UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"

Pós-Graduação em Ciência da Computação

Ingrid Marçal

Uma abordagem para o apoio à identificação de interesses transversais: diretrizes para a escolha de técnicas de identificação de interesses transversais

UNESP
2017

Ingrid Marçal

Uma abordagem para o apoio à identificação de interesses transversais: diretrizes para a escolha de técnicas de identificação de interesses transversais

Orientador: Prof. Dr. Rogério Eduardo Garcia

Dissertação apresentada ao Instituto de Biociências, Letras e Ciências Exatas (IBILCE – UNESP – São José do Rio Preto) como parte dos requisitos para a obtenção do título de mestre em Ciência da Computação.

UNESP
2017

Marçal, Ingrid.
Uma abordagem para o apoio à identificação de interesses transversais: diretrizes para a escolha de técnicas de identificação de interesses transversais / Ingrid Marçal . -- São José do Rio Preto, 2017
77 f. : il., tabs.

Orientador: Rogério Eduardo Garcia
Dissertação (mestrado) – Universidade Estadual Paulista “Júlio de Mesquita Filho”, Instituto de Biociências, Letras e Ciências Exatas

1. Ciência da Computação. 2. Engenharia de Software. 3. Análise de sistemas. 4. Modularização. 5. Transversalidade. I. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. II. Título.

CDU – 518.721

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Ingrid Marçal

Uma abordagem para o apoio à identificação de interesses transversais: diretrizes para a escolha de técnicas de identificação de interesses transversais

Dissertação apresentada ao Instituto de Biociências, Letras e Ciências Exatas (IBILCE – UNESP – São José do Rio Preto) como parte dos requisitos para a obtenção do título de mestre em Ciência da Computação.

Comissão Examinadora

Prof. Dr. Rogério Eduardo Garcia
UNESP – Presidente Prudente
Orientador

Prof. Dr. Reginaldo Ré
UTFPR – Campo Mourão

Prof. Dr. Ives Pola
UNESP – Presidente Prudente

Presidente Prudente, 26 de Junho de 2017

Agradecimentos

Faço minhas as palavras de *Brendan Pietsch* e culpo a todos vocês. Realizar todo o trabalho e escrever esta dissertação foi um exercício de sofrimento. Permito talvez a um leitor casual eximir-se da culpa de tal sofrimento, mas àqueles que tiveram papel importante no prolongamento da minha agonia com o apoio e encorajamento indispensáveis... bem, você sabe quem você é, e você me deve uma.

Depois de tanto tempo trabalhando sobre o tema desta dissertação, não me é possível escrever uma lista completa de todos aqueles que de algum modo me ajudaram. São muitos e alguns eu nem mesmo sei o nome, por exemplo aquele cara que postou um ”Não desista, Deus está contigo!” em letras garrafais sob um fundo marítimo. Por isso me perdoem os não citados, serão muitos.

Por falar em Deus, obrigada meu Pai e meu amigo que nunca me permitiu permanecer no chão por muito tempo. Sem Ti nenhuma dessas letras teriam sentido.

Prossigo por ordem cronológica de culpa.

Obrigada mãe, pai e irmã, também conhecidos por Fátima, Lourival, e Fernanda. Vocês são minha base e meu modelo para tudo o que faço. E tudo o que faço é para tentar retribuir ao menos o mínimo do amor e apoio que sempre tive.

Obrigada a todos os professores que tive e que contribuíram para a descoberta de tudo o que sei hoje. Em especial ao Ronaldo Celso Messias Corrêa, por me apoiar desde o primeiro momento na universidade.

Obrigada Rogério Eduardo Garcia pelos ensinamentos tanto sobre nossa profissão quanto sobre a vida. Você se tornou minha referência de ética e docência, tenho imenso orgulho em ser sua orientanda. Obrigada por tudo que fez por mim durante os últimos vários anos de orientação. *Tá pronto!*, ou não.

Obrigada Jaqueline Batista pela companhia durante a maior parte da jornada. Sozinha teria sido muito mais difícil enfrentar as diversas viagens e o microondas com microorganismos vivos e verdes que usávamos no hotel.

Obrigada a todos os meus alunos que ao final de cada aula contribuíram para que eu tivesse certeza de que escolhi uma profissão que vou amar por muito tempo.

I never saw a wild thing sorry for itself. A small bird will drop frozen dead from a bough without ever having felt sorry for itself.

D. H. Lawrence

Begin at the beginning, the King said gravely, “and go on till you come to the end: then stop.”

Lewis Carroll, *Alice in Wonderland*

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Promover a evolução de software sem provocar qualquer degradação em sua arquitetura e projeto é um desafio para a Engenharia de Software. A divisão de código em módulos é um dos mecanismos utilizados por metodologias convencionais para tentar minimizar o grau de acoplamento entre elementos de software e facilitar a evolução. A modularização dos interesses que compõem o sistema melhora a compreensão de código e reduz a complexidade das atividades de manutenção. Apesar dos esforços para prover a modularização de interesses em sistemas de software, alguns não são facilmente encapsulados e têm sua implementação espalhada por diversos módulos. Isso gera dependências lógicas entre um grande número de elementos de software. Tais interesses são chamados de interesses transversais. Técnicas para a identificação de interesses transversais apresentam baixa precisão nos resultados obtidos, principalmente devido a não conformidade dos objetivos de identificação às características da técnica escolhida. Este trabalho propõe um conjunto de diretrizes para apoiar a escolha de técnicas de identificação de interesses transversais. O conjunto de diretrizes colabora para a adequação das técnicas ao seu propósito e, por consequência, para a obtenção de resultados mais significantes.

Palavras-chave: Identificação de interesses transversais, modularização, técnicas de identificação

Abstract

Promoting software evolution without causing any degradation in its architecture and design is a challenge for software engineering. The implementation of code into modules is one of the mechanisms used by conventional programming methodologies to try to minimize the degree of coupling between elements of software and facilitate a development and evolution. The modularization of concerns that make up the system improves the source code comprehension and reduces the complexity of maintenance activities. Despite efforts to provide the modularization of concerns in software systems, some are not easily encapsulated and have their implementation spread across several modules. This generates logical dependencies between a large number of software elements. Such concerns are called crosscutting concerns. Techniques for the identification of crosscutting concerns show low precision in the obtained results, mainly due to the nonconformity of the identification objectives to the characteristics of the chosen technique. This work proposes a set of guidelines to support the choice of techniques for identifying crosscutting concerns. The set of guidelines helps to adequate the techniques to their purpose and, consequently, to obtain meaningful results.

Keywords: crosscutting concern identification, modularization, identification techniques

CAPÍTULO 1

Introdução

Uma maneira de evidenciar o entendimento de um sistema de software é explicar o programa, sua estrutura, seu comportamento e suas relações com o domínio da aplicação em termos diferentes daqueles utilizados durante a codificação. Por exemplo, para um ser humano a afirmação (1) "o programa soma dois números" é diferente da afirmação (2) `soma = numero1 + numero2; return soma;` (Biggerstaff et al., 1993).

A afirmação (1) é uma expressão informal que remete a conceitos do mundo real. A afirmação (2) é uma expressão formal, restrita e que segue uma gramática predeterminada por uma linguagem de programação. Para que seja possível criar, manutenir, explicar, reutilizar e documentar sistemas de software é necessário que esse possa ser representado em ambas as formas (Biggerstaff et al., 1993).

Os conceitos do mundo real relacionados ao domínio de um sistema são chamados de *interesses* (IEEE, 2000). O desenvolvimento de software é a tradução de interesses em estruturas algorítmicas adequadas. Quando o relacionamento entre interesses é complexo, sua representação em código é dificultada. Por isso, o código que implementa um software é decomposto em módulos menores, com interdependência mínima, e que representam um único interesse cada (Chang e Lu, 2009).

A programação modular permite que um módulo seja desenvolvido sem ser preciso conhecer todo o código do sistema, além de facilitar a reutilização e reescrita de código sem a necessidade de revisar todo o sistema (Parnas, 1972).

Para possibilitar a programação modular, vários paradigmas de programação foram desenvolvidos ao longo de mais de 40 anos de história da programação – imperativo, funcional, e orientação a objetos são exemplos. Cada paradigma de programação é baseado em abstrações algorítmicas específicas e representam maneiras diferentes de raciocinar sobre os interesses que compõem um sistema. Por exemplo, o paradigma imperativo permite a implementação de inte-

resses em sub-rotinas que são agrupadas em módulos com diferentes níveis de interdependência (acoplamento). O paradigma orientado a objeto permite a implementação de interesses em objetos e classes de objetos que encapsulam dados e operações que descrevem o comportamento dos objetos.

No entanto, há interesses que são difíceis de serem modularizados usando paradigmas de programação convencionais. Tais interesses são chamados de *interesses transversais*. A atividade de identificação de interesses transversais é apresentada na Seção 1.1. Na Seção 1.2 é apresentada a relação entre os paradigmas de programação mais usados atualmente e interesses de difícil modularização, além de uma alternativa para melhorar o encapsulamento dos mesmos. Em seguida são estabelecidos a motivação, justificativa, problema abordado, e objetivos deste trabalho, respectivamente nas Seções 1.3 e 1.4.

1.1 Modularização de interesses

Como já mencionado, um sistema de software é composto por diversos interesses e o relacionamento entre eles. No início do desenvolvimento, o sistema é decomposto em unidades modulares de modo que cada interesse é representado por um único módulo cuja implementação é minimamente dependente dos demais. O modo como os módulos de um sistema são organizados dá origem à estrutura de decomposição do mesmo, que uma vez estabelecida implica que todo novo interesse a ser implementado deve ser compatível à ela. Ao longo das atividades de manutenção, interesses já implementados são modificados, ou pode ocorrer a adição de novos interesses cuja implementação deixa de ser compatível com a estrutura modular original. Quando há a incompatibilidade entre o código que implementa um interesse e a estrutura modular do sistema, o interesse é chamado de *transversal* (Nguyen et al., 2011).

Para exemplificar a ocorrência de interesses transversais em um sistema de software, no Código 1.1 são mostradas duas operações comuns em sistemas bancários, *depósito* e *saque*. As operações `desbloquear()` e `bloquear()` são usadas para permitir a ocorrência de transações concorrentes sem afetar a integridade do banco de dados da agência bancária. Chamadas a `desbloquear()` e `bloquear()` são adicionadas a todas as operações que alteram os dados de uma conta. Como os métodos `desbloquear()` e `bloquear()` entrecortam a implementação das funcionalidades básicas *depósito* e *saque* eles são chamados de interesses transversais.

Código 1.1: As operações de bloqueio e desbloqueio são interesses transversais

```
public class ContaBancaria{
        Database db = new Database();
        public void deposito(double valor){
                db.bloquear();
                db.executar(valor);
                db.desbloquear();
        }
```

```
        public void saque (double valor){
                db.bloquear();
                db.executar(valor);
                db.desbloquear();
        }
        public static void main(String... args){
                ConBancaria depositar = new ContaBancaria();
                depositar.deposito(30.0);}
}
```

O paradigma de programação orientado a objetos (OO) permite a utilização de classes, objetos, e métodos como estruturas algorítmicas para encapsular e abstrair interesses em diferentes níveis de granularidade (Harbulot, 2006). Assim, é possível, por exemplo, implementar o **gerenciamento de contas** (*depósito* e *saque*) e o **controle de concorrência** (`desbloquear()` e `bloquear()`) em classes diferentes cujo relacionamento se dá por meio do envio de mensagens entre objetos dessas classes.

No exemplo representado pelo Código 1.1[1] é mostrado que interesses transversais têm relacionamentos complexos com outros interesses no sistema de tal modo que é difícil implementá-los como um único objeto ou método. Observa-se que não é possível implementar o interesse controle de concorrência em uma única classe e ao mesmo tempo impedir o espalhamento de chamadas aos métodos `desbloquear()` e `bloquear()` por todas as classes que realizam alterações em contas bancárias, como a classe `ContaBancaria`. Diversos estudos (Eaddy et al., 2008, Filho et al., 2006, Garcia et al., 2005, Greenwood et al., 2007) demonstram empiricamente que a existência de interesses transversais degrada a qualidade de código e tem impacto negativo sobre métricas de qualidade interna (Khoshgoftaar et al., 1999), por exemplo acoplamento e número de linhas de código, e qualidade externa (Emam, 2000), por exemplo manutenibilidade e número de defeitos observáveis.

Para diminuir o impacto negativo da presença de interesses transversais em sistemas de software, é possível usar a *orientação a aspectos* que permite implementar interesses transversais em unidades modulares chamadas *aspectos*. Os aspectos encapsulam o comportamento de um interesse transversal e o adiciona ao comportamento de outros interesses em tempo de compilação, e assim elimina o espalhamento e emaranhamento de código (Laddad, 2003). Desse modo, o desenvolvimento de software orientado a aspectos provê melhor separação de interesses, o que por sua vez aumenta as oportunidades de reúso de código e a coesão entre módulos (Kiczales e Hilsdale, 2001).

Programar orientado a aspectos não é uma tarefa trivial, principalmente para refatorar um software já existente. Antes de refatorar interesses transversais em aspectos, é preciso identificá-los por meio dos sintomas que apresentam. A refatoração para aspectos do exemplo mostrado no

[1]É importante notar que o exemplo dado é meramente ilustrativo. Atualmente os Sistemas Gerenciadores de Banco de Dados possuem mecanismos automáticos para o controle do grau de isolamento de transações que preservam a integridade dos dados. De modo que é desnecessária a implementação de métodos para bloqueio e desbloqueio em um contexto real.

Código 1.1 é mostrada no Código 1.2 e o processo geral para a identificação e posterior refatoração de interesses transversais para aspectos é ilustrado na Figura 1.1.

Código 1.2: Refatoração para aspectos do exemplo mostrado no Código 1.1

```
public class ContaBancaria{
        Database db = new Database()
        public void deposito(double valor){
                db.executar(valor);
        }
        public void saque (double valor){
                db.executar(valor);
        }
        public static void main(String... args){
                ConBancaria depositar = new ContaBancaria();
                depositar.deposito(30.0);
        }
}

public aspect AspectoContaBancaria{
        pointcut bloquear() : call(* Database.executar());
        pointcut desbloquear() : call(* Database.executar());
        before() : bloquear(){
                System.out.println("Bloqueou");
        }
                after() : desbloquear(){
                System.out.println("Desbloqueou");
        }
}
```

Observa-se na Figura 1.1 que os interesses transversais *#1*, *#2*, e *#3* têm sua implementação espalhada pelos módulos *M1*, *M2*, e *M3* de um sistema orientado a objetos. A identificação de interesses transversais é feita por meio da aplicação de técnicas de identificação que indicam os fragmentos de código (métodos, linhas de código, ou classes) que os implementam. Uma vez identificados, é possível usar os mecanismos oferecidos pela orientação a aspectos para modularizar os interesses transversais: *pontos de junção* (momento bem definido da execução de um programa), *ponto de corte* (agrupa um conjunto de pontos de junção), *advice* (fragmento de código que é executado em cada ponto junção em um ponto de corte). Mais detalhes sobre os mecanismos de programação oferecidos pela orientação a aspectos fogem ao escopo deste trabalho, mas podem ser encontrados em Laddad (2003).

A orientação a aspectos permite implementar interesses transversais em um único módulo (aspecto) e separá-los dos demais (*M1, M2, M3, e M4*). É importante enfatizar que para refatorar interesses transversais é necessário identificar todos os fragmentos de código que os compõem. Por isso, que principalmente para objetivos de refatoração, a precisão da técnica de identificação é de extrema importância. Se a técnica não for capaz de identificar todos os fragmentos de

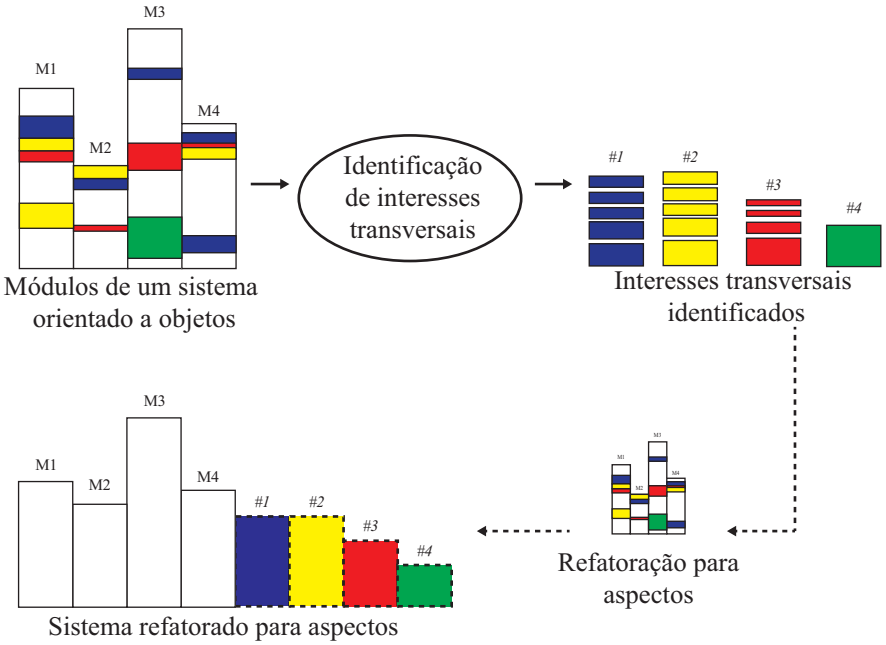

Figura 1.1: Processo de refatoração de interesses trasversais para aspectos.

código que correspondem a um interesse transversal, a refatoração do mesmo é mais difícil.

1.2 Identificação de Interesses transversais

Por mais de uma década pesquisadores têm desenvolvido técnicas e ferramentas para a identificação de interesses transversais em sistemas que não foram desenvolvidos usando a orientação a aspectos (Breu e Krinke, 2004, Marin et al., 2004, Bruntink et al., 2005, Moldovan e Serban, 2006, Zhang e Jacobsen, 2007, Zhang et al., 2008, Cojocar, 2012). Como enfatizado na Seção 1.1 e observado na literatura (Marin et al., 2004), o principal objetivo da identificação de interesses transversais é refatorá-los para aspectos de modo a obter um sistema potencialmente mais fácil de ser compreendido e manutenido (Cojocar, 2012). No entanto, nem todo interesse transversal pode/deve ser refatorado para aspecto. Há interesses transversais que podem ser eliminados por meio de mecanismos de refatoração orientados a objeto, e há aqueles que não impactam negativamente na qualidade de software, se bem documentados (Mulder, 2009). Além disso, os problemas relacionados a interesses transversais podem existir mesmo em sistemas desenvolvidos usando a orientação a aspectos desde o início.

Por exemplo, considerando a implementação do gerenciamento de contas bancárias refatorado para aspectos mostrada no Código 1.2. A implementação de uma operação para a transferência de valores entre contas também requer os mecanismos de bloqueio e desbloqueio para o controle de concorrência. Se os desenvolvedores da nova funcionalidade não souberem sobre a existência do aspecto já existente ou sua relação com a nova função, pode ocorrer a implementação de rotinas redundantes ou a omissão de rotinas importantes, o que pode levar a erros de consistência durante a execução do sistema (Nguyen et al., 2011). Logo, observa-se que a identificação de interesses transversais também pode auxiliar na manutenção, evolução e na

compreensão de programas, pois se identificados é possível analisar e entender os interesses transversais de modo isolado sem a necessidade de analisar código de outros interesses (Roy et al., 2007).

As técnicas atuais procuram por sintomas de *espalhamento* e/ou *emaranhamento* para identificar interesses transversais. Espalhamento significa que o código que implementa o interesse transversal está espalhado por diversos módulos, e o emaranhamento significa que mais de um interesse é implementado no mesmo módulo sem muita clareza sobre quais fragmentos de código implementam cada interesse (Cojocar, 2012). Os sintomas que caracterizam o espalhamento e emaranhamento se manifestam em código de modos diferentes, por exemplo múltiplas chamadas ao mesmo método a partir de diferentes módulos.

Várias técnicas de identificação têm sido propostas (Breu e Krinke, 2004, Marin et al., 2004, Bruntink et al., 2005, Moldovan e Serban, 2006, Zhang e Jacobsen, 2007, Zhang et al., 2008), e cada uma adota uma abordagem diferente com base em determinadas características de transversalidade. Mesmo com o uso de técnicas (semi)automatizadas, a identificação de interesses transversais em código é uma tarefa difícil, suscetível a erros e que toma tempo devido à complexidade, tamanho e falta de documentação dos sistemas de software atuais.

As técnicas atuais para a identificação de interesses transversais requerem o envolvimento humano para serem aplicadas (Detalhes no Capítulo 3). Mais comumente, o esforço manual exigido está relacionado à análise das *sementes* resultantes da aplicação da técnica. Isso porque o processo de identificação produz elementos de código, chamados de sementes, que podem ou não fazer parte da implementação de um interesse transversal. Por exemplo, `bloquear` e `desbloquear` no exemplo anterior fazem parte da implementação do interesse transversal *controle de concorrência*, logo, são *sementes* do mesmo interesse transversal.

A análise manual de sementes é necessária para classificá-las ou não como parte da implementação de um interesse transversal. Assim, observa-se que sementes identificadas são candidatas a interesse transversal que devem ser submetidas à confirmação por um avaliador humano. Se não confirmadas, as sementes são chamadas de *falso-positivos*. Há casos em que uma técnica não identifica determinada semente já conhecida, nesse caso a semente não identificada é chamada de *falso-negativo*. É importante notar que um dos motivos para a existência de falso-negativos pode ser a configuração incorreta da técnica e não sua incapacidade para identificar determinado interesse transversal.

Sob o contexto descrito, uma técnica ideal não identifica falso-positivos e extingue falso-negativos. Alcançar tal cenário é o maior desafio das técnicas de identificação atuais, visto que a maioria não apresenta números expressivos em precisão.

1.3 Motivação e Justificativa

As técnicas atuais para a identificação de interesses transversais são imprecisas (Mens et al., 2008). A precisão de uma técnica é a razão entre a quantidade de sementes identificadas e a quantidade de sementes identificadas que fazem parte da implementação de um interesse

transversal. Baixa precisão significa que uma quantidade muito pequena do total de sementes identificadas estão de fato relacionadas a algum interesse transversal. A qualidade do resultado de uma técnica ajuda a aliviar o esforço manual requerido para separar falso-positivos e sementes que implementam algum interesse transversal (Zhang et al., 2008).

Nesse contexto, este trabalho é motivado pela busca por melhores resultados no processo de identificação de interesses transversais. Esse processo é prejudicado, principalmente pela inconsistência entre as características dos interesses transversais que se deseja identificar e as características de transversalidade que a técnica de identificação escolhida é capaz de analisar. Prover tal consistência permite que os resultados obtidos sejam validados de modo justo, ou seja, de acordo com aquilo que a técnica é capaz de identificar. Como consequência, a precisão também melhora.

Um primeiro passo em direção ao melhor entendimento da relação técnica-interesse transversal foi dado por Marin et al. (2005) ao propor uma classificação comum para interesses transversais de acordo com suas características. Tal classificação é apresentada na Seção 4.5. Porém, também é preciso compreender as características das técnicas de identificação para que se possa relacionar uma técnica à identificação de determinado sintoma de transversalidade. Compreender as características de uma técnica significa analisar quais sintomas ela consegue ou não identificar e por qual motivo.

1.4 Formulação do Problema e Objetivos

A maior parte dos problemas relacionados às técnicas de identificação atuais são devidos à dificuldade em relacionar as capacidades de identificação da técnica ao conjunto de sintomas que caracterizam um interesse como transversal. As técnicas de identificação atuais são baseadas em suposições sobre o modo como interesses transversais são implementados. Por isso, elas se tornam especialistas na identificação de determinadas características de transversalidade, por exemplo padrões de sintaxe ou de execução. No entanto, interesses transversais podem apresentar diferentes conjuntos de características de transversalidade, o que torna impraticável identificar todos os interesses transversais em um sistema utilizando apenas uma técnica (Nguyen et al., 2011). Além disso, podem ocorrer problemas como a incompatibilidade entre técnica e sistema analisado (Roy et al., 2007);

Assim, este trabalho tem por objetivo oferecer suporte à escolha da técnica de identificação a ser utilizada por meio da elaboração de um conjunto de diretrizes capazes de auxiliar na escolha da técnica ou do conjunto de técnicas mais apropriadas para que seja mantida a consistência entre as características dos interesses transversais que se deseja identificar e as características que a técnica é capaz de identificar.

Adicionalmente, é proposto um catálogo de sementes que implementam interesses transversais e que foram confirmadas e documentados na literatura. O catálogo auxilia na avaliação da proposta deste trabalho por permitir a comparação e validação de técnicas de identificação escolhidas com o apoio do conjunto de diretrizes elaboradas.

Dentre as contribuições deste trabalho, destaca-se também, a análise documentada do comportamento de interesses transversais e sementes que compõem o catálogo, além da análise detalhada das capacidades e limitações de técnicas atuais para a identificação de interesses transversais. Considera-se que tais contribuições são importantes para a melhoria das técnicas existentes e para a criação de novas técnicas.

1.5 Organização

Para discorrer sobre o tema e objetivos deste trabalho, o texto está organizado da seguinte maneira: os conceitos relacionados a interesses transversais, características de transversalidade, e a classificação de interesses transversais são discutidos no Capítulo 2. No Capítulo 3 são apresentadas as principais abordagens utilizadas pelas técnicas de identificação de interesses transversais atuais. Também é feita uma análise das técnicas para identificar os fatores que têm maior impacto no processo de identificação. A metodologia utilizada para a realização da proposta deste trabalho é descrita no Capítulo 4 enquanto que no Capítulo 5 as diretrizes propostas neste trabalho são apresentadas e discutidas. As diretrizes são aplicadas a um sistema de software real e é feita uma discussão sobre as vantagens de seu uso. Por fim, no Capítulo 6 são demonstradas as ameaças ao trabalho proposto e delineadas as conclusões obtidas. Também são elencados os trabalhos futuros e as publicações fruto do trabalho apresentado.

CAPÍTULO 2

Interesses Transversais

Interesses transversais têm sido classificados, explorados e definidos na literatura por meio de estudos empíricos e teóricos (Marin et al., 2005, Ducasse et al., 2006, Marin et al., 2006, 2007, Figueiredo et al., 2009, da Silva et al., 2009). Compreendê-los é um fator crucial para criar técnicas que os identifiquem, melhorar as técnicas já existentes, e para estabelecer critérios de escolha da técnica a ser utilizada.

Para discutir sobre interesses transversais, os sintomas que apresentam e suas classificações, é necessário entender a relação existente entre os conceitos: características de transversalidade, sintomas de transversalidade, interesses transversais, tipos de interesse transversal, sementes e técnicas de identificação.

Interesses transversais podem ser classificados por *tipos* que descrevem o conjunto de sintomas que exibem. As técnicas de identificação localizam interesses transversais buscando por tais sintomas, que também podem ser chamados de *características de transversalidade* ou *sintomas de transversalidade*. Sendo assim, ao longo dos próximos capítulos, os termos *sintoma*, *característica de transversalidade* e *sintoma de transversalidade* são usados como sinônimos.

Na Figura 2.1 é ilustrado o relacionamento entre interesses transversais, tipos, sintomas, e técnicas. À esquerda, é representado um arquivo de código fonte de um sistema qualquer. Nesse arquivo, alguns fragmentos do código fonte exibem sintomas de transversalidade (representados pelos padrões em linhas verticais e horizontais), logo são trechos do código fonte que implementam interesses transversais, também conhecidos por *sementes*. Os sintomas representados na Figura 2.1 são relacionados a interesses transversais classificados como *Tipo 1* e *Tipo 2*, à direita. Ao meio, tem-se uma *Técnica A* que é capaz de identificar apenas os sintomas representados pelas linhas horizontais. Como a *Técnica A* é capaz de identificar os sintomas descritos pelo *Tipo 1*, pode-se dizer que ela é capaz de identificar interesses transversais daquele tipo. Além disso, observa-se que a *Técnica A* não é capaz de identificar interesses transversais do

Tipo 2, pois não identifica nenhum sintoma relacionado ao *Tipo 2* (linhas verticais).

Há três observações importantes que devem ser consideradas sobre o exemplo da Figura 2.1: (1) vários sintomas diferentes podem coexistir em um sistema; (2) consequentemente, um sistema pode conter sementes de interesses transversais classificados como de tipos distintos; (3) e uma técnica de identificação pode não identificar todo sintoma exibido em um sistema, sendo assim ela não é capaz de identificar todos os tipos de interesses transversais que o mesmo implementa.

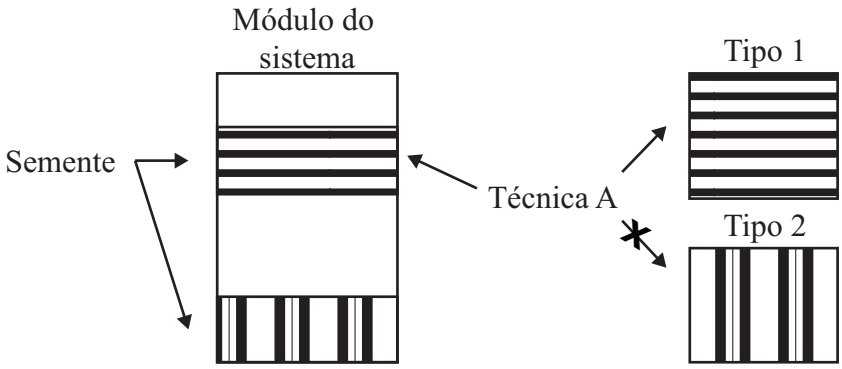

Figura 2.1: Relação entre interesses transversais, tipos, sintomas, e técnicas de identificação.

Embora não seja objetivo do presente trabalho sumarizar os diversos modos como interesses transversais se manifestam em sistemas de software, optou-se por explorar com algum detalhe a explicação de sintomas, interesses transversais e classificações, pois é importante para a formulação das diretrizes propostas nesta dissertação. Inicialmente os sintomas são discutidos e relacionados a dependências entre artefatos de software na Seção 2.1. Em seguida, na Seção 2.2, são apresentadas as classificações de interesses transversais por tipo propostas na literatura. A Seção 2.3 conclui o capítulo.

2.1 Sintomas de Transversalidade

O relacionamento entre os interesses implementados em um sistema gera dependências. Quando um interesse é transversal sua relação de dependência com outros interesses é complexa, e causa problemas de modularização, entendimento, e manutenção. Sintomas de transversalidade refletem relações de dependência problemáticas entre interesses transversais e outros interesses.

Há dois tipos de relações de dependência: sintática e lógica. Dependências sintáticas são estabelecidas pela relação entre elementos específicos do código fonte, como instruções, funções/métodos, e classes. Além disso, podem ser referentes a dados ou a relações funcionais, por exemplo, um método *A* faz chamada a um método *B*. A dependência lógica (ou acoplamento lógico), por sua vez, é um tipo de dependência implícita e difícil de ser identificada diretamente no código fonte, mas que pode ser observada por meio da análise do relacionamento entre artefatos, por exemplo classes que são frequentemente alteradas juntas tendem a ser logicamente acopladas.

A seguir é feita uma discussão sobre sintomas de transversalidade sob a perspectiva das dependências sintática (ou estrutural) e lógica discutidas anteriormente.

2.1.1 Discussão

Fowler et al. (1999) propuseram um catálogo composto por 22 anomalias estruturais de código que prejudicam sua manutenção e compreensão, as considerando, portanto, oportunidades de refatoração. Além disso, os autores elaboraram um guia para a refatoração de tais anomalias utilizando o paradigma orientado a objetos. No entanto, observa-se que alguns dos problemas estruturais documentados no catálogo proposto não são facilmente resolvidos utilizando mecanismos orientados a objetos. Por exemplo:

- *Código duplicado:* o mesmo código aparece em mais de um lugar no projeto. Por exemplo, a mesma expressão é implementada em dois métodos de uma mesma classe, em duas classes na mesma estrutura de herança, ou em duas classes não relacionadas. Código duplicado pode não corresponder a fragmentos de código idênticos, mas similares, ou diferentes mas que executam a mesma funcionalidade.

- *Alterações divergentes:* caracterizado quando uma classe frequentemente sofre diferentes tipos de alterações por diferentes motivos. Por exemplo, um conjunto de n métodos de uma mesma classe são constantemente alterados por motivos distintos.

- *Cirurgia de rifle:* é o oposto de *Alterações divergentes*, sempre que uma alteração é feita em código, é necessário fazer várias pequenas alterações em classes diferentes para que a estrutura do código seja adaptada à mudança.

A duplicação e a alteração frequente de código em decorrência de atividades de manutenção ou adição de novas funcionalidades são sintomas de transversalidade conhecidos na literatura e não são facilmente refatoráveis, pois não há mecanismos específicos que deem suporte à modularização total de código transversal (Massicotte et al., 2007). Obviamente, tem-se a orientação a aspectos que ajuda a minimizar os problemas causados por interesses transversais, porém com limitações como discutido na Seção 1.2.

Sendo assim, é possível afirmar que a identificação de anomalias estruturais pode levar à identificação de interesses transversais. Bernardi e Di Lucca (2009) concordam que propriedades estruturais podem ser exploradas na busca por sintomas de transversalidade, porém é um erro buscar por interesses transversais analisando as propriedades estruturais de um único interesse. Isso porque, segundo Bernardi e Di Lucca (2009), a transversalidade só passa a existir quanto é estabelecida uma relação de dependência entre dois interesses, um entrecorta o outro (van den Berg et al., 2006). Logo, a análise dos relacionamentos de dependência entre interesses é uma estratégia mais eficiente para a busca por sintomas de transversalidade.

Da Silva (2009) confirmam empiricamente a hipótese de Bernardi e Di Lucca (2009) utilizando a estratégia de identificação proposta por Marinescu (2004) para demonstrar que a exploração de sintomas puramente estruturais para identificar interesses transversais é limitante.

Os experimentos executados por Marinescu (2004) são baseados em métricas de modularização para medir a qualidade estrutural de um sistema de software. Por exemplo, para detectar o sintoma *Cirurgia de Rifle*, Marinescu (2004) define-se a seguinte regra:

```
Shotgun-Surgery := (CM.topValues(20%) and (CC, HigherThan(5)))
```

CC (Changing Classes) e *CM* (Changing Method) medem o acoplamento e representam, respectivamente, o número de métodos distintos que fazem chamada a outro método e o número de classes que fazem chamadas a determinado método. Os valores *topValues* e *HigherThen* determinam o limite para as métricas *CC* e *CM*. Assim, para que a regra definida indique o sintoma *Cirurgia de Rifle* em determinado método, é necessário que seu valor de *CM* seja entre os 20% maiores valores para todos os métodos do sistema, e ao menos 5 classes devem fazer chamadas a ele.

Da Silva (2009) aplicaram a estratégia de Marinescu (2004) ao sistema web *Health Watcher*. Na Figura 2.2 os componentes em cinza implementam o padrão *Observer* de tal sistema (Gamma et al., 1995). A aplicação da regra apresentada anteriormente nessa instância de *Observer* resulta em *CC = 0* para a interface *Subject*. Isso indica que nenhuma classe acessa *Subject* diretamente, logo não é identificada nenhuma anomalia estrutural que se relacione a ela. Entretanto, alterações na interface *Subject* podem ocasionar modificações nas classes que a implementa. Por exemplo, renomear o método `removeObserver` ocasiona alterações nas classes `Complaint`, `Employee`, e `HealthUnit` (Da Silva, 2009).

A análise do exemplo anterior mostra que os elementos do padrão *Observer*, que é um interesse transversal (Da Silva, 2009, Marin et al., 2004), não são identificados. Isso acontece pois a abordagem de identificação executada é incapaz de descrever um problema de transversalidade cujo sintoma só é percebido se for analisada a dependência lógica entre os módulos que implementam interesses distintos (Relacionamento entre *Subject* e as classes *HealthUnit*, *Complaint*, e *Employee*). Também é possível observar a importância em manter a consistência entre os sintomas de transversalidade e a abordagem adotada para identificar interesses transversais. A estratégia de detecção de Marinescu (2004) não é errada, e é eficiente na identificação de padrões estruturais em código (dependências sintáticas), porém é incapaz de detectar interesses transversais relacionados ao padrão *Observer*, que dependem do contexto ao qual pertencem para serem identificados.

Na Figura 2.2 a relação de dependência lógica implícita entre as classes *HealthUnit*, *Complaint*, *Employee* e *Subject* poderia ser identificada por outras abordagens, por exemplo a análise de alterações feitas em conjunto.

É importante observar que os sintomas de transversalidade gerados por problemas relacionados a dependências sintáticas e lógicas variam, por exemplo duplicação de código, alterações em cascata, padrões de chamadas a métodos, entre outros. Assim, interesses transversais podem exibir diversos sintomas diferentes. Para facilitar o entendimento e identificação de interesses transversais, a literatura mostra tentativas em classifica-los de acordo com o conjunto de sinto-

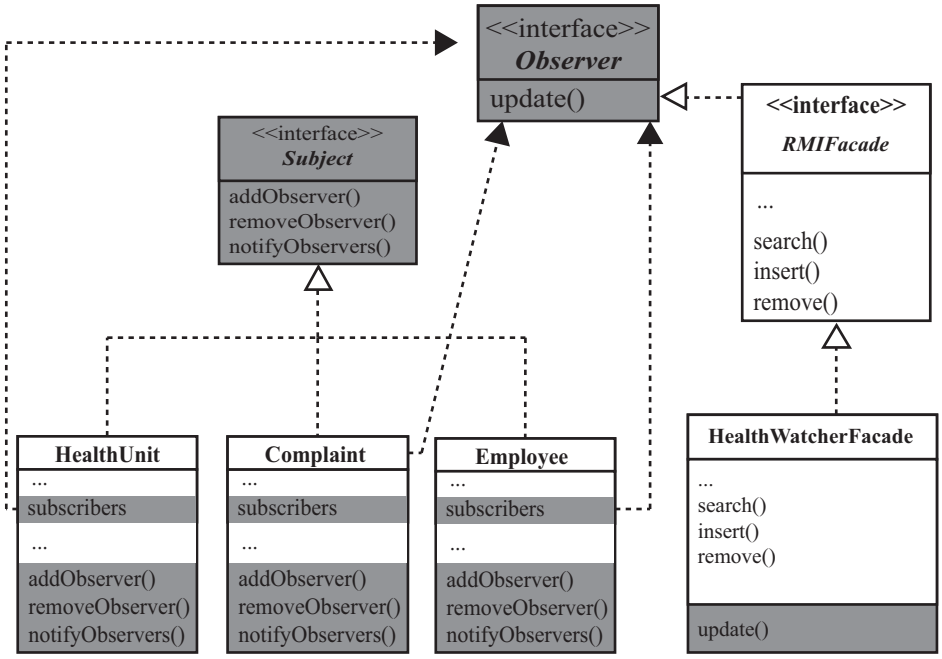

Figura 2.2: O padrão ***Observer*** no sistema HealthWatcher (Adaptado de Da Silva (2009))

mas que exibem. A seguir são apresentadas as classificações de interesses transversais por tipo propostas na literatura.

2.2 Tipos de Interesses Transversais

Marin et al. (2005) propõem a classificação de interesses transversais em tipos que os descreve e classifica com base em três características: (1) objetivo do interesse; (2) idioma de implementação; e (3) mecanismo de linguagem orientado a aspectos que dá suporte à modularização de cada instância do interesse.

Os tipos propostos por Marin et al. (2005) são sempre associados à menor unidade que pode ser usada para expressar individualmente e modularizar um interesse transversal. Ou seja, o mecanismo de linguagem orientado a aspectos capaz de modularizar cada semente de um interesse transversal. Consequentemente, a implementação de um interesse transversal pode ser expressa como uma combinação de uma ou mais sementes de um tipo.

Marin et al. (2005) definem um novo tipo sempre que: (1) não é possível compor o novo tipo utilizando aqueles já definidos, e (2) o novo tipo não pode ser decomposto em unidades menores. Os autores ressaltam que nem todos os tipos precisam necessariamente ser associados a um mecanismo de linguagem orientado a aspectos, eles podem ser referentes a mecanismos desejados os quais poderiam ajudar na evolução de linguagens de programação.

A classificação proposta por Marin et al. (2005) dá um nome para cada tipo para prover uma denominação comum a todas as sementes do interesse transversal correspondente. Além do nome, é descrito o objetivo do tipo em termos de comportamento, estrutura, e/ou requisitos exigidos pela implementação. O idioma de implementação descreve como, geralmente, é a implementação dos interesses transversais que correspondem ao tipo. Finalmente, cada tipo

é associado a um mecanismo de linguagem orientada a aspectos que pode ser utilizado para refatorar suas instâncias. Os treze diferentes tipos de interesses transversais definidos por Marin et al. (2005) são descritos a seguir.

Comportamento consistente (*Consistent Behavior*) são interesses transversais que implementam determinado comportamento de maneira consistente na execução de métodos. Tal comportamento pode ser capturado por um ponto de corte durante uma possível refatoração para aspectos. Esse tipo é caracterizado por chamadas a métodos que implementam uma funcionalidade desejada. Notificações a *listeners* é um exemplo de interesse transversal desse tipo.

Imposição de contrato (*Contract Enforcement*) corresponde a interesses transversais que implementam regras de contrato, por exemplo pré e pós validação de condicionais. Esse tipo é caracterizado por chamadas a métodos que implementam a verificação de condições no sistema. Assim como o tipo anterior, interesses transversais do tipo imposição de contrato podem ser capturados por um ponto de corte durante a refatoração para aspectos. É possível observar que os tipos *Imposição de Contrato* e *Comportamento Consistente* têm características em comum, no entanto eles têm objetivos diferentes de acordo com a definição de Marin et al. (2005): interesses transversais do tipo *Imposição de Contrato* garantem determinadas condições devido a relações entre um método e seus chamadores.

Papéis entrelaçados (*Entangled Roles*) são interesses transversais que estendem determinado método de modo que ele passa a exercer um segundo papel ou responsabilidade cuja implementação é emaranhada a seu papel principal. O idioma de implementação que descreve esse tipo é a implementação de um método com funcionalidades referentes a um interesse que não o principal tratado pelo método. Esse tipo de interesse transversal é capturado por pontos de corte e *advices* do tipo *around* durante a refatoração para aspectos. De acordo com Marin et al. (2005) esse tipo são especificamente relacionados a sobreposição de papéis a nível de métodos. Por exemplo, na estrutura do Java Swing, é comum classes assumirem ambos os papéis *view* e *controller*: um item de menu em uma interface é habilitado/desabilitado se o comando a ser executado ao selecionar o item o habilita ou desabilita. Apesar de o método responsável por habilitar/desabilitar o item de menu ser parte da interface *view*, sua execução é baseada em decisões do *controller*.

Camada de redirecionamento (*Redirection Layer*) são interesses transversais que definem uma camada de interface a um objeto (adiciona uma funcionalidade ou altera o contexto) e encaminha chamadas a ele. São geralmente implementados por uma camada de redirecionamento que possui métodos responsáveis por repassar chamadas a outros métodos. Esse tipo de interesse transversal é capturado por pontos de corte e *advices* do tipo *around* durante a refatoração para aspectos. Os padrões *decorator* e *adapter* são exemplos desse tipo.

Adição de Variabilidade (*Add Variability*) correspondem a interesses transversais que usam métodos como parâmetro. Interesses transversais desse tipo criam objetos de tipos (anônimos) que implementam uma ação única e específica: `ActionListener.actionPerformed()`, `Command.execute()`, `Runnable.run()`. Esse tipo generaliza interesses que

são implementados por um conjunto de métodos. Os métodos assumem responsabilidade pela chamada a determinado objeto, e possivelmente executam tarefas adicionais como modificação de parâmetros de chamadas.

Exposição de Contexto (*Expose Context*) são interesses transversais que expõem o contexto de métodos chamadores aos métodos chamados por meio da passagem de informações a cada um dos métodos na pilha de chamadas (não é necessário passar as informações como um conjunto de parâmetros para cada método no fluxo de controle). Esse tipo também é conhecido como *Wormhole* (Laddad, 2003). O idioma de implementação é caracterizado pela adição de argumentos a cada método na pilha de chamadas. Esse tipo de interesse transversal também pode ser capturado por pontos de corte e *advices*, nos quais o ponto de corte captura o contexto a ser passado.

Sobreposição de Papéis (*Role Superimposition*) é um tipo de interesse transversal cujo objetivo é implementar um papel ou responsabilidade secundária específica. É caracterizado pela implementação de uma interface, ou implementação direta de métodos que poderiam ser abstraídos pela definição de uma interface. Podem ser capturados por um ponto de inserção (introduction mechanisms) durante a refatoração para aspectos. São exemplos desse tipo os padrões *Observer*, *Command*, e *Visitor* (Marin et al., 2004).

Classes de Suporte para Sobreposição de Papéis (*Support Classes for Role Superimposition*) é um tipo de interesse transversal que usa classes aninhadas para tornar o relacionamento entre classe explícito. Assim torna-se possível sobrepor determinado papel a uma hierarquia de classes. Não há um idioma de implementação específico para esse tipo mas é possível observá-lo em comentários e documentação do sistema.

Propagação de Exceção (*Exception Propagation*) corresponde a interesses transversais que implementam a propagação de exceção para eventos em que nem o método nem os chamadores têm uma resposta apropriada. Geralmente, é um tipo caracterizado pela propagação da exceção para os métodos chamadores.

Declaração de Cláusula *throws* (*Declare Throws Clause*) são interesses transversais que adicionam uma exceção específica à cláusula *throws* para todos os métodos dentro de determinado contexto.

Imposição de Design (*Design Enforcement*) é um tipo em que há a imposição de determinado tipo de estrutura ao sistema. Por exemplo, todas as classes em uma hierarquia devem declarar construtores sem argumentos.

Imposição de Comportamento Dinâmico (*Dynamic Behavior Enforcement*) são interesses transversais que implementam a imposição de regras para o uso de objetos, por exemplo obrigatoriedade de inicialização antes do uso.

É possível observar que os tipos *Imposição de Contrato* e *Comportamento Consistente* têm características em comum, no entanto eles têm objetivos diferentes de acordo coma definição de Marin et al. (2005): interesses transversais do tipo *Imposição de Contrato* garantem deter-

minadas condições devido às relações entre um método e seus chamadores.

Os tipos *Comportamento Consistente*, *Imposição de Contrato*, *Papéis Entrelaçados*, *Camada de Redirecionamento*, *Adição de variabilidade* e *Exposição de Contexto* têm em comum o objetivo de prover consistência em relação a determinado comportamento do sistema. Assim, tais tipos podem ser generalizados em uma categoria de tipos comportamentais. O que difere cada um deles é o idioma de implementação que os caracteriza. Isso implica que em uma atividade de refatoração para aspecto são necessários mecanismos diferentes para que cada um deles sejam refatorados corretamente (Marin et al., 2005).

A classificação proposta por Marin et al. (2005) estabelece uma linguagem comum para a descrição de situações que incorrem em interesses transversais. Além disso, a classificação proposta é capaz de auxiliar pesquisadores a compreender os resultados obtidos por técnicas de identificação de interesses transversais por meio do mapeamento de sintomas identificados para os tipos genéricos de interesses.

A classificação de interesses transversais proposta por Ducasse et al. (2006) é mais genérica que a apresentada por Marin et al. (2005) e, diferentemente, independe do paradigma de programação usado. Ducasse et al. (2006) propõe metáforas para a classificação de interesses transversais que facilitam o entendimento de sintomas de transversalidade. A classificação é estendida por da Silva et al. (2009) e Figueiredo et al. (2009) resultando em 13 sintomas, 3 deles descritos a seguir.

Ovelha Negra (*Black Sheep*): Sintoma caracterizado pelo espalhamento de fragmentos de código em diversos módulos do sistema. Não há um módulo cuja maior parte do código correspondente é dedicado à implementação do interesse. Este sintoma é ilustrado na Figura 2.3 em que as caixas representam módulos de um sistema e as áreas em cinza indicam os fragmentos de código que implementam o interesse.

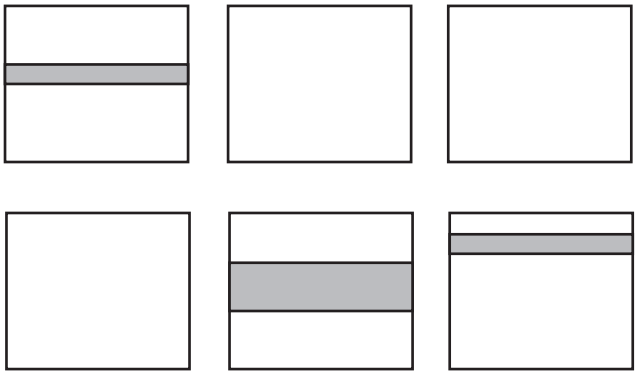

Figura 2.3: Representação do sintoma *Ovelha negra* em diversos módulos de um sistema (Adaptado de (Da Silva, 2009))

Polvo (*Octopus*): Interesses transversais do tipo Polvo são relativamente bem modularizados, porém há código que o implementa espalhado por outros módulos. O módulo dedicado somente à implementação de parte do interesse é o 'corpo' do polvo, e os demais fragmentos de código espalhados são os 'tentáculos'. O sintoma é ilustrado pela Figura 2.4, em que uma caixa

que representa um módulo é totalmente dedicada à implementação do interesse, e as demais caixas indicam o espalhamento dos 'tentáculos' do polvo.

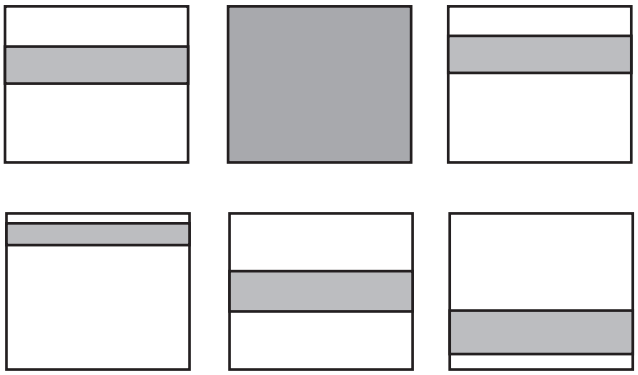

Figura 2.4: Representação do sintoma *Polvo* em diversos módulos de um sistema (Adaptado de (Da Silva, 2009))

Dominador (*God Concern*): Sintoma relativo a interesses que implementam diversas funcionalidades do sistema. Esse tipo de interesse apresenta alto grau de espalhamento e compõe a maior parte do sistema. O sintoma é ilustrado na Figura 2.5.

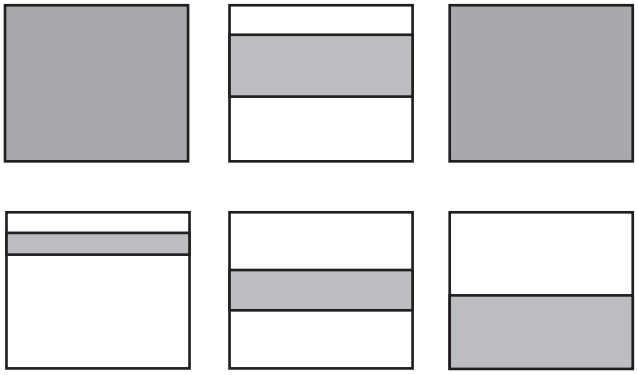

Figura 2.5: Representação do sintoma *Dominador* em diversos módulos de um sistema (Adaptado de (Da Silva, 2009))

Os três sintomas descritos são interesses implementados em módulos parcialmente ou totalmente dedicados a eles. Há ainda sintomas relativos a herança que envolvem estruturas hierárquicas, são eles: Planta Trepadeira (*Clibing Plant*) e Doença Hereditária (*Hereditary Desease*); sintomas relativos a acoplamento, caracterizados pela existência de acoplamento entre os componentes do sistema que implementam um interesse, são eles: Interesse Enraizado (*Tree Root*), Tsunami, Serpente (*King Snake*), e Rede Neural (*Neural Network*); e sintomas relativos a problemas de modularidade, são eles: Cópia Carbono (*Copy Cat*), Ovelha Dolly (*Dolly Sheep*), Interesse Exclusivo de Dados (*Data Concern*), e Interesse Comportamental (*Behavioural Concern*).

2.3 Considerações Finais

Neste capítulo foi demonstrado que a transversalidade é caracterizada e classificada de maneiras diferentes na literatura. O principal problema relacionado às múltiplas definições e classificações é que a validação dos resultados produzidos pelas técnicas de identificação de interesses transversais é ambígua. Ou seja, duas pessoas podem discordar acerca de um mesmo candidato a interesse transversal sendo que uma define tal candidato como real-positivo e a outra como falso-positivo.

Como a interpretação de resultados é sujeita ao indivíduo que interpreta, resultados da replicação de determinada técnica tornam-se subjetivos. A comparação entre técnicas também é complexa, não somente pelas diferentes interpretações acerca dos resultados obtidos mas também pelas especificidades de cada uma.

Além da falta de consenso sobre o que é a transversalidade de interesses, muitos autores associam transversalidade diretamente a aspectos (orientação a aspectos). Em geral, as técnicas para identificação de interesses transversais retornam conjuntos de elementos de código (classes, métodos, fragmentos de código) que, de acordo com determinada definição de transversalidade, são candidatos a aspectos. No entanto, quando um elemento de código é apontado como pertencente a um interesse transversal ele não necessariamente deve ser refatorado para um aspecto.

A associação direta interesse transversal-aspecto pode tornar-se um problema, pois nem sempre são identificados todos os elementos de código que implementam determinado interesse transversal. Se o objetivo da identificação é a refatoração de um interesse transversal para aspecto, todos os elementos que o implementa devem ser identificados. Desse modo, é necessário definir de modo não ambíguo qual o tipo de interesse transversal que se deseja identificar, e escolher a técnica mais apropriada para esse tipo, de modo que sejam obtidos resultados mais precisos.

Neste capítulo foram apresentadas as principais características de interesses transversais e foram dados vários exemplos das diferentes definições de transversalidade e interpretação de resultados obtidos na busca por interesses transversais. Foram apresentados os tipos de interesses transversais propostos pela classificação de Marin et al. (2005) e Ducasse et al. (2006). Detalhes sobre as principais abordagens para a identificação de interesses transversais adotadas pelas técnicas propostas na literatura são apresentados no Capítulo 3.

CAPÍTULO 3

Técnicas para a identificação de interesses transversais

As técnicas para a identificação de interesses transversais propostas na literatura implementam diferentes abordagens de identificação. As mais comuns empregam um algoritmo ou uma combinação de algoritmos baseados em estruturas de árvore, algoritmos de pesquisa em grafos, detecção de clones, análise estrutural de componentes e de execução, análise formal de conceitos, análise fan-in, regras de associação, e análise histórica.

Além disso, o processo de identificação de interesses transversais pode ser analisado sob duas perspectivas diferentes: geral e específica. A primeira diz respeito às etapas executadas para a identificação de interesses transversais em qualquer técnica de identificação. A segunda diz respeito às etapas de identificação executadas por uma abordagem/técnica ou ferramenta específica.

As etapas de identificação comuns a todas as técnicas de identificação de interesses transversais são ilustradas na Figura 3.1.

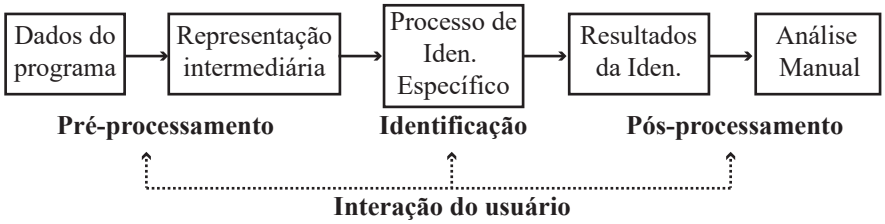

Figura 3.1: Etapas do processo de identificação de interesses transversais (Adaptado de Delfim (2013))

De acordo com as etapas de identificação mostradas na Figura 3.1, o primeiro passo para executar a identificação de interesses transversais é a coleta de dados do programa em análise,

e se necessário, o pre-processamento dos mesmos para que sirvam como entrada para a técnica escolhida. O pre-processamento resulta em uma representação intermediária do programa, que é submetida à técnica de identificação escolhida e pode requerer a intervenção manual do usuário, por exemplo para a criação de casos de uso na aplicação de técnicas dinâmicas (Tonella e Ceccato, 2004).

É importante notar que nem sempre o pre-processamento dos dados de entrada é manual. Por exemplo, a ferramenta FINT (Marin et al., 2004) faz a identificação de candidatos a interesses transversais sem requerer nenhum tipo de pre-processamento manual. Ao finalizar a atividade de identificação a ferramenta/abordagem escolhida apresenta os resultados obtidos que são analisados manualmente pelo usuário. Essa é a etapa de pós-processamento.

O processo de identificação específico, envolve atividades que podem ser executados manualmente ou não, como por exemplo a configuração de filtros e limiares para a aplicação de técnicas específicas. Como exemplo de funcionamento do processo específico de identificação, duas técnicas são apresentadas a seguir: Análise Fan-in e Detecção de Clones.

3.1 Análise Fan-in

Marin et al. (2004) propuseram a análise fan-in para a identificação de interesses transversais em sistemas de software após observar que, geralmente, interesses transversais exibem alto valor fan-in. A análise fan-in calcula o valor da métrica fan-in – obtida estaticamente a partir do código fonte – para todos os métodos em um sistema orientado a objetos. Marin et al. (2004) definem o valor fan-in de um método m como o número de métodos em um sistema S que faz chamada à m.

Apesar de ser uma técnica estática, o algoritmo para cálculo do valor fan-in elaborado por Marin et al. (2004) trata o polimorfismo fazendo com que uma chamada a um método *m* contribua também para o valor fan-in de todos os métodos que o refina. Desse modo métodos polimórficos não são punidos. O processo de identificação específico estabelecido pela análise fan-in é ilustrado na Figura 3.2 e consiste de três etapas que são descritas a seguir:

- Cálculo do valor fan-in para todos os métodos do sistema. Tal cálculo depende do algoritmo elaborado e de como tal algoritmo trata casos específicos como o polimorfismo.
- Filtragem de resultado, que deve ser limitado a um conjunto de métodos cujos valor fan-in é maior que um valor mínimo previamente definido. Além disso, devem ser descartados métodos utilitários, assessores e modificadores para evitar ruídos nos resultados.
- Análise manual dos métodos resultantes do processo de identificação para validá-los ou não como parte da implementação de um interesse transversal.

Os experimentos em que a análise fan-in foi aplicada para a identificação de interesses transversais (Marin et al., 2004) apresentaram precisão aproximada de 33%, ou seja, 33% dos

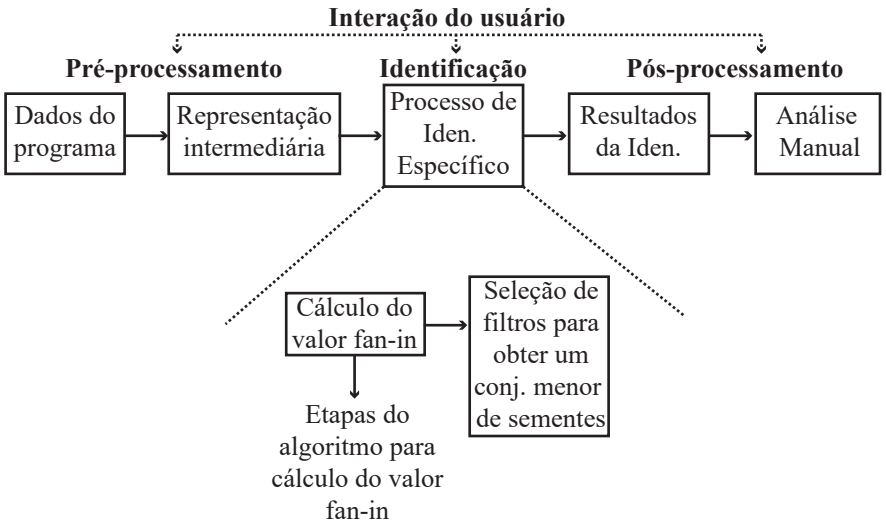

Figura 3.2: Perspectiva específica do processo de identificação de interesses transversais - Análise Fan-in (Adaptado de Delfim (2013)).

métodos resultantes correspondiam à implementação de algum interesse transversal. Nós reaplicamos a análise fan-in utilizando uma abordagem histórica para a identificação de interesses transversais e foi obtida uma precisão de até 39% (o experimento realizado é descrito em Marcal et al. (2017)).

3.2 Detecção de Clones

A duplicação de código é um dos sintomas que interesses transversais podem apresentar, pois como são incompatíveis com a estrutura do sistema sua implementação acaba espalhada por diversos trechos de código (principalmente devido ao reuso de código). A detecção de duplicação de código como técnica para a identificação de interesses transversais pode adotar diferentes abordagens que implicam em diferentes processos específicos de identificação. Por exemplo, em Bruntink et al. (2004) são utilizados três detectores de duplicação diferentes, um baseado em grafos de dependência, um segundo em árvores de sintaxe abstrata, e o terceiro em *tokens*. Tais detectores são descritos a seguir.

Detecção de Clones em Grafos de Dependência utiliza grafos de dependência para identificar duplicação de código. Cada instrução (*statement*) no código de um programa é representada por um nó do grafo e as arestas representam relações de dependência entre as instruções. A duplicação de código é identificada por meio da comparação entre vários grafos de dependência gerados a partir do programa em análise.

Detecção de Clone em Árvores de Sintaxe Abstrata (AST) utiliza a árvore de sintaxe abstrata do código fonte. Os algoritmos de detecção de duplicação de código desse tipo buscam por sub-árvores semelhantes na AST e retornam pares de instruções similares que são potenciais sementes de interesses transversais.

Detecção de Clone por análise de *Tokens* aplica análise léxica ao código fonte e subsequentemente os *tokens* gerados são utilizados como base para a detecção de duplicação de código.

A maioria dos detectores de clones produzem como resultado pares de clones, ou seja pares de fragmentos de código que são similares. Após a identificação dos pares de clones utilizando os detectores, Bruntink et al. (2004) agrupam os clones obtidos em grupos chamados de *classes de clones*. Todos os fragmentos de código de uma mesma classe são clones entre si.

As classes são então avaliadas para determinar a eficiência do detector para a identificação de interesses transversal. Idealmente cada classe deve conter todo trecho de código relacionado a um determinado interesse transversal. Quanto menos classes forem necessárias para capturar toda a extensão da implementação de um interesse transversal, mais eficiente é o detector.

3.2.1 Análise, Comparação, e Combinação de Técnicas

Nesta seção é feita uma análise descritiva qualitativa das técnicas de identificação de interesses transversais consideradas pra este trabalho. O objetivo de tal análise é fazer um levantamento e descrever os principais atributos de cada abordagem adotada para a identificação de interesses transversais.

Inicialmente foram selecionados os critérios a serem utilizados para a análise das técnicas. Kellens et al. (2007) apresentam um conjunto de critérios para comparar técnicas de identificação de interesses transversais. Os critérios foram elaborados para permitir a análise de características específicas das técnicas. Assim, é possível contrastar as restrições impostas por cada técnica, o tipo de análise executada, o grau de automação e a escalabilidade. Utilizaremos um subconjunto dos critérios estabelecidos por Kellens et al. (2007) para a análise de técnicas para a identificação de interesses transversais. A seguir é apresentada uma breve descrição dos critérios definidos em Kellens et al. (2007).

Dados estáticos ou dinâmicos Critério relativo ao ***tipo de dados*** que a técnica analisa. Dados podem ser obtidos por meio de processamento estático do código fonte do sistema, ou dinamicamente por meio da execução de casos de uso pré-definidos, por exemplo. Há técnicas híbridas que executam a análise sobre dados obtidos estática e dinamicamente.

Análise léxica, estrutural, ou comportamental Critério referente ao ***tipo de análise*** executada sobre os dados obtidos a partir de determinado programa. A análise léxica preocupa-se com sequências de caracteres e expressões regulares. Já a análise estrutural/comportamental preocupa-se com árvores de derivação sintática, tipo de informação e envio de mensagens, por exemplo.

Granularidade Critério relativo à ***granularidade*** das sementes resultantes da aplicação de uma técnica. Há técnicas em que os resultados correspondem a métodos, outras a fragmentos de código ou classes (quando o sistema em análise é orientado a objetos, por exemplo).

Espalhamento ou emaranhamento de código Critério relativo aos ***sintomas de transversalidade*** que a técnica consegue identificar. É importante notar que espalhamento e emaranhamento é uma classificação geral para diversos sintomas que podem ser apresentados por sementes.

Tipo de envolvimento do usuário Critério relativo ao ***esforço manual requerido*** para aplicar a técnica. Por exemplo, análise de sementes resultantes.

Sistema de maior porte usado para avaliação empírica Critério relativo ao tamanho do maior sistema de software ao qual a técnica foi aplicada experimentalmente. A precisão de uma técnica quando utilizada em sistemas de pequeno porte pode não se repetir em sistemas de grande porte (problema de escalabilidade). Isso ocorre devido a vários fatores como complexidade computacional da técnica e aumento exponencial do esforço manual requerido para a aplicação da mesma. Por isso, esse critério é importante.

Pré-condições Critério referente às ***premissas*** assumidas pela técnica em relação à implementação de interesses transversais. Por exemplo, convenções para a nomenclatura de métodos e classes.

3.3 Critérios de Análise

A partir de uma revisão sistemática da literatura realizada para este trabalho (Marçal et al., 2016), 25 técnicas para a identificação de interesses transversais foram selecionadas. As técnicas foram analisadas e divididas em grupos de técnicas que compartilham do mesmo método de análise para identificar interesses transversais. Os grupos foram nomeados como segue: **(1)** Análise de alterações concomitantes em código; **(2)** Análise de chamadas a métodos; **(3)** Análise de clustering; **(4)** Análise de grafos; **(5)** Análise de link; **(6)** Análise de rastros de execução; **(7)** Análise formal de conceitos; **(8)** Análise de clones; **(9)** Análise de padrões de linguagem; e **(10)** Análise histórica. A seguir é descrita as características das técnicas que compõem cada um dos grupos.

- **(1)** São técnicas que analisam dependências lógicas com base em padrões de alterações concomitantes em artefatos de software. Por exemplo, um conjunto de classes, métodos, ou pacotes que são alterados frequentemente em um mesmo *commit*.

- **(2)** Técnicas que procuram por interesses transversais por meio da análise de padrões de invocação a métodos. Em alguns casos, padrões de término de execução de métodos também são analisados. As técnicas desse grupo têm sua análise baseada apenas em código fonte e o que é possível extrair diretamente dele, por exemplo Árvores de Sintaxe Abstrata.

- **(3)** Técnicas que utilizam algoritmos de *clustering* para tentar agrupar componentes que implementam interesses transversais. Por exemplo, agrupar todos os métodos que implementam determinado interesse transversal em um mesmo *cluster*.

- **(4)** Técnicas que elaboram uma representação em grafo para o sistema e analisa as dependências de elementos de software utilizando o grafo gerado. Por exemplo, Grafos de Controle de Fluxo podem ser usados para analisar o comportamento do sistema de modo estático (sem precisar executar o programa).

- **(5)** Técnicas que utilizam a análise de *link* para analisar dependências implícitas e explícitas entre elementos do programa, na busca por interesses transversais.

- **(6)** Técnicas em que o sistema em análise é executado para cenários de caso de teste bem definidos. Os traços de execução são obtidos e analisados na busca por interesses transversais.

- **(7)** Técnicas que empregam a análise formal de conceitos utilizando componentes de software para localizar interesses transversais.

- **(8)** Técnicas que aplicam algoritmos de detecção de duplicação de código para localizar interesses transversais.

- **(9)** Técnicas que buscam por padrões de código referentes a convenções de nomenclatura utilizadas ao longo da codificação. Tais padrões podem indicar trechos de código relacionados/similares podem indicar a existência de interesses transversais no sistema.

- **(10)** Técnicas que analisam dados provenientes do repositório de software para analisar os aspectos evolutivos do sistema e identificar interesses transversais.

Os grupos de técnicas foram individualmente analisados e posteriormente comparados entre si. Assim, os critérios de análise foram selecionados com o objetivo de identificar e entender os fatores com maior potencial de impacto na escolha da técnica a ser aplicada para a identificação de interesses transversais. Os critérios são: tipo de dado que analisa, granularidade dos resultados, tipo de análise que executa, sintoma de transversalidade que identifica, e esforço manual requerido para aplicação. Para analisar os sintomas de transversalidade relacionados a cada grupo de técnicas, foram considerados apenas dois sintomas gerais: espalhamento e emaranhamento, para simplificar a análise. Os critério descritos foram obtidos a partir do trabalho elaborado por Kellens et al. (2007), porém foram empregados não para a comparação entre técnicas, mas para compreender os fatores de maior impacto nos resultados da aplicação de técnicas de identificação.

Para facilitar a compreensão, os critérios de análise são mostrados separadamente de acordo com o grupo de técnicas correspondente. Na Tabela 3.2 são apresentados os tipos de dados de entrada aceitos por cada grupo, na Tabela 3.3 são analisados de acordo com a granularidade de

resultados. Na Tabela 3.4 são mostrados os tipos de análises executadas, enquanto que na Tabela 3.5 é mostrada a granularidade dos resultados da análise. Por fim, na Tabela 3.6 é mostrado o tipo de esforço manual requerido. Na Tabela 3.1 são listadas as técnicas que implementam cada uma das abordagens de identificação consideradas para este trabalho.

Tabela 3.1: Técnicas de identificação selecionadas na amostragem teórica.

Técnica
Marin et al. (2004)
Tourwe e Mens (2004)
Bruntink et al. (2004)
Breu e Krinke (2004)
Tonella e Ceccato (2004)
Tourwe e Mens (2004)
Bruntink et al. (2004)
Breu (2005)
Bruntink et al. (2005)
Shepherd et al. (2005)
Canfora et al. (2006)
Krinke (2006)
Breu e Zimmermann (2006)
Qu e Liu (2007a)
Zhang e Jacobsen (2007)
Zhang et al. (2008)
Ishio et al. (2008)
Krinke (2008)
Cojocar e Czibula (2008)
Ceccato e Tonella (2009)
Maisikeli e Mitropoulos (2010)
Huang et al. (2010)
Qu et al. (2011)
Zhang e Jacobsen (2012)
Junior et al. (2012)
McFadden e Mitropoulos (2012)
Marcal et al. (2017)

Na Seção 3.3.1 é feita uma avaliação comparativa dos grupos de técnicas considerados neste trabalho.

3.3.1 Análise e Comparação de Técnicas

A partir dos dados reportados na tabela 3.4 observa-se que a maior parte das técnicas propostas na literatura realizam análises estática e estrutural de métodos, e requerem intervenção manual somente para a análise das sementes resultantes. Além disso, poucas técnicas executam análises a nível de linha de código ou classe. Observamos que isso acontece pois tentar identificar interesses transversais a nível de linha de código pode elevar o número de sementes resultantes e consequentemente o esforço manual requerido para validação de resultados. A

Tabela 3.2: Tipo de dado de entrada por abordagem adotada pelas técnicas atuais para a identificação de interesses transversais.

Grupo de técnicas	**Tipo de dado de entrada**	
	Estático	**Dinâmico**
Análise de alterações concomitantes em código	X	
Análise de chamadas a métodos	X	
Análise de clustering	X	
Análise de grafos	X	X
Análise de link	X	
Análise de rastros de execução		X
Análise formal de conceitos	X	X
Análise de clones	X	
Análise de padrões de linguagem	X	
Análise histórica	X	

Tabela 3.3: Granularidade de resultados da aplicação de técnicas atuais para a identificação de interesses transversais.

Grupo de técnicas	**Granularidade**			
	Fragmento de código	**Linha de código**	**Método**	**Classe**
Análise de alterações concomitantes em código		X		
Análise de chamadas a métodos			X	
Análise de clustering			X	
Análise de grafos	X		X	
Análise de link				X
Análise de rastros de execução				
Análise formal de conceitos			X	X
Análise de clones	X			
Análise de padrões de linguagem	X			
Análise histórica			X	

capacidade de uma técnica em indicar em um arquivo fonte a linha de código exata onde ocorre a transversalidade facilita a identificação e análise de sementes, porém tal benefício só é obtido caso a técnica apresente resultados muito precisos, o que é difícil, visto que as técnicas que fazem análise a nível de linha de código geralmente retornam muitas sementes falso-positivas.

O inverso é observado em técnicas com granularidade alta, como por exemplo as que resultam em classes que contém interesses transversais. Nesse caso, apesar de o número de sementes ser menor, o esforço de análise ainda é alto, pois é necessário analisar toda a extensão da classe para identificar os trechos de código que implementam interesses transversais. Apenas duas das técnicas consideradas para este trabalho têm granularidade a nível de classe como mostrado na Tabela 3.3.

Tabela 3.4: Tipo de análise executada nas abordagens adotadas pelas técnicas atuais para a identificação de interesses transversais.

Grupo de técnicas	**Tipo de análise**			
	Estrutural	**Comportamental**	**Léxica**	**Histórica**
Análise de alterações concomitantes em código	X			
Análise de chamadas a métodos	X	X		
Análise de clustering	X	X		
Análise de grafos	X	X	X	
Análise de link	X	X		
Análise de rastros de execução	X	X		
Análise formal de conceitos	X	X	X	
Análise de clones	X		X	
Análise de padrões de linguagem			X	
Análise histórica				X

Tabela 3.5: Sintomas de transversalidade analisados nas abordagens adotadas pelas técnicas atuais para a identificação de interesses transversais.

Grupos de técnicas	**Sintoma de Transversalidade**	
	Espalhamento	**Emaranhamento**
Análise de alterações concomitantes em código	X	
Análise de chamadas a métodos	X	
Análise de clustering	X	X
Análise de grafos	X	
Análise de link	X	
Análise de rastros de execução	X	X
Análise formal de conceitos	X	X
Análise de clones	X	
Análise de padrões de linguagem	X	
Análise histórica	X	

Os dados mostrados na Tabela 3.6 deixam claro que apesar de automatizar a busca por interesses transversais, nenhuma das técnicas analisadas é capaz de identificar interesses transversais sem o envolvimento humano. Todas requerem a análise manual das sementes resultantes, fato que está relacionado a três assertivas para toda técnica de identificação de interesses transversais: (1) O número de sementes resultantes deve ser limitado de modo que seja viável a análise manual das mesmas. (2) A limitação do número de sementes resultantes não deve prejudicar a precisão da técnica nem se contrapor aos objetivos da identificação. (3) A validação de resultados é subjetiva, pois depende do conhecimento e definição de transversalidade adotada pelo avaliador.

As assertivas (1) e (2) dão origem à tripla restrição *precisão* $vs.$ *número de sementes* $vs.$ *esforço manual* ilustrada na Figura 3.3. Qualquer perturbação em um dos fatores impacta diretamente nos demais, como descrito a seguir.

Tabela 3.6: Esforço manual requerido nas abordagens adotadas pelas técnicas atuais para a identificação de interesses transversais.

Grupo de técnicas	**Esforço Manual Exigido**
Análise de alterações concomitantes em código	Análise de sementes resultantes
Análise de chamadas a métodos	Análise de sementes e configuração de parâmetros de identificação (filtros e thresholds)
Análise de clustering	Análise de sementes resultantes
Análise de grafos	Análise de sementes resultantes
Análise de link	Análise de sementes resultantes
Análise de rastros de execução	Pré-processamento de dados e análise de sementes resultantes
Análise formal de conceitos	Pré-processamento de dados e análise de sementes resultantes
Análise de clones	Análise de sementes resultantes
Análise de padrões de linguagem	Análise de sementes resultantes
Análise histórica	Análise de sementes resultantes

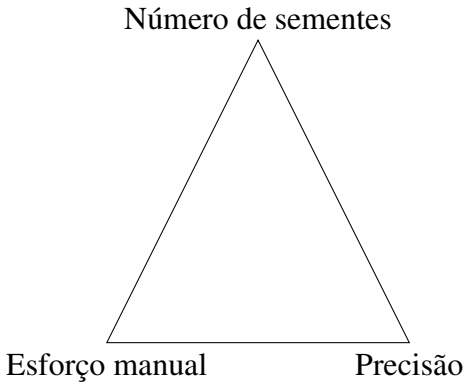

Figura 3.3: Tripla Restrição para Técnicas de Identificação de Interesses Transversais.

É imediato que se tomada uma técnica pouco precisa e que retorna poucas sementes, as chances de nenhuma delas fazer parte da implementação de interesses transversais é alta. No entanto, uma técnica pouco precisa que resulta em muitas sementes, talvez identifique interesses transversais, apesar de sua imprecisão. Nesse último caso, o esforço manual para descarte de falso-positivos nos resultados obtidos aumenta.

Para uma técnica precisa (por exemplo 95% de precisão), um número baixo de sementes resultantes significa que é possível identificar interesses transversais e o esforço manual para a análise de resultados é baixo. No entanto, apesar da precisão, a técnica pode não ser indicada para sistemas de grande porte que potencialmente têm um número elevado de interesses transversais. Resultados com poucas sementes podem refletir o fato de que a técnica é incapaz de identificar toda a extensão do código que implementa os vários interesses transversais em um sistema de grande porte. Nesse caso, atividades de refatoração e entendimento de código

requerem maior esforço por parte do desenvolvedor, que deve, a partir das poucas sementes reportadas, analisar todo o código fonte do programa para identificar toda a implementação de um interesse transversal.

Se uma técnica é precisa e é capaz de identificar muitas sementes, o esforço manual para análise dos resultados aumenta proporcionalmente, pois é preciso descartar falso-positivos, visto que apesar da precisão não há garantia de que todas as sementes reportadas realmente correspondem à implementação de interesses transversais. Ainda não foi reportada na literatura nenhuma técnica com precisão igual a 100% para qualquer cenário de aplicação.

Paralelo ao desafio imposto pela tríplice restrição à proposição e uso de técnicas para a identificação de interesses transversais, a subjetividade da validação manual das sementes resultantes influencia na precisão da técnica. Um exemplo de validação subjetiva pode ser observado em Breu e Zimmermann (2006) em que métodos resultantes do processo de identificação são validados como transversais se forem invocados de modo *semelhante* em diferentes módulos do sistema (mesmo contexto local e mesmos parâmetros de chamada) e se puderem ser implementados/refatorados para aspecto. Breu e Zimmermann (2006) explicam que durante a validação de sementes não é considerado se a refatoração do método para aspecto poderia ou não melhorar a qualidade de código, e em caso de dúvida quanto à possível implementação do método como aspecto a semente é classificada como falso-positivo.

A validação de sementes feita em Breu e Zimmermann (2006) é subjetiva pois (1) depende do conhecimento do avaliador acerca das possibilidades de implementação de interesses transversais como aspectos, e (2) depende da definição de similaridade contextual de chamada a métodos. É importante notar que a subjetividade dificulta a replicação de técnicas de identificação e a comparação de resultados, já que critérios de validação vagos como o usados em Breu e Zimmermann (2006) impedem a equivalência de análise de sementes em uma possível replicação.

A aplicação de técnicas dinâmicas em sistemas de software é limitada pelos trechos de código do sistema passíveis de execução em algum caso de teste. Por isso, como observado na Tabela 3.4, o tipo de análise executada por técnicas dinâmicas é classificado como comportamental. Geralmente, técnicas dinâmicas são executadas a nível de métodos e exigem a elaboração de casos de uso com alta abrangência, ou seja que permitam exercitar a maior quantidade de código possível. Como nem sempre é viável elaborar casos de uso para todos os cenários de execução possíveis, os resultados da análise dinâmica podem não incluir interesses transversais cuja implementação não fora executada. Além disso, a elaboração dos casos de uso anterior à aplicação da técnica é um esforço manual adicional. Assim observa-se que técnicas dinâmicas requerem mais trabalho manual do que determinadas técnicas estáticas como por exemplo as propostas em Bruntink et al. (2004), Canfora et al. (2006), Cojocar e Czibula (2008). Há poucos estudos que tentam combinar o uso de dados estáticos e dinâmicos em uma única técnica (Breu, 2005, Qu e Liu, 2007b, Krinke, 2008). Tal combinação tem por objetivo minimizar a limitação imposta pela dificuldade em exercitar todos os cenários de execução possíveis em técnicas

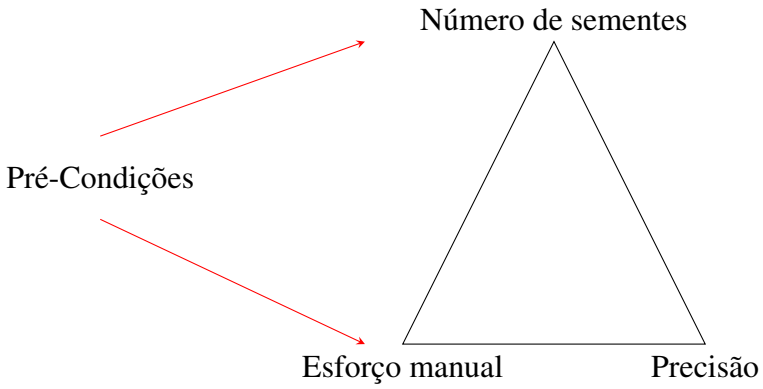

Figura 3.4: As pré-condições assumidas por técnicas de identificação podem afetar o número de sementes identificadas e o nível de esforço manual requerido. Indiretamente também afetam a precisão.

dinâmicas.

Técnicas baseadas em análise de *tokens* são aplicadas em dados estáticos e, geralmente, assumem que interesses transversais são implementados de modo específico e de acordo com convenções de codificação bem estabelecidas e aceitas pelos desenvolvedores.

Assumir que interesses transversais são implementados seguindo convenções de codificação específicas, ou que exibem características únicas e específicas pode afetar tanto a precisão quanto o número de sementes resultantes (Consequentemente afeta o esforço manual – Figura 3.4). O impacto negativo em potencial dessas pré-condições é minimizado proporcionalmente ao nível de conhecimento que o usuário da técnica tem sobre o código que implementa o sistema em análise. Além disso, Li e Prasad (2005) reportam que apesar de desenvolvedores entenderem a importância do uso de convenções de codificação, eles podem não segui-las, por exemplo quando o desenvolvimento precisa ser acelerado.

3.3.2 Combinação de Técnicas

Marin et al. (2006) propuseram um *framework* para a combinação de diferentes técnicas de identificação de interesses transversais com o objetivo de melhorar a precisão dos resultados obtidos. Observa-se em Marin et al. (2006) que a precisão e o número de sementes relevantes identificadas podem ser melhorados com a combinação de técnicas baseadas em diferentes abordagens de identificação.

O tipo de combinação mais simples é a aplicação de duas técnicas distintas e a análise da intersecção dos resultados obtidos. No entanto, para isso é necessário haver compatibilidade de granularidade entre as técnicas. Além disso, elas devem ser dedicadas à identificação dos mesmos tipos de interesses transversais. Isso porque caso as técnicas combinadas identifiquem tipos distintos de interesses transversais, os resultados formarão conjuntos de sementes disjuntos e a intersecção de resultados será vazia. Esse tipo de combinação melhora a precisão dos resultados desde que as técnicas escolhidas sejam compatíveis.

Outro modo comum de fazer a combinação entre técnicas é tomar a união dos resultados

das mesmas. Esse tipo de combinação não requer a compatibilidade entre os tipos de interesses transversais identificáveis como é exigido na combinação pela tomada da intersecção de resultados. Além disso, o número de sementes resultantes pode ser maior. Isso significa que a combinação de técnicas pela união de sementes resultantes é uma boa opção para a identificação da maior quantidade possível de código que implementa interesses transversais. É importante notar que esse aumento no número de sementes resultantes altera o equilíbrio da tripla restrição ilustrada na Figura 3.3.

A combinação entre técnicas de identificação de interesses transversais é difícil principalmente devido às características específicas do processo de identificação de cada técnica, e também pela granularidade a qual tal processo é aplicado. Técnicas de diferente granularidade são difíceis de combinar. Para auxiliar no processo de combinação, em (Marcal et al., 2017) nós propomos um *framework* para a combinação de técnicas de identificação em diferentes níveis de granularidade (Marcal et al., 2017). Tal *framework* consiste no uso de técnicas com granularidade alta para selecionar os dados a serem analisados pela técnica com granularidade baixa. Experimentos executados constataram melhora significante na relevância dos resultados obtidos pelas técnicas combinadas.

3.4 Considerações finais

Neste capítulo foi demonstrado que as técnicas para a identificação de interesses transversais propostas na literatura adotam abordagens de identificação diferentes. Mesmo técnicas pertencentes a um mesmo grupo podem adotar algoritmos de identificação distintos. Por exemplo, diferentes algoritmos de *clustering* ou algoritmos para detecção de duplicação de código.

Toda técnica de identificação assume premissas sobre o modo como interesses transversais são implementados no sistema, e isso pode afetar a precisão dos resultados obtidos. No entanto, umas vez conhecidas pelo analista, é possível utilizar as premissas adotadas por cada técnica para direcionar sua aplicação à identificação de tipos específicos de interesses transversais, e assim obter resultados mais relevantes.

A combinação de técnicas de identificação é difícil devido às especificidades de cada uma, como granularidade e métodos de busca por interesses transversais. Há diferentes maneiras de combinar técnicas de identificação, por exemplo tomando a intersecção ou união dos resultados de duas ou mais técnicas.

A análise descritiva qualitativa baseada em critérios específicos das técnicas para a identificação de interesses transversais propostas na literatura ajudou a identificar três parâmetros que têm grande impacto na aplicação e resultados de uma técnica. Tais parâmetros são: (1) tipo de dado de entrada, (2) tipo de análise, e (3) granularidade. Observou-se que o tipo de dado de entrada depende do sistema em análise, enquanto que o tipo de análise depende do tipo de dado selecionado. A granularidade influencia no esforço manual para a aplicação das técnicas de identificação e tem impacto direto sobre o número de sementes resultantes.

Para aprofundar a análise sobre os parâmetros identificados foi realizado um estudo qualita-

tivo dos mesmos baseado na *Teoria Fundamentada em Dados*. A metodologia adotada permitiu confirmar e entender melhor os parâmetros preliminarmente observados, além de estendê-los para que fosse possível elaborar as diretrizes propostas neste trabalho. A metodologia adotada e a aplicação da *Teoria Fundamentada em Dados* são descritas no Capítulo 4.

CAPÍTULO 4

Metodologia

A proposta deste trabalho é resultado direto da análise qualitativa de dados sobre técnicas para a identificação de interesses transversais. A Teoria Fundamentada em Dados (TDF) (Strauss e Corbin, 1990, Walker e Myrick, 2006, Rondon e Pastor, 2007) foi adotada como metodologia de pesquisa para auxiliar na explicação dos fatos observados durante a investigação de conceitos relacionados à identificação semi-automática de interesses transversais. A análise qualitativa dos dados extraídos da literatura por meio do uso da TDF deu origem às diretrizes propostas neste trabalho. Para explicar melhor a metodologia de pesquisa utilizada, a seguir são descritos os principais conceitos relacionados à TFD e como ela é utilizada. As ameaças da TFD à validade do estudo elaborado também é discutido a seguir.

4.1 Conceitos da Teoria Fundamentada em Dados

A Teoria Fundamentada em Dados é um método de pesquisa qualitativa originalmente proposto por dois sociologistas, *Barney Glaser* e *Anselm Strauss*, como alternativa aos métodos de pesquisa tradicionais constituídos por formulação de hipótese, técnicas de verificação e análises quantitativas. A TFD tem por objetivo permitir a formulação de teorias a partir da análise criteriosa de dados que são organizados em categorias. Tais categorias são obtidas por meio da análise do domínio do problema abordado pela pesquisa (Willig, 2013).

Para identificar, refinar, integrar, e por fim formular uma teoria, são usadas estratégias como análise comparativa constante, amostragem teórica e codificação. A seguir são descritas as principais estruturas analíticas usadas no método da TFD de acordo com Willig (2013).

4.1.1 Categorias

Categorias designam um conjunto de instâncias que compartilham determinadas características e que descrevem suas instâncias em baixo nível de abstração. Assim, categorias podem ser

também chamadas de rótulos, conceitos, ou categorias descritivas (Strauss e Corbin, 1990). Por exemplo, C, Java, e Pascal são instâncias que podem ser agrupadas na categoria "Linguagem de Programação". Com o progresso da pesquisa é possível identificar categorias em níveis mais elevados de abstração. Quando o nível de abstração de uma categoria aumenta, ela deixa de ser descritiva e passa a ser analítica, pois refere-se à interpretação de suas instâncias e não mais à descrição de características comuns de suas instâncias.

Tanto categorias analíticas quanto descritivas são baseadas na identificação de relações de similaridade e diferenças entre os dados analisados (Dey, 1999), no entanto enquadram-se em diferentes níveis de abstração.

4.1.2 Codificação

O processo de identificação de categorias é chamado de codificação e é realizado durante a análise dos dados coletados para a pesquisa. Inicialmente são identificadas categorias descritivas e, posteriormente, analíticas. Categorias analíticas são utilizadas para integrar categorias descritivas. Tal análise caracteriza o processo de comparação constante de dados (Strauss e Corbin, 1990).

A codificação ocorre em três estágios (Strauss e Corbin, 1990). **(1)** Inicialmente é realizada a ***codificação aberta*** em que os dados são analisados e categorias descritivas iniciais são identificadas. Aqui as categorias refletem características gerais do problema investigado.

Após a codificação aberta, é realizada a **(2)** *codificação axial*, em que os dados já categorizados são analisados novamente (processo de comparação constante) para que as categorias já definidas sejam refinadas, se necessário subdivididas em novas categorias, ou reagrupadas. O principal objetivo da codificação axial é tornar explícitos os relacionamentos existentes entre categorias e subcategorias.

A partir do refinamento das categorias por meio da codificação axial, é realizada a ***codificação seletiva***. Essa codificação envolve a seleção e identificação de uma categoria principal capaz de relacionar todas as demais categorias. O processo de codificação seletiva envolve a validação e refinamento dos relacionamentos identificados na codificação axial para que seja possível integrar as categorias identificadas e formular a teoria final. Neste trabalho a codificação seletiva não é realizada, pois não há uma única categoria (mas sim um conjunto) central que possa ser considerada como o conceito fundamental para a formulação das diretrizes propostas.

4.1.3 Comparação Constante

A comparação constante das informações obtidas a partir da análise de dados novos e já categorizados garante que o processo de codificação seja baseado na identificação sistemática de similaridades e diferenças entre as categorias. Após a identificação de características comuns entre as instâncias de uma mesma categoria, é preciso observar as diferenças entre essas instâncias para que seja possível identificar subcategorias. A categoria "Linguagem de Programação" exemplificada anteriormente pode ser usada para ilustrar o processo de comparação constante.

C#, Java e Pascal têm como característica em comum o fato de serem linguagens de programação. A análise das diferenças entre tais instâncias resulta na classificação de C# e Java como linguagens de programação orientadas a objetos e pascal como linguagem estruturada. Assim, a partir da comparação das instâncias da categoria "Linguagens de Programação" emergem duas subcategorias "linguagens de programação orientadas a objetos" e "linguagens de programação estruturadas".

4.1.4 Amostragem e Saturação Teórica

A amostragem teórica é iniciada com a coleta, análise, e codificação aberta de dados, que dão origem às primeiras categorias descritivas. Após a identificação das categorias iniciais, mais dados são coletados e analisados. Por meio dos processos de codificação e comparação constante as categorias são expandidas e subdivididas a partir dos novos dados até atingir o ponto de saturação teórica. Idealmente, o processo de coleta de dados (amostragem teórica) continua até que a saturação teórica seja atingida. Ou seja, a coleta e codificação de dados devem ser realizadas até que não seja mais possível identificar novas categorias e até que não ocorra mais o surgimento de novas categorias a partir da comparação constante das instâncias em categorias existentes.

4.1.5 O Processo de Análise de Dados

Ao longo do processo de coleta da amostragem teórica, é feita a comparação e codificação constante de dados. Além disso, é mantido um registro textual do processo de desenvolvimento da teoria. Isso significa que todo o processo de identificação das categorias é registrado para que seja possível justificar o nome dado às categorias e para permitir a rastreabilidade entre categorias e seus relacionamentos. O registro também permite identificar alterações nos critérios e perspectivas de análise dos dados coletados. Os registros feitos podem ser longos ou curtos, abstratos ou concretos.

O processo de análise de dados proposto pela TFD é de natureza indutiva, e não baseia-se na confirmação ou refutação de uma hipótese pré-estabelecida (Rodrigues et al., 2004). Sendo assim, a teoria resultante é estruturada de acordo com aquilo que é descoberto ao longo da investigação sobre os dados obtidos na amostragem teórica. Para reduzir a subjetividade e garantir a validade do método é preciso aplicar a TFD de modo objetivo. A objetividade também garante a possibilidade de replicação do estudo e permite que outros pesquisadores reavaliem a pesquisa e demonstrem que os dados coletados são claros em significado e alheios ao viés do pesquisador (Yamaguchi, 2010, Rodrigues et al., 2004).

O problema da saturação teórica:

O ponto de saturação teórica não é claramente definido na literatura a respeito da TFD, o que dificulta a definição de um ponto de parada para a coleta de dados. É preciso estar certo de que novos dados do domínio do problema não mais contribuem para a pesquisa. Além disso, o uso

da TFD pode invalidar o estudo caso não seja adotado o rigor necessário para análise e coleta de dados (Yamaguchi, 2010). Neste trabalho, tentamos minimizar o problema da saturação teórica coletando a amostragem teórica por meio de uma Revisão Sistemática da Literatura (SLR) que segue um protocolo formal e bem definido para a busca e análise de dados na literatura (Marçal et al., 2016, Biolchini et al., 2005).

A identificação de categorias e características:

Para identificar de maneira significante as categorias (descritivas/analíticas) e características gerais do problema investigado, há dois métodos de codificação: codificação por *micro-análise* e codificação por *pontos-chave*.

Codificação por Micro-Análise:

A codificação por micro-análise pode ser utilizada tanto no estágio de codificação aberta quanto axial para ajudar na análise dos dados coletados. Esse tipo de codificação consiste na fragmentação dos textos obtidos na literatura palavra por palavra, linha por linha, ou página por página. Para cada fragmento é atribuído um significado/código, que com o avanço da análise se transforma em uma categoria (Yamaguchi, 2010, Allan, 2003). Desse modo, para executar a codificação por micro-análise é necessário percorrer fragmento por fragmento de texto para identificar informações relevantes, o que faz com que esse seja um processo demorado e inviável para amostragens teóricas volumosas como a deste trabalho, por exemplo. Além disso, a análise de fragmentos de texto isolados, fora de seu contexto, pode levar a interpretações diferentes de seu significado real. Um exemplo de codificação por micro-análise é mostrado na Tabela 4.1.

Tabela 4.1: Exemplo de codificação aberta utilizando o método de micro-análise - Adaptado de Allan (2003)

Texto	**Código**
Do meu ponto de vista	ponto de vista pessoal
o maior desafio é	afirmação
em mudanças na tecnologia	mudanças na tecnologia
ou melhoria do produto	mudanças no produto
você	alteração de pronome
nunca pode garantir que	afirmação de incerteza

Codificação por Pontos-Chave:

Na codificação por pontos-chave são analisados os conceitos relacionados a cada fragmento de texto e não o significado. Neste trabalho, utilizamos a codificação por pontos-chave, de modo que todas as características de técnicas de identificação consideradas originam diretamente dos dados analisados na literatura. Posteriormente, tais características são organizadas em categorias e a relação entre as categorias identificadas dão suporte à elaboração da das diretrizes propostas (Yamaguchi, 2010). Um exemplo de codificação pelo método de pontos-chave

é mostrado na Tabela 4.2.

Tabela 4.2: Exemplo de codificação aberta utilizando o método de ponto-chave - Adaptado de Allan (2003)

Ponto-chave	Código
O agendamento de alterações é considerado essencial e parte integral do processo de software	alterações e processo de software
Problemas no gerenciamento de configuração podem ser resolvidos envolvendo pessoas em discussões	problemas de comunicação
Software é controlado em pré-produção e produção usando o gerenciamento de configuração	controle de software
Empresa X precisava desenvolver um sistema de controle de versão para sistemas de software. [Comentário: Isso implica que não há um sistema de controle de versão adequado disponível no mercado.]	ferramentas complexas e sistemas de controle de versão

4.2 Aplicação da Teoria Fundamentada em Dados Neste Trabalho

A amostragem teórica para este trabalho foi obtida por meio de uma Revisão Sistemática da Literatura (Marçal et al., 2016) e seguiu um protocolo formal para a coleta e análise de dados a partir da literatura. Os resultados obtidos na revisão sistemática serviram de fonte de dados. Todo o conjunto de publicações obtido a partir da revisão sistemática realizada serviu como base para a identificação de características, que foram classificadas em categorias. Posteriormente, as categorias identificadas tornaram-se parâmetros a serem considerados na escolha da técnica a ser utilizada na identificação de interesses transversais. As categorias identificadas e suas características influenciaram diretamente na elaboração das diretrizes propostas.

4.2.1 Amostragem Teórica

Para obter a amostragem teórica necessária para a elaboração das diretrizes propostas, a busca por estudos relacionados à identificação de interesses transversais foi definida e planejada em termos de *escopo de busca*, *métodos de busca* e *termo de busca*.

Escopo: Para garantir a relevância dos estudos obtidos, a busca foi limitada por período de publicação. Assim, foram considerados todos os estudos publicados entre os anos 2000 e 2016[1], inclusive.

Método: A busca por estudos relevantes foi feita de modo automático, ou seja, o termo de busca foi simplesmente executado nas bases de dados escolhidas. As seguintes bases de dados

[1] A amostragem teórica foi refinada e incrementada ao longo do tempo de elaboração desta dissertação, por isso o período de busca por estudos é até o ano de 2016

foram selecionadas:

ACM Digital Library IEEE Xplore Science Direct Elsevier SpringLink Willey Inter Science Journal Finder

Termo de busca: O seguinte termo de busca foi utilizado nas bases de dados escolhidas:

((crosscutting OR crosscut OR cross-cutting) AND (concern OR concerns)) AND (("aspect mining") OR ("separation of concerns") OR ("code mining") OR "coding mining")

Diferentes bases de dados utilizam sintaxes de busca diferentes. Assim, ao realizar a busca, é criada uma string de busca semanticamente equivalente à apresentada para cada base de dados. A string de busca poderia ter sido refinada de modo a retornar resultados mais relevantes em relação ao domínio sobre técnicas de identificação de interesses transversais. No entanto, optou-se por permitir o retorno de resultados mais amplos, e por vezes relacionados a áreas fracamente correlatas. Isso possibilitou obter informações não somente sobre técnicas de identificação e interesses transversais, mas também sobre como a área de estudo foco deste trabalho se relaciona com outras áreas do conhecimento. Por exemplo, com a área de recuperação da informação e mineração de dados.

Critérios de Seleção Do conjunto de estudos retornados pela busca, foram incluídos na amostragem teórica apenas aqueles com algum resultado experimental. Tendo sido excluídos artigos referentes a:

- Early Aspects;
- Técnicas não automáticas;
- Editoriais, revisões, tutoriais, sumários e painéis.

Para autores que aplicaram a mesma técnica em duas publicações diferentes, foram consideradas as melhorias feitas na abordagem proposta e ambos os estudos são incluídos na amostragem.

Controle de Qualidade da Amostragem Teórica Para cada estudo selecionado foi aplicado um *formulário de qualidade*. Na Tabela 4.4 são listadas as questões que foram respondidas durante a análise dos estudos obtidos para avaliar sua qualidade. As questões para avaliação de qualidade são adaptadas de Dyba e Dingsayr (2008). No entanto, ao invés de usar uma escala de dois pontos como a adotada pelos autores, foi utilizada uma escala de três pontos, como

a empregada por Ali et al. (2010). Assim, foi possível pontuar estudos em que há descrições limitadas da metodologia utilizada e dos resultados obtidos. A escala utilizada é mostrada na Tabela 4.3 e permite analisar se o estudo *atende*, *atende parcialmente* ou *não atende* ao critério definido por cada questão da Tabebla 4.4.

Tabela 4.3: Escala utilizada para avaliação dos estudos selecionados

Ponto	Significado
0	Não atende
0,5	Atende parcialmente
1	Atende completamente

Tabela 4.4: Questões de avaliação de qualidade dos estudos selecionados

Nº	Questão
Q1	O estudo é baseado em alguma metodologia de pesquisa bem definida?
Q2	O principal foco do estudo é a identificação/compreensão de interesses transversais?
Q3	O estudo apresenta informações experimentais ou aplicações reais do objeto de estudo?
Q4	Há uma descrição satisfatória do contexto no qual o estudo está inserido?
Q5	Está claro como os resultados foram extraídos?
Q6	Os métodos de extração dos resultados foram bem justificados?
Q7	Os métodos de extração dos resultados foram bem detalhados?
Q8	Houve uma descrição clara do processo de análise dos resultados?
Q9	Os resultados foram bem justificados?
Q10	Informações contraditórias foram consideradas para a avaliação dos resultados?
Q11	Foram utilizadas métricas apropriadas para a avaliação dos resultados?
Q12	Os pesquisadores examinam criticamente a própria influência sobre os resultados obtidos durante a coleta e análise dos dados?
Q13	Os resultados são claramente descritos?
Q14	Os pesquisadores discutiram a credibilidade dos resultados da pesquisa?
Q15	As limitações/desvantagens da abordagem são discutidas?
Q16	Há uma discussão adequada dos resultados tanto positivos quanto negativos do estudo?
Q17	A conclusão exposta pelos pesquisadores é justificada pelos resultados?
Q18	Os pesquisadores discutem a contribuição do estudo feito para a engenharia de software e a identificação e compreensão de interesses transversais?

As 18 questões elaboradas correspondem a quatro critérios de análise que foram considerados importantes para garantir a qualidade da amostragem teórica:

- Filtro inicial: As questões de 1 a 3 avaliam se o estudo deve ser mantido para a análise. Uma pontuação é atribuída a cada questão. A soma dos pontos obtidos classifica o estudo de acordo com a Tabela 4.5. Estudos com pontuação maior ou igual a 2 continuaram a ser analisados e passaram pela avaliação de *Rigor*, *Credibilidade* e *Relevância*.

- Rigor: As questões de 4 a 12 são referentes ao rigor com o qual o estudo foi conduzido. Os pontos atribuídos para cada questão foram/ somados. A soma dos pontos obtidos classificou o rigor do estudo de acordo com a Tabela 4.5.
- Credibilidade: As questões 13 a 17 são referentes à credibilidade do estudo. Os pontos atribuídos para cada questão foram somados. A soma dos pontos para cada questão classifica a credibilidade do estudo de acordo com a Tabela 4.5.
- Relevância: A questão 18 é referente a relevância do estudo para a Engenharia de Software e a pesquisa na área de identificação de interesses transversais. A relevância do estudo foi classificada de acordo com a Tabela 4.5.

Tabela 4.5: Escala utilizada para avaliação dos estudos selecionados

Pontuação	Classificação
Filtro Inicial	
0 - 0,5	Pobre
1 - 1,5	Mediano
2 - 3	Bom
Rigor	
0 - 5	Pobre
5,5 - 7,5	Mediano
8 - 9	Bom
Credibilidade	
0 - 1,5	Pobre
2 - 3	Mediano
3,5 - 5	Bom
Relevância	
0	Pobre
0,5	Mediano
1	Bom
Classificação final	
0 - 10	Pobre
10,5 - 13	Mediano
13,5 - 15,5	Bom
16 - 18	Muito Bom

Para a classificação final do estudo em análise, foi feita a somatória de todos os pontos referentes aos filtros *inicial*, *rigor*, *credibilidade* e *relevância*. A soma obtida classificou o estudo de acordo com a Tabela 4.5. Somente os estudos classificados como *Bom* ou *Muito Bom* na classificação final são incluídos na amostragem teórica.

Tamanho da Amostragem Inicialmente foram obtidos 1289 estudos dos quais 1068 foram excluídos pelo título, pois claramente não tinham relação direta com a proposição ou avaliação de abordagens para a identificação de interesses transversais. Esses estudos foram armazenados e consultados sempre que necessário para obter informações sobre a relação entre interesses

transversais e outras áreas de estudo relacionadas à Engenharia de Software. Outros 169 estudos foram excluídos após a leitura, por não apresentarem informações corretas ou por apresentarem informações incompletas que não adicionariam fatos interessantes ao objetivo deste trabalho. Os 52 estudos que restaram foram avaliados de acordo com os critérios de qualidade estabelecidos. Após a avaliação de qualidade, restaram 25 estudos classificados como *Bom* ou *Muito Bom*. A quantidade de estudos excluídos por cada filtro aplicado é mostrada na Tabela 4.6, enquanto que o número de estudos obtidos de acordo com a classificação de qualidade é mostrado na Tabela 4.7.

Tabela 4.6: Quantidade de estudos excluídos por cada filtro aplicado

Excluído por	Nº de Estudos
Título	1068
Após a leitura	169
Após a avaliação de qualidade	27
Total	1264

Tabela 4.7: Número de estudos incluídos na amostragem teórica após a filtragem de acordo com a classificação.

Classificação	Nº de Estudos
Pobre	10
Mediano	17
Bom	15
Muito bom	10
Total	52

4.2.2 Codificação

Cada um dos estudos da amostragem teórica passou pelos processos de codificação aberta e axial. A codificação aberta foi executada iterativamente, o que significa que a categorização de pontos-chave foi feita paralelamente à leitura e análise do texto e foi repetida várias vezes com o propósito de refinamento. Foram extraídos somente trechos que foram julgados como uma descrição importante para compreender os fatores que influenciam o uso, elaboração, e escolha de técnicas para a identificação de interesses transversais. Também foram categorizados trechos de texto relacionados a características de transversalidade. Parte da codificação aberta feita sobre o artigo em Tonella e Ceccato (2004) é mostrada na Tabela 4.8.

Em seguida foi executada a codificação axial, em que as categorias e suas características já analisadas são revistas para possibilitar o refinamento das mesmas. Nós definimos como objetivo para a codificação axial reduzir o número de categorias e aumentar a relevância das mesmas e de suas características em relação a fatores de impacto na escolha de técnicas de identificação de interesses transversais. Com isso, ao final da codificação axial foi possível obter um conjunto de características que serviram de parâmetro para a elaboração das diretrizes

propostas. As categorias e características finais obtidas após a codificação axial são mostradas na Tabela 4.9.

Tabela 4.8: Codificação aberta utilizando o método de ponto-chave

Ponto-chave/característica	Categoria
"(...) we do not rely on naming/coding coventions, thus requiering minimal knowledge about the system under analysis (it is sufficient to define the use-cases for the main functionalities)" (Tonella e Ceccato, 2004)	Critérios para a identificação de interesses transversais
"(...) The results we obtained are very encouraging, although their generalization to larger programs is hard to make. The granularity of the use-cases might affect the quality of the resulting concept lattice, possibly resulting in false-positives (too fine grained) or false-negatives (too high-level functionalities)" (Tonella e Ceccato, 2004)	Granularidade e aplicação da técnica em sistemas de grande porte
"Execution traces are obtained by running an instrumented version of the program under analysis for a set of scenarios (use cases). The relationship between execution traces and executed computational units is subjected to concept analysis." (Tonella e Ceccato, 2004)	Tipo de dado utilizado para análise
"(...) We implemented our own tool, Dynamo, to trace method executions of Java programs and we used ToscanaJ for concept lattice construction and visualization. The tracing tool is a modification of the Java compiler javac, developed by Sun Microsystems, version 1.4.0. It includes a facility to enable and disable tracing during execution" (Tonella e Ceccato, 2004)	Ferramentas para a identificação de interesses transversais
"(...) Interpretation of the concept lattice for migration to AOP is instead a humanintensive activity." (Tonella e Ceccato, 2004)	Esforço manual
"(...) However, it is interesting to note that we had no previous knowledge of our case study" (Tonella e Ceccato, 2004)	Conhecimento necessário sobre o sistema em análise
"(...) In this paper, we propose to use dynamic analysis in order to exercise the computational units involved in the main application functionalities" (Tonella e Ceccato, 2004)	Tipo de análise realizada sobre os dados obtidos.

4.3 Considerações Finais

A Teoria Fundamentada em Dados foi considerada adequada como metodologia de pesquisa para este trabalho por ser indutiva, e possibilitar a validação de análises baseadas em dados que são constantemente comparados e categorizados para facilitar a formulação de uma teoria final. No caso deste trabalho, a formulação das diretrizes propostas.

De acordo com o processo definido pela TFD, foi extraída, inicialmente, uma amostragem de estudos documentados na literatura sobre a identificação e entendimento de interesses transversais em sistemas de software. A execução de uma revisão sistemática da literatura

Tabela 4.9: Resultado da codificação axial

Ponto-chave/característica	Categoria
As técnicas para a identificação de interesses transversais analisam dados de dois tipos diferentes: estáticos e dinâmicos	Tipo de dado de entrada
As técnicas para a identificação de interesses transversais são capazes de executar três tipos de análise diferentes: sintática, estrutural, e comportamental	Tipo de análise
Os dados que são extraídos e analisados pelas técnicas de identificação podem exibir diferentes níveis de granularidade. Por exemplo, classe, método, linha de código, e fragmento de código	Granularidade
Toda técnica para a identificação de interesses transversais requer algum tipo de envolvimento manual do analista	Esforço manual

mostrou-se adequada para este propósito pelo rigor que exige tanto para coleta quanto para análise e validação de dados.

O processo de codificação por pontos-chave foi escolhido por sua viabilidade e por possibilitar a identificação dos fatores de maior impacto na escolha de técnicas de identificação de interesses transversais. A codificação axial, por sua vez, possibilitou simplificar e refinar as categorias e características obtidas pela codificação por pontos-chave. Por meio da codificação axial, os pontos-chave selecionados na fase anterior da codificação foram agrupados nas seguintes categorias: **(1)** Tipo de dado de entrada, **(2)** Tipo de análise, **(3)** Granularidade, e **(4)** Esforço manual.

As quatro categorias identificadas e suas características foram analisadas em relação aos grupos de técnicas de identificação definidos no Capítulo 3. Tal análise permitiu a elaboração das diretrizes para a escolha de técnicas de identificação de interesses transversais descritas e discutidas no Capítulo 5;

CAPÍTULO 5

Diretrizes para a Escolha de Técnicas de Identificação

Mens et al. (2008) afirmam que um dos problemas da área de pesquisa em identificação de interesses transversais é que as técnicas atuais são de propósito geral, e que a maioria delas não são dedicadas à busca de instâncias de um tipo específico de interesse transversal. No entanto, ao longo deste trabalho foi possível observar que o problema indicado não está relacionado às técnicas, mas sim ao modo como elas são aplicadas. Ou seja, as técnicas propostas na literatura são especialistas na identificação de determinadas características que correspondem ao modo como sintomas de transversalidade aparecem no programa. Mesmo sendo especialistas na identificação de sintomas específicos, tais técnicas são aplicadas a um contexto geral, na esperança de que seja possível identificar qualquer sintoma. Não há na literatura um guia que possa servir de apoio para o analista/usuário escolher qual técnica de identificação aplicar de modo que seja mantida a consistência entre os objetivos da identificação e as capacidades da técnica.

Kellens et al. (2007) propuseram um conjunto de critérios de comparação e análise de técnicas de identificação de interesses transversais. A partir dos critérios estabelecidos, foi elaborada uma taxonomia que objetiva a ajudar a compreender as características específicas de técnicas propostas na literatura. No entanto, não há preocupação em descrever como a escolha de uma técnica afeta os resultados da identificação ou em como manter a compatibilidade entre aquilo que a técnica é capaz de identificar e o que se deseja identificar.

Marin et al. (2006) propõem um *framework* para facilitar a aplicação sistemática de técnicas de identificação. No entanto, o objetivo para o uso do *framework* é auxiliar na descrição e análise dos resultados da identificação. Logo, observa-se que nenhum dos trabalhos (Kellens et al., 2007, Marin et al., 2006) preocupa-se com a compatibilidade da técnica a ser aplicada

aos objetivos da identificação. Sendo assim, neste capítulo são apresentadas diretrizes para a escolha de técnicas de identificação. Espera-se que o uso de tais diretrizes auxilie na garantia de compatibilidade entre a especialidade de uma técnica e os sintomas de transversalidade que se deseja identificar.

5.1 Contextualização

A escolha apropriada da técnica de identificação tem potencial para melhorar a significância dos resultados obtidos e minimizar o esforço manual necessário para validar sementes. A proposta deste trabalho representa um ponto de partida para discutir melhores maneiras para aplicar as técnicas de identificação propostas na literatura. Logo, não temos a intenção de fornecer um conjunto definitivo de diretrizes capaz de solucionar de uma vez por todas os problemas relacionados à identificação de interesses transversais. Além disso, tais diretrizes devem evoluir conforme a área de pesquisa em identificação de interesses transversais evolui.

É importante notar também que as diretrizes propostas não garantem, por si só, a melhora na precisão dos resultados de técnicas de identificação, mas acreditamos que elas ajudam a direcionar sua escolha e aplicação de modo que os resultados obtidos sejam mais relevantes.

Este capítulo é organizado da seguinte maneira. Primeiro são apresentadas as diretrizes propostas, em seguida é feita uma discussão sobre elas e sobre o que abordam. A discussão apresentada faz um paralelo entre características de técnicas de identificação, sintomas de transversalidade e objetivos de identificação. A seguir são descritos os relacionamentos e dependências entre as diretrizes propostas. Por último é explicado como utilizá-las por meio de um exemplo de aplicação.

5.2 Diretrizes

Nesta seção são apresentadas as diretrizes propostas neste trabalho. Elas foram elaboradas a partir da análise sistemática das características das técnicas de identificação propostas na literatura descritas no Capítulo 3, e dos fatores que influenciam a aplicação e os resultados das mesmas na identificação de interesses transversais discutidos na Seção 3.3.1. Inicialmente, as diretrizes são apresentadas de maneira direta. Em seguida é feita uma discussão sobre o que é abordado em cada diretriz e sua influência no processo de identificação de interesses transversais. Também é descrito como as diretrizes podem ser usadas como apoio no processo de escolha da técnica de identificação a ser utilizada.

5.2.1 *[D1] - Defina o que deve ser identificado*

Defina o tipo de interesse transversal a ser identificado. A escolha de um tipo permite a descrição de propriedades específicas do mesmo, que auxiliam a reconhecê-lo no sistema a

ser analisado. Para definir um tipo pode-se utilizar as classificações de tipo documentadas na literatura (Marin et al., 2005, Figueiredo et al., 2009) e descritas no Capítulo 3.

É importante notar que as classificações propostas têm níveis de abstração diferentes, enquanto Marin et al. (2005) categoriza sintomas que podem ser diretamente mapeados para padrões de linguagem utilizados em código orientado a objetos (mais especificamente Java), Figueiredo et al. (2009) definem categorias abstratas e independentes de linguagem ou paradigma de programação. A classificação de tipos proposta em Figueiredo et al. (2009) impõe um esforço de mapeamento maior entre sintoma e sua representação no código fonte devido ao nível de abstração considerado.

5.2.2 *[D2] - Estabeleça o tipo de dado a ser analisado*

Para analisar um sistema na busca de interesses transversais é necessário extrair dados do mesmo. A extração de dados do sistema pode ser feita diretamente no código fonte (Dados *estáticos*), por meio da execução de cenários de caso de teste que exercitam um conjunto de funcionalidades do sistema *(Dados dinâmicos*), ou a partir do histórico de desenvolvimento (*Dados históricos*).

A escolha do tipo de dado deve considerar sua disponibilidade. Ou seja, se é ou não possível extrair o tipo de dado desejado a partir do sistema. Assim, antes de estabelecer quais dados serão usados deve-se investigar o sistema com base em perguntas simples: **(1)** "*O sistema é mantido em um repositório de software?*", **(2)** "*É possível elaborar um conjunto significativo de cenários de casos de uso para exercitar as principais funcionalidades do sistema?*", **(3)** "*O código fonte do sistema está disponível para análise?*". Tais perguntas ajudam a descobrir se o tipo de dado desejado pode ou não ser extraído do sistema. Se as respostas para **(1)**, **(2)**, e **(3)** forem afirmativas, por exemplo, significa que é possível extrair, respectivamente, dados históricos, dinâmicos, e estáticos do sistema em análise.

5.2.3 *[D3] - Defina o tipo de análise*

A análise executada para a identificação de interesses transversais pode ser feita de quatro maneiras. **(1)** A análise estrutural, explora como artefatos são decompostos em hierarquias e as dependências existentes entre eles. Por exemplo, como um conjunto de classes compõem um pacote e quais as relações de dependência entre classes. **(2)** A análise comportamental investiga o uso de mecanismos internos e chamadas a APIs que ocorrem durante a execução do sistema (Reimanis e Izurieta, 2016). **(3)** A análise sintática explora elementos extraídos diretamente do código fonte. Por exemplo, *tokens* e assinaturas de métodos, para reconhecer padrões de linguagem. **(4)** A análise histórica investiga repositórios de software para obter dados sobre sua evolução.

Nota-se que as técnicas atuais para identificação de interesses transversais podem empregar

um único tipo de análise ou uma combinação delas, como pode ser observado na discussão apresentada no Capítulo 3. O propósito de combinar mais de um tipo de análise na mesma técnica é identificar a maior quantidade possível de sintomas de transversalidade. Logo, é preciso considerar que diferentes análises levam à identificação de sintomas distintos, e consequentemente tipos diferentes de interesses transversais. Ao escolher qual análise realizar sobre os dados extraídos de um sistema (*diretriz [D2]*) deve-se manter a consistência com os objetivos definidos para a identificação (*diretriz [D1]*).

5.2.4 *[D4] - Determine o tempo e o esforço disponíveis*

Até a presente data não há nenhum mecanismo que permita a identificação totalmente automática e precisa de interesses transversais. O envolvimento humano é necessário e, geralmente, toma tempo. A quantidade de tempo necessária para fazer a identificação de interesses transversais varia de acordo com a técnica de identificação utilizada. Logo, é importante definir o tempo disponível para que seja possível escolher a melhor abordagem de identificação de acordo com as necessidades de quem vai aplicá-la. Por exemplo, se o tempo disponível para análise for pouco, e a elaboração de cenários de casos de uso for considerada muito complexa para o prazo disponível, técnicas que utilizam dados dinâmicos e análises dinâmicas são descartadas. Do mesmo modo, o esforço deve ser considerado.

5.2.5 *[D5] - Relacione as definições de [D1], [D2], [D3], e [D4]*

O tipo de interesse transversal que se deseja identificar (*[D1]*), o tipo de dado a ser utilizado (*D2*), o tipo de análise (*[D3]*), e o esforço e tempo disponíveis (*[D4]*) têm relações de dependência tais que a definição de qualquer uma das diretrizes tem impacto sobre as demais.

O tipo de análise deve ser compatível com o tipo de dado definido. Por exemplo, para executar a análise histórica é mandatório o uso de dados históricos. A análise escolhida, por sua vez, deve ser capaz de identificar o tipo de interesse transversal definido na *[D1]*.

Por exemplo, a análise estrutural em dados estáticos é capaz de identificar os sintomas exibidos pelo tipo *Consistent Behavior* descrito no Capítulo 3. No entanto, esse tipo de análise toma tempo para ser executada e pode requerer a definição de limiares e filtros. Tais fatores impactam diretamente no tempo necessário para a atividade de identificação.

Adicionalmente, a granularidade de uma técnica é um fator que deve ser analisado com atenção. Em primeiro lugar porque tem influência sobre os valores de *recall* obtidos. Em segundo lugar, como tem influência sobre o *recall*, indiretamente também tem impacto sobre a tríplice restrição discutida na Seção 3.3.1. Logo, é possível observar que a granularidade, mesmo que indiretamente, pode aumentar ou diminuir o esforço manual requerido e a precisão dos resultados obtidos na aplicação de técnicas de identificação. A Figura 5.1 ilustra como a granularidade se relaciona com a tríplice restrição já discutida.

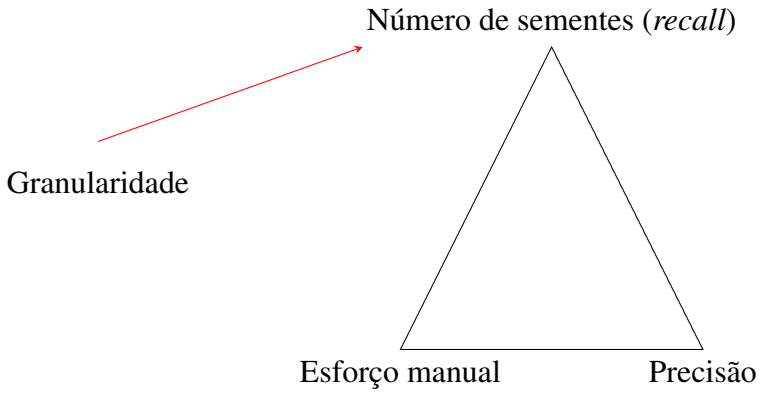

Figura 5.1: Como tem influência direta sobre o *recall*, a granularidade de uma técnica também pode impactar no esforço manual e precisão.

Conclui-se então, que a dependência entre as definições feitas em todas as diretrizes deve ser considerada, principalmente, para manter a compatibilidade e consistência.

5.2.6 *[D6] - Escolha a técnica e ferramenta a serem utilizadas*

A definição consistente do objetivo da identificação, tipo de dados, e tipo de análise auxilia na decisão sobre qual técnica de identificação de interesses transversais utilizar. No entanto, nem todas as técnicas propostas na literatura foram implementadas em ferramentas que podem ser utilizadas para aplicar a identificação. Há casos em que a técnica de identificação proposta é associada a uma ferramenta, porém não disponibilizada para uso, ou desatualizada, impossibilitando o uso. Logo, observa-se que várias das técnicas propostas na literatura são difíceis de serem aplicadas em um contexto de uso real devido à falta ou desatualização das ferramentas que as implementam.

É importante considerar a disponibilidade ou não de uma ferramenta para a identificação, pois a falta da mesma pode impossibilitar ou aumentar o tempo necessário para a identificação de interesses transversais. Como regra geral, o uso real de técnicas de identificação requer a existência de alguma ferramenta para que possa ser aplicada.

5.3 Discussão

Nesta seção são discutidos os conceitos referentes à identificação de interesses transversais abordados em cada uma das diretrizes propostas. Tal discussão pode ser usada como apoio à aplicação das diretrizes, pois fornece informações sobre como selecionar cada um dos parâmetros para a escolha de técnicas de identificação.

5.3.1 Objetivo da identificação (*[D1]*)

O problema abordado por este trabalho é a incompatibilidade gerada pela aplicação não direcionada de técnicas para a identificação de interesses transversais. Em outras palavras, as técnicas propostas na literatura são aplicadas sem considerar suas capacidades de identificação, e para propósitos gerais de identificação. Busca-se identificar qualquer tipo de interesse transversal.

É importante considerar que todas as técnicas de identificação propostas na literatura, até a data atual, realizam a busca por interesses transversais com base em premissas sobre o modo como interesses transversais são implementados em código. Isso foi discutido no Capítulo 3. As premissas adotadas pelas técnicas de identificação faz com que a precisão dos resultados obtidos fique atrelada ao modo como o sistema analisado é estruturado, ou se comporta em execução. Assim como apontado por Mens et al. (2008), a menos que uma única técnica adote um conjunto de premissas, ela identificará somente interesses transversais específicos que exibem sintomas particulares no sistema.

Além disso, as técnicas não são capazes de considerar pequenas variações em padrões de linguagem utilizados durante a codificação. Essas variações podem ser devido a erros de digitação, por exemplo. Não ser capaz de considerar tais variações impacta diretamente nos resultados obtidos, pois mesmo que implementem interesses transversais, as variações não são identificadas Mens et al. (2008).

Desse modo, para que seja possível escolher uma técnica de identificação de interesses transversais adequada, ou mesmo executar algum processo para a identificação, é preciso definir um objetivo em termos dos sintomas que se deseja identificar e como tais sintomas são representados em código ou no comportamento do sistema.

5.3.2 Tipo de dado: Estático, dinâmico, ou histórico (*[D2]*)

Todas as técnicas de identificação de interesses transversais analisam dados provenientes do sistema que se deseja investigar. Tais dados podem ser estáticos, dinâmicos, ou históricos. A diferença entre eles consiste no modo como podem ser obtidos a partir do sistema em análise e o tipo de informação que é possível obter ao processá-los. Além disso, observamos que o tipo de dado que será analisado determina o nível de conhecimento que o analista deve ter sobre o sistema para propósitos de identificação de interesses transversais. A seguir cada tipo de dado é discutido isoladamente.

Tipo de dado estático

Todo sistema de software permite a extração de dados estáticos, contanto que seu código fonte seja disponibilizado. Mesmo código não compilado pode fornecer dados estáticos para propósitos de identificação de interesses transversais (Kellens et al., 2007).

Dados estáticos são compostos por sequências de caracteres, expressões regulares e *tokens*, que podem ser usados para a elaboração de Árvores de Sintaxe Abstrata (*Abstract Syntax Trees* - AST's) e Grafos de Dependência do Programa (Program Dependence Graphs - PDG), por exemplo. AST's PDG's são estruturas de dados comumente utilizadas por técnicas de identificação para localizar código que implementa interesses transversais Bruntink et al. (2005).

Para que seja possível usar dados estáticos na identificação de interesses transversais e obter bons resultados é necessário que o sistema em análise tenha sido implementado de acordo com padrões de linguagem bem definidos. No entanto, nem sempre boas práticas de programação são empregadas na construção de sistemas de software. Padrões de linguagem, ou convenções de nomenclatura, quando definidos, podem não ser seguidos ou serem pouco seguidos.

Apesar do risco representado pela premissa de que o sistema em análise foi implementado seguindo rigorosos padrões de linguagem para o processo de identificação de interesses transversais, a maioria das técnicas automatizadas são capazes de coletar dados estáticos sem envolvimento humano. O que torna o uso de dados estáticos atraente do ponto de vista de minimização do esforço manual para a identificação de interesses transversais.

Dados estáticos também podem ser utilizados para analisar o comportamento de um sistema. Neste caso, a análise é relativamente simples e não requer a execução do programa (a execução é necessária com o uso de dados dinâmicos). No entanto, atualmente o desenvolvimento de software é apoiado por múltiplas bibliotecas dinamicamente lincadas, vinculação dinâmica, polimorfismo, *threads*, entre outros recursos bastante úteis que fazem com que a análise de comportamento se torne imprecisa se for baseada somente em dados estáticos (Gosain e Sharma, 2015). Consequentemente, a identificação de interesses transversais usando dados estáticos pode ser imprecisa.

Várias técnicas de identificação analisam dados estáticos Bruntink et al. (2004), Canfora et al. (2006), Cojocar e Czibula (2008), Zhang e Jacobsen (2012), Tonella e Ceccato (2004), Marin et al. (2004), Tourwe e Mens (2004), Junior et al. (2012), Krinke (2006), McFadden e Mitropoulos (2012), Zhang et al. (2008), Qu et al. (2011), Ishio et al. (2008), Qu e Liu (2007a), Bruntink et al. (2005), Breu e Zimmermann (2006), Zhang e Jacobsen (2007), Shepherd et al. (2005), Huang et al. (2010). O tipo de interesse transversal que cada uma delas é capaz de identificar varia de acordo com o que elas reconhecem como sintoma de transversalidade.

Tipo de dado dinâmico

Dados dinâmicos são compostos por traços de execução, que são sequencias de saídas e chamadas a métodos. A análise dos resultados de técnicas de identificação baseadas em dados dinâmicos[1] permite observar que o uso de traços de execução limita o conjunto de tipos de interesses transversais que são possíveis de identificar.

Por exemplo, sintomas caracterizados por dependências implícitas envolvendo componentes

[1]Foi realizada a análise de resultados documentados e discutidos na literatura.

de herança não são facilmente identificáveis analisando traços de execução. Isso porque dependências implícitas não exibem padrões bem definidos de chamada/saída de métodos. Além disso, dependências implícitas podem ser geradas por elementos que não métodos, por exemplo atributos. As técnicas de identificação atuais baseadas em traços de execução analisam somente dependências entre métodos (Breu, 2005, Krinke, 2008).

Geralmente, para obter dados dinâmicos é preciso ter algum conhecimento sobre o sistema em análise. O conhecimento sobre o sistema é necessário para elaborar cenários de casos de uso que permitam observar o comportamento do sistema conforme ele responde a requisições do usuário. Consequentemente, diferentes sequências de comportamento podem ocorrer de acordo com a interação do usuário (Cockburn, 2000). Imaginar todas as sequências de comportamento que podem ocorrer durante a execução de um programa e traduzi-las para cenários de caso de teste que irão exercitar a maior quantidade de código possível requer conhecer em detalhes as funcionalidades do sistema.

É importante notar que quando se utiliza dados dinâmicos para identificar interesses transversais só é possível identificar sintomas nos trechos de código executados por algum caso de teste. Por isso que é importante conseguir executar a maior quantidade de código possível, sob o risco de não identificar interesses importantes. Considerando as classificações de interesses transversais em tipos descritas no Capítulo 3, o uso de dados dinâmicos permite identificar interesses classificados como *Tsunami*, *Method Consistent Behavior*, e *Contract Enforcement*, por exemplo.

Tipo de dado histórico

Dados históricos são obtidos de repositórios de software, por exemplo de sistemas de controle de versão (*VCS*). Repositórios de software geralmente não são utilizados para propósitos de mineração, no entanto é possível explorar os dados que eles armazenam para extrair informações implícitas sobre um sistema de software (Mulder, 2009). Obviamente que para usar dados históricos na busca por interesses transversais é preciso extrair dados do repositório de software, sendo assim não é possível usar dados históricos para identificar interesses transversais em sistemas que não são armazenados em repositórios de software.

O uso de dados históricos permite a análise de relacionamentos entre artefatos de software que não são possíveis de identificar usando somente o código fonte. Por exemplo, acoplamentos lógicos que são referentes a artefatos não relacionados estruturalmente mas que possuem dependências implícitas e evolutivas. Tais dependências só são observadas ao analisar artefatos que evoluem juntos ao longo do desenvolvimento de software (Mulder, 2009, D'Ambros e Lanza, 2006).

Como definir

Para escolher entre o uso de dados estáticos, dinâmicos, ou históricos é necessário considerar as vantagens de desvantagens no uso de cada tipo. Poucas técnicas de identificação propostas na literatura utilizam dados dinâmicos para buscar por interesses transversais (Tonella e Ceccato, 2004, Breu e Krinke, 2004, Krinke, 2008). Isso pode ser justificado pela dificuldade em elaborar um conjunto de cenários de caso de teste para capturar interesses transversais de modo preciso.

Se o propósito da análise do sistema é identificar e observar interesses mal modularizados que impactam negativamente em cenários de execução específicos, o uso de dados dinâmicos é apropriado. No entanto, se o propósito da análise é localizar toda a extensão da implementação de determinado interesse (para objetivos de refatoração, por exemplo), então o uso de dados estáticos é mais apropriado. Nenhuma técnica proposta até o momento é capaz de identificar a extensão total de interesses transversais, mas se bem aplicadas é possível identificar a maior parte do código correspondente a interesses transversais. Se o objetivo é identificar interesses transversais que não exibem padrões explícitos de execução ou implementação, o uso de dados históricos é uma boa escolha.

5.3.3 Tipo de análise: Estrutural, comportamental, sintática ou histórica (*[D3]*)

As técnicas automatizadas para a identificação de interesses transversais executam diferentes tipos de análises sobre os dados coletados em um sistema. As análises podem ser estruturais, comportamentais, sintáticas, ou históricas.

Como mencionado nas diretrizes propostas, a análise estrutural explora relações de dependência e composição hierárquica de artefatos. A análise comportamental observa os mecanismos internos que são acionados durante a execução do programa (Reimanis e Izurieta, 2016). A análise sintática preocupa-se com comparação de *tokens*, identificadores e padrões de linguagem exibidos em código. Finalmente, a análise histórica investiga o histórico de desenvolvimento do sistema na busca por dependências implícitas.

É importante notar que cada tipo de análise requer um tipo específico de dado para que possa ser executada propriamente. Dados estáticos são adequados, principalmente, para análises estruturais e sintáticas, enquanto que dados dinâmicos são indicados para análises comportamentais e estruturais. Por último, dados históricos são adequados para técnicas que executam análise histórica.

Análise estrutural

A análise estrutural permite minimizar a complexidade de um sistema para que seja possível investigar seus artefatos em qualquer nível de granularidade e a partir de qualquer perspectiva. Por exemplo, a análise da relação de dependência entre métodos, classes, e pacotes. Conforme

o sistema evolui e interesses transversais são adicionados ao código, a estrutura do programa é modificada. Tais modificações e o impacto delas nas relações de dependência entre artefatos podem ser explorados para identificar sintomas de transversalidade (Khan et al., 2015).

Vários tipos de interesses transversais podem ser identificados por meio de análise estrutural, por exemplo *Consistent Behavior*, *Tree Root*, *Tsunami*, e *God Concern*. Esses tipos são facilmente identificados, pois têm sintomas que exibem padrões claro, por exemplo padrões de chamadas a métodos, que podem ser identificados por meio da análise da estrutura do programa (Krinke, 2008, Zhang e Jacobsen, 2012, Krinke, 2006, Qu e Liu, 2007a, Qu et al., 2011).

Análise sintática

A análise sintática baseia-se em convenções de nomenclatura usadas para definir nomes de atributos e identificadores ao longo da codificação do sistema. Técnicas de identificação que executam análise sintática podem ser classificadas em duas categorias: técnicas baseadas em tipos ou técnicas baseadas em texto (Junior et al., 2012).

Técnicas baseadas em tipo tentam identificar o uso recorrente tipos e variáveis de tipos específicos. Por exemplo, é possível identificar toda ocorrência em código do tipo `Connection` do pacote `java.sql` (Junior et al., 2012). Logo, observa-se claramente que para aplicar técnicas de identificação baseadas em análise sintática, é recomendado certificar-se de que o código em análise foi escrito de acordo com convenções de nomenclatura bem definidas.

Interesses transversais podem exibir sintomas implícitos que são difíceis de serem identificados por meio da análise de padrões de linguagem ou relações estruturais. *Expose context*, *Redirection layer*, *Design enforcement* (Marin et al., 2005), *Neural network*, *King snake*, *Hereditary disease* (Figueiredo et al., 2009) são exemplos.

É fato que sistemas de software não exibem apenas características estruturais ou históricas. Sistemas de software também apresentam significantes características comportamentais que podem ser exploradas para identificar interesses transversais. Embora a análise estática seja uma boa abordagem para a identificação de interesses transversais que exibem sintomas estruturais/sintáticos, ela é limitada e imprecisa na avaliação de comportamento.

Análise Comportamental

A análise comportamental não baseia-se em convenções de nomenclatura, e isso é uma de suas vantagens. No entanto, sua generalização para sistemas de grande porte pode se tornar inviável. Isso porque seriam gerados conjuntos de traço de execução muito grandes que inviabilizariam a análise. Além disso, a granularidade dos casos de uso elaborados para a execução do sistema pode afetar a qualidade da análise (Tonella e Ceccato, 2004).

Adicionalmente, a consistência entre o tipo de dado escolhido o tipo de análise executada deve ser garantida pelo analista. Por exemplo, se o sistema a ser analisado permitir a extra-

ção somente de dados estáticos, não é recomendado executar análises dinâmicas (é difícil, senão impossível, analisar o comportamento de um *token* específico em tempo de execução, por exemplo.)

Análise histórica

A análise histórica é utilizada para identificar interesses transversais com base na hipótese de que os mesmos são modificados ao longo do processo de desenvolvimento de software.

Ao contrário da análise comportamental, a análise histórica beneficia-se de projetos em grande escala com histórico de desenvolvimento bem estabelecido. Logo, é um tipo de análise escalável. Para projetos de pequeno porte, com pouco ou nenhum dado histórico disponível, a análise histórica torna-se imprecisa ou impossível de ser aplicada (quando não há histórico de desenvolvimento).

É possível observar que o uso de repositórios de software para a identificação de interesses transversais é apropriado, principalmente, para a identificação de alterações concomitantes em artefatos de software. Isso porque, dois artefatos alterados em conjunto em um mesmo *commit* ao longo de vários *commits* têm maiores chances de ser relacionados, ou seja, há entre eles algum tipo de dependência lógica.

Como definir

Para definir o tipo de análise a ser executada é preciso considerar o tipo de dado selecionado na diretriz *[D1]*. Isso porque cada tipo de análise requer tipos de dados específicos para poder executar.

A análise sintática requer que o código fonte do sistema esteja disponível e que seja possível executar análises sobre o mesmo. O tipo de dado nesse caso deve ser estático e seu formato depende da granularidade escolhida, por exemplo: métodos, classes, fragmentos de código, ou linhas de código.

A análise estrutural pode ser feita sobre dados estáticos ou dinâmicos, depende da abordagem de identificação. Sendo assim, a escolha da análise estrutural não limita o tipo de dado a ser escolhido, apesar de ser um pouco mais complexa de ser feita se utilizados apenas dados históricos, por exemplo.

A análise comportamental, assim como a estrutural também pode ser feita utilizando dados estáticos ou dinâmicos, porém como já mencionado, o uso de dados estáticos para a análise comportamental é pouco eficiente. Já a análise histórica requer dados referentes ao histórico de desenvolvimento do sistema.

5.3.4 Tempo de análise e esforço manual (*[D4]*)

Apesar de serem chamadas de automáticas na literatura, as técnicas de identificação de interesses transversais não são totalmente automáticas. Todas requerem algum tipo de esforço manual, seja para a definição de filtros e limiares, para o pré-processamento dos dados de entrada ou para a confirmação manual de resultados como observado na discussão do Capítulo 3.

Para escolher a técnica mais apropriada de acordo com os objetivos da identificação é recomendado considerar o tempo necessário para a execução de todo o processo de identificação e o esforço manual necessário.

Geralmente, técnicas com nível de granularidade baixa requerem maior esforço manual se usadas em sistemas de grande porte, isso porque o número de elementos de código resultantes aumenta proporcionalmente ao tamanho do sistema. No entanto, baixa granularidade permite localizar mais facilmente trechos de código que implementam interesses transversais, o que não acontece na granularidade alta.

Por exemplo, se o resultado do processo de identificação for composto por classes indicadas como potenciais interesses transversais (granularidade alta), é preciso inspecionar toda a extensão da classe para localizar os trechos de código correspondentes à funcionalidade transversal.

A definição de limiares e filtros tem grande impacto nos resultados de uma técnica de identificação, e podem ser difíceis de determinar. Por exemplo, é difícil definir o número inicial de *centróides* para a aplicação do algoritmo *k-means* (Thammano e Kesisung, 2013) em uma técnica para a identificação de interesses transversais baseada em agrupamento ou *clustering* (Cojocar e Czibula, 2008).

5.3.5 Como as diretrizes propostas se relacionam (*[D5]*)

As diretrizes propostas evidenciam a relação de dependência entre os parâmetros: objetivo de identificação, tipo de dado, tipo de análise, e tempo e esforço. O modo como as diretrizes foram elaboradas permite uma abordagem *top-down* para a definição de tais parâmetros. Ou seja a escolha de uma técnica é feita inicialmente definindo-se um objetivo de identificação, seguido da definição dos parâmetros de análise, que irão permitir a escolha de uma ferramenta, que por sua vez é utilizada para atingir os objetivos de identificação.

É importante observar que, o que é chamado de parâmetro nesta seção, é resultado da análise das técnicas de identificação feita no Capítulo 3 e da aplicação da TFD descrita no Capítulo 4. A partir do *framework* para a análise proposto por Kellens et al. (2007) as técnicas foram analisadas, e foi notado claramente que *tipo de dado*, *tipo de análise*, e *esforço manual* são os fatores de maior impacto nos resultados das abordagens para a identificação de interesses transversais, e que são dependentes entre si.

Adicionalmente, ao executar a análise qualitativa da literatura utilizando a TFD no Capítulo 4, foi possível reduzir todas as categorias e características relacionadas a técnicas de identifica-

ção para quatro categorias (na codificação axial): *tipo de dado*, *tipo de análise*, *granularidade* e *esforço manual*. O cruzamento entre os fatores de impacto descobertos na análise preliminar das técnicas de identificação e as categorias e características resultantes da TDF resultou nos parâmetros: tipo de dados, tipo de análise, e esforço manual requerido.

Ao longo da análise sobre as técnicas de identificação e do resultado obtido aplicando a TFD, foi possível estabelecer a natureza da dependência entre os parâmetros identificados, o que possibilitou refiná-los e transformá-los no conjunto de diretrizes apresentado anteriormente.

Conclui-se então, que ao escolher os objetivos, tipo de dados, tipo de análise, e definir o tempo e esforço necessários para a identificação, conforme o fluxo das diretrizes *D1* a *D2*, é preciso manter a consistência entre os parâmetros selecionados. A não compatibilidade entre as definições, pode levar a redefinição de cada parâmetro. Por exemplo, a escolha do tipo de análise deve ser consistente com o tipo de dados definido, pois cada tipo de análise requer um tipo de dado específico para ser executada. O fluxo de definições proposto pelas diretrizes é mostrado na Figura 5.2.

5.3.6 Técnica e ferramentas para a identificação de interesses transversais (*[D6]*)

A escolha de uma técnica para a identificação de interesses transversais e, consequentemente, da ferramenta automatizada para executar o processo de identificação têm influência sobre a escalabilidade da abordagem adotada. A escalabilidade é uma das propriedades mais importantes para uma técnica de identificação de interesses transversais. Principalmente devido ao fato de que quanto maior o sistema a ser analisado mais difícil é de fazer a identificação manual de interesses transversais, ou qualquer tipo de anomalia estrutural ou comportamental do sistema.

Assim, é essencial que abordagens para a identificação de interesses transversais sejam implementadas em ferramentas que possam ser utilizadas por um analista. No entanto, há técnicas de identificação propostas na literatura que não foram implementadas ou não disponibilizam ferramentas que possam ser utilizadas por analistas ou outros pesquisadores (Maisikeli e Mitropoulos, 2010, Qu e Liu, 2007a).

Quando uma técnica de identificação é proposta e não é disponibilizada nenhum tipo de ferramenta que automatize a abordagem de identificação proposta, seu uso é dificultado, e leva à redefinição do planejamento para a identificação de interesses transversais.

Além de considerar a existência/disponibilização ou não de ferramentas para automatizar a aplicação de uma técnica, é necessário avaliar o tipo de envolvimento manual requerido pela ferramenta, por exemplo para a configuração de limiares e filtros. Isso porque a definição de limiares e filtros, geralmente requer que o analista tenha conhecimento sobre o sistema em análise ou sobre a abordagem de identificação adotada.

A escolha da técnica e da ferramenta a serem utilizadas não é a última das diretrizes por

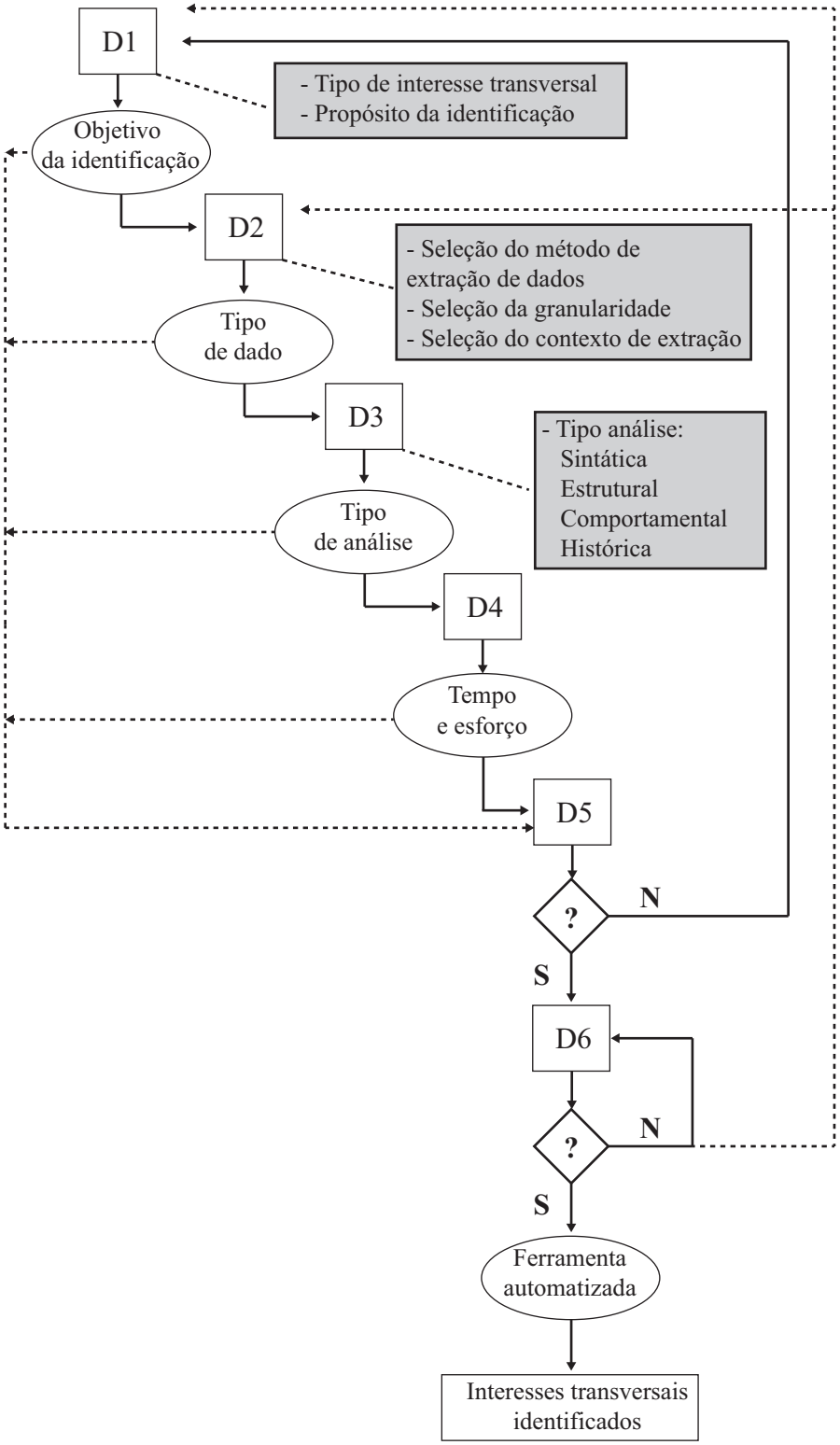

Figura 5.2: Uso das diretrizes para a escolha de técnicas de identificação de interesses transversais

acaso. Todo o fluxo de definições e planejamento guiado pelas diretrizes *D1*, *D2*, *D3*, e *D4* culmina na escolha de uma técnica que forneça apoio automático para a identificação de interesses transversais.

A definição da diretriz *D6* pode alterar todo o fluxo da aplicação das diretrizes e requerer o retorno às definições de *D2*, que levará à seleção de uma abordagem de identificação diferente, ou a *D1* em que os objetivos de identificação devem ser revistos por não haver uma técnica que auxilie a alcançá-los. O fluxo de definições proposto pelas diretrizes até a escolha da ferramenta/técnica de identificação e obtenção de resultados é mostrado na Figura 5.2.

5.4 Aplicação das Diretrizes Propostas

Nesta seção as diretrizes propostas são aplicadas para a identificação de interesses transversais em um sistema de software específico. O objetivo da aplicação das diretrizes é fornecer um exemplo de uso e possibilitar sua validação. Além disso, é proposto um catálogo de sementes composto pelas sementes identificadas pela técnica escolhida com o auxílio das diretrizes propostas e as sementes documentadas e confirmadas na literatura para o sistema em análise.

Esta seção é estruturada do seguinte modo: Inicialmente é definido o sistema que será utilizado para identificar interesses transversais. Em seguida, as diretrizes propostas são utilizadas para determinar qual técnica deve ser utilizada para fazer a identificação de interesses transversais. Os resultados obtidos são discutidos e são apresentadas as conclusões finais sobre o uso das diretrizes propostas.

5.4.1 Escolha do objeto de análise e construção do oráculo

O *framework* JHotDraw v5.4b1 foi escolhido como objeto de análise por dois motivos: (1) é um projeto *open-source*, de modo que qualquer pesquisador pode repetir a aplicação da técnica que será escolhida; (2) ele tem sido utilizado como *benchmark* para a maioria dos experimentos com técnicas de identificação de interesses transversais propostas na literatura. E assim, apesar de não se conhecer todos os interesses transversais implementados no JHotDraw, a maioria é reportada na literatura, o que facilita a análise dos resultados aqui obtidos.

Para possibilitar a validação dos resultados alcançados por meio da aplicação da técnica de identificação escolhida, foi construído um *oráculo* composto por todas as sementes identificadas e reportadas na literatura.

5.4.2 Escolha da Técnica de Identificação

Nesta seção é descrito como as diretrizes propostas foram empregadas para escolher a técnica de identificação a ser utilizada. Inicialmente é descrito como foi definido o objetivo da identificação *(D1)*, seguido da definição dos principais parâmetros de escolha *(D2), (D3)* e *(D4)*. A seguir o relacionamento entre os parâmetros definidos e o objetivo da identificação são confrontados para verificar a consistência entre eles *(D5)*. Por último é investigada qual técnica proposta na literatura corresponde às definições feitas e se ela dispõe de uma ferramenta

automatizada e de uso livre.

[D1]: Objetivo da identificação:

O *framework* JHotDraw foi inicialmente desenvolvido como exercício para a aprendizagem de padrões de projeto. Padrões provêm soluções que podem ser usadas e adaptadas para melhorar a modularização e reuso de código. No entanto, atividades como a notificação de alterações de modelo a *Observers* ou a invocação de alguma funcionalidade do sistema por meio do padrão *Command*, exemplificam que o uso de padrões de projeto gera espalhamento de código. A análise do padrão *Command* evidencia a existência de interesses que implementam determinados comportamentos do sistema de maneira consistente por meio da execução recorrente de um conjunto de métodos específicos. Por exemplo, a chamada de métodos para checar a possibilidade de execução de comandos definidos no *framework* JHotDraw.

Observou-se que a implementação consistente dos padrões *Command* e *Observer* faz com que ocorra alterações em cascata em classes espalhadas por mais de uma hierarquia. Sendo assim, a partir da análise das características dos padrões *Observer* e *Command* no JHotDraw, foi definido a busca por interesses transversais do tipo *Consistent Behavior* que define sintomas relacionados à implementação de determinados comportamentos de maneira consistente.

[D2] [D3] [D4]: Tipo de dado, tipo de análise, tempo e esforço necessários:

A análise necessária para analisar os padrões de chamadas a métodos no sistema pode ser feita estaticamente com base em dados retirados diretamente do código fonte. Como utilizados dados estáticos, não é possível executar uma análise dinâmica. Logo, a análise deve ser estrutural/comportamental sobre dados estáticos.

O tempo e esforço definidos para a identificação de interesses transversais levou em consideração uma aplicação única da técnica. Assim, foi determinado que os resultados serão avaliados por uma pessoa e o tempo disponível para aplicação e análise é de 1 (uma) semana.

Assim, as definições referentes às diretrizes *[D2], [D3],* e *[D4]* são sumarizadas na Tabela 5.1.

Tabela 5.1: Definições para as diretrizes *[D2], [D3],* e *[D4]*

D1: *Consistent Behavior*
D2: Dados estáticos
D3: Estrutural
D4: 1 semana / 1 pessoa

[D5]: Relacionar as definições em D1, D2, D3, e D4:

O objetivo definido para a identificação considera a análise de chamadas frequentes e consistentes a métodos que envolvem os padrões de projeto que implementam o JHotDraw. Desse modo, a granularidade a ser considerada para análise é a de métodos. De acordo com a literatura e como foi descrito nos Capítulos 3 e 4 a análise de chamadas a métodos pode ser feita tanto por meio da utilização de dados dinâmicos quanto estáticos. Devido à restrição de tempo e esforço, optou-se pela análise estática que não requer a elaboração de casos de uso e execução de uma aplicação que implemente o *framework* em análise.

Um vez que foi definido o uso de dados estáticos provenientes diretamente do código fonte, eliminou-se a possibilidade da execução de análises históricas. Assim, é possível optar entre as análises léxica e estrutural/comportamental. Como o objetivo é buscar por interesses transversais analisando o comportamento do sistema (porém de maneira estática) sob o ponto de vista da execução de métodos, e padrões de chamadas a eles, optou-se pela análise estrutural.

[D1]: Escolha da técnica e da ferramenta:

A escolha da técnica a ser utilizada deve considerar os seguintes parâmetros definidos: uso de dados estáticos, análise comportamental, pouco esforço manual, e a análise deve ser capaz de identificar padrões de chamadas a métodos analisando dados estáticos. Ao selecionar a técnica a ser aplicada é imprescindível que ela tenha sido implementada em alguma ferramenta de identificação de uso livre.

Para encontrar a técnica adequada de acordo com os parâmetros e objetivos definidos, as técnicas selecionadas para análise durante a elaboração da revisão sistemática e aplicação da TDF foram sumarizadas de acordo com tais parâmetros como mostra a Tabela 5.2, os artigos que propõem cada uma delas são referenciados na Tabela 5.3.

Tabela 5.2: Mapeamento das técnicas de identificação propostas na literatura em relação à suas características

		Análise de co-change	Análise de chamadas a métodos	Análise de *clustering*	Análise de grafos	Análise de link	Análise de rastros de execução	Análise formal de conceitos	Análise de clones	Análise de padrões de linguagem	Análise histórica
Tipo de dado	Estática	[1]	[2] [3] [4] [5] [6] [7]	[8] [9] [4]	[10] [11] [12] [13] [7]	[14]	[2] [9]	[15] [6] [13]	[16] [17]	[18][5] [19]	
	Dinâmica		[2]	[9]			[21][2] [9]	[22] [23] [6]			
	Histórico										[20] [25]
Tipo de Análise	Sintática		[5]					[15]	[16]	[18] [5] [19]	
	Estrutural	[1]	[2] [3] [4] [6][7]	[8] [24] [9][4]	[10] [11] [12] [13] [6]	[14]	[2][9]	[6][13]	[16] [17]		
	Histórica										[20] [25]
	Comport.		[2]	[9]			[21][2] [9]	[22] [23] [6]			
Granularidade	Classe					[14]		[15]			
	Methods		[2] [3] [4] [6] [7]	[8] [24] [9] [4]	[10] [11] [12] [13] [6]		[21] [2] [9]	[22] [15] [23][6] [13]			[20] [25]
	Code fragment		[5]						[16] [17]	[18][5] [19]	
	Linhas de código	[1]									
Esforço Man.	Pré		[3][4]	[4]			[21]	[22] [23]		[18]	
	Pós	[1]	[2][3] [4] [5] [6] [7]	[8][24] [9] [4]	[10] [11] [12] [13] [6]	[14]	[21] [2] [9]	[22] [15] [23] [6][13]	[16] [17]	[18][5][19]	[20] [25]

Tabela 5.3: Técnicas de identificação selecionadas na amostragem teórica.

Código	Referência
[1]	Canfora et al. (2006)
[2]	Breu (2005)
[3]	Marin et al. (2004)
[4]	Zhang et al. (2008)
[5]	Ishio et al. (2008)
[6]	Qu e Liu (2007a)
[7]	Zhang e Jacobsen (2007)
[8]	Cojocar e Czibula (2008)
[9]	Maisikeli e Mitropoulos (2010)
[10]	Krinke (2008)
[11]	Zhang e Jacobsen (2012)
[12]	Krinke (2006)
[13]	Qu et al. (2011)
[14]	Huang et al. (2010)
[15]	Tourwe e Mens (2004)
[16]	Bruntink et al. (2004)
[17]	Bruntink et al. (2005)
[18]	Junior et al. (2012)
[19]	Shepherd et al. (2005)
[20]	Breu e Zimmermann (2006)
[21]	Breu e Krinke (2004)
[22]	Tonella e Ceccato (2004)
[23]	Ceccato e Tonella (2009)
[24]	McFadden e Mitropoulos (2012)
[25]	Marcal et al. (2017)

A Tabela 5.2 mostra o cruzamento entre todos os possíveis parâmetros e as técnicas propostas na literatura. De acordo com o objetivo de identificação definido é necessário uma técnica que seja capaz de identificar padrões de execução de métodos, e portanto pertencentes ao grupo de técnicas da terceira coluna (Análise de chamadas a métodos). O cruzamento dos parâmetros definidos e o grupo de técnicas escolhido mostra que é possível escolher entre as técnicas [2], [3], [4], [6], e [7]. A análise de cada uma delas levou à seleção da técnica [3], principalmente por ser implementada em uma ferramenta de uso livre, e simples de ser utilizada. A técnica [3] corresponde à Análise Fan-in.

A seguir é mostrado como a Análise Fan-in foi aplicada, e os resultados obtidos são analisados para validar o uso das diretrizes propostas.

5.4.3 Aplicação da Técnica Selecionada

Para executar a aplicação da análise fan-in, foi utilizada a ferramenta FINT 0.6[2] (Marin et al., 2004). FINT é um *plugin* para a IDE *Eclipse* que computa automaticamente o valor fan-in

[2]http://swerl.tudelft.nl/bin/view/AMR/FINT

de cada método que compõe o sistema em análise e facilita a filtragem e validação de resultados. Sendo assim, a ferramenta da suporte a:

- *Cálculo do valor fan-in:* Para cada método implementado, a ferramenta fornece uma lista dos métodos que fazem chamada a ele. O número de métodos chamadores é o valor fan-in do método que é chamado. A ferramenta considera apenas chamadas feitas em locais diferentes do código fonte (o que é especialmente importante ao tratar de métodos polimórficos).

- *Filtragem de métodos:* A FINT permite definir um limiar para o valor fan-in, além de possibilitar a aplicação de filtros para métodos utilitários como *setters* e *getters* que, geralmente, geram ruídos nos resultados. Além disso, é possível definir quais classes, pacotes, ou métodos devem ser analisados e quais devem ser ignorados.

- *Análise de resultados:* A ferramenta FINT facilita a análise das sementes resultantes diretamente no código fonte do sistema. Os métodos são listados com o respectivos valores fan-in e uma lista de métodos chamadores. O limiar para o valor fan-in é utilizado para determinar os métodos que aparecerão na lista de resultados. Desse modo, foi possível investigar cada método chamador para identificar os padrões de invocação indicativos do tipo de interesse *Consistent Behavior*.

Para a aplicação da análise fan-in foram filtrados todos os métodos utilitários (*setters* e *getters*), pois não são interesses transversais, apesar de exibirem alto valor fan-in.

Foi definido um limiar igual a 10 para o valor fan-in, ou seja foram considerados apenas métodos com valor $Fan-in \geq 10$. Esse valor foi determinado por meio da análise do número de sementes retornadas pela ferramenta para cada valor fan-in entre 0 e 20, observando a precisão da identificação a cada valor limiar. O mesmo valor foi usado em Marin et al. (2004), pois provê um bom equilíbrio entre o número total de métodos do sistema em análise e o número total de métodos resultantes como potenciais interesses transversais.

Como não há disponível na literatura o conjunto de todas as sementes implementadas no JHotDraw, ou qualquer outro sistema, os resultados foram validados com base no oráculo criado para este trabalho. A análise fan-in foi aplicada a 60 revisões do *framework* JHotDraw v5.4b1, os resultados obtidos para todas as aplicações executadas são discutidos a seguir.

Resultados:

A análise dos resultados obtidos teve como foco a investigação das características das sementes resultantes para que fosse possível confirmar que a análise fan-in é uma técnica realmente adequada para a identificação de interesses transversais do tipo *Consistent Behavior*. A confirmação de tal adequação, por sua vez, valida o uso das diretrizes propostas que levaram à escolha da análise fan-in como técnica de identificação a ser usada no experimento apresentado. A análise fan-in foi executada em 60 revisões do *framework* JHotDraw v5.4b1. Logo, foi

possível avaliar também se a adequação da técnica é independente do estágio de desenvolvimento do sistema em análise, além de permitir a análise histórica das sementes implementadas no *framework* JHotDraw v5.4b1.

Para cada semente retornada pela análise fan-in em cada um dos experimentos foram documentados os seguintes dados: tamanho em linhas de código (descontadas linhas em branco), tipo e quantidade de alterações sofridas ao longo das revisões consideradas, tipo do interesse que implementam, classe e pacote ao qual pertencem. É importante lembrar que a análise fan-in é executada a nível de método (granularidade).

Os dados obtidos a partir dos resultados da aplicação da análise fan-in foram manualmente processados e com eles foi gerado um *treemap* para que fosse possível visualizar a correlação entre as sementes e outros elementos do sistema e extrair informações sobre o modo como a técnica identifica interesses transversais, e assim validar sua compatibilidade com os objetivos de identificação definidos.

Uma visão geral das sementes identificadas ao longo de todos os experimentos realizados é mostrada na Figura 5.3. O tamanho de cada nó representa o tamanho da semente em linhas de código. As cores representam o quanto cada semente foi alterada ao longo do desenvolvimento do *framework*.

Por um lado, a análise do *treemap* mostrado na Figura 5.3 permite observar que a análise fan-in é capaz de identificar sementes relacionadas a vários tipos de interesses transversais. Dentre eles *Role Superimposition*, *Entangled Roles*, *Add Variability*, *Redirection Layer*, e *Contract Enforcement*. Isso confirma o que foi apontado no Capítulo 2, ou seja, que um tipo de interesse transversal descreve um conjunto de sintomas que são analisados e identificados por uma técnica. Desse modo, é possível que uma única técnica seja capaz de identificar sementes correspondentes a mais de um tipo de interesse transversal mesmo sendo ela especialista na identificação de um conjunto de sintomas específicos (A análise fan-in analisa o sintoma de espalhamento de chamadas a métodos).

Por outro lado, é possível observar também a predominância de sementes classificadas como *Consistent Behavior* (destacadas pelo retângulo em amarelo à esquerda da Figura 5.3), que ocupa todos os nós mais à esquerda da Figura 5.3. O que também confirma a hipótese de que técnicas para a identificação de interesses transversais são especialistas na identificação de determinado tipo de interesse transversal.

O fato de os mesmos sintomas de transversalidade estarem relacionados a diferentes tipos de interesses pode sugerir que as classificações propostas na literatura são ambíguas. Assim, se a definição do objetivo de identificação (*diretriz D1*) for baseada em tipos, a precisão dos resultados dificilmente será próxima ou igual a 100%. No entanto, a ambiguidade pode refletir pequenas variações na estrutura ou comportamento de interesses transversais, que alteram o contexto e significado do interesse, de modo que a única alternativa é classifica-lo como um novo tipo, mesmo que exiba sintomas semelhantes a um tipo já existente.

Uma observação importante a ser feita é a de que ao considerar como objetivo da iden-

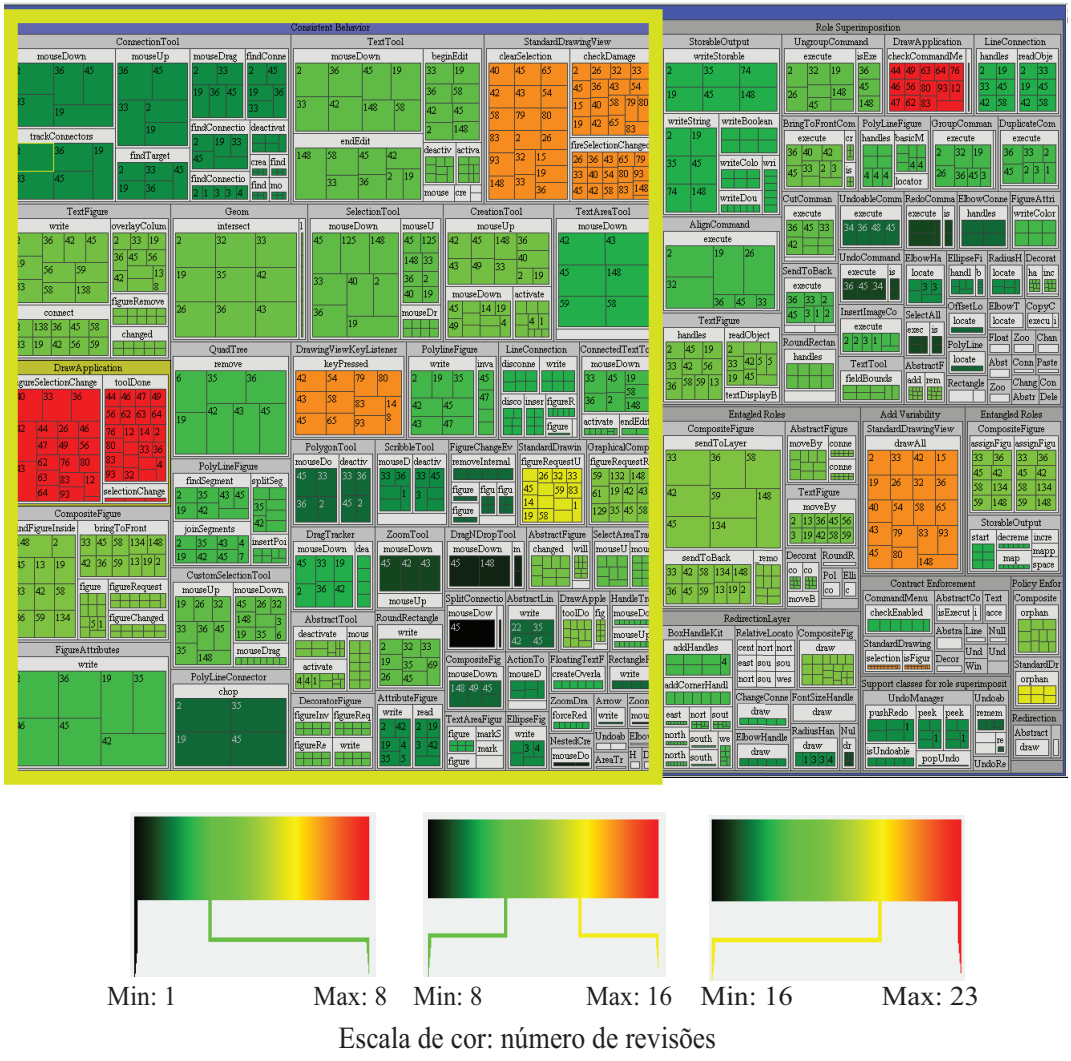

Figura 5.3: Sementes identificadas pela Análise Fan-in

tificação apenas sementes do tipo *Consistent Behavior* e classificar todas as demais como falso-positivos, as sementes confirmadas de outros tipos (por exemplo, *Role Superimposition*, *Entangled Roles*, *Add Variability*) contribuirão para diminuir a precisão da técnica em termos do número de sementes validadas. No entanto, como pode ser visualizado na Figura 5.3 a técnica foi capaz de identificar os interesses com maior impacto negativo na modularidade do sistema em análise. Isso é evidenciado pelo alto número de revisões relacionadas às sementes do tipo *Consistent Behavior* representadas pela cor vermelha, laranja e amarelo na escala de cor.

Mesmo que a precisão da técnica aplicada em termos numéricos seja baixa, ela foi capaz de identificar elementos do código cujo impacto negativo na qualidade do sistema é maior, e assim deveriam ser revistos com maior prioridade que os interesses transversais de outros tipos também identificados. Uma comparação da relevância das sementes identificadas para três tipos de interesses transversais é mostrada na Figura 5.4, a composição das sementes do tipo *Consistent Behavior* é mostrada na Figura 5.5.

Fica em evidência então, a relação entre o objetivo da identificação, o tipo de dado, o tipo

de análise executada, e os sintomas de transversalidade que o sistema apresenta.

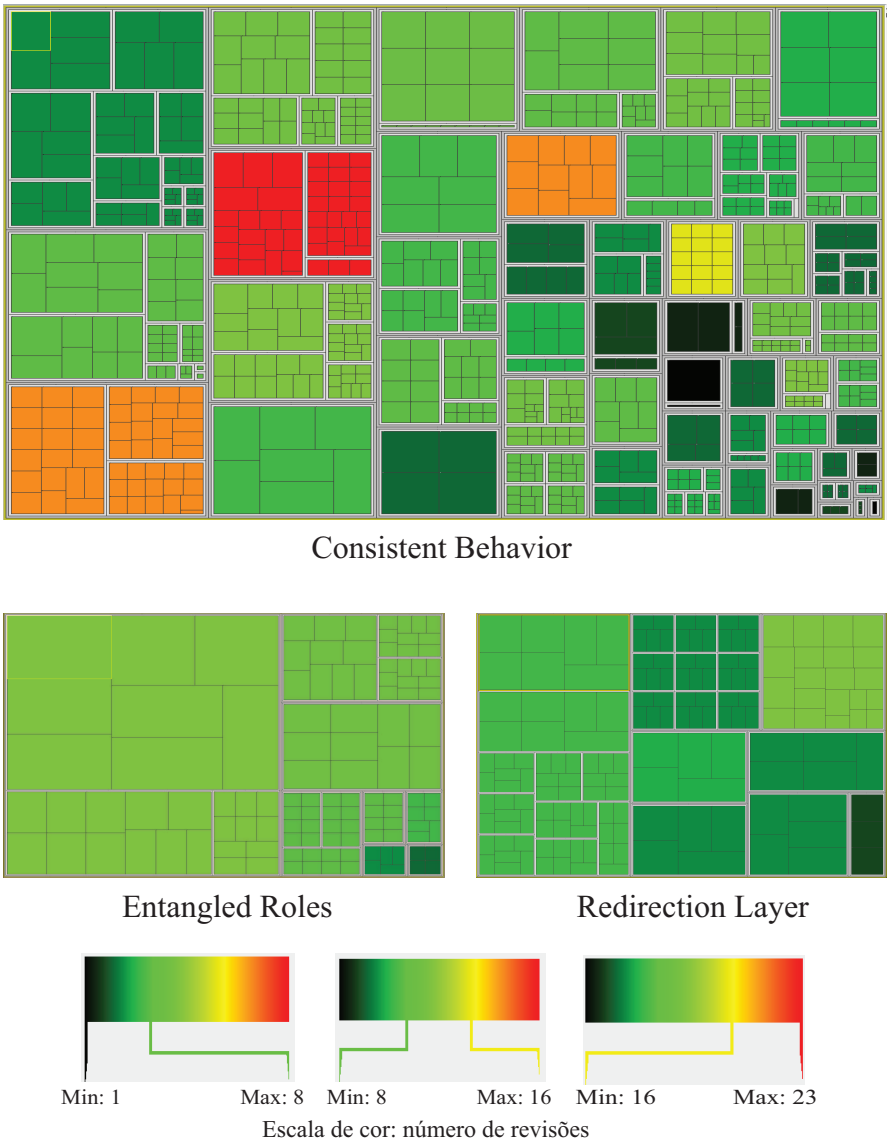

Figura 5.4: Os interesses transversais do tipo *Consistent Behavior* são mais relevantes, pois mostram maior impacto em relação a problemas de alterações em cascata.

Para a definição dos objetivos é preciso reconhecer os problemas estruturais ou comportamentais que o sistema apresenta, para então escolher a técnica mais apropriada para identificar os elementos causadores de tais problemas. A escolha da técnica mais apropriada, por sua vez é atrelada à definição do tipo de dado e tipo de análise mais adequadas para identificar o problema que o sistema exibe.

5.5 Considerações finais

Neste capítulo foram apresentadas as diretrizes propostas e foi feita a utilização das mesmas para selecionar a técnica ser utilizada para a identificação de interesses transversais no *framework* JHotDraw. Os resultados obtidos e discutidos demonstram que a técnica selecionada

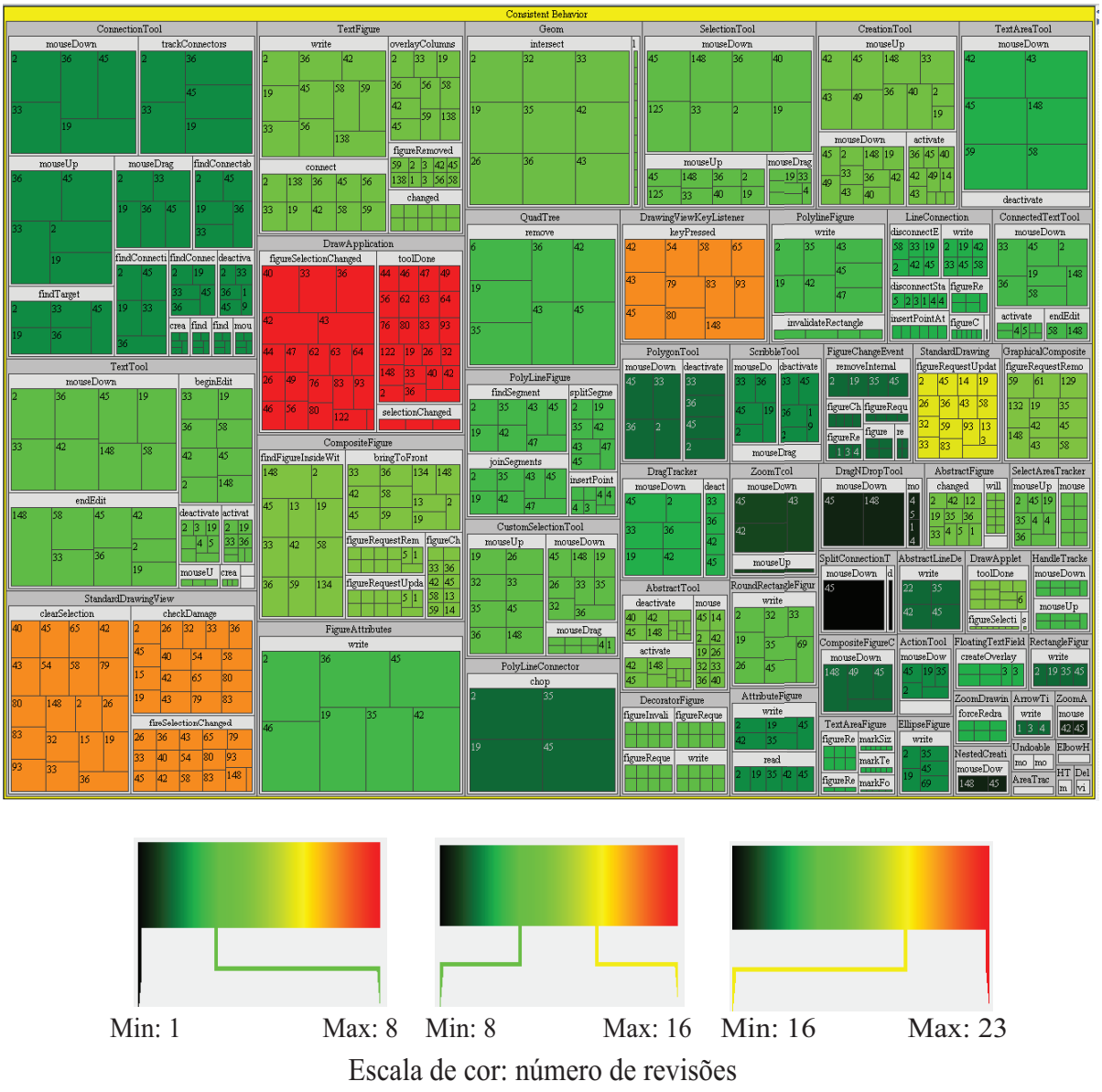

Figura 5.5: As sementes do tipo *Consistent Behavior* são frequentemente alteradas

com o uso das diretrizes propostas é adequada para a identificação de interesses transversais do tipo definido como objetivo da identificação. Assim, as diretrizes propostas foram validadas com base nas várias aplicações da técnica selecionada.

Para possibilitar a avaliação da análise fan-in ao longo das 60 revisões consideradas, foram feitas 2000 análises relacionadas a 150 sementes do *framework* JHotDraw documentadas na literatura. Observou-se que apesar de os interesses transversais serem modificados ao longo do tempo, a técnica é hábil em identificar o tipo *Consistent Behavior* em qualquer estágio do desenvolvimento, restringido às 60 revisões consideradas.

CAPÍTULO 6

Conclusão

Os problemas causados pela existência de interesses transversais têm sido amplamente estudados e demonstrados na literatura. Para minimizar tais problemas são aplicadas técnicas de refatoração na tentativa de modularizar interesses transversais. Porém, para que seja possível encapsular interesses transversais, primeiro é necessário identificá-los no sistema em análise.

Várias técnicas de identificação são propostas na literatura com o objetivo de automatizar o processo de identificação de interesses transversais. No entanto, tais técnicas demonstram problemas relacionados à precisão. Nós observamos que a precisão das técnicas de identificação vigentes, geralmente, é devido à não utilização das mesmas de modo direcionado. Ou seja, não é estabelecido um objetivo de identificação, e as técnicas são escolhidas sem considerar nenhuma de suas características e capacidades de identificação em relação aos conhecidos sintomas de transversalidade.

Neste trabalho foi proposto um conjunto de diretrizes para auxiliar na escolha da técnica de identificação apropriada considerando características das técnicas atuais e como tais características se relacionam com sintomas de transversalidade.

6.1 Contribuições

O conjunto de diretrizes estabelecido é a maior contribuição deste trabalho. São 6 diretrizes que consideram diferentes características de técnicas para a identificação de interesses transversais. As diretrizes foram aplicadas e demonstraram sua utilidade para a seleção de técnicas capazes de resultar em sementes significativas em relação ao objetivo de identificação definido.

Os resultados obtidos por meio da aplicação da técnica de Análise Fan-in, escolhida por meio do uso das diretrizes, também é uma contribuição importante. Primeiramente porque para validar as sementes resultantes foi necessário elaborar um oráculo para a comparação de resultados. Tal oráculo foi composto pelas sementes validadas e documentadas na literatura. Além

disso, dados sobre as sementes foram armazenados em uma base de dados que será disponibilizada para pesquisadores da área que poderão contribuir para que tal base se torne mais completa ao passar do tempo.

A aplicação da análise Fan-in foi repetida e 60 revisões do *framework* JHotDraw, o que permitiu obter uma perspectiva histórica da evolução das sementes e para constatar que apesar de as sementes serem modificadas ao longo do desenvolvimento, a técnica escolhida por meio das diretrizes propostas, é capaz de identificar interesses do tipo objetivo em diferentes momentos do ciclo de vida de desenvolvimento.

Além disso, para que fosse possível a elaboração das diretrizes propostas, foi realizada uma análise detalhada das capacidades e limitações de técnicas atuais para a identificação de interesses transversais em relação a suas principais características.

6.2 Limitações

A utilidade do conjunto de diretrizes propostas foi validado pela aplicação de uma única técnica escolhida por meio delas. Para avaliar a eficácia e a eficiência do uso das diretrizes propostas, planejamos utilizar as diretrizes definindo outros objetivos de identificação.

As diretrizes propostas foram elaboradas a partir da análise de uma amostragem teórica restrita de 25 técnicas. Uma extensão está sendo planejada de modo que novas técnicas sejam incluídas na amostragem teórica para possível refinamento das diretrizes elaboradas.

A precisão obtida pela técnica selecionada com o uso das diretrizes não foi considerada, isso porque buscou-se validar se a técnica escolhida resultaria em sementes relevantes em relação ao objetivo de identificação. Pretende-se fazer um estudo sobre a precisão das técnicas selecionadas para que seja possível refinar as diretrizes propostas.

6.3 Trabalhos Futuros

O principal trabalho futuro é a implementação de uma ferramenta que possa ser utilizada para aplicar as diretrizes de modo automático com base nas características das técnicas de identificação existentes. Desse modo o usuário da ferramenta define somente o objetivo da identificação em termos das características de transversalidade que se deseja identificar. Então, a ferramenta retorna as técnicas existentes mais apropriadas.

O oráculo criado corresponde ao catálogo de sementes propostas. Tal catálogo pode ser estendido para que outras sementes de outros sistemas de software sejam disponibilizadas para pesquisadores e analistas que queiram validar os resultados de suas abordagens de identificação.

O estudo realizado neste trabalho sugere também que é possível refinar as classificações de interesses transversais por tipo propostas na literatura, para minimizar a ambiguidade do conjunto de sintomas que representa cada tipo.

6.4 Produção Bibliográfica

Foi publicado um artigo relacionado à análise de técnicas para a identificação de interesses transversais, em 2016, como segue:

- MARÇAL, I.; GARCIA, R. E.; ELER, D. M.; OLIVETE JUNIOR, C.; CORREIA, R. C. M. Techniques for the identification of crosscutting concerns: A systematic literature review Cham: Springer International Pu- blishing, p. 569−579, 2016.

Adicionalmente dois artigos estão em fase de correção para submissão, como segue:

- MARCAL, I.; GARCIA, R. E.; ELER, M. D.; CORREIA, R. C. M.; JU- NIOR, C. O. A framework to combine different granularity-level cross- cutting concern identification techniques. No Prelo, 2017.
- MARCAL, I.; GARCIA, R. E.; ELER, M. D.; CORREIA, R. C. M.; JU- NIOR, C. O. Guidelines for Choosing Adequate Crosscutting Concern Identification Techniques: towards improving results precision. No Prelo, 2017.

Referências Bibliográficas

ALI, M. S.; BABAR, M. A.; CHEN, L.; STOL, K.-J. A systematic review of comparative evidence of aspect-oriented programming. *Information and Software Technology*, v. 52, n. 9, p. 871 – 887, 2010.

ALLAN, G. A Critique of using Grounded Theory as a Research Method. *Electronic Journal of Business Research Methods*, v. 2, n. 1, 2003.

VAN DEN BERG, K.; CONEJERO, J. M.; HERNÁNDEZ, J. Analysis of crosscutting across software development phases based on traceability. In: *Proceedings of the 2006 International Workshop on Early Aspects at ICSE*, New York, NY, USA: ACM, 2006, p. 43–50.

BERNARDI, M. L.; DI LUCCA, G. A. A role-based crosscutting concerns mining approach to evolve java systems towards aop. In: *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, New York, NY, USA: ACM, 2009, p. 63–72.

BIGGERSTAFF, T.; MITBANDER, B.; WEBSTER, D. The concept assignment problem in program understanding. In: *Software Engineering, 1993. Proceedings., 15th International Conference on*, 1993.

BIOLCHINI, J.; ET AL. Systematic review in software engineering. *System Eng. and Comp. Sci. Dept. COPPE/UFRJ, Tech. Rep. ES*, v. 679, n. 05, p. 45, 2005.

BREU, S. Extending dynamic aspect mining with static information. In: *Fifth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'05)*, 2005, p. 57–65.

BREU, S.; KRINKE, J. Aspect mining using event traces. In: *Automated Software Engineering, 2004. Proceedings. 19th International Conference on*, 2004, p. 310–315.

BREU, S.; ZIMMERMANN, T. Mining aspects from version history. In: *Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on*, 2006, p. 221–230.

BRUNTINK, M.; VAN DEURSEN, A.; VAN ENGELEN, R.; TOURWE, T. On the use of clone detection for identifying crosscutting concern code. *Software Engineering, IEEE Transactions on*, v. 31, n. 10, p. 804–818, 2005.

BRUNTINK, M.; VAN DEURSEN, A.; TOURWE, T.; VAN ENGELEN, R. An evaluation of clone detection techniques for crosscutting concerns. In: *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, 2004, p. 200–209.

CANFORA, G.; CERULO, L.; PENTA, M. D. On the use of line co-change for identifying crosscutting concern code. In: *2006 22nd IEEE International Conference on Software Maintenance*, 2006, p. 213–222.

CECCATO, M.; TONELLA, P. Dynamic aspect mining. *IET Software*, v. 3, n. 4, p. 321–336, 2009.

CHANG, H.-F.; LU, S.-Y. Decomposition and traceability in software design. In: *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, 2009, p. 13–18.

COCKBURN, A. *Writing effective use cases*. 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

COJOCAR, G. Aspect mining. past, present, future. In: *Studia Univ. Babes-Bolyai, Informatica*, v. LVII, 2012.

COJOCAR, G.; CZIBULA, G. On clustering based aspect mining. In: *Intelligent Computer Communication and Processing, 2008. ICCP 2008. 4th International Conference on*, 2008, p. 129–136.

DA SILVA, B. Um método de refatoração para modularização de interesses transversais. 2009.

D'AMBROS, M.; LANZA, M. Reverse engineering with logical coupling. In: *2006 13th Working Conference on Reverse Engineering*, 2006, p. 189–198.

DELFIM, F. M. Uma abordagem usando visualização de software como apoio à refatoração para aspectos. 2013.

DEY, I. *Grounding grounded theory: Guidelines for qualitative inquiry*. Academic Press, 1999.

DUCASSE, S.; GIRBA, T.; KUHN, A. Distribution map. In: *Proceedings of the 22Nd IEEE International Conference on Software Maintenance*, Washington, DC, USA: IEEE Computer Society, 2006, p. 203–212.

DYBA, T.; DINGSAYR, T. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, v. 50, p. 833 – 859, 2008.

EADDY, M.; ZIMMERMANN, T.; SHERWOOD, K.; GARG, V.; MURPHY, G.; NAGAPPAN, N.; AHO, A. Do crosscutting concerns cause defects? *Software Engineering, IEEE Transactions on*, v. 34, n. 4, p. 497–515, 2008.

EMAM, K. E. A methodology for validating software product metrics. 2000.

FIGUEIREDO, E.; SILVA, B.; SANT'ANNA, C.; GARCIA, A.; WHITTLE, J.; NUNES, D. Crosscutting patterns and design stability: An exploratory analysis. In: *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, 2009, p. 138–147.

FILHO, F. C.; CACHO, N.; FIGUEIREDO, E.; MARANHÃO, R.; GARCIA, A.; RUBIRA, C. M. F. Exceptions and aspects: The devil is in the details. In: *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA: ACM, 2006, p. 152–162.

FOWLER, M.; KENT, B.; BRANT, J. *Refactoring: Improving the design of existing code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Design patterns: Elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

GARCIA, A.; SANT'ANNA, C.; FIGUEIREDO, E.; KULESZA, U.; LUCENA, C.; VON STAA, A. Modularizing design patterns with aspects: A quantitative study. In: *Proceedings of the 4th International Conference on Aspect-oriented Software Development*, New York, NY, USA: ACM, 2005, p. 3–14.

GOSAIN, A.; SHARMA, G. *A survey of dynamic program analysis techniques and tools* Cham: Springer International Publishing, p. 113–122, 2015.

GREENWOOD, P.; BARTOLOMEI, T.; FIGUEIREDO, E.; DOSEA, M.; GARCIA, A.; CACHO, N.; SANT'ANNA, C.; SOARES, S.; BORBA, P.; KULESZA, U.; RASHID, A. On the impact of aspectual decompositions on design stability: An empirical study. In: *Proceedings of the 21st European Conference on Object-Oriented Programming*, Berlin, Heidelberg: Springer-Verlag, 2007, p. 176–200.

HARBULOT, B. *Separating concerns in scientific software using aspect-oriented programming*. Tese de Doutoramento, University of Manchester, 2006.

HUANG, J.; LU, Y.; YANG, J. Aspect mining using link analysis. In: *2010 Fifth International Conference on Frontier of Computer Science and Technology*, 2010, p. 312–317.

IEEE Ieee recommended practice for architectural description of software-intensive systems. *IEEE Std 1471-2000*, p. i–23, 2000.

ISHIO, T.; DATE, H.; MIYAKE, T.; INOUE, K. Mining coding patterns to detect crosscutting concerns in java programs. In: *2008 15th Working Conference on Reverse Engineering*, 2008, p. 123–132.

JUNIOR, P. A. P.; MENDES, W.; DE CAMARGO, V. V.; PENTEADO, R. A. D.; COSTA, H. A. X. Mining crosscutting concerns with comscid: A rule-based customizable mining tool. In: *2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI)*, 2012, p. 1–9.

KELLENS, A.; MENS, K.; TONELLA, P. Transactions on aspect-oriented software development iv. cáp. A Survey of Automated Code-level Aspect Mining Techniques, Berlin, Heidelberg: Springer-Verlag, p. 143–162, 2007.

KHAN, T.; BARTHEL, H.; EBERT, A.; LIGGESMEYER, P. Visual analytics of software structure and metrics. In: *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*, 2015, p. 16–25.

KHOSHGOFTAAR, T.; ALLEN, E.; JONES, W.; HUDEPOHL, J. Classification tree models of software quality over multiple releases. In: *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on*, 1999, p. 116–125.

KICZALES, G.; HILSDALE, E. Aspect-oriented programming. *SIGSOFT Softw. Eng. Notes*, v. 26, n. 5, p. 313–, 2001.

KRINKE, J. Mining control flow graphs for crosscutting concerns. In: *Reverse Engineering, 2006. WCRE '06. 13th Working Conference on*, 2006, p. 334–342.

KRINKE, J. Mining execution relations for crosscutting concerns. *IET Software*, v. 2, n. 2, p. 65–78, 2008.

LADDAD, R. *Aspectj in action: Practical aspect-oriented programming*. Greenwich, CT, USA: Manning Publications Co., 2003.

LI, X.; PRASAD, C. Effectively teaching coding standards in programming. In: *Proceedings of the 6th Conference on Information Technology Education*, SIGITE '05, New York, NY, USA: ACM, 2005, p. 239–244 (*SIGITE '05*,).

MAISIKELI, S. G.; MITROPOULOS, F. J. Aspect mining using self-organizing maps with method level dynamic software metrics as input vectors. In: *2010 2nd International Conference on Software Technology and Engineering*, 2010, p. V1–212–V1–217.

MARÇAL, I.; GARCIA, R. E.; ELER, D. M.; OLIVETE JUNIOR, C.; CORREIA, R. C. M. *Techniques for the identification of crosscutting concerns: A systematic literature review* Cham: Springer International Publishing, p. 569–579, 2016.

MARCAL, I.; GARCIA, R. E.; ELER, M. D.; CORREIA, R. C. M.; JUNIOR, C. O. A framework to combine different granularity-level crosscutting concern identification techniques. *No Prelo*, 2017.

MARIN, M.; VAN DEURSEN, A.; MOONEN, L. Identifying aspects using fan-in analysis. In: *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, 2004, p. 132–141.

MARIN, M.; MOONEN, L.; VAN DEURSEN, A. A classification of crosscutting concerns. In: *Proceedings of the 21st IEEE International Conference on Software Maintenance*, Washington, DC, USA: IEEE Computer Society, 2005, p. 673–676.

MARIN, M.; MOONEN, L.; VAN DEURSEN, A. *Documenting typical crosscutting concerns*. Relatório Técnico, Deft University of Technology, 2007.

MARIN, M.; MOONEN, L.; DEURSEN, A. V. A common framework for aspect mining based on crosscutting concern sorts. In: *2006 13th Working Conference on Reverse Engineering*, 2006, p. 29–38.

MARINESCU, R. Detection strategies: metrics-based rules for detecting design flaws. In: *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, 2004, p. 350–359.

MASSICOTTE, P.; BADRI, L.; BADRI, M. Towards a tool supporting integration testing of aspect-oriented programs. v. 6, n. 1, p. 67–89, 2007.

MCFADDEN, R.; MITROPOULOS, F. Aspect mining using model-based clustering. In: *Southeastcon, 2012 Proceedings of IEEE*, 2012, p. 1–8.

MENS, K.; KELLENS, A.; KRINKE, J. Pitfalls in aspect mining. In: *Reverse Engineering, 2008. WCRE '08. 15th Working Conference on*, 2008, p. 113–122.

MOLDOVAN, G.; SERBAN, G. Aspect mining using a vector-space model based clustering approach. In: *Proceedings of Workshop on Linking Aspect Technology and Evolution (LATE'06)*, 2006.

MULDER, F. Identifying cross-cutting concerns using software repository mining. 2009.

NGUYEN, T. T.; NGUYEN, H. V.; NGUYEN, H. A.; NGUYEN, T. N. Aspect recommendation for evolving software. In: *Proceedings of the 33rd International Conference on Software Engineering*, New York, NY, USA: ACM, 2011, p. 361–370.

PARNAS, D. L. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, v. 15, n. 12, p. 1053–1058, 1972.

QU, L.; LIU, D. Aspect mining using method call tree. In: *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, 2007a, p. 407–412.

QU, L.; LIU, D. Extending dynamic aspect mining using formal concept analysis. In: *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*, 2007b, p. 564–567.

QU, L.; YIN, G.; YANG, J.; HOU, X. Aspect mining using relative reduced concept lattice. In: *2011 3rd International Conference on Computer Research and Development*, 2011, p. 183–187.

REIMANIS, D.; IZURIETA, C. Towards assessing the technical debt of undesired software behaviors in design patterns. In: *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*, 2016, p. 24–27.

RODRIGUES, A.; ALBUQUERQUER, C. H. L.; BENTO, C. A.; VIEIRA, J. M.; SILVA, J. G. O. B. Grounded theory: Problemas de alicerçagem. 2004.

RONDON, J.; PASTOR, A. Applying grounded theory to study the implementation of an inter-organizational information system. v. 5, 2007.

ROY, C. K.; UDDIN, M. G.; ROY, B.; DEAN, T. R. Evaluating aspect mining techniques: A case study. In: *15th IEEE International Conference on Program Comprehension (ICPC '07)*, 2007, p. 167–176.

SHEPHERD, D.; POLLOCK, L.; TOURWÉ, T. Using language clues to discover crosscutting concerns. In: *Proceedings of the 2005 Workshop on Modeling and Analysis of Concerns in Software*, MACS '05, New York, NY, USA: ACM, 2005, p. 1–6 (*MACS '05*,).

DA SILVA, B.; FIGUEIREDO, E.; GARCIA, A.; NUNES, D. Refactoring of crosscutting concerns with metaphor-based heuristics. *Electronic Notes in Theoretical Computer Science*, v. 233, p. 105 – 125, 2009.

STRAUSS, A.; CORBIN, J. *Basics of qualitative research: grounded theory procedures and techniques*. Sage Publications, 1990.

THAMMANO, A.; KESISUNG, P. Enhancing k-means algorithm for solving classification problems. In: *2013 IEEE International Conference on Mechatronics and Automation*, 2013, p. 1652–1656.

TONELLA, P.; CECCATO, M. Aspect mining through the formal concept analysis of execution traces. In: *11th Working Conference on Reverse Engineering*, 2004, p. 112–121.

TOURWE, T.; MENS, K. Mining aspectual views using formal concept analysis. In: *Source Code Analysis and Manipulation, 2004. Fourth IEEE International Workshop on*, 2004, p. 97–106.

WALKER, D.; MYRICK, F. Grounded theory: An exploration of process and procedure. *Qualitative Health Research*, v. 16, n. 4, p. 547–559, 2006.

WILLIG, C. *Introducing qualitative research in psychology*. 3 ed. 2013.

YAMAGUCHI, J. K. Diretrizes para a escolha de técnicas de visualização aplicadas no processo de extração do conhecimento. 2010.

ZHANG, C.; JACOBSEN, H.-A. Efficiently mining crosscutting concerns through random walks. In: *AOSD '07: Proceedings of the 6th international conference on Aspect-oriented software development*, New York, NY, USA: ACM Press, 2007, p. 226–238.

ZHANG, C.; JACOBSEN, H.-A. Mining crosscutting concerns through random walks. *IEEE Transactions on Software Engineering*, v. 38, n. 5, p. 1123–1137, 2012.

ZHANG, D.; GUO, Y.; CHEN, X. Automated aspect recommendation through clustering-based fan-in analysis. In: *Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on*, 2008, p. 278–287.

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

TERMO DE REPRODUÇÃO XEROGRÁFICA

Autorizo a reprodução xerográfica do presente Trabalho de Conclusão, na íntegra ou em partes, para fins de pesquisa.

São José do Rio Preto, 17 / 07 / 2017

Assinatura do autor