado ENTAO
6                 CDO <- CDO + 2;
7         FIM-SE
8         CDLOC <- (-? a n_metodos);
9 FIM
```

O procedimento da listagem 6.5 refere-se ao impacto da refatoração OA Mover Método de Classe para Intertipo. Nesse algoritmo, pode haver impacto somente na medição de CDLOC. Haverá diminuição em duas unidades se não houver trechos de código imediatamente adjacentes ao método movido para aspecto, isto é, serão eliminadas duas alternâncias de código relacionadas ao interesse. Não há alteração na medição de CDO, pois o método continua sendo contabilizado mesmo estando em aspecto. A variação pela métrica CDC está sendo considerada nos próprios algoritmos que usam este procedimento.

Listagem 6.5: Rotina de impacto para Mover Método de classe para intertipo em aspecto

```
1
2 PROCEDIMENTO calculaImpactoMoverMetodoParaAspecto(var CDLOC)
3 INICIO
4         SE nao houver trechos de codigo relacionados ao interesse
              imediatamente adjacentes ao metodo ENTAO
5                 CDLOC <- CDLOC - 2;
6         FIM-SE
7 FIM
```

A listagem 6.6 define o procedimento para a análise de impacto da refatoração Extrair Fragmento para Adendo. Para este caso, a medição de CDO pode variar em duas situações: primeiro, se a operação que contém o fragmento não tiver outro relacionado ao interesse, então a operação deixará de ser computada na medição de CDO (CDO - 1); segundo, se o fragmento for extraído para um novo adendo, deve-se aumentar em 1 unidade o valor de CDO, pois um adendo é considerado como uma operação. Caso o fragmento seja extraído para um adendo já existente em algum aspecto, então não há aumento na medição de CDO.

Adicionalmente, deve-se verificar como fica a alternância de código relacionado ao interesse referente ao fragmento extraído. Se não houver trechos de código relacionados ao interesse imediatamente adjacentes ao fragmento a ser extraído então elimina-se duas unidades de CDLOC. Além disso, pode haver na mesma operação que contém o fragmento extraído variáveis locais usadas no fragmento a serem também extraídas. Nesse caso, se essas variáveis não estiverem adjacentes a outros trechos relacionados ao interesse, há também uma diminuição em 2 unidades na medição de CDLOC pela extração das variáveis.

Listagem 6.6: Rotina de impacto para Extrair fragmento para adendo

```
1
2  PROCEDIMENTO calculaImpactoExtrairFragmentoParaAdendo(var CDO, var
       CDLOC)
3  INICIO
4           SE a operacao que contem o fragmento nao tiver outro fragmento
                relacionado ao interesse ENTAO
5                   CDO <- CDO - 1;
6           FIM-SE
7           SE for criado um adendo exclusivo para o fragmento extraido
                ENTAO
8                   CDO <- CDO + 1;
9           FIM-SE
10          SE nao houver trechos de codigo relacionados ao interesse
                imediatamente adjacentes ao fragmento ENTAO
11                  CDLOC <- CDLOC - 2;
12                  PARA CADA variavel local usada apenas pelo fragmento
                        extraido E nao adjacente a outro trecho de codigo
                        relacionado ao interesse FACA
13                          CDLOC <- CDLOC - 2;
14                  FIM-PARA
15          FIM-SE
16 FIM
```

Por fim, a listagem 6.7 mostra o procedimento para calcular o impacto da refatoração OA de trocar *implements* ou *extends* por *declare parents* em aspecto. Nesse caso, haverá sempre uma diminuição em 2 unidades na medição de CDLOC se não houver código relacionado ao interesse adjacente à declaração de *implements* ou *extends*, pois assim haverá eliminação de alternância de código relacionado ao interesse.

Listagem 6.7: Rotina de impacto para trocar *implements* ou *extends* por *declare parents* em aspecto

```
1
2  PROCEDIMENTO calculaImpactoMoverImplementsExtends(var CDLOC)
3  INICIO
4           SE nao houver trechos de codigo relacionados ao interesse
                imediatamente adjacentes a declaracao de implements/extends
                ENTAO
```

```
5                   CDLOC <- CDLOC - 2;
6           FIM-SE
7   FIM
```

6.3 Algoritmos de Análise de Impacto das Refatorações

Nesta seção, são definidos os algoritmos referentes à análise de impacto das refatorações para modularização de interesses transversais apresentadas no capítulo 5. Os algoritmos podem ser utilizados antes da aplicação das refatorações para auxiliar no processo de decidir quando aplicar ou não as refatorações, podendo ser implementado por ferramentas de apoio.

Estes algoritmos usam os procedimentos definidos na seção anterior conforme cada refatoração do capítulo 5. Da mesma maneira que na seção anterior, as listagens de 6.8 a 6.16 também são apresentadas em pseudocódigo seguindo o português estruturado (MANZANO; OLIVEIRA, 2008) e podem ser estendidas caso seja necessário alterar os passos de refatoração correspondentes ou caso seja necessário incluir novas métricas na análise de impacto. Como mencionado na seção anterior, vale ressaltar que, para a implementação dos algoritmos, deve-se assumir que os sintomas de um determinado interesse sejam conhecidos, isto é, previamente detectados. Além disso, os componentes e suas partes que realizam o interesse sintomático em questão (operações, atributos, relacionamentos e trechos de código) devem estar devidamente mapeados. Portanto, as entradas para os algoritmos das listagens 6.8 a 6.16 são: o sintoma (correspondente ao algoritmo) detectado em um dado interesse e as partes dos componentes que participam do sintoma na realização do interesse. Como saída para os algoritmos, tem-se os valores referentes às variações das medições de CDC, CDO e CDLOC que são calculados ao longo dos mesmos. Para deixar mais clara a apresentação das listagens dos algoritmos, tais entradas e saídas não foram explicitamente descritas, embora devam ser consideradas.

A listagem 6.8 mostra o algoritmo que calcula o impacto da refatoração para Polvo (seção 5.4). Esse algoritmo, assim como todos os outros, se inicia com uma declaração de variáveis, seguida da inicialização das mesmas. Posteriormente, o algoritmo consiste em um laço de repetição para cada componente corpo ou tentáculo a ser refatorado. Em cada componente pode haver chamadas para as rotinas de impacto definidas através de procedimentos apresentados na seção anterior (6.2), conforme a situação concreta de cada corpo ou tentáculo. Ao fim de cada iteração do laço, verifica-se se o componente ainda possui partes relacionadas ao interesse. Caso negativo, diminiu-se em 1 unidade a medição de CDC. Este último *SE* encontra-se também em todos os demais algoritmos desta seção.

Listagem 6.8: Algoritmo para análise de impacto da refatoração para Polvo

```
1
2  PROGRAMA AnaliseImpacto_Refat_Polvo
3  var
4      n_op_modificadas, n_trechos, n_metodos: Inteiro;
5      CDC, CDO: Inteiro
6      CDLOC: Inteiro ou Intervalo discreto;
7  INICIO
8          CDC, CDO, CDLOC, n_op_modificadas, n_trechos, n_metodos <- 0;
9          CDC <- CDC + 1; //referente ao aspecto como novo componente
               relacionado ao interesse.
10         PARA CADA componente corpo ou tentaculo a ser refatorado FACA
11                 SE houver heranca ou implementacao de interface
                       relacionada ao interesse ENTAO
```

```
12                      calculaImpactoMoverImplementsExtends(CDLOC);
13                  FIM-SE
14                  PARA CADA atributo relacionado ao interesse FACA
15                      calculaImpactoMoverAtributoParaAspecto(
                            n_op_modificadas, n_trechos, n_metodos, CDO
                            , CDLOC);
16                  FIM-PARA
17                  PARA CADA metodo relacionado ao interesse FACA
18                       calculaImpactoMoverMetodoParaAspecto(CDLOC);
19                  FIM-PARA
20                  SE houver fragmento de codigo relacionado ao interesse
                        ENTAO
21                      PARA CADA fragmento FACA
22                              calculaImpactoExtrairFragmentoParaAdendo
                                    (CDO);
23                      FIM-PARA
24                  FIM-SE
25                  SE todas as partes do interesse foram eliminadas do
                        componente ENTAO
26                      CDC <- CDC - 1;
27                  FIM-SE
28          FIM-PARA
29  FIM
```

A listagem 6.9 refere-se ao algoritmo para análise de impacto da refatoração para Ovelha Negra (seção 5.3). Este algoritmo é estrutura de maneira semelhante ao anterior, porém com as diferenças da refatoração para Ovelha Negra.

Listagem 6.9: Algoritmo para análise de impacto da refatoração para Ovelha Negra

```
1
2  PROGRAMA AnaliseImpacto_Refat_OvelhaNegra
3  var
4      n_op_modificadas, n_trechos, n_metodos: Inteiro;
5      CDC, CDO: Inteiro
6      CDLOC: Inteiro ou Intervalo discreto;
7  INICIO
8          CDC, CDO, CDLOC, n_op_modificadas, n_trechos, n_metodos <- 0;
9          CDC <- CDC + 1; //referente ao aspecto como novo componente
               relacionado ao interesse.
10         PARA CADA componente do interesse a ser refatorado FACA
11                 PARA CADA atributo relacionado ao interesse FACA
12                     calculaImpactoMoverAtributoParaAspecto(
                           n_op_modificadas, n_trechos, n_metodos, CDO
                           , CDLOC);
13                 FIM-PARA
14                 PARA CADA metodo relacionado ao interesse FACA
15                      calculaImpactoMoverMetodoParaAspecto(CDLOC);
16                 FIM-PARA
17                 SE houver fragmento de codigo relacionado ao interesse
                       ENTAO
18                     PARA CADA fragmento FACA
19                             calculaImpactoExtrairFragmentoParaAdendo
                                   (CDO);
20                     FIM-PARA
21                 FIM-SE
22                 SE todas as partes do interesse foram eliminadas do
                       componente ENTAO
23                     CDC <- CDC - 1;
```

```
24                      FIM-SE
25              FIM-PARA
26  FIM
```

A listagem 6.10 mostra o algoritmo para análise de impacto da refatoração para Dominador (seção 6.10). Nesse caso, o algoritmo prevê a criação de mais de um aspecto caso o interesse dominador seja decomposto em outros interesses e caso seja necessário mais de um aspecto. Nos outros casos, considera-se, a princípio, um aspecto criado para modularizar o interesse transversal.

Listagem 6.10: Algoritmo para análise de impacto da refatoração para interesse Dominador

```
1
2  PROGRAMA AnaliseImpacto_Refat_Dominador
3  var
4      n_op_modificadas, n_trechos, n_metodos: Inteiro;
5      CDC, CDO: Inteiro
6      CDLOC: Inteiro ou Intervalo discreto;
7  INICIO
8          CDC, CDO, CDLOC, n_op_modificadas, n_trechos, n_metodos <- 0;
9          SE houver uma decomposicao do interesse em outros ENTAO
10                 PARA CADA novo interesse FACA
11                         CDC <- CDC + 1; //referente a criacao de um
                               novo aspecto para cada novo interesse.
12                 FIM-PARA
13         FIM-SE
14         PARA CADA componente a ser refatorado FACA
15                 SE houver heranca ou impl. de interface relacionada ao
                       interesse ENTAO
16                         calculaImpactoMoverImplementsExtends(CDLOC);
17                 FIM-SE
18                 PARA CADA atributo relacionado ao interesse FACA
19                         calculaImpactoMoverAtributoParaAspecto(
                               n_op_modificadas, n_trechos, n_metodos, CDO
                               , CDLOC);
20                 FIM-PARA
21                 PARA CADA metodo relacionado ao interesse FACA
22                          calculaImpactoMoverMetodoParaAspecto(CDLOC);
23                 FIM-PARA
24                 SE houver fragmento de codigo relacionado ao interesse
                       ENTAO
25                         PARA CADA fragmento FACA
26                                 calculaImpactoExtrairFragmentoParaAdendo
                                       (CDO);
27                         FIM-PARA
28                 FIM-SE
29                 SE todas as partes do interesse foram eliminadas do
                       componente ENTAO
30                         CDC < - CDC - 1;
31                 FIM-SE
32         FIM-PARA
33 FIM
```

A listagem 6.11 demonstra o algoritmo para análise de impacto da refatoração para Planta Trepadeira (seção 5.6). Esse algoritmo está centrado em dois laços principais: o primeiro para o impacto na refatoração do componente raiz, e o outro para o impacto na refatoração dos componentes galhos.

Listagem 6.11: Algoritmo para análise de impacto da refatoração para interesse Planta T.

```
1
2  PROGRAMA AnaliseImpacto_Refat_Planta
3  var
4     n_op_modificadas, n_trechos, n_metodos: Inteiro;
5     CDC, CDO: Inteiro
6     CDLOC: Inteiro ou Intervalo discreto;
7  INICIO
8           CDC, CDO, CDLOC, n_op_modificadas, n_trechos, n_metodos <- 0;
9           CDC <- CDC + 1; //referente ao aspecto como novo componente
                relacionado ao interesse.
10          PARA CADA componente raiz a ser refatorado FACA
11                  PARA CADA atributo relacionado ao interesse FACA
12                          calculaImpactoMoverAtributoParaAspecto(
                                n_op_modificadas, n_trechos, n_metodos, CDO
                                , CDLOC);
13                  FIM-PARA
14                  PARA CADA metodo relacionado ao interesse FACA
15                           calculaImpactoMoverMetodoParaAspecto(CDLOC);
16                  FIM-PARA
17                  SE houver fragmento de codigo relacionado ao interesse
                        ENTAO
18                          PARA CADA fragmento FACA
19                                  calculaImpactoExtrairFragmentoParaAdendo
                                        (CDO);
20                          FIM-PARA
21                  FIM-SE
22                  SE todas as partes do interesse foram eliminadas do
                        componente ENTAO
23                          CDC <- CDC - 1;
24                  FIM-SE
25          FIM-PARA
26          PARA CADA componente galho a ser refatorado FACA
27                  SE houver heranca ou implementacao de interface
                        relacionada ao interesse ENTAO
28                          calculaImpactoMoverImplementsExtends(CDLOC);
29                  FIM-SE
30                  PARA CADA atributo relacionado ao interesse FACA
31                          calculaImpactoMoverAtributoParaAspecto(
                                n_op_modificadas, n_trechos, n_metodos, CDO
                                , CDLOC);
32                  FIM-PARA
33                  PARA CADA metodo relacionado ao interesse FACA
34                           calculaImpactoMoverMetodoParaAspecto(CDLOC);
35                  FIM-PARA
36                  SE houver fragmento de codigo relacionado ao interesse
                        ENTAO
37                          PARA CADA fragmento FACA
38                                  calculaImpactoExtrairFragmentoParaAdendo
                                        (CDO);
39                          FIM-PARA
40                  FIM-SE
41                  SE todas as partes do interesse foram eliminadas do
                        componente ENTAO
42                          CDC <- CDC - 1;
43                  FIM-SE
44          FIM-PARA
45 FIM
```

A listagem 6.12 refere-se ao algoritmo de impacto da refatoração para Doença Hereditária (seção 5.7). Esse algoritmo é semelhante aos anteriores, porém com a diferença de que há um laço específico para o impacto da refatoração sobre os componentes descendentes sadios.

Listagem 6.12: Algoritmo para análise de impacto da refatoração para interesse Doença Hereditária

```
1
2  PROGRAMA AnaliseImpacto_Refat_DoencaHereditaria
3  var
4      n_op_modificadas, n_trechos, n_metodos: Inteiro;
5      CDC, CDO: Inteiro
6      CDLOC: Inteiro ou Intervalo discreto;
7  INICIO
8          CDC, CDO, CDLOC, n_op_modificadas, n_trechos, n_metodos <- 0;
9          CDC <- CDC + 1; //referente ao aspecto como novo componente
               relacionado ao interesse.
10         PARA CADA componente Origem da Doenca a ser refatorado FACA
11                 PARA CADA atributo relacionado ao interesse FACA
12                         calculaImpactoMoverAtributoParaAspecto(
                               n_op_modificadas, n_trechos, n_metodos, CDO
                               , CDLOC);
13                 FIM-PARA
14                 PARA CADA metodo relacionado ao interesse FACA
15                         calculaImpactoMoverMetodoParaAspecto(CDLOC);
16                 FIM-PARA
17                 SE houver fragmento de codigo relacionado ao interesse
                       ENTAO
18                         PARA CADA fragmento FACA
19                                 calculaImpactoExtrairFragmentoParaAdendo
                                       (CDO);
20                         FIM-PARA
21                 FIM-SE
22                 SE todas as partes do interesse foram eliminadas do
                       componente ENTAO
23                         CDC <- CDC - 1;
24                 FIM-SE
25         FIM-PARA
26         PARA CADA componente descendente sadio FACA
27                 SE o seu ancestral direto for movido completamente para
                        aspecto ENTAO
28                         calculaImpactoMoverImplementsExtends(CDLOC);
29                 FIM-SE
30         FIM-PARA
31         PARA CADA componente descendente a ser refatorado FACA
32                 SE houver heranca ou implementacao de interface
                       relacionada ao interesse ENTAO
33                         calculaImpactoMoverImplementsExtends(CDLOC);
34                 FIM-SE
35                 PARA CADA atributo relacionado ao interesse FACA
36                         calculaImpactoMoverAtributoParaAspecto(
                               n_op_modificadas, n_trechos, n_metodos, CDO
                               , CDLOC);
37                 FIM-PARA
38                 PARA CADA metodo relacionado ao interesse FACA
39                         calculaImpactoMoverMetodoParaAspecto(CDLOC);
40                 FIM-PARA
```

```
41                  SE houver fragmento de codigo relacionado ao interesse
                        ENTAO
42                      PARA CADA fragmento FACA
43                              calculaImpactoExtrairFragmentoParaAdendo
                                    (CDO);
44                      FIM-PARA
45                  FIM-SE
46                  SE todas as partes do interesse foram eliminadas do
                        componente ENTAO
47                      CDC <- CDC - 1;
48                  FIM-SE
49          FIM-PARA
50  FIM
```

O algoritmo da listagem 6.13 calcula o impacto das refatorações para os sintomas relativos a acoplamento (seção 5.8). Assim como há um conjunto de passos de refatoração unificado para todos os sintomas relativos a acoplamento, há também um único algoritmo que calcula o impacto para esses casos.

Listagem 6.13: Algoritmo para análise de impacto da refatoração para sintomas relativos a acomplamento

```
1
2  PROGRAMA AnaliseImpacto_Refat_SintomasRelativosAcoplamento
3  var
4     n_op_modificadas, n_trechos, n_metodos: Inteiro;
5     CDC, CDO: Inteiro
6     CDLOC: Inteiro ou Intervalo discreto;
7  INICIO
8           CDC, CDO, CDLOC, n_op_modificadas, n_trechos, n_metodos <- 0;
9           CDC <- CDC + 1; //referente ao aspecto como novo componente
                relacionado ao interesse.
10          PARA CADA componente a ser refatorado FACA
11                  PARA CADA atributo relacionado ao interesse FACA
12                          calculaImpactoMoverAtributoParaAspecto(
                                n_op_modificadas, n_trechos, n_metodos, CDO
                                , CDLOC);
13                  FIM-PARA
14                  PARA CADA metodo relacionado ao interesse FACA
15                          calculaImpactoMoverMetodoParaAspecto(CDLOC);
16                  FIM-PARA
17                  SE houver fragmento de codigo relacionado ao interesse
                        ENTAO
18                      PARA CADA fragmento FACA
19                              calculaImpactoExtrairFragmentoParaAdendo
                                    (CDO);
20                      FIM-PARA
21                  FIM-SE
22                  SE todas as partes do interesse foram eliminadas do
                        componente ENTAO
23                      CDC <- CDC - 1;
24                  FIM-SE
25          FIM-PARA
26  FIM
```

A listagem 6.14 mostra o algoritmo que calcula o impacto da refatoração para Cópia Carbono (seção 5.9). Como a refatoração para Ovelha Dolly é a mesma da Cópia Carbono, este algoritmo também pode ser usado para o impacto da refatoração para Ovelha Dolly. Nesse caso, a estrutura interna da listagem difere um pouco das demais. Há

uma laço maior para cada componente cópia que verifica as diversas possibilidades de refatoração conforme está definida na seção 5.9

Listagem 6.14: Algoritmo para análise de impacto das refatorações para Cópia Carbono e Ovelha Dolly

```
1
2  PROGRAMA AnaliseImpacto_Refat_CopiaCarbono_OvelhaDolly
3  var
4     n_op_modificadas, n_trechos, n_metodos: Inteiro;
5     CDC, CDO: Inteiro
6     CDLOC: Inteiro ou Intervalo discreto;
7  INICIO
8     CDC, CDO, CDLOC, n_op_modificadas, n_trechos, n_metodos <- 0;
9     CDC <- CDC + 1; //referente ao aspecto como novo componente
          relacionado ao interesse.
10    PARA CADA copia FACA
11       SE os componentes que possuem as copias estao envolvidos em uma
             relacao de heranca em comum ENTAO
12          Selecionar uma das copias do grupo para ser herdada de um
               componente ancestral;
13          SE a copia for um fragmento de codigo ENTAO
14             calculaImpactoExtrairMetodo(CDO, CDLOC);
15             calculaImpactoGeneralizarMethod(CDC, CDLOC);
16             PARA as demais copias do grupo FACA
17                SE a operacao que contem o fragmento copiado nao tiver
                     outro fragmento relacionado ao interesse ENTAO
18                   CDO <- CDO - 1;
19                FIM-SE
20             FIM-PARA
21          SENAO SE a copia for um atributo ENTAO
22             calculaImpactoGeneralizarField(CDC, CDLOC, CDO);
23             PARA as demais copias do grupo FACA
24                SE houver metodos acessores para o atributo no
                     componente e ja existirem tambem no componente
                     ancestral que recebeu o atributo ENTAO
25                   CDO <- CDO - 2; //referente a remocao de 2
                        metodos acessores.
26                FIM-SE
27                SE nao houver trechos de codigo relacionados ao
                     interesse imediatamente adjacentes ao atributo
                     ENTAO
28                   CDLOC <- CDLOC - 2;
29                FIM-SE
30             FIM-PARA
31          SENAO SE a copia for um metodo ENTAO
32             calculaImpactoGeneralizarMethod(CDC, CDLOC);
33             PARA as demais copias do grupo FACA
34                SE nao houver trechos de codigo relacionados ao
                     interesse imediatamente adjacentes ao metodo ENTAO
35                   CDLOC <- CDLOC - 2;
36                FIM-SE
37                CDO <- CDO - 1;//referente a remocao do metodo pois ele
                        esta sendo herdado do componente ancestral.
38             FIM-PARA
39          FIM-SE
40       SENAO SE a copia for um fragmento ENTAO
41          calculaImpactoExtrairFragmentoParaAdendo(CDO);
42       SENAO SE a copia for um atributo ENTAO
```

```
43              calculaImpactoMoverAtributoParaAspecto(n_op_modificadas,
                    n_trechos, n_metodos, CDO, CDLOC);
44          SENAO SE a copia for um metodo ENTAO
45              calculaImpactoMoverMetodoParaAspecto(CDLOC);
46          FIM-SE
47          SE o componente que possui a copia nao tiver mais partes
                relacionadas ao interesse ENTAO
48              CDC <- CDC - 1;
49          FIM-SE
50      FIM-PARA
51  FIM
```

Em seguida, a listagem 6.15 demonstra o algoritmo para a análise de impacto da refatoração para Interesse Exclusivo de Dados (seção 5.11). Esse algoritmo considera a estratégia de refatoração que utiliza aspectos, embora a refatoração na seção 5.11 inclua uma alternativa sem o uso de aspectos. Nesse caso, como o interesse é composto apenas de atributos, há apenas um laço de repetição para cada atributo a ser refatorado que chama o procedimento *calculaImpactoMoverAtributoParaAspecto* e ao fim verifica se cada componente refatorado deixou de fazer parte do interesse em questão.

Listagem 6.15: Algoritmo para análise de impacto da refatoração para interesse Exclusivo de Dados

```
1
2  PROGRAMA AnaliseImpacto_Refat_ExclusivoDados
3  var
4      n_op_modificadas, n_trechos, n_metodos: Inteiro;
5      CDC, CDO: Inteiro
6      CDLOC: Inteiro ou Intervalo discreto;
7  INICIO
8          CDC, CDO, CDLOC, n_op_modificadas, n_trechos, n_metodos <- 0;
9          CDC <- CDC + 1; //referente ao aspecto como novo componente
               relacionado ao interesse.
10         PARA CADA atributo a ser refatorado FACA
11                 calculaImpactoMoverAtributoParaAspecto(n_op_modificadas
                       , n_trechos, n_metodos, CDO, CDLOC);
12                 SE todos os atributos do componente relacionados ao
                       interesse foram eliminados ENTAO
13                         CDC <- CDC - 1;
14                 FIM-SE
15         FIM-PARA
16 FIM
```

Na listagem 6.16 está o algoritmo para cálculo do impacto sobre a refatoração para Interesse Comportamental (seção 5.12). De maneira semelhante à situação anterior, considera-se também nesta listagem a estratégia de refatoração que utiliza aspectos. Nesse caso, como o interesse é composto apenas de métodos, o laço concentra-se em chamar o procedimento *calculaImpactoMoverMetodoParaAspecto* e verificar se cada componente refatorado deixou de participar do interesse.

Listagem 6.16: Algoritmo para análise de impacto da refatoração para interesse Comportamental

```
1
2  PROGRAMA AnaliseImpacto_Refat_Comportamental
3  var
4      CDC, CDLOC: Inteiro
5  INICIO
```

```
6           CDC, CDLOC, n_op_modificadas, n_trechos, n_metodos <- 0;
7           CDC <- CDC + 1; //referente ao aspecto como novo componente
                relacionado ao interesse.
8           PARA CADA metodo a ser refatorado FACA
9                   calculaImpactoMoverMetodoParaAspecto(CDLOC);
10                  SE todos os metodos do componente relacionados ao
                        interesse foram eliminados ENTAO
11                          CDC <- CDC - 1;
12                  FIM-SE
13          FIM-PARA
14  FIM
```

6.4 Considerações Finais do Capítulo

Neste capítulo apresentou-se os algoritmos de análise de impacto sobre a aplicação das refatorações propostas no capítulo anterior. Estes algoritmos compõem o método de refatoração do presente trabalho e estão divididos em duas partes: a primeira reúne as rotinas de impacto referentes à aplicação de refatorações de granularidade mais fina; e a segunda parte compreende os algoritmos maiores, para o cálculo da análise de impacto das refatorações para os sintomas e que realizam chamadas às rotinas de impacto. O cálculo do impacto é feito sobre três métricas para separação de interesse (CDC, CDO e CDLOC) e se baseia em um estudo anterior proposto em (PAGLIARI; NUNES, 2007) e (PAGLIARI, 2007). Na prática, estes algoritmos de análise de impacto são essenciais para a realização de uma avaliação quantitativa e objetiva sobre as possibilidades, prós e contras quanto à aplicação das refatorações. É possível também que essa proposta seja estendida através da inclusão de novas métricas modificando os algoritmos existentes ou até mesmo criando novos algoritmos seguindo a mesma abordagem para novas refatorações. O próximo capítulo apresenta o trabalho de avaliação realizado para verificar a aplicabilidade do método de refatoração.

7 AVALIAÇÃO DO MÉTODO

Este capítulo tem como objetivo apresentar o trabalho de avaliação realizado a fim de verificar a aplicabilidade do método de refatoração para modularização de interesses transversais. Para tanto, dois estudos de caso foram conduzidos envolvendo os sistemas de software *Health Watcher* e *Mobile Media*. Estes sistemas são aplicações reais e reúnem um conjunto heterogêneo de interesses transversais disponíveis para análise. Juntos somam 22 interesses (12 do *Mobile Media* e 10 do *Health Watcher*). Além disso, estes dois sistemas têm sido alvos de diversos estudos (FILHO et al., 2006) (GREENWOOD et al., 2007) (FIGUEIREDO et al., 2008) (FIGUEIREDO; CACHO, 2008) (SANT'ANNA, 2008).

O capítulo encontra-se organizado da seguinte forma: a seção 7.1 descreve o procedimento de estudo realizado no processo de avaliação; em seguida, as seções 7.2 e 7.3 descrevem os sistemas alvos do estudo, respectivamente o *Mobile Media* e o *Health Watcher*; posteriormente, na seção 7.4 são apresentados os resultados da etapa de detecção dos sintomas; na seção 7.5, os resultados da análise de impacto são expostos e discutidos; a seção 7.6 discute e apresenta os resultados da aplicação das refatorações; depois, a seção 7.7 expõe as limitações do estudo; e, por fim, a seção 7.8 conclui este capítulo.

7.1 Procedimentos de Estudo

Segundo Mens e Tourwé (2004), um processo de refatoração envolve outras atividades além de simplesmente aplicar as refatorações, como por exemplo: identificar as oportunidades de refatoração; determinar quais refatorações devem ser aplicadas para as oportunidades identificadas; e avaliar o efeito da refatoração em atributos de qualidade do software (compreensão, manutenibilidade etc.) ou do processo (produtividade, custo etc.). Portanto, neste trabalho procurou-se não somente aplicar as refatorações, mas também avaliar o método de refatoração como um todo, que envolve a aplicabilidade: da abordagem de detecção e classificação de interesses através de sintomas proposto em (FIGUEIREDO et al., 2009) e apresentada no capítulo 4; dos algoritmos para análise de impacto das refatoraçãoes (capítulo 6); e das refatorações voltadas para modularização de interesses transversais (capítulo 5).

O protocolo de estudo foi conduzido de acordo com uma única metodologia para os dois estudos de caso. Portanto, para ambos os sistemas-alvo realizaram-se os mesmos passos:

1. ***Identificação dos interesses transversais como oportunidades para refatoração:*** selecionar os interesses transversais e seus componentes a serem avaliados e classificados de acordo com os sintomas sensíveis ao interesse discutidos no capítulo

4.

2. ***Coleta de dados através de métricas:*** aplicação de três métricas para medir a separação dos interesses selecionados no passo anterior. As três métricas usadas (CDC, CDO e CDLOC) foram apresentadas e discutidas no capítulo 3).

3. ***Execução de heurísticas para detecção dos sintomas:*** usar as heurísticas apresentadas na seção 4.3 para identificar os sintomas sensíveis ao interesse sobre os interesses transversais selecionados no passo 1. Este passo irá indicar, de acordo com a identificação dos sintomas, as refatorações possíveis de serem aplicadas.

4. ***Aplicação dos algoritmos para análise de impacto das refatorações:*** aplicar os algoritmos de análise de impacto apresentados no capítulo 6 para as refatorações correspondentes aos sintomas de cada interesse em questão.

5. ***Aplicação das refatorações:*** aplicar as refatorações demonstradas no capítulo 5 para os sintomas sensíveis ao interesse. Nesta etapa, todas as refatorações devem ser aplicadas sobre a mesma versão do sistema, sem ter sofrido alguma reestruturação prévia. Portanto, não serão aplicadas refatorações sobre versões já refatoradas do sistema.

6. ***Nova coleta de dados:*** reaplicação do conjunto de métricas utilizado no passo 2, porém sobre as versões refatoradas dos interesses.

7. ***Análise de resultados:*** realização de uma análise quantitativa e qualitativa sobre os resultados obtidos antes e após a aplicação das refatorações. Adicionalmente, confrontar os resultados levantados sobre a aplicação dos algoritmos de análise de impacto com a coleta de dados da própria realização da refatoração.

7.2 O Sistema *Mobile Media*

O *Mobile Media*[1] (YOUNG; MURPHY, 2005) (YOUNG, 2005) consiste em uma linha de produto de software (CLEMENTS; NORTHROP, 2002) (POHL; BöCKLE; LINDEN, 2005) para aplicações de dispositivos móveis (aparelhos celulares, *smartphones* etc.) incluindo funcionalidades como manipulação de fotos, músicas, vídeos e mensagens instantâneas. Este sistema tem aproximadamente 3000 linhas de código e foi desenvolvido baseado em uma versão inicial também de uma linha de produto de software chamada *Mobile Photo* (YOUNG, 2005) desenvolvido na Universidade de *British Columbia*. Para a implementação do *Mobile Media*, os desenvolvedores estenderam a implementação inicial (*Mobile Photo*) incluindo novas *features* mandatórias, opcionais e alternativas como forma de agregar novas funcionalidades ao longo das versões da linha de produto do *Mobile Media* (FIGUEIREDO; CACHO, 2008).

As *features* inclusas em uma linha de produto de software são frequentemente descritas através de uma notação gráfica chamada de Modelo de *Features* (POHL; BöCKLE; LINDEN, 2005). A figura 7.1 apresenta uma visão simplificada do modelo de *features* do *Mobile Media*. Este modelo representa as *features* incluídas até a última versão disponível durante o presente estudo. De acordo com o modelo da figura, as *features* alternativas referem-se aos tipos de mídias suportadas: foto (*Photo*), música (*Music*) e/ou vídeo (*Video*). Exemplos de *features* mandatórias são as operações básicas sobre as mídias (*Basic*

[1] http://sourceforge.net/projects/mobilemedia/

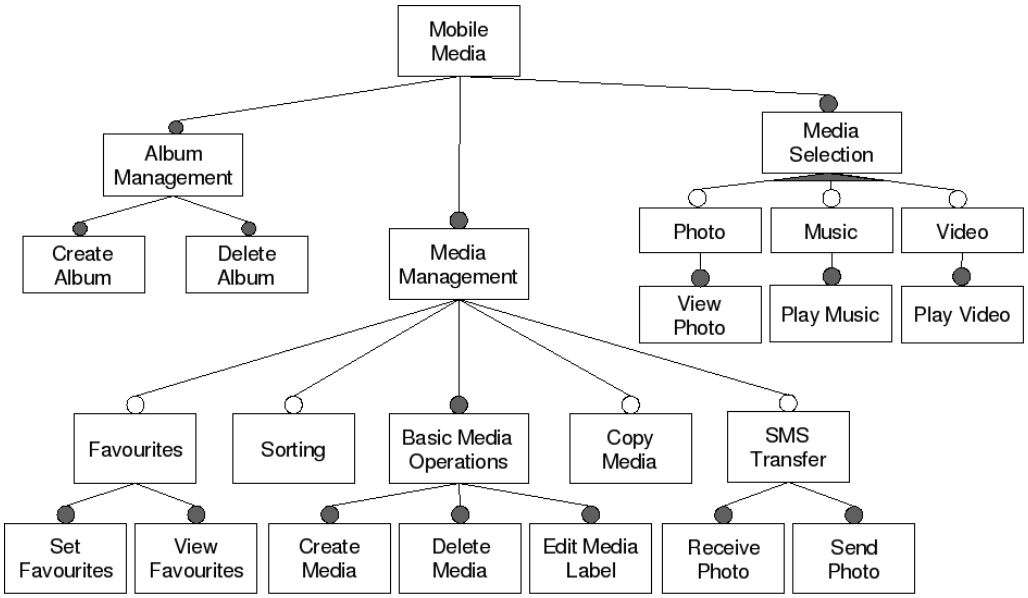


Figura 7.1: Modelo de *features* do *Mobile Media*

Media Operations), criar e deletar álbum (*Album Management*), entre outras. As *features* opcionais referem-se ao gerenciamento de mídias, por exemplo, indicar e visualizar favoritos (*Set/View Favourites*), transferir mensagens instatâneas (*SMS Transfer*), entre outras. As *features* mandatórias do *Mobile Media* são aplicáveis a todos os dispositivos móveis que suportam a plataforma *Java Micro Edition* (Java ME) (SUN, 2008). As *features* opcionais e alternativas são configuráveis dependendo do suporte disponível das APIs em cada dispositivo. O *Mobile Media* foi desenvolvido para uma família de quatro marcas de dispositivos móveis: Nokia, Motorola, Siemens e RIM (YOUNG; MURPHY, 2005) (YOUNG, 2005).

A linha de produto do *Mobile Media* inclui um total de sete cenários de modificação, incorporando assim 7 versões a partir da versão inicial que seria o *Mobile Photo*. Os cenários compreendem diferentes tipos de mudanças incluindo *features* mandatórias, opcionais e alternativas, assim como interesses de requisitos não-funcionais. A tabela 7.1 (adaptada de (FIGUEIREDO; CACHO, 2008)) resume sequencialmente as modificações realizadas da linha de produto do *Mobile Media*. Cada linha da tabela refere-se aos diferentes cenários correspondentes às suas versões, sendo que para cada versão (coluna 1) há uma descrição (coluna 2) e o tipo de modificação (coluna 3).

Para a realização deste estudo, selecionou-se a última versão estável do *Mobile Media* (até o momento de escrita deste trabalho) - a versão 07. Esta versão possui 12 interesses catalogados, que são: *CaptureMedia*, *Controller*, *CopyMedia*, *Exception Handling*, *Favourites*, *LabelMedia*, *Music*, *Persistence*, *Photo*, SMS, *Sorting*, *Video*.

7.3 O Sistema *Health Watcher*

O segundo alvo de estudo envolveu o software *Health Watcher* (SOARES; LAUREANO; BORBA, 2002). O *Health Watcher* é um sistema de informação baseado na *web* com o principal objetivo de automatizar o registro, encaminhamento e gerenciamento de reclamações para o sistema público de saúde ou órgãos de vigilância sanitária sobre restaurantes, açougues, supermercados e similares. Dessa maneira, o público em geral pode

Tabela 7.1: Sumário dos cenários no *Mobile Media*

Versão	Descrição	Tipo de Mudança
V00	Núcleo do *Mobile Photo*.	
V01	Tratamento de exceção incluído.	Inclusão de um interesse não funcional.
V02	Duas novas *features* adicionadas: uma para contar o número de visualizações das fotos e ordená-las pela frequência em que são vistas. E outra para editar o nome da foto.	Inclusão de *feature* opcional e mandatória.
V03	Nova *feature* adicionada para permitir que usuários indiquem e vejam suas fotos favoritas.	Inclusão de *feature* opcional.
V04	Nova *feature* adicionada para permitir que usuários matenham múltiplas cópias de suas fotos.	Inclusão de *feature* opcional.
V05	Nova *feature* adicionada para o envio de fotos via SMS (mensagens instantâneas)	Inclusão de *feature* opcional.
V06	Nova *feature* adicionada para armazenar, tocar e organizar músicas. O gerenciamento de fotos (criar, deletar, e renomear) tornou-se uma *feature* alternativa. Todas as funcionalidades estendidas (ordenação, favoritos e transferência de SMS) continuam sendo providas.	Mudança de uma *feature* mandatória em duas *features* alternativas.
V07	Nova *feature* adicionada para gerenciar vídeos.	Inclusão de *feature* alternativa.

registrar reclamações através de uma interface *web* para que as entidades responsáveis possam investigar os casos e realizar as ações necessárias.

O *Health Watcher* foi escolhido como alvo de um dos estudos de caso por ser um sistema real, atualmente com implementações em Java e em AspectJ, cada uma com aproximadamente 4000 linhas de código. A primeira versão da implementação em Java do *Health Watcher* foi distribuída em 2001 pelo Sistema Público de Saúde da cidade do Recife. Desde então, o *Health Watcher* vem sofrendo atualizações e modificações incrementais ao longo de novas versões. Além disso, este sistema tem sido alvo de estudos relacionados à avaliação de técnicas de modularização através de aspectos, como em (SOARES; LAUREANO; BORBA, 2002), (FILHO et al., 2006), (GREENWOOD et al., 2007) e (SANT'ANNA, 2008).

Os interesses do sistema *Health Watcher* considerados neste estudo de caso envolvem também a utilização de padrões de projeto. São eles: concorrência, distribuição, persistência, *Command*, *Observer*, *State*, duas instâncias do padrão *Abstract Factory* e duas instâncias do *Adapter*.

7.4 Detecção dos Sintomas

As heurísticas de detecção de sintomas sensíveis ao interesse discutidas no capítulo 4 foram aplicadas para os 12 interesses da versão 07 do *Mobile Media*, assim como para os 10 interesses do *HealthWatcher*. A realização das heurísticas de detecção envolve a definição de limiares para alguns sintomas. Estes limiares definem a condição mínima para que um determinado interesse se encaixe na detecção do sintoma e podem variar de acordo com a necessidade de cada usuário do método de detecção. Neste estudo foram considerados os limiares mostrados na tabela 7.2.

Assim, para um sintoma ser classificado como Polvo é necessário ter ao menos 1 componente como corpo (alta dedicação) e 2 componentes como tentáculos (baixa dedi-

cação), sendo que os limiares de alta e baixa dedicação são, respectivamente, $\geq 67\%$ e $\leq 33\%$. Para o Tsunami, por exemplo, considerou-se a necessidade de ter no mínimo 3 componentes participantes. Alguns sintomas não dependem de limiares e, por isso, não aparecem na tabela 7.2. O ajuste dos limiares implica na reclassificação dos interesses e deve variar com a necessidade de cada usuário (profissional desenvolvedor, projetista etc.) em cada contexto. Maiores explicações sobre as heurísticas encontram-se na seção 4.3.

Tabela 7.2: Limiares paras as heurístidas de detecção dos sintomas

Sintoma	**Limiar**
Polvo	Mínimo de 1 componente corpo e 2 componentes tentáculos. Dedicação baixa ($\leq 33\%$). Dedicação alta ($\geq 67\%$).
Ovelha Negra	Mínimo de 3 componentes participantes. Dedicação baixa ($\leq 33\%$).
Dominador	Dominante em mais de 33% do sistema.
Planta Trepadeira	Mínimo de 3 componentes participantes.
Doença Hereditária	Mínimo de 1 componente descendente sadio.
Enraizado	Mínimo de 3 componentes ramos.
Serpente	Mínimo de 3 componentes participantes.
Tsunami	Mínimo de 3 componentes participantes.
Rede Neural	Mínimo de 2 componente na camada de entrada e Mínimo de 2 na camada de saída.
Tsunami	Mínimo de 3 componentes participantes.

Após a aplicação das heurísticas considerando os limiares apresentados na tabela 7.2, foi possível identificar diversos sintomas sobre os 22 interesses dos dois sistemas. As tabelas 7.3 e 7.4 resumem a classificação desses interesses quanto aos 13 sintomas considerados no estudo para o *Mobile Media* e o *Health Watcher*, respectivamente.

Tabela 7.3: Classificação dos interesses no *Mobile Media* versão 7

Interesse	**PO**	**ON**	**DM**	**PL**	**DH**	**TS**	**EN**	**SE**	**RN**	**CC**	**OD**	**ED**	**CO**
CaptureMedia	x					x	x	x					
Controller	x			x									
CopyMedia	x												
Exception Handling	x			x	x	x	x		x	x			
Favourites		x											
LabelMedia	x			x		x	x	x	x				
Music	x					x	x	x	x				
Persistence	x			x		x	x	x	x	x			
Photo	x					x	x	x	x				
SMS	x					x	x	x					
Sorting		x						x					
Video	x					x	x	x	x				

LEGENDA: **PO** - Polvo, **ON** - Ovelha Negra, **DM** - Dominador, **PL** - Planta Trepadeira, **DH** - Doenção Hereditária, **TS** - Tsunami, **EN** - Enraizado, **SE** - Serpente, **RN** - Rede Neural, **CC** - Cópia Carbono, **OD** - Ovelha Dolly, **ED** - Exclusivo de Dados, **CO** - Comportamental.

Como é possível notar, em ambas as tabelas, 4 sintomas não foram detectados nos interesses. No caso do *Mobile Media* (tabela 7.3), não houve a ocorrência dos sintomas

Dominador, Ovelha Dolly, Interesse Exclusivo de Dados e Comportamental. Enquanto que no *Health Watcher* não foram encontrados: Ovelha Negra, Ovelha Dolly, Interesse Exclusivo de Dados e Comportamental. Comparando os dois casos, percebe-se que coincide a ausência de 3 interesses: Ovelha Dolly, Interesse Exclusivo de Dados e Comportamental. Estes 3 sintomas são relacionados a situações muito peculiares e já era prevista a dificuldade de encontrá-los nos estudos de caso. Consequentemente, não foi possível aplicar os algoritmos de análise de impacto e as refatorações sobre interesses com estes sintomas.

Já o sintoma Dominador foi encontrado em apenas 1 interesse de um dos sistemas-alvo - o *Health Watcher*. Este sintoma opõem-se justamente às idéias básicas de modularidade e revela-se, portanto, um problema grave de separação de interesses. O Dominador é de fato raro de ocorrer em sistemas que já sofreram diversas manutenções e que já se encontram estáveis. Ele foi encontrado no interesse *Command* do *Health Watcher*, pois grande parte da implementação de funcionalidades deste sistema se concentra na estrutura do padrão de projeto *Command*.

Tabela 7.4: Classificação dos interesses no *Health Watcher*

Interesse	**PO**	**ON**	**DM**	**PL**	**DH**	**TS**	**EN**	**SE**	**RN**	**CC**	**OD**	**ED**	**CO**
Concorrência	x						x						
Distribuição	x					x	x		x	x			
Persistência	x					x	x		x	x			
Abstract Factory (1)				x		x		x					
Abstract Factory (2)	x							x	x				
Adapter (1)						x		x					
Adapter (2)						x							
Command			x	x		x				x			
Observer	x			x	x					x			
State				x		x	x	x	x				

LEGENDA: **PO** - Polvo, **ON** - Ovelha Negra, **DM** - Dominador, **PL** - Planta Trepadeira, **DH** - Doenção Hereditária, **TS** - Tsunami, **EN** - Enraizado, **SE** - Serpente, **RN** - Rede Neural, **CC** - Cópia Carbono, **OD** - Ovelha Dolly, **ED** - Exclusivo de Dados, **CO** - Comportamental.

Os sintomas menos encontrados no *Mobile Media* (tabela 7.3) foram, em ordem crescente de ocorrência: Doença Heteditária, Cópia Carbono e Ovelha Negra. Em contrapartida, os mais recorrentes foram, também em ordem crescente de ocorrência: Planta Repadeira e Rede Neural, detectados em 4 e 6 interesses respectivamente; em seguida, Interesse Enraizado, Serpente e Tsunami, os três com 8 ocorrências; e por fim, Polvo, encontrado em 10 interesses do *Mobile Media*.

Já observando a frequência dos sintomas no *Health Watcher* (tabela 7.4), nota-se que os menos encontrados foram: Dominador e Doença Hereditária, com apenas 1 ocorrência. A maioria dos sintomas foram detectados 4 ou 5 vezes, com exceção do Tsunami verificado em 7 dos 10 interesses.

Vale observar a existência de sobreposição de sintomas na maioria dos interesses analisados em ambos os estudos. Observado a tabela 7.3 referente ao *Mobile Media*, em somente um caso, o interesse *CopyMedia*, foi detectado apenas 1 sintoma. Em todos os demais interesses foram detectados ao menos 2 sintomas, chegando aos casos maiores de: 7 sintomas encontrados nos interesses *Persistence* e *Exception Handling*; e 6 sintomas no *LabelMedia*. Pelo *Health Watcher*, tabela 7.4, também é possível notar apenas 1 interesse com somente 1 sintoma, o *Adapter (1)*.

Vale ressaltar que a existência de sobreposição de sintomas em um mesmo interese não significa que todos os componentes participantes do interesse estão necessariamente envolvidos em todos os sintomas encontrados no mesmo. Isto é, existem sintomas que envolvem apenas uma parte dos componentes de um interesse. Por exemplo, o conjunto de componentes envolvidos no sintoma Polvo do interesse *CaptureMedia* (8 componentes) é diferente do conjunto de componentes envolvidos no sintoma Serpente (apenas 4 componentes). Possivelmente estes dois conjuntos teriam uma interseção, porém isto também não é condição necessária.

A densidade média de sintomas no *Mobile Media* é de 4.08 sintomas por interesse, enquanto que no *Health Watcher* é de 3.4. Este valor é obtido pela soma dos sintomas encontrados em cada interesse dividido pela quantidade de interesses analisados.

7.5 Análise de Impacto das Refatorações

Para a realização da análise de de impacto antes da aplicação das refatorações, foi feita uma seleção prévia dos interesses em ambos os estudos de caso. Esta escolha foi feita conciliando-se dois critérios: o tamanho dos interesses, isto é, a quantidade de componentes participantes; e a cobertura dos sintomas detectados sobre os interesses. Selecionou-se, portanto, 6 interesses do total de 12 existentes na versão 07 do *Mobile Media* e 8 interesses dos 10 do *Health Watcher*.

No *Mobile Media* os 6 interesses considerados para os trabalhos de refatoração (análise de impacto e aplicação) foram: *CaptureMedia*, *CopyMedia*, *Favourites*, *LabelMedia*, SMS e *Sorting*. A tabela 7.5 reapresenta a classificação dos interesses do *Mobile Media*, porém exibindo apenas os seis interesses selecionados.

Tabela 7.5: Classificação dos interesses selecionados para refatoração no *Mobile Media* versão 7

Interesse	**PO**	**ON**	**DM**	**PL**	**DH**	**TS**	**EN**	**SE**	**RN**	**CC**	**OD**	**ED**	**CO**
CaptureMedia	x					x	x	x					
CopyMedia	x												
Favourites		x											
LabelMedia	x			x		x	x	x	x				
SMS	x					x	x	x					
Sorting		x						x					

LEGENDA: **PO** - Polvo, **ON** - Ovelha Negra, **DM** - Dominador, **PL** - Planta Trepadeira, **DH** - Doenção Hereditária, **TS** - Tsunami, **EN** - Enraizado, **SE** - Serpente, **RN** - Rede Neural, **CC** - Cópia Carbono, **OD** - Ovelha Dolly, **ED** - Exclusivo de Dados, **CO** - Comportamental.

Assim, no estudo do *Mobile Media* foi possível cobrir todos os sintomas detectados com exceção do Doença Hereditária. Este sintoma foi detectado no interesse de tratamento de exceção e optou-se por não incluí-lo no estudo devido ao tamanho do mesmo. Como a realização do estudo foi feita com pouco auxílio ferramental e assim suscetível a erros, dada a ausência de ferramentas para refatorações orientadas a aspectos e para coleta de dados através de métricas, evitou-se o interesse *Exception Handling* para diminuir a possibilidade de inserção de erros durante o estudo.

Os mesmos critérios foram usados para seleção dos interesses no estudo do *Health Watcher*, que foram: Concorrência, *Abstract Factory (1)*, *Abstract Factory (2)*, *Adapter (1)*, *Adapter (2)*, *Command*, *Observer* e *State*. Com estes 8 interesses selecionados para a análise de impacto e posterior aplicação das refatorações, conseguiu-se cobrir todos os

sintomas que aparecem nos interesses considerados na etapa de detecção dos sintomas (seção 7.4).

A realização da análise de impacto das refatorações sobre os interesses nos dois estudos de caso se desdobrou em analisar o impacto de 39 possíveis refatorações para modularização de interesses transversais, conforme a seguinte divisão:

- 18 ocorrências de sintomas sobre os 6 interesses selecionados do *Mobile Media*; mais
- 21 ocorrências de sintomas sobre os 8 interesses selecionados do *Health Watcher*.

Apesar de existirem 24 possíveis refatorações para o *Health Watcher*, de acordo com as 24 ocorrências de sintomas, 3 casos não tiveram análise de impacto, pois decidiu-se não refatorá-los. Estas 3 situações referem-se ao sintoma Planta Trepadeira nos interesses *Abstract Factory (1)*, *Command* e *State*. Nesses casos, as implementações dos componentes envolvidos no sintoma Planta Trepadeira não justificavam a realização de refatoração, pois haviam poucos componentes envolvidos. Assim, visivelmente, os esforços de executar a análise de impacto e a refatoração significariam um trabalho que não seria compensando pelos ganhos insignificantes na separação de interesses. As análises de impacto foram feitas, portanto, para os casos em que iria se aplicar a refatoração na prática.

Esta etapa do estudo foi feita através da aplicação dos algoritmos definidos no capítulo 6. Estes algoritmos computam três métricas de separação de interesses (CDC, CDO e CDLOC), a depender de informações de entrada que variam conforme cada contexto e implementação dos interesses.

As tabelas 7.6 e 7.7 reúnem os resultados obtidos pela realização da análise de impacto para as refatorações dos interesses e seus sintomas detectados, em ambos os estudos. As duas tabelas encontram-se organizadas da seguinte forma: a primeira coluna indica o interesse; a segunda refere-se ao sintoma do interesse; a terceira coluna informa a quantidade de componentes que participam do sintoma no interesse; em seguida, a quarta coluna indica a quantidade de componentes a serem refatorados, isto é, o número de componentes considerados caso a refatoração ocorra; e, por fim, a variação de unidades que pode ocorrer em cada métrica de separação de interesses, ou seja, o impacto em si da análise feita seguindo os algoritmos.

Antes de analisar as tabelas, vale relembrar que quanto menor o valor para as três métricas (CDC, CDO e CDLOC) melhor será a separação de interesses. Em outras palavras, um interesse tende a estar melhor separado, e assim melhor modularizado, se ele estiver espalhado por menos componentes (CDC) e por menos operações (CDO), e se ele estiver entrelaçado por menos linhas de código (CDLOC). Estas métricas são algumas das exploradas em estudo feito e discutido no capítulo 3.

Observa-se, portanto, que as medições indicaram melhorias, de maneiral geral. Isto é, obteve-se valores negativos ou neutros (sem variação) para a maioria dos casos da análise de impacto em ambos os estudos.

Vale destacar, entretanto, que os resultados com a métrica CDO se comportam de maneira diferente pela seguinte razão: alguns passos de refatoração muitas vezes indicam a necessidade de criação de adendos nos aspectos para introduzir fragmentos extraídos que se encontram entrelaçados nos componentes, sendo um adendo considerado também como uma operação. Assim, sempre que o passo de refatoração for a aplicação de "Extrair fragmento para adendo" existe a possibilidade de o valor de CDO aumentar 1 unidade pela criação de adendos. Portanto, a análise de impacto verifica as seguintes possibilidades:

Tabela 7.6: Resultados da análise de impacto para refatorações no *Mobile Media*

Análise de Impacto das Refatorações no *Mobile Media*						
Interesse	**Sintoma**	**Participantes do sintoma**	**Componentes a serem refatorados**	**Variação das Medições de Separação de Interesses**		
				CDC	**CDO**	**CDLOC**
CaptureMedia	Polvo	8	5	-4	0	-14
	Tsunami	7	4	-3	0	-10
	Enraizado	5	3	-2	0	-6
	Serpente	4	3	-2	0	-6
CopyMedia	Polvo	9	8	-7	-2	-30
Favourites	Ovelha Negra	5	5	-4	-2	-30
LabelMedia	Polvo	19	18	-15	12	-170
	Rede Neural	14	13	-9	9	-132
	Tsunami	10	9	-6	4	-108
	Enraizado	6	6	-4	5	-92
	Serpente	4	4	-2	5	-82
	Planta T.	4	4	-1	1	-14
SMS	Polvo	14	6	-5	0	-20
	Tsunami	12	6	-5	0	-20
	Enraizado	8	4	-3	0	-8
	Serpente	5	3	-2	0	-12
Sorting	Ovelha Negra	5	5	-4	0	-36
	Serpente	3	3	-2	0	-26

1. Uma operação possui um fragmento de código a ser extraído para um novo adendo e existem outros fragmentos relacionados ao interesse ainda na mesma operação - CDO aumenta em 1 unidade.

2. Uma operação possui um fragmento de código a ser extraído para um novo adendo e **não** existem outros fragmentos relacionados ao interesse na mesma operação - CDO não varia, pois a diminuição de 1 unidade pela operação que deixará de ser relacionada ao interesse é compensada pelo aumento em 1 unidade com a criação do adendo.

3. Alternativamente, se houver adendos já criados no aspecto e o fragmento a ser extraído puder ser introduzido por algum adendo já existente, não há a soma de 1 unidade pela criação de um novo adendo e assim o cômputo geral desta métrica não teria variação no caso 1 ou teria uma redução de 1 unidade para o caso 2.

Além desta questão, na maioria dos casos, é possível notar que os sintomas de um interesse que envolvem um maior número de componentes na refatoração (coluna 4) tendem a ter um melhor resultado no impacto de acordo com as medições. Ou seja, na maioria dos interesses, quanto maior o número de componentes a serem refatorados para um determinado sintoma, menor será o valor das medições de separação de interesses. A figura 7.2 demonstra graficamente esta observação considerando o interesse *Observer* no *Health Watcher*. Cada ponto no gráfico representa uma variação na medição de CDC, CDO e CDLOC sobre o eixo vertical. O eixo horizontal representa o número de componentes a serem refatorados em cada sintoma do interesse. Observado o sintoma Polvo, que envolve o maior número de componentes para refatoração no interesse *Observer*, tem-se que a variação de CDC é -14 quando 15 componentes são refatorados. Os valores correspondentes

Tabela 7.7: Resultados da análise de impacto para refatorações no *Health Watcher*

Análise de Impacto das Refatorações no *Health Watcher*						
Interesse	**Sintoma**	**Participantes do sintoma**	**Componentes a serem refatorados**	**Variação das Medições de Separação de Interesses**		
				CDC	**CDO**	**CDLOC**
Concorrência	Polvo	7	5	-4	0	-26
	Enraizado	4	3	-2	0	-14
Abstract Factory (1)	Tsunami	5	1	0	0	-2
	Serpente	3	1	0	0	-2
	Planta T.	3	0	-	-	-
Abstract Factory (2)	Polvo	5	2	-1	0	-4
	Rede Neural	5	2	-1	0	-4
	Serpente	3	2	-1	0	-2
Adapter (1)	Tsunami	4	1	0	0	-4
	Serpente	3	1	0	0	-4
Adapter (2)	Tsunami	4	1	0	0	-4
Command	Dominador	33	1	0	-1	-12
	Tsunami	33	1	0	-1	-12
	Cópia Carbono	31	30	0	1	0
	Planta T.	32	0	-	-	-
Observer	Polvo	17	17	-16	-15	-54
	Cópia Carbono	10	10	-4	-11	-44
	Planta T.	6	6	-5	-9	-42
	Doença H.	3	3	-2	0	-4
State	Rede Neural	16	4	-3	0	-144
	Enraizado	9	5	-4	0	-146
	Tsunami	5	2	-1	0	-56
	Serpente	3	2	0	0	-54
	Planta T.	3	0	-	-	-

à variação de CDO e CDLOC para este caso também representam uma alta variação (-14 e -50 respectivamente). Já observando o sintoma Doença Hereditária, que envolve o menor número de componentes refatorados neste interesse, tem-se também uma menor variação nas medições de CDC, CDO e CDLOC (-2, 0 e -4 respectivamente).

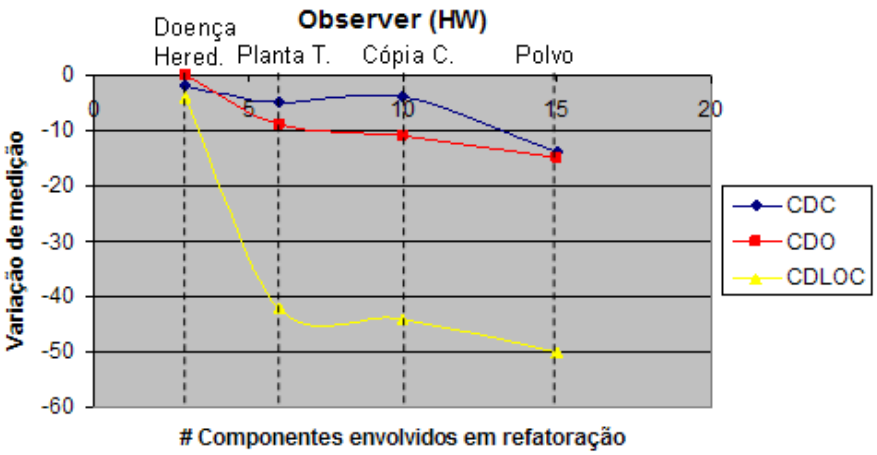


Figura 7.2: Variação de medição e componentes refatorados em cada sintoma do *Observer*

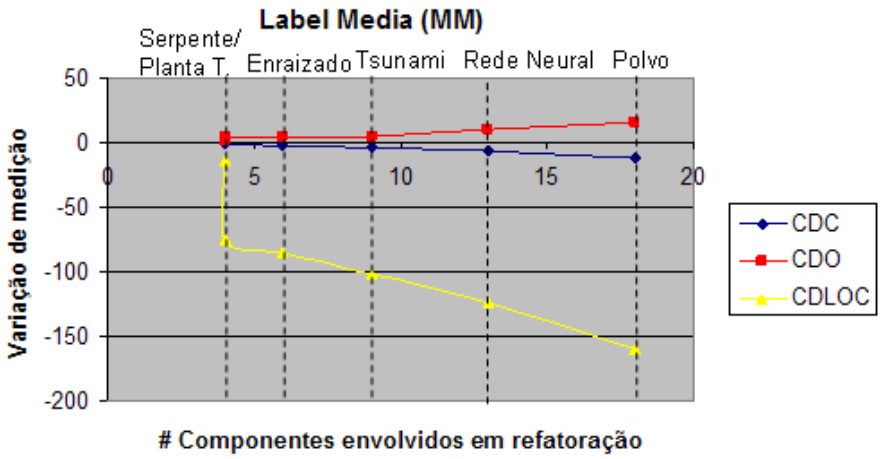


Figura 7.3: Variação de medição e componentes refatorados em cada sintoma do *Label Media*

Entretanto, deve-se ressaltar algumas exceções a essa última observação. Pela tabela 7.6, no interesse *LabelMedia*, apesar de a análise com as métricas CDC e CDLOC melhorar (isto é, os valores das medições diminuem), o mesmo não acontece com a variação do CDO, como demonstra o gráfico da figura 7.3. Neste interesse, a análise de impacto das refatorações para os sintomas com mais componentes envolvidos indica uma variação positiva do CDO (o traçado correspondente a CDO cresce na figura 7.3). Isto significa uma piora no espalhamento do interesse sobre operações, apesar de melhorar o entrelaçamento e a difusão do interesse sobre componentes (o traçado correspondente a CDC e CDLOC decresce). Tal fato pode ser explicado pelo modo em que se encontra a implementação. Existem muitos trechos de código entrelaçados pelos componentes envolvidos neste interesse. A grande quantidade desses fragmentos entrelaçados provocariam também a criação de uma grande quantidade de adendos para introduzí-los uma vez que eles sejam extraídos para aspecto. Consequentemente, a criação de muitos adendos resulta em variação positiva na medição do CDO. Tal questão pode ser reforçada ao observar a medição do CDLOC. A variação chegando a valores abaixo de 100 unidades negativas, na refatoração de alguns sintomas, indica a existência de muitos fragmentos de código entrelaçados que passariam a estar concentrados em aspectos, sendo introduzidos por adendos, e assim diminuindo a medição de CDLOC.

Conforme tabela 7.7, a exceção no estudo do *Health Watcher* é o sintoma Cópia Carbono no interesse *Command*. Neste caso, a análise de impacto não apontou variação nas medições de CDC e CDLOC, além de computar uma unidade positiva para CDO. Tal fato deve-se à maneira com que o interesse está implementado e também pela refatoração considerada na análise de impacto. Nesta situação específica, a análise de impacto considerou a alternativa da refatoração para Cópia Carbono que não utiliza aspectos. Além disso, outros fragmentos dos componentes permanecem dedicados ao interesse, apesar de as partes copiadas terem sido refatoradas. Em contraste com esta situação, observa-se também a análise de impacto para refatoração dos sintomas Tsunami e Dominador do *Command* que, apesar de considerar a refatoração de apenas um componente, possuem melhorias na medição de CDO e CDLOC e neutralidade na medição de CDC.

Estas últimas observações também se repetem na aplicação das refatorações, isto é, tanto na análise de impacto quanto na real aplicação das refatorações, na maioria dos interesses e sintomas é possível observar que a variação nas medições de CDC, CDO e

CDLOC tendem a melhorar na medida em que se tem um maior número de componentes envolvidos. Tal observação é compatível com outro estudo semelhante realizado em (GARCIA et al., 2006). A sessão seguinte demonstra os dados da aplicação das refatorações.

7.6 Aplicação de Refatorações

De acordo com o procedimento de estudo apresentado na seção 7.1 levantou-se dados através de métricas antes e após a aplicação das refatorações a fim de fazer uma comparação com informações quantitativas coletadas nestes dois momentos. Em ambos os estudos (*Health Watcher* e *Mobile Media*), a comparação foi feita sob o foco das métricas de separação de interesses (CDC, CDLOC e CDO), as mesmas consideradas nos algoritmos de análise de impacto.

Aplicou-se, portanto, 39 refatorações correspondendo a 39 sintomas dos 22 interesses sob estudo. É importante destacar, nesse momento, que intencionalmente decidiu-se aplicar refatorações para cada sintoma detectado em um interesse, com exceção de 3 casos no *Health Watcher* - os mesmos mencionados na seção 7.5. Faz parte do objetivo também confrontar os resultados de acordo com as diversas refatorações possíveis para cada interesse conforme os sintomas detectados nele.

As tabelas 7.8 e 7.9 reúnem, respectivamente, para o estudos do *Mobile Media* e *Health Watcher*, os resultados obtidos pelas métricas CDC, CDO e CDLOC ao longo das refatorações aplicadas para cada sintoma dos interesses.

Tabela 7.8: Variação das medições após refatorações do *Mobile Media*

Variação da Separação de Interesses após Refatorações						
Interesse	**Sintoma**	**Participantes do sintoma**	**Componentes refatorados**	**Variação das Medições de Separação de Interesses**		
				CDC	**CDO**	**CDLOC**
CaptureMedia	Polvo	8	5	-4	0	-14
	Tsunami	7	4	-3	0	-10
	Enraizado	5	3	-2	0	-6
	Serpente	4	3	-2	0	-6
CopyMedia	Polvo	9	8	-7	-2	-30
Favourites	Ovelha Negra	5	5	-2	2	-18
LabelMedia	Polvo	19	18	-12	15	-160
	Rede Neural	14	13	-7	10	-124
	Tsunami	10	9	-4	5	-102
	Enraizado	6	6	-3	5	-86
	Serpente	4	4	-1	5	-76
	Planta T.	4	4	0	1	-14
SMS	Polvo	14	6	-5	0	-20
	Tsunami	12	6	-5	0	-20
	Enraizado	8	4	-3	0	-8
	Serpente	5	3	-2	0	-12
Sorting	Ovelha Negra	5	5	-2	3	-28
	Serpente	3	3	0	3	-14

O mesmo comportamento das medições pode ser observado em relação à análise de impacto (tabelas 7.6 e 7.7):

- a peculiaridade na medição de CDO;
- as exceções - o interesse *LabelMedia* no *Mobile Media* e o sintoma Cópia Carbono no interesse *Command* do *Health Watcher*;
- a tendência geral de que quanto mais componentes refatorados melhores os resultados nas medições de separação de interesses.

Tabela 7.9: Variação das medições após refatorações do *Health Watcher*

Variação da Separação de Interesses após Refatorações						
Interesse	**Sintoma**	**Participantes do sintoma**	**Componentes refatorados**	**Variação das Medições de Separação de Interesses**		
				CDC	**CDO**	**CDLOC**
Concorrência	Polvo	7	5	-4	0	-26
	Enraizado	4	3	-2	0	-14
Abstract Factory (1)	Tsunami	5	1	0	0	-2
	Serpente	3	1	0	0	-2
	Planta T.	3	0	-	-	-
Abstract Factory (2)	Polvo	5	2	-1	0	-4
	Rede Neural	5	2	-1	0	-4
	Serpente	3	2	-1	0	-2
Adapter (1)	Tsunami	4	1	0	0	-4
	Serpente	3	1	0	0	-4
Adapter (2)	Tsunami	4	1	0	0	-4
Command	Dominador	33	1	0	-1	-12
	Tsunami	33	1	0	-1	-12
	Cópia Carbono	31	30	0	1	0
	Planta T.	32	0	-	-	-
Observer	Polvo	17	15	-14	-15	-50
	Cópia Carbono	10	10	-4	-11	-44
	Planta T.	6	6	-5	-9	-42
	Doença H.	3	3	-2	0	-4
State	Rede Neural	16	4	-3	0	-144
	Enraizado	9	5	-4	0	-146
	Tsunami	5	2	-1	0	-56
	Serpente	3	2	0	0	-54
	Planta T.	3	0	-	-	-

Comparando (i) os dados da análise de impacto e (ii) os dados da aplicação das refatorações é possível observar que houve diferenças nas medições para alguns interesses. Isto pode ser notado ao confrontar: os dados das tabelas 7.6 e 7.8, no caso do *Mobile Media*; e os dados das tabelas 7.7 e 7.9, no caso *Health Watcher*.

Pelo estudo do *Mobile Media* as medições para os interesses *Favourites*, *Sorting* e *LabelMedia* obtidas pela análise de impacto e pela aplicação das refatorações foram diferentes. No caso do *Health Watcher* houve diferença entre a análise de impacto e a refatoração em apenas um sintoma do interesse *Observer*. A tabela 7.10 destaca estas situações.

Em todos os casos da tabela 7.10 a análise de impacto trouxe melhores resultados, isto é, apontou melhorias que na prática não foram exatamente iguais. Entretanto, pode-se verificar que não houve discrepância entre as variações. Isto se justifica pela existência

Tabela 7.10: Diferenças entre análise de impacto e refatoração no *Mobile Media* e *Health Watcher*

Sistema	Interesse	Sintoma	Variação das Medições					
			Análise de Impacto			Refatoração		
			CDC	CDO	CDLOC	CDC	CDO	CDLOC
Health Watcher	*Observer*	Polvo	-16	-15	-54	-14	-15	-50
Mobile Media	*Favourites*	Ovelha Negra	-4	-2	-30	-2	2	-18
	LabelMedia	Polvo	-15	12	-170	-12	15	-160
		Rede Neural	-9	9	-132	-7	10	-124
		Tsunami	-6	4	-108	-4	5	-102
		Enraizado	-4	5	-92	-3	5	-86
		Serpente	-2	5	-82	-1	5	-76
		Planta T.	-1	1	-14	0	1	-14
	Sorting	Ovelha Negra	-4	0	-36	-2	3	-28
		Serpente	-2	0	-26	0	3	-14

de alguns casos específicos de implementação que não foram cobertos pelas refatorações na prática. Assim, a análise de impacto projeta um valor que não foi exatamente o mesmo quando se aplicou a refatoração.

Duas situações provocaram esta diferença: (i) trechos relacionados ao interesse em que não se aplicava algum passo de refatoração - isto é, a refatoração não cobriu alguma situação específica encontrada no código; (ii) trechos relacionados ao interesse onde decidiu-se não refatorar, por decisões particulares do desenvolvedor - isto pode envolver situações em que a reestruturação provocaria outras necessidades de refatoração, ou afetaria negativamente na modularização de outros componentes, por exemplo.

Já era esperado que a análise de impacto não oferecesse resultados exatamente iguais ao da aplicação real da refatoração, pois não é possível garantir a cobertura das diversas possibilidades de implementação que se pode encontrar nos interesses sintomáticos. No entanto, como este fato não ocorreu para todos os casos, e ainda assim a diferença não foi discrepante, não tomou-se esta observação como um problema. Pode-se notar que, mesmo com a diferença de valores, o comportamento dos resultados foi o mesmo.

7.7 Limitações de Estudo

Esta seção discute algumas limitações referentes à realização dos estudos de caso durante o processo de avaliação da proposta. Primeiramente, as conclusões obtidas neste capítulo restringem-se aos sistemas alvos de avaliação (*Mobile Media* e *Health Watcher*) e aos seus respectivos interesses transversais levados em consideração. Portanto, os pontos fortes e limitações a partir dos resultados da aplicação do método podem não ser diretamente generalizados para outros contextos. Não é possível garantir que a mesma análise feita a partir dos resultados obtidos possa se repetir em diferentes situações.

Entretanto, durante a realização dos dois estudos de caso buscou-se maximizar a consistência no procedimento de avaliação em prol de sua confiabilidade. Com isso, permitiu-se construir um trabalho de avaliação útil sobre o quanto aplicável pode ser o método de refatoração e o quanto podem ser factíveis futuras investigações nesse campo de estudo. Além disso, o trabalho de avaliação também permitiu verificar a aplicabilidade dos sintomas sensíveis ao interesse, bem como suas heurísticas de detecção.

Uma outra questão que pode interferir nas conclusões dos estudos é o fato de que o processo de coleta de dados através de métricas impacta diretamente na detecção de alguns sintomas como, Ovelha Negra, Polvo e Dominador, pois suas heurísticas de detecção dependem diretamenta das medições. Como nem todas as métricas podem ser adotadas automaticamente através de ferramentas, algumas delas podem ser usadas somente de maneira manual. Portanto, a realização da coleta de dados feita manualmente é suscetível a erros, demandando um tempo alto para a conclusão do trabalho e sucessivas revisões por pessoas diferentes.

O trabalho de avaliação mostrou que o método de refatoração proposto é aplicável e promissor na eliminação ou minização de estruturas transversais identificadas pelos sintomas em desenho de software. O número de sistemas alvo considerado na avaliação não é estatisticamente relevante. Entretanto, considera-se que os dois alvos (*Mobile Media* e *Health Watcher*) possuem características e interesses transversais heterogêneos, além de já terem sido alvos de diversos outros trabalhos, como em (FILHO et al., 2006) (GREENWOOD et al., 2007) (FIGUEIREDO et al., 2008) (FIGUEIREDO; CACHO, 2008) (SANT'ANNA, 2008).

7.8 Considerações Finais do Capítulo

Este capítulo apresentou o trabalho de avaliação realizado através de dois estudos de caso, envolvendo dois sistemas-alvo (*Mobile Media* e *Health Watcher*), com o intuito de verificar a aplicabilidade do método de refatoração que é o objetivo principal da presente proposta. Além disso, descreveu-se os sistemas envolvidos nos estudos de caso e principalmente os resultados obtidos em termos quantitativos, que permitem realizar também análises qualitativas. Além de apresentar os resultados, discutiu-se também os valores obtidos tanto pela análise de impacto quanto pela aplicação das refatorações. Esta etapa do trabalho é importante para o fortalecimento da proposta. Apesar de o estudo possuir limitações foi possível verificar a aplicabilidade do método proposto a fim de melhorar a modularização de interesses transversais, como apresentado na seção anterior (7.7). A seguir, o último capítulo realiza as últimas considerações e discute possíveis direções para trabalhos futuros.

8 CONCLUSÃO

A modularização de sistemas é algo que permeia a Engenharia de Software há décadas. Durante o processo de construção e evolução de software, componentes e interesses mal modularizados afetam negativamente os atributos de qualidade do mesmo. De maneira geral, o desenvolvimento de software orientado a aspectos (DSOA), emergindo da própria programação orientada a aspectos (POA), tem sido estudado e aplicado como alternativa para modularização de interesses transversais. Uma das direções de trabalho e pesquisa consiste em técnicas de refatoração utilizadas para aplicar transformações em sistemas de software usando construções da POA (capítulo 2).

Entretanto, existem duas vertentes no que tange os trabalhos de refatorações que utilizam a orientação a aspectos. Primeiramente, existem as refatorações que lidam com a POA, porém não se encontram focadas nem direcionadas à modularização de interesses. Estas simplesmente reestruturam elementos ou um conjunto de elementos da programação OO para a programação OA, ou até mesmo realizam refatorações internas ao código OA. Em uma outra linha, existem as propostas de refatoração que buscam pela modularização de interesses, isto é, estão motivadas primordialmente pela reestruturação de um interesse como um todo que, por sua vez, envolve a busca por modificações nos componentes participantes do mesmo.

O presente trabalho propõe um método de refatoração para modularização de interesses transversais procurando não carregar as limitações de outras propostas como discutidas na seção 2.3 do capítulo 2. Este método integra uma abordagem de classificação e detecção de interesses (capítulo 4) independente de linguagem ou tecnologia a um conjunto de refatorações proposto no capítulo 5. Além disso, faz parte também do método, uma estratégia para realização de análise de impacto como um suporte adicional a possíveis aplicações de cada refatoração (capítulo 6).

Dessa maneira, as contribuições deste trabalho dividiram-se nos seguintes itens:

- apresentar as propostas de refatorações que envolvem a orientação a aspectos existentes na literatura e discutir seus objetivos e limitações;
- apresentar um estudo sobre métricas orientadas a interesses e estabeler uma análise crítica sobre as mesmas;
- criar um método de refatoração que inclui um catálogo de refatorações para modularização de interesses transversais integrado à técnica de identificação e classificação de interesses baseada em sintomas;
- propor algoritmos para apoiar a análise de impacto das refatorações, sendo mais um recurso que compõe o método;

- e, por fim, apresentar uma avaliação quantitativa de tal abordagem através de dois estudos de caso que permitiram analisar mais concretamente as potencialidades e limitações do método.

Finalmente, conclui-se que, apesar da existência de trabalhos importantes sobre refatorações OA, ainda faltam esforços de refatoração direcionados à modularização de interesses transversais, que é onde o DSOA concentra sua maior vantagem. É nesse sentido que se motivou o desenvolvimento da presente proposta. O método desenvolvido foi aplicado em dois estudos de caso que possibilitaram verificar a viabilidade do mesmo. Conseguiu-se, de acordo com os resultados apresentados e discutidos no capítulo 7, verificar quantitativamente que as refatorações aplicadas em dois sistemas alvos melhoraram o atributo de qualidade "Separação de Interesses" conforme medições através de três métricas conhecidas. Os algoritmos de análise de impacto conseguiram apontar resultados, na maioria das vezes, iguais aos obtidos pela própria realização das refatorações. Em poucos casos, devidamente justificados, a análise de impacto não apontou números iguais aos da aplicação da refatoração correspondente. Entretanto, de maneira geral, ainda nesses casos em que os números não foram iguais, os algoritmos de análise de impacto trouxeram resultados com comportamento semelhante aos das respectivas refatorações.

Foi possível verificar também a vantagem de se possuir refatorações de granularidade alta disponíveis para serem reusadas sempre que um sintoma possa ser identificado em um interesse. De modo contrário, não seria possível garantir que profissionais sempre usariam a mesma configuração de refatorações de granularidade baixa (quais e em que sequência) para modularizar interesses transversais organizados de maneira semelhante.

8.1 Trabalhos Futuros

As contribuições alcançadas no presente contexto representam um primeiro esforço em direção à possibilidade de aplicar, em larga escala, refatorações voltadas especificamente para a melhoraria da modularização de interesses transversais em sistemas de software. Portanto, além das metas atingidas, é possível traçar alguns trabalhos futuros.

Estudos Experimentais de Avaliação

O desenvolvimento de novos estudos experimentais sobre o método proposto são desejáveis a fim de reunir um conjunto maior de resultados e informações interessantes à melhoria e ao refinamento do método. Estes estudos podem envolver novos sistemas alvos com interesses transversais heterogêneos e novas metas de avaliação e comparação como, por exemplo: verificar quais sintomas representam melhores resultados na maioria dos casos de refatoração; correlação entre interesses refatorados e atributos de qualidade dos mesmos em versões posteriores do software; correlação entre instabilidade de desenho do software antes e depois das refatorações; etc.

Prova Formal de Preservação do Comportamento

A maioria dos trabalhos de refatoracão concentra-se na indicação de testes como verificação da preservação do comportamento. Portanto, é interessante estudar métodos formais para a prova de preservação do comportamento das refatorações de granularidade alta como as propostas aqui. Uma outra questão a responder seria: provando a preservação do comportamento isoladamente das refatorações de granularidade fina e utilizando-as para compor as refatorações de granularidade alta torna-se possível garantir também a

preservação do comportamento das últimas?

Extensão do Método

Propõe-se também como trabalhos futuros o refinamento da classificação definida por Figueiredo *et al.* (2009) para atender a outras "formas" ou "tipos" de interesses transversais que eventualmente possam ser levantados. Consequentemente, isto provocaria ajustes na definição do catálogo de refatorações. Uma outra linha de melhoria do método seria a possibilidade de extensão da abordagem de análise de impacto em duas direções: primeiramente, os algoritmos podem ser estendidos para analisar o impacto sobre novas métricas voltadas para outros atributos de qualidade do software; e, além disso, novos algoritmos podem também ser criados na ocorrência de novas refatorações que possam cobrir novas formas de interesses transversais que por ventura possam ser catalogadas.

Suporte Ferramental

É interessante para a melhoria da aplicabilidade do método e para possibilitar uma melhor avaliação do mesmo a construção de alguns suportes ferramentais. Isto diminui o custo de aplicação do método, na medida que aumenta a produtividade no seu uso, além de diminuir tarefas manuais ou semi-automáticas que são suscetíveis a erros. Atualmente, está sendo desenvolvida na Universidade de Lancaster uma ferramenta para detecção automática de sintomas em interesses transversais, no contexto do doutorado do autor principal da abordagem de classificação de interesses por sintomas (FIGUEIREDO et al., 2009). Além disso, deseja-se automatizar um conjunto de refatorações OO e OA ainda não disponíveis em ferramentas, pois elas são os passos de refatoração que compõem as refatorações maiores para modularização de interesses. Um outro ponto seria a implementação dos algoritmos de análise de impacto integrada a uma ferramenta de desenvolvimento para auxiliar desenvolvedores e projetistas no processo de decisão entre aplicar ou não as refatorações.

REFERÊNCIAS

BANIASSAD, E. et al. Discovering Early Aspects. **IEEE Software**, Los Alamitos, CA, USA, v.23, n.1, p.61–70, 2006.

BINKLEY, D. et al. Tool-Supported Refactoring of Existing Object-Oriented Code into Aspects. **IEEE Transactions on Software Engineering**, Los Alamitos, CA, USA, v.32, n.9, p.698–717, 2006.

BOIS, B. D. **A Study of Quality Improvements By Refactoring**. 2006. Tese (Doutorado em Ciência da Computação) — University of Antwerp, Antwerp - Belgium.

BOIS, B. D.; MENS, T. Describing the Impact of Refactorings on Internal Program Quality. In: INTERNATIONAL WORKSHOP ON EVOLUTION OF LARGE-SCALE INDUSTRIAL SOFTWARE APPLICATIONS - ELISA'03, 2003, Amsterdan, Netherlands. **Anais...** [S.l.: s.n.], 2003. p.37–48.

BOOCH, G. **Object-oriented analysis and design with applications (2nd ed.)**. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1994.

BRIAND, L. C.; DALY, J. W.; WüST, J. A Unified Framework for Cohesion Measurement in Object-OrientedSystems. **Empirical Software Engineering**, Hingham, MA, USA, v.3, n.1, p.65–117, 1998.

BRIAND, L. C.; DALY, J. W.; WüST, J. K. A Unified Framework for Coupling Measurement in Object-Oriented Systems. **IEEE Transactions on Software Engineering**, Los Alamitos, CA, USA, v.25, n.1, p.91–121, 1999.

BRUNTINK, M. et al. An Evaluation of Clone Detection Techniques for Identifying Crosscutting Concerns. In: ICSM '04: 20TH IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2004, Washington, DC, USA. **Anais...** IEEE Computer Society, 2004. p.200–209.

CACHO, N. et al. Composing design patterns: a scalability study of aspect-oriented programming. In: AOSD '06: 5TH INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, 2006, New York, NY, USA. **Anais...** ACM, 2006. p.109–121.

CECCATO, M.; TONELLA, P. Measuring the Effects of Software Aspectization. In: WORKSHOP ON ASPECT REVERSE ENGINEERING, 1., 2004. **Anais...** [S.l.: s.n.], 2004.

CHIDAMBER, S.; KEMERER, C. A Metrics Suite for Object Oriented Design. **IEEE Transactions on Software Engineering**, Los Alamitos, CA, USA, v.20, n.6, p.476–493, 1994.

CLEMENTS, P.; NORTHROP, L. **Software Product Lines**: practices and patterns. Boston, MA, USA: Addison-Wesley Professional, 2002.

COLE, L.; BORBA, P. Deriving refactorings for aspectJ. In: OOPSLA '04: COMPANION TO THE 19TH ANNUAL ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES, AND APPLICATIONS, 2004, New York, NY, USA. **Anais...** ACM, 2004. p.202–203.

DIJKSTRA, E. W. **A Discipline of Programming**. [S.l.]: Prentice Hall, Inc., 1976.

DUCASSE, S.; GIRBA, T.; KUHN, A. Distribution Map. In: ICSM '06: 22ND IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2006. **Anais...** [S.l.: s.n.], 2006. p.203–212.

EADDY, M.; AHO, A.; MURPHY, G. C. Identifying, Assigning, and Quantifying Crosscutting Concerns. In: ACOM '07: FIRST INTERNATIONAL WORKSHOP ON ASSESSMENT OF CONTEMPORARY MODULARIZATION TECHNIQUES, 2007, Washington, DC, USA. **Anais...** IEEE Computer Society, 2007. p.2.

EADDY, M. et al. Do Crosscutting Concerns Cause Defects? **IEEE Transactions on Software Engineering**, Piscataway, NJ, USA, v.34, n.4, p.497–515, 2008.

ELRAD, T.; FILMAN, R. E.; BADER, A. Aspect-oriented Programming: introduction. **Communications of ACM**, New York, NY, USA, v.44, n.10, p.29–32, 2001.

FIGUEIREDO, E.; CACHO, N. Evolving software product lines with aspects: an empirical study on design stability. In: ICSE '08: 13TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2008, New York, NY, USA. **Anais...** ACM, 2008. p.261–270.

FIGUEIREDO, E. et al. On the Maintainability of Aspect-Oriented Software: a concern-oriented measurement framework. In: CSMR'08: 12TH EUROPEAN CONFERENCE OF SOFTWARE MAINTENANCE AND REEGINEERING, 2008. **Anais...** [S.l.: s.n.], 2008. p.183–192.

FIGUEIREDO, E. et al. Crosscutting Patterns and Design Stability: an exploratory analysis. In: ICPC '09: 17TH INTERNATIONAL CONFERENCE ON PROGRAM COMPREHENSION, 2009. **Anais...** A publicar, 2009.

FILHO, F. C. et al. Exceptions and aspects: the devil is in the details. In: SIGSOFT '06/FSE-14: 14TH ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, 2006, New York, NY, USA. **Anais...** ACM, 2006. p.152–162.

FOWLER, M.; BECK, K.; BRANT, J.; OPDYKE, W.; ROBERTS, D. **Refactoring**: improving the design of existing code. [S.l.]: Addison-Wesley Professional, 1999.

GAMMA, E. et al. **Design patterns**: elements of reusable object-oriented software. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

GARCIA, A. F. et al. Modularizing Design Patterns with Aspects: a quantitative study. **T. Aspect-Oriented Software Development I**, [S.l.], v.3880, p.36–74, 2006.

GARCIA, V. et al. Manipulating crosscutting concerns. In: LATIN AMERICAN CONFERENCE ON PATTERNS LANGUAGES OF PROGRAMMING (SUGARLOAF-PLOP'04), 4., 2004. **Anais...** [S.l.: s.n.], 2004.

GHEYI, R.; MASSONI, T. Formal refactorings for object models. In: OOPSLA '05: COMPANION TO THE 20TH ANNUAL ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS, 2005, New York, NY, USA. **Anais...** ACM, 2005. p.208–209.

GREENWOOD, P. et al. On the Impact of Aspectual Decompositions on Design Stability: an empirical study. In: ECOOP, 2007. **Anais...** Springer, 2007. p.176–200. (Lecture Notes in Computer Science, v.4609).

HANENBERG, S.; OBERSCHULTE, C.; UNLAND, R. Refactoring of Aspect-Oriented Software. In: NET.OBJECTDAYS CONFERENCE (NODE'03), 2003. **Anais...** [S.l.: s.n.], 2003.

HANNEMANN, J. Aspect-Oriented Refactoring: classification and challenges. In: INTERNATIONAL WORKSHOP ON LINKING ASPECT TECHNOLOGY AND EVOLUTION AT AOSD'06, 5., 2006. **Anais...** [S.l.: s.n.], 2006.

HANNEMANN, J.; KICZALES, G. Design pattern implementation in Java and aspectJ. **SIGPLAN Not.**, New York, NY, USA, v.37, n.11, p.161–173, 2002.

HANNEMANN, J.; MURPHY, G. C.; KICZALES, G. Role-based refactoring of crosscutting concerns. In: AOSD '05: 4TH INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, 2005, New York, NY, USA. **Anais...** ACM, 2005. p.135–146.

IEEE. **Standard Glossary of Software Engineering Terminology**. [S.l.: s.n.], 1990. Std. 610.12-1990.

IWAMOTO, M.; ZHAO, J. Refactoring Aspect-Oriented Programs. In: INTERNATIONAL WORKSHOP ON ASPECT-ORIENTED MODELING AT UML'2003, 4., 2003. **Anais...** [S.l.: s.n.], 2003.

KENDALL, J.; KENDALL, K. Metaphors and Methodologies: living beyond the systems machine. **MIS Quaterly**, University of Minessota, v.17, n.2, p.149–171, 2003.

KERIEVSKY, J. **Refactoring to Patterns**. [S.l.]: Pearson Higher Education, 2004.

KOZEN, D. **The Design and Analysis of Algorithms**. New York, USA: Springer-Verlag, 1991.

LADDAD, R. **AspectJ in Action**: practical aspect-oriented programming. Greenwich, CT, USA: Manning Publications Co., 2003.

LADDAD, R. **Aspect Oriented Refactoring**. [S.l.]: Addison-Wesley Professional, 2006.

LANZA, M.; MARINESCU, R.; DUCASSE, S. **Object-Oriented Metrics in Practice**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

LOPEZ-HERREJON, R. E.; APEL, S. Measuring and Characterizing Crosscutting in Aspect-Based Programs: basic metrics and case studies. In: FASE'07: INTERNATIONAL CONFERENCE ON FUNDAMENTAL APPROACHES TO SOFTWARE ENGINEERING, 2007. **Anais...** Springer, 2007. p.423–437. (Lecture Notes in Computer Science, v.4422).

MANZANO, J.; OLIVEIRA, J. **Algoritmos - Lógica para Desenvolvimento de Programação de Computadores**. 21.ed. São Paulo: Ed. Érica, 2008.

MARIN, M.; MOONEN, L.; DEURSEN, A. van. An Approach to Aspect Refactoring based on Crosscutting Concern Types. **SIGSOFT Software Engineering Notes**, New York, NY, USA, v.30, n.4, p.1–5, 2005.

MARIN, M.; MOONEN, L.; DEURSEN, A. van. A Classification of Crosscutting Concerns. In: ICSM '05: 21ST IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2005, Washington, DC, USA. **Anais...** IEEE Computer Society, 2005. p.673–676.

MARINESCU, R. Detection Strategies: metrics-based rules for detecting design flaws. In: ICSM '04: 20TH IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2004, Washington, DC, USA. **Anais...** IEEE Computer Society, 2004. p.350–359.

MASSONI, T.; GHEYI, R.; BORBA, P. A model-driven approach to formal refactoring. In: OOPSLA '05: COMPANION TO THE 20TH ANNUAL ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS, 2005, New York, NY, USA. **Anais...** ACM, 2005. p.124–125.

MENS, T.; TOURWÉ, T. A Survey of Software Refactoring. **IEEE Transactions on Software Engineering**, Piscataway, NJ, USA, v.30, n.2, p.126–139, 2004.

MEYER, B. **Object-oriented Software Construction**. 2.ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.

MONTEIRO, M. P. **Refactorings to Evolve Object-Oriented Systems with Aspect-Oriented Concepts**. 2005. Tese (Doutorado em Ciência da Computação) — Departamento de Informática, Universidade do Minho, Portugal.

MONTEIRO, M. P.; FERNANDES, J. M. L. Towards a catalogue of refactorings and code smells for aspectj. **Transactions on Aspect Oriented Software Development (TAOSD) - Lecture Notes in Computer Science**, [S.l.], n.3880, p.214–258, 2006.

MURPHY-HILL, E.; PARNIN, C.; BLACK, A. P. How We Refactor, and How We Know It. In: ICSE '09: 14TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2009, New York, NY, USA. **Anais...** ACM, 2009.

OPDYKE, W. F. **Refactoring Object-Oriented Frameworks**. 1992. Tese (Doutorado em Ciência da Computação) — , Urbana-Champaign, IL, USA.

PAGLIARI, L. **Avaliação Quantitativa de Refatorações Orientadas a Aspectos**. 2007. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Porto Alegre - Brasil.

PAGLIARI, L.; NUNES, D. Um Processo Para Avaliação Quantitativa de Refatorações de Software. In: VI JORNADAS PERUANAS DE COMPUTACIóN, 2007, Peru. **Anais...** SPC, 2007.

PARNAS, D. L. On the criteria to be used in decomposing systems into modules. **Communications of ACM**, New York, NY, USA, v.15, n.12, p.1053–1058, 1972.

PIVETA, E. K. **Improving the Search for Refactoring Opportunities on Object-Oriented and Aspect-Oriented Software**. 2009. Tese (Doutorado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Porto Alegre - Brasil.

PIVETA, E. K. et al. Detecting Bad Smells in AspectJ. **Journal of Universal Computer Science**, [S.l.], v.12, n.7, p.811–827, 2006.

PIVETA, E. K. et al. Avoiding Bad Smells in Aspect-Oriented Software. In: SEKE, 2007. **Anais...** Knowledge Systems Institute Graduate School, 2007. p.81–.

POHL, K.; BöCKLE, G.; LINDEN, F. van der. **Software Product Line Engineering**: foundations, principles, and techniques. Berlin Heidelberg New York: Springer, 2005.

ROBERTS, D. B. **Practical Analysis for Refactoring**. 1999. Tese (Doutorado em Ciência da Computação) — University of Illnois at Urbana-Champaign, USA.

ROBILLARD, M. P.; MURPHY, G. C. Representing concerns in source code. **ACM Transactions on Software Engineering Methodologies**, New York, NY, USA, v.16, n.1, p.3, 2007.

RURA, S. **Refactoring Aspect-Oriented Software**. [S.l.]: Computer Science at Williams College, Massachuusetts, 2003. Undergraduate Thesis.

SANT'ANNA, C. **On the Modularity of Aspect-Oriented Design**: a concern-driven measurement approach. 2008. Tese (Doutorado em Ciência da Computação) — Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro - Brazil.

SANT'ANNA, C. et al. On the Reuse and Maintenance of Aspect-Oriented Software: an assessment framework. In: XVII BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 2003. **Anais...** [S.l.: s.n.], 2003.

SANT'ANNA, C. et al. On the Modularity of Software Architectures: a concern-driven measurement framework. In: ECSA, 2007. **Anais...** Springer, 2007. p.207–224. (Lecture Notes in Computer Science, v.4758).

SILVA, B. et al. Refactoring of Crosscutting Concerns with Metaphor-Based Heuristics. **Eletronic Notes on Theoretical Computer Science**, [S.l.], v.233, p.105–125, 2009.

SOARES, S.; LAUREANO, E.; BORBA, P. Implementing distribution and persistence aspects with aspectJ. In: OOPSLA '02: 17TH ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS, 2002, New York, NY, USA. **Anais...** ACM, 2002. p.174–190.

SULTANA, N.; THOMPSON, S. Mechanical verification of refactorings. In: PEPM '08: ACM SIGPLAN SYMPOSIUM ON PARTIAL EVALUATION AND SEMANTICS-BASED PROGRAM MANIPULATION, 2008, New York, NY, USA. **Anais...** ACM, 2008. p.51–60.

SUN. **The Java ME Platform**. 2008.

TARR, P. et al. N degrees of separation: multi-dimensional separation of concerns. In: ICSE '99: 21ST INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1999, Los Alamitos, CA, USA. **Anais...** IEEE Computer Society Press, 1999. p.107–119.

WONG, W. E.; GOKHALE, S. S.; HORGAN, J. R. Quantifying the closeness between program components and features. **Journal of System and Software**, New York, NY, USA, v.54, n.2, p.87–98, 2000.

YOUNG, T. **Using AspectJ to Build a Software Product Line for Mobile Devices**. 2005. Dissertação (Mestrado em Ciência da Computação) — University of British Columbia, British Columbia - Canada.

YOUNG, T.; MURPHY, G. Using AspectJ to Build a Product Line for Mobile Devices. In: AOSD '05: 4TH INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT (DEMO SESSION), 2005. **Anais...** [S.l.: s.n.], 2005.