MÉTODO DE EXTRAÇÃO DE LINHA DE PRODUTOS DE SOFTWARE BASEADO EM TESTES

ALCEMIR RODRIGUES SANTOS

MÉTODO DE EXTRAÇÃO DE LINHA DE PRODUTOS DE SOFTWARE BASEADO EM TESTES

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Eduardo Magno Lages Figueiredo
Coorientador: Pedro de Alcântara dos Santos Neto

Belo Horizonte
Março de 2013

Santos, Alcemir Rodrigues
S237m Método de extração de linha de produtos de software baseado em testes / Alcemir Rodrigues Santos. — Belo Horizonte, 2013
xxv, 85 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Eduardo Magno Lages Figueiredo
Coorientador: Pedro de Alcântara dos Santos Neto

1. Computação – Teses. 2. Engenharia de software – Teses. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*32 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Método de extração de linha de produtos de software baseado em testes

ALCEMIR RODRIGUES SANTOS

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Eduardo Magno Lages Figueiredo
PROF. EDUARDO MAGNO LAGES FIGUEIREDO - Orientador
Departamento de Ciência da Computação - UFMG

PROF. PEDRO DE ALCANTARA DOS SANTOS NETO - Coorientador
Departamento de Informática e Estatística - UFPI

PROF. EDUARDO SANTANA DE ALMEIDA
Departamento de Ciência da Computação - UFBA

PROF. MARCO TÚLIO DE OLIVEIRA VALENTE
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 01 de março de 2013.

A mim.

Agradecimentos

Agradecer é difícil. Principalmente quando se tem muito o que agradecer. Tentarei ser breve.

Ao agradecer, muitos nomes vêm à mente, outros nem tanto. Helvécio, Maxwell, Pablo, Seu Jacinto, à vocês e suas famílias, obrigado. Aos companheiros do Judô Renato Marques e do "Fasta, bem aí, fasta!" que sempre no retorno para casa me ajudavam a recarregar às baterias para continuar lutando, esse trecho é pra vocês, obrigado.

Enquanto uns vem à mente, outros não saem dela. E por maior que seja o tempo que passe longe destes sua ligação aumenta ainda mais. PAI, MÃE, IRMÃO seu amor e apoio incondicionais me tornaram ainda mais persistente. Um OBRIGADO em letras garrafais para vocês.

Existem ainda os que caminharam com você durante todo o tempo desta caminhada, mesmo que em times diferentes, nunca jogaram contra. Às amizades feitas aqui na UFMG, professores e orientadores, muito obrigado.

Existe também alguém que mesmo sem os laços de sangue, sentia uma verdadeira necessidade de estar sempre ali, ao lado, querendo saber de todos os detalhes, dando força a cada novo desespero. De mãos dadas, de peito aberto. Literalmente, lutando pela vitória junto comigo. Jômazya Frota, o meu sincero muito obrigado!

"Remember, misery is comfortable.
It's why so many people prefer it.
Happiness takes effort."
(David Wong)

Resumo

Muitos sistemas de software foram desenvolvidas como produtos individuais antes da abordagem de Linha de Produtos de Software (LPS) emergir. Embora algumas abordagens promissoras foram propostas, extrair uma LPS de produtos de software existentes ainda é caro e demorado. Este trabalho apresenta um método para extrair uma LPS a partir de produtos únicos que se baseia em testes de software já desenvolvidos. Nós avaliamos testes como o principal meio para localizar código de características e diferentes tipos de artefatos existentes para apoiar a técnica de localização baseada em testes. Realizou-se três estudos de caso iniciando a partir da derivação do modelo de características da LPS até a localização do código das características. Os nossos resultados indicaram (i) interessantes índices de precisão de localização de sementes das características, semente é uma pequena porção do código de característica que permite a identificação da porção restante, e (ii) boas taxas de cobertura para localizar o código completo da característica.

Palavras-chave: Localização de características, Linha de Produtos de Software, Teste de software.

Abstract

Many software systems have been developed as single products before Software Product Lines (SPLs) have emerged. Although some promising approaches have been proposed, extracting an SPL from existing software products is still expensive and time consuming. This work presents a method to extract an SPL from single products that relies on software testing already developed. We evaluate testing as the main mean to locate feature code and different sorts of existing artifacts to support the test-based location. We conduct three case studies starting from the derivation of the SPL feature model to the feature code location. Our results indicate (i) interesting rates of precision for feature seed location, where seed means a small portion of the feature code that allows the identification of the remaining portion, and (ii) good rates of recall for locating the whole feature code.

Keywords: Feature Location, Software Testing, Software Product Line.

Lista de Figuras

2.1 Relação entre variabilidade no mundo real e um modelo do mundo real. . . 7

4.1 Ilustração do método proposto para localização de características. 26
4.2 Modelo de caso de uso 28
4.3 Mapeamento entre suítes de teste para características através dos requisitos. 32
4.4 Processo de 4 etapas para localização do código de uma característica. . . 33
4.5 Parte de código executado por uma suíte de teste. 34
4.6 Interseção entre os conjuntos de código executados pelas suítes de teste. . . 35
4.7 União entre os conjuntos de código executado pelas suítes de teste. 36

5.1 Relacionamento TaBuLeTa-ConcernMapper-EclEmma-Eclipse. 39
5.2 Definição do modelo de interesse do ConcernMapper na forma Bakus-Naur. 40
5.3 Modelo de uma suíte de testes gerada pela TaBuLeTa. 41
5.4 Interação com usuário da visão de mapeamento. 42
5.5 Visualização do Mapeamento de Testes para Características. 42
5.6 Interações do usuário para o lançamento de ações na TaBuLeTa. 43
5.7 Visualização do menu de lançamento de ações da TaBuLeTa. 43
5.8 Visualização de Métricas. 44

6.1 Ilustração das métricas utilizadas. 49
6.2 Modelo de características do JBook. 55
6.3 Modelo de características do WebStore. 55
6.4 Modelo de características do WEKA. 55
6.5 “Precisão” para o JBook na localização parcial do código-fonte. 56
6.6 “Acurácia” para o JBook na localização parcial do código-fonte. 56
6.7 “Precisão” para o WebStore na localização parcial do código-fonte. 58
6.8 “Acurácia” para o WebStore na localização parcial do código-fonte. 58
6.9 “Precisão” para o JBook na localização completa do código-fonte. 59
6.10 “Cobertura” para o JBook na localização completa do código-fonte. 60

6.11 “Acurácia” para o JBook na localização completa do código-fonte. 61
6.12 “Precisão” para o WebStore na localização completa do código-fonte. . . . 62
6.13 “Cobertura” para o WebStore na localização completa do código-fonte. . . 63
6.14 “Acurácia” para o WebStore na localização completa do código-fonte. . . . 64
6.15 “Precisão” para o WEKA na localização parcial do código-fonte. 65
6.16 “Precisão” para o WEKA na localização completa do código-fonte. 66
6.17 “Cobertura” para o WEKA na localização completa do código-fonte. 66

A.1 Botões na visão *Text2Feature Mapping*. 85

Lista de Tabelas

3.1 Lista de fontes de pesquisa. 15
3.2 Comparativo entre os trabalhos relacionados. 24

4.1 Relacionamento entre Requisitos e Características. 31

5.1 Elementos de implementação da TaBuLeTa. 41

6.1 Características analisadas em cada um dos sistemas alvo. 52
6.2 Exemplos Características identificadas pelas heurísticas. 54
6.3 Acertos e erros na anotação das linhas de código no JBook. 61
6.4 Acertos e erros na anotação das linhas de código no WebStore. 63

Lista de Acrônimos

ACM	Associação para Maquinaria de Computação
API	Interface de Programação de Aplicativos
CAVE	Extração de Similaridade e Variabilidade
DFT	Rastro Dinâmico de Característica
IDE	Ambiente de Desesenvolvimento Integrado
J2EE	Java, Edição Empresarial
J2ME	Java, Edição Micro
KLOC	Mil Linhas de Código
LPS	Linha de Produtos de Software
SWEBOK	Manual do Conhecimento de Engenharia de Software
TaBuLeTa	Ferramenta de Localização de Características Baseado em Testes
WEKA	Ambiente para Análise de Conhecimento de Waikato

Sumário

Agradecimentos **ix**

Resumo **xiii**

Abstract **xv**

Lista de Figuras **xvii**

Lista de Tabelas **xix**

Lista de Acrônimos **xxi**

1 Introdução **1**

1.1 Motivação 1

1.2 Esboço da Solução 2

1.3 Escopo 2

1.4 Contribuições 3

1.5 Organização desta Dissertação 4

2 Fundamentação Teórica **5**

2.1 Linha de Produtos de Software 5

2.1.1 Variabilidade 6

2.1.2 Engenharia de Domínio 8

2.1.3 Engenharia de Aplicação 8

2.2 Teste de Software 9

2.2.1 Níveis do Teste 9

2.2.2 Reuso de Testes 10

2.3 Resumo 11

3 Revisão de Literatura **13**

3.1 Formulação da Questão de Pesquisa . . . 13
3.2 Seleção de Fontes de Pesquisa . . . 14
3.3 Seleção de Estudos . . . 15
3.4 Sumarização de Resultados . . . 15
3.4.1 Manutenção de Linha de Produtos de Software (LPS) . . . 16
3.4.2 Testes e LPS . . . 17
3.4.3 Localização de Características . . . 17
3.4.4 Migração de Software para Linha de Produtos de Software . . . 20
3.5 Trabalhos Relacionados: em poucas palavras . . . 21
3.6 Resumo . . . 23

4 Método Proposto **25**
4.1 Visão Geral do Método . . . 25
4.2 Construção do Modelo de Características . . . 27
4.3 Mapeamento de Requisitos para Características . . . 30
4.4 Mapeamento de Testes para Características . . . 31
4.5 Localização das Características no Código-fonte . . . 33
4.5.1 Localização parcial do código-fonte . . . 34
4.5.2 Localização completa do código-fonte . . . 35
4.6 Resumo . . . 36

5 TaBuLeTa **37**
5.1 Arquitetura . . . 38
5.2 Projeto e Implementação . . . 39
5.3 TaBuLeTa em Ação . . . 41
5.4 Resumo . . . 44

6 Avaliação **47**
6.1 Definição . . . 47
6.2 Planejamento . . . 49
6.3 Sistemas Alvo . . . 50
6.3.1 JBook . . . 51
6.3.2 WebStore . . . 51
6.3.3 WEKA . . . 51
6.4 Coleta de Dados . . . 52
6.5 Construção do Modelo de Características . . . 53
6.6 Localização Parcial do Código-fonte . . . 54
6.6.1 JBook . . . 55

6.6.2 WebStore 57
6.7 Localização Completa do Código-fonte 59
6.7.1 JBook 59
6.7.2 WebStore 62
6.8 Avaliação de Escalabilidade: Estudo de Caso WEKA 64
6.9 Discussão das Questões de Pesquisa 67
6.9.1 Localização de Semente 67
6.9.2 Localização do Código da Característica 67
6.10 Limitações e Ameaças à Validade 68
6.10.1 Validade Interna 68
6.10.2 Validade Externa 69
6.10.3 Validade de Construção 70
6.10.4 Validade de Conclusão 71
6.11 Resumo 71

7 Conclusão **73**
7.1 Considerações Finais 73
7.2 Trabalhos Futuros 74

Referências Bibliográficas **77**

Anexo A Como usar a ferramenta TaBuLeTa **83**
A.1 Requisitos Para Instalação 83
A.2 Instalação 83
A.3 Como utilizar 84

Capítulo 1

Introdução

Este capítulo apresenta o contexto em que o presente trabalho está inserido e suas pretensões, bem como a sua motivação (Seção 1.1). Além disto, apresenta-se um esboço da solução (Seção 1.2) e uma discussão sobre o escopo do trabalho (Seção 1.3). Com isso, pretende-se elucidar os questionamentos iniciais sobre o mesmo. São enumeradas algumas das contribuições (Seção 1.4) e ao final detalha-se como o restante do trabalho está organizado (Seção 1.5).

1.1 Motivação

A engenharia de Linha de Produtos de Software (LPS) é um paradigma de desenvolvimento de aplicações de software usando uma arquitetura comum e componentes personalizados a fim de satisfazer necessidades específicas das partes interessadas [Pohl et al., 2005]. Este paradigma evita a duplicação de partes de código de produtos únicos, que são sistemas previamente desenvolvidos sem o planejamento de reuso sistemático. Muitos sistemas de software foram desenvolvidos antes da abordagem de LPS ser concebida. Existe, portanto, a possibilidade de que a equipe de desenvolvimento do sistema de software perceba que a adoção desta abordagem possa lhe trazer benefícios. Entretanto, reconstruir o sistema inteiro como uma LPS pode ser muito oneroso, o que poderia fazer a equipe dispensar a adoção da abordagem.

Este problema merece atenção das comunidades acadêmica e industrial em função da necessidade da redução dos custos envolvidos. Um deles é a refatoração de todo o código fonte utilizando a abordagem LPS. Caso o sistema seja muito grande, a tarefa de refatoração completa pode tornar-se prolongada e consequentemente passível de erros [Kästner et al., 2011]. Além disto, os arquitetos de software precisam mapear as características atualmente implementadas para a docu-

mentação de arquitetura para verificar se estão de acordo com o que foi especificado. Em LPS, uma característica é um incremento na funcionalidade de um programa [Batory, 2005]. Assim, a existência de interesses transversais (certas funcionalidades de um sistema cujas implementações ficam espalhadas por diferentes módulos [Kiczales et al., 1997, Bergmans & Aksit, 2001]) dificulta a localização manual da implementação destas características [Adams et al., 2010].

1.2 Esboço da Solução

O método proposto neste trabalho para extração de LPS com foco na localização de características[1] promove o reuso de artefatos de requisitos e testes de software. Além disso, o método é apoiado por uma ferramenta que provê automação parcial. A automação permite a redução do esforço necessário na execução da tarefa de extração de LPS.

Com este trabalho, pretende-se minimizar problema da refatoração de sistemas desenvolvidos como produtos únicos. Isto é feito através da automação da localização de características fazendo o reuso de artefatos de teste já desenvolvidos para o sistema. Realizou-se um levantamento bibliográfico para entender as alternativas já propostas para que seja possível identificar suas limitações e propor uma nova abordagem viável e de baixo custo.

Tem-se como hipótese que o reuso dos testes de software podem ser de grande valia na identificação das características do software. Nossa motivação em reusar artefatos de teste se deu a partir do sucesso obtido em estudos anteriores [Santos et al., 2010, Fé et al., 2010, Santos et al., 2011] envolvendo o reuso de testes em diferentes contextos.

1.3 Escopo

Como a localização de características é apenas parte da extração de uma LPS, um conjunto de fatores relacionados serão deixados de fora do escopo deste trabalho. Assim, as questões enumerados abaixo não são diretamente estudadas:

- Impacto em outras etapas do processo de extração de uma LPS. Apesar do método proposto ter interesse na extração, ele apoia principalmente a localização de características. O principal interesse é encontrar o código que implementa

[1] Neste documento o termo 'característica(s)' refere-se à tradução do termo 'feature' que é comumente utilizado no contexto de linha de produtos de software em inglês.

determinada característica. Portanto, está fora do escopo o esforço adicional necessário para finalizar a extração da LPS.

- Tipos de usuários. Inicialmente, qualquer indivíduo (analista de teste, gerente de teste, arquiteto de software, desenvolvedor, *etc.*) efetivamente envolvido no processo de extração da LPS pode fazer uso do método proposto. No entanto, está fora do escopo deste trabalho, dispor de dados para fundamentar a melhor alocação de quaisquer deles à localização de característica com o método proposto.
- O método proposto é dividido em quatro passos e avaliar passos iniciais do método está fora do escopo. O primeiro passo do método proposto já foi avaliado em outros trabalhos. O segundo e o terceiro já exigiriam um trabalho adicional e que seria necessário tempo adicional ao disponível para um mestrado.
- Documentar a variabilidade adicionada na extração de LPS. Além do modelo de características derivado no primeiro passo não é definido artefatos adicionais para a documentação da linha de produtos construída a partir do uso do método proposto.

1.4 Contribuições

Como resultado do trabalho apresentado nesta dissertação, as seguintes contribuições podem ser destacadas:

- Uma **revisão de literatura sobre os métodos de localização de características** de software desenvolvido como produtos únicos, seja para a utilização em LPS, seja para a manutenção e evolução de software. Isto provê à comunidade acadêmica o estado-da-arte neste campo (Capítulo 3);
- A **definição de um novo método de localização de características** de software desenvolvido como produto único que eventualmente necessite de uma extração para LPS (Capítulo 4);
- O **desenvolvimento de uma ferramenta** para prover apoio ferramental com automação parcial do método proposto (Capítulo 5);
- A **definição, planejamento e análise de um estudo experimental** para verificar a viabilidade e eficiência do método proposto (Capítulo 6).

Em adição às contribuições enumeradas, um artigo apresentando parte dos resultados desta dissertação foi publicado:

Santos, A. R. *et.al.*. "Uso de testes na identificação de características e extração de linha de produtos de software". Anais do II Workshop de Teses e Dissertações do III Congresso Brasileiro de Software: Teoria e Prática (CBSoft), Natal, 2012.

Santos, A. R. *et.al.*. "Test-based SPL extraction: an exploratory study". In proceedings of 28^{th} Symposium on Applied Computing (SAC'2013). Coimbra - Portugal. 2013.

1.5 Organização desta Dissertação

O resto desta dissertação está organizado como segue:

Capítulo 2: detalha a fundamentação teórica necessária para o entendimento dos conceitos utilizados neste trabalho como, linha de produtos e teste de software.

Capítulo 3: descreve uma revisão da literatura dos trabalhos relacionados ao tema desta dissertação, com foco na localização de características.

Capítulo 4: apresenta o método proposto para extração de LPS com foco na localização de características em código desenvolvido como produto único.

Capítulo 5: apresenta a ferramenta desenvolvida para prover apoio automatizado ao método proposto.

Capítulo 6: descreve o planejamento, a execução e apresenta os resultados de experimentos realizados durante a avaliação do método proposto.

Capítulo 7: finaliza esta dissertação apresentando as conclusões, limitações e direções para trabalhos futuros.

Anexo A: apresenta como instalar e utilizar a ferramenta desenvolvida neste trabalho e detalhada no Capítulo 5.

Capítulo 2

Fundamentação Teórica

O bom entendimento de um trabalho acadêmico começa pelo entendimento dos conceitos que o permeiam. Este capítulo apresenta conceitos relevantes para este trabalho. A Seção 2.1 introduz os conceitos de Linha de Produtos de Software (LPS). A Seção 2.2 apresenta Teste de Software e seus diferentes níveis. Por fim, é apresentado um resumo do capítulo na Seção 2.3.

2.1 Linha de Produtos de Software

A engenharia de LPS é um paradigma de desenvolvimento de aplicações de software usando uma arquitetura comum e componentes personalizados a fim de satisfazer necessidades específicas das partes interessadas [Pohl et al., 2005]. Os componentes personalizados adicionam funcionalidades a um ou mais produtos da linha. Isto é conhecido como variabilidade e será detalhada na Seção 2.1.1. Este paradigma permite o reuso eficiente de software e tem ganho popularidade com o passar dos anos.

Adotar LPS é uma decisão, em geral, de caráter econômico e reforçada pela possibilidade de produção em larga escala, reuso, melhoria no processo de desenvolvimento, entre outras [van der Linden et al., 2007]. Por outro lado, a adoção de LPS possui a desvantagem da demanda de investimentos extras na construção de software reutilizável, além de mudanças na organização.

Superados os obstáculos da implantação, a utilização da abordagem de LPS no desenvolvimento de software traz consigo algumas vantagens, como, *(i)* redução do custo de desenvolvimento, *(ii)* aumento da qualidade dos produtos e a *(iii)* redução do tempo de entrega. Estudos indicam que a partir de um conjunto de três produtos a abordagem de LPS já mostra benefícios sobre abordagens de produtos únicos

[van der Linden et al., 2007, Pohl et al., 2005]. A seguir são descritas algumas características as quais acredita-se que contribuem para isto.

Redução do custo de desenvolvimento: o reuso de artefatos, incluindo código fonte e documentação, leva a redução do trabalho repetido e do custo de evolução da linha de produtos;

Aumento da qualidade: o uso compartilhado de componentes comuns a todos os produtos proporciona a redução da quantidade de código a ser testado. Isto possibilita o aumento do tempo disponível para as atividades de testes e que consequentemente aumenta o nível de qualidade do produto.

Redução do tempo de entrega: a equipe de desenvolvimento deverá desenvolver apenas as novas características requisitadas e suas interfaces com o restante do sistema ao invés de um novo produto por inteiro.

De acordo com Pohl e outros, a experiência adquirida com projetos de reuso de software na década de 90 mostra que sem um direcionamento apropriado, os custos de reuso de software tornam-se maiores que desenvolver artefatos do zero [Pohl et al., 2005]. Portanto, planejar o reuso é de grande importância durante todo o processo de desenvolvimento de software. Além disto, a customização em massa de produtos de software requer o gerenciamento da variabilidade entre diferentes produtos de software. Neste ponto, a engenharia de LPS apresenta-se atualmente como uma das formas mais eficientes de planejar o reuso de artefatos durante o desenvolvimento [Pohl et al., 2005].

2.1.1 Variabilidade

Variabilidade é um dos conceitos chave em LPS, consequentemente, para este trabalho também, vez que este tem como objetivo prover apoio à extração de uma linha de produtos. Por extração de LPS entende-se a marcação do código-fonte com a intenção de produzir diferentes variantes (ver Definição 2) da linha de produtos dado um ponto de variação (ver Definição 1). Ou seja, produzir a variabilidade de uma LPS a partir de um único produto ou de um conjunto deles pertencentes ao mesmo domínio.

Em linguagem natural, o termo *variabilidade* refere-se à habilidade ou a tendência de mudar. No caso de engenharia de LPS a variabilidade não acontece por acaso, mas sim produzida com um propósito [Pohl et al., 2005].

Definição 1. *Ponto de variação é uma representação de um sujeito de variabilidade dentro de um artefato de domínio enriquecido por informação contextual (Figura 2.1). Um sujeito de variabilidade é um item variável do mundo real ou uma propriedade de tal item. [Pohl et al., 2005]*

Definição 2. *Variante é uma representação de um objeto de variabilidade dentro de um artefato de domínio. Um objeto de variabilidade é uma instância particular de um sujeito de variabilidade (Figura 2.1) [Pohl et al., 2005].*

O método aqui proposto (Capítulo 4) busca a extração de uma LPS através da localização de características. Essas características podem representar uma variabilidade externa (Definição 3) ou interna (Definição 4). O que será feito a partir da identificação de pontos de variação.

Definição 3. *Variabilidade externa é a variabilidade dos artefatos de domínio que é perceptível para o usuário (ver Figura 2.1). [Pohl et al., 2005]*

Definição 4. *Variabilidade interna é a variabilidade dos artefatos de domínio que não é perceptível para o usuário (ver Figura 2.1). [Pohl et al., 2005]*

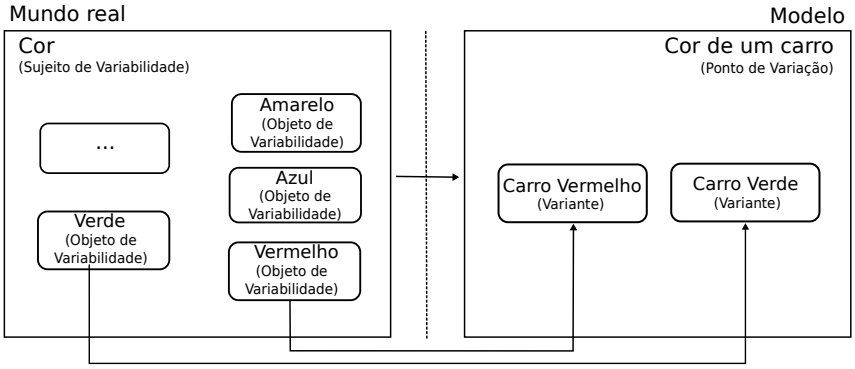

Figura 2.1. Relação entre variabilidade no mundo real e um modelo do mundo real. Adaptado de Pohl e outros [Pohl et al., 2005]

O relacionamento entre as conceitos apresentados pelas Definições 1, 2, 3 e 4 é inlustrado na Figura 2.1. No mundo real, um sujeito de variabilidade "Cor" possui diversas instâncias denominadas objetos de variabilidade. Cada sujeito de variabilidade no mundo real tem um ponto de variação correspondente em um modelo. No caso da Figura 2.1 o sujeito "Cor" corresponde ao ponto de variação "Cor de um carro". Da mesma forma, cada objeto de variabilidade pode ter uma variante associada no modelo. No exemplo utilizado as variante seriam os carros "Vermelho" e "Verde".

O paradigma de engenharia de LPS é dividido em dois processos: a engenharia de domínio (Seção 2.1.2) e a engenharia de aplicação (Seção 2.1.3). Cada um deles é diretamente relacionado com variabilidade. O primeiro define os pontos de variação da linha de produtos e o segundo explora esta definição possibilitando a o reuso sistemático de artefatos da linha de produtos. Esses processos são comentados a seguir.

2.1.2 Engenharia de Domínio

Este é o processo em que a plataforma comum a ser reutilizada é estabelecida, bem como a variabilidade existente na linha de produtos. Todos os tipos de artefatos de software (requisitos, implementações, testes, etc.) compõem esta plataforma. Assim, o reuso sistemático e consistente destes artefatos pode ser facilitado com o estabelecimento de elos de rastreabilidade entre eles [Pohl et al., 2005].

Definição 5. *Engenharia de domínio é o processo de engenharia de linha de produtos de software no qual são definidas e realizadas a comunalidade e a variabilidade [Pohl et al., 2005].*

Assim, os principais objetivos do processo de engenharia de domínio de uma LPS são *(i)* definir a variabilidade existente, *(ii)* definir o escopo, ou seja, o conjunto de aplicações para a qual ela foi planejada, e *(iii)* definir e construir os artefatos para realizar variabilidade desejada. O método proposto procura cobrir estes pontos em seus passos iniciais

2.1.3 Engenharia de Aplicação

A engenharia de aplicação tem a responsabilidade de derivação dos produtos da linha definidos no processo de engenharia de domínio. Assim como, a garantia da construção da variabilidade de acordo com as necessidades específicas da aplicação também é responsabilidades deste processo.

Definição 6. *Engenharia de aplicação é o processo de engenharia de linha de produtos de software no qual os produtos da linha são construídos através do reuso dos artefatos do domínio e explorando a variabilidade da linha de produtos [Pohl et al., 2005].*

Assim, os principais objetivos do processo de engenharia de aplicação de uma LPS são no momento do desenvolvimento de uma aplicação da linha de produtos *(i)* alcançar um alto reuso dos recursos de domínio, *(ii)* explorar a comunalidade e a variabilidade,*(iii)* documentar os artefatos da aplicação, *(iv)* vincular a variabilidade

de acordo com suas necessidades, e *(v)* estimar o impacto das diferenças entre requisitos de aplicação e domínio na arquitetura, componentes e testes. O passo final do método proposto é o encarregado da engenharia de aplicação.

2.2 Teste de Software

O método proposto é baseado em teste de software, portanto, esta seção apresenta esse conceito com mais detalhes. Teste de software consiste na verificação dinâmica do comportamento de um programa dado um conjunto finito de casos de teste adequadamente selecionados a partir do usualmente infinito domínio de execuções, contra o comportamento esperado [Abran et al., 2004].

2.2.1 Níveis do Teste

De acordo com o alvo do teste, o teste de software pode ser classificado de diferentes formas. O Manual do Conhecimento de Engenharia de Software (SWEBOK) [Abran et al., 2004] o classifica como *(i) teste de unidade*, *(ii) teste de integração*, *(iii) teste de sistema*. Mais detalhes sobre essas classificações são apresentadas a seguir.

1. **Teste de Unidade:** têm por objetivo verificar um elemento que possa ser logicamente tratado como uma unidade de implementação. Este tipo de teste se aplica a um conjunto de um ou mais partes de um programa de computador que exercitam os requisitos funcionais de subprogramas, sub-rotinas ou métodos de um sistema de software [Myers, 2004]. Ou seja, eles testam individualmente as unidades de um sistema [Sommerville, 2006]. O teste de unidade pode ocorrer de forma paralela para os vários módulos que compõe o sistema. Assim, se existirem dez programadores codificando o sistema, os mesmos programadores podem também codificar testes de unidade para verificar o que foi desenvolvido por eles.

2. **Teste de Integração:** têm por objetivo verificar as interfaces entre as partes de uma arquitetura do produto. Ou seja, é o processo de verificação da interação entre componentes de software [Abran et al., 2004]. Os testes de integração são guiados pela arquitetura do sistema e, exceto em casos de sistemas pequenos e simples é dada preferência por testar a integração de todos os componentes. Entretanto, sempre que acontece uma integração entre componentes de um sistema de grande porte é recomendado a execução de testes de integração.

   Para realização do teste de integração, é necessário definir uma estratégia de integração. De forma geral, a estratégia de integração determina como os diversos

componentes do sistemas são agrupados e como a verificação do funcionamento, agora em conjunto, é realizada. Estratégias de integração bem conhecidas são a "bottom-up" e a "top-down". Resumidamente, a primeira define uma estratégia de integração começando pelos módulos localizados inferiormente na hierarquia de unidades até chegar ao módulo principal. Por outro lado, a segunda define o caminho inverso, começando pela integração do módulo principal a cada um dos módulos adicionais.

De qualquer forma, o mais indicado para o desenvolvimento profissional de produtos de software é utilizar uma estratégia de integração incremental, em que o programa seja construído e testado em pequenos segmentos. Para a integração incremental é preciso lançar mão de testes de integração sempre que necessária a interface entre os diferentes módulos a serem integrados. À medida que um sistema de software evolui os testes de integração são executados periodicamente. A execução periódica dos testes de integração, de forma sistematizada, é conhecida como integração contínua [Duvall et al., 2007].

3. **Teste de Sistema:** têm por objetivo executar o sistema sob o ponto de vista de seu usuário final, verificando o correto atendimento a todos os requisitos funcionais e não-funcionais. Grande parte das falhas funcionais do sistema de software devem ter sido encontradas durante os testes de unidade e integração. Entretanto, o Teste de Sistema está interessado com o comportamento de um sistema de software como um todo [Abran et al., 2004].

Por ser comumente realizado durante o ciclo de desenvolvimento, o teste de integração de software foi escolhido para ser incorporado ao método proposto neste trabalho. Além disso, os testes de integração entre os módulos fornece naturalmente informações sobre o relacionamento entre partes isoladas do sistema.

Para melhor entendimento do conceito de teste de integração, considere o seguinte cenário. Dado um sistema bancário composto por diferentes módulos. Ao tentar transferir uma determinada quantidade monetária a uma outra pessoa, algumas restrições devem ser satisfeitas. Por exemplo, o número identificador da conta que receberá o valor transferido deve ser um número válido e a quantidade a ser transferida não pode ser maior que a disponível. Os módulos envolvidos nesta operação são, hipoteticamente, o módulo de "Exibição de Saldo Atual" e "Transferência" que, possivelmente, já passaram por testes de unidade. Entretanto, com estes novos requisitos que aparecem da integração entre os módulos, eles devem passar pelo teste de integração. Este último é justamente a verificação dos módulos combinados que já testados individualmente.

2.2.2 Reuso de Testes

Embora o reuso de software seja uma ideia largamente aceita na comunidade devido seus benefícios inerentes, não se conseguiu encontrar muitos exemplos do reuso de artefatos de teste de software. No entanto, alguns trabalhos vêm sendo conduzidos ao longo do tempo para investigar o reuso destes, por exemplo, na *geração de documentação de usuário* [Santos et al., 2010] e na *geração de testes de desempenho e estresse* [Fé et al., 2010].

A geração manual de documentação de usuário durante o desenvolvimento de software é onerosa e a mantê-la atualizada é uma tarefa árdua. Além disto, esta atividade não é indicada na maioria dos processos ágeis de desenvolvimento de software devido, principalmente, ao custo de manutenção. Santos e outros [Santos et al., 2010] apresentam uma alternativa à criação manual da documentação de usuário e da especificação de testes baseada na geração semi-automática a partir de *scripts* de testes funcionais. Em seu estudo experimental, Santos e outros mostraram que é possível reduzir o esforço na criação destes documentos mantendo o mesmo nível de qualidade da geração manual.

Em adição, embora a atividade de teste esteja bem difundida nas organizações desenvolvedoras de software, a automação por completo ainda é um grande desafio, especialmente em software que possuem diversas regras e manipulam muitos dados provenientes de bancos de dados. A geração de dados para o teste torna essa prática muito onerosa. Quando o assunto é a geração de dados para teste de desempenho e estresse, esse problema é ainda maior, chegando a inviabilizar o teste em alguns casos. Fé e outros [Fé et al., 2010] apresentaram um mecanismo para a geração de dados para o teste de desempenho e estresse reutilizando os testes funcionais conseguindo reduzir drasticamente o esforço para criação de tais testes [Santos et al., 2011]. Estes exemplos, além de fundamentar a escolha do reuso de artefatos de teste de software na tentativa de localização de características, servem também para motivar o seu uso nas atividades do dia-a-dia de desenvolvimento de novos produtos.

2.3 Resumo

Este capítulo apresentou a os conceitos básicos para o entendimento deste trabalho. Entre eles, *(i)* LPS, *(ii)* variabilidade, *(iii)* características em LPS e *(iv)* teste de software:

LPS: permite o reuso eficiente de uma arquitetura comum, facilitando a adição de componentes personalizados a fim de satisfazer necessidades específicas das partes interessadas [Pohl et al., 2005].

Variabilidade: é a possibilidade de produzir diferentes variantes (ver Definição 2) da linha de produtos, dado um ponto de variação (ver Definição 1).

Característica de uma LPS: é um incremento na funcionalidade de um programa [Batory, 2005].

Teste de software: consiste na verificação dinâmica do comportamento de um programa dado um conjunto finito de casos de teste adequadamente selecionados a partir do usualmente infinito domínio de execuções, contra o comportamento esperado [Abran et al., 2004].

O capítulo seguinte apresenta um revisão da literatura em busca de trabalhos com tema relacionados.

Capítulo 3

Revisão de Literatura

A abordagem de Linha de Produtos de Software (LPS) é utilizada para a construção de grandes famílias de produtos de software reutilizando um conjunto de artefatos e componentes [Clements & Northrop, 2001]. Esta abordagem tem ganho popularidade na última década e muitos trabalhos anteriores [Apel & Beyer, 2011, Ghanam & Maurer, 2010, Eisenbarth et al., 2003, Antoniol & Guéhéneuc, 2005, Poshyvanyk et al., 2007, Alves et al., 2005, Valente et al., 2012, Couto et al., 2011] atacaram problemas referentes a LPS, o que proporcionou um inegável ganho de maturidade. Este fato fez com que a abordagem passasse a ter ampla utilização em escala industrial [van der Linden et al., 2007, Bass et al., 1998, Pohl et al., 2005].

Este capítulo apresenta uma revisão da literatura referente aos trabalhos que abordaram a localização de características em software e a extração de LPS inspirado nos trabalhos de Petersen e outros [Petersen et al., 2008] e Kitchenham e outros [Kitchenham et al., 2009]. O procedimento de revisão foi definido como apresentado adiante, onde serão abordadas: a formulação das questões de pesquisa (Seção 3.1); a seleção de fontes de pesquisa (Seção 3.2); a seleção de estudos (Seção 3.3). Por fim, na Seção 3.4 é feita a sumarização de resultados, além do detalhamento dos trabalhos relacionados encontrados.

3.1 Formulação da Questão de Pesquisa

Dado o contexto em que este trabalho está inserido, o foco da questão de pesquisa é identificar iniciativas e relatos de experiências em engenharia de software relacionados com a localização de características em software e extração de LPS. Assim, os seguintes parâmetros para ajuste da qualidade e amplitude da pesquisa foram definidos:

Problema: LPS tem se tornado largamente utilizado na indústria, mas muitos do sistemas de software já desenvolvidos foram feitos como produto único. Portanto, se faz necessário recorrer a um método de migração destes produtos para a abordagem de LPS.

Questão: Quais as iniciativas que foram conduzidas com o propósito de migração de produtos de software para a abordagem de LPS?

Palavras-chave e sinônimos: definiu-se algumas palavras-chave para a pesquisa à literatura, bem como sinônimos, acrônimos e palavras que pudessem expressar objetivos semelhantes de pesquisa. As palavras-chave foram: (i) método; (ii) localização; (iii) característica; (iv) extração; (v) linha de produtos de software. Abaixo, os sinônimos e o acrônimo definidos em cada caso:

- **Método:** abordagem, mecanismo, modo, plano, processo, técnica;
- **Localização:** identificação;
- **Característica:** interesse;
- **Extração:** desenvolvimento, implementação, construção, evolução;
- **Linha de Produtos de Software:** LPS.

Intervenção: extração de LPS.

Controle: nenhum tipo de controle foi exigido.

Efeito: identificação de iniciativas relacionadas à extração de LPS.

Medida de Êxito: número de iniciativas identificadas.

População: publicações que levem em consideração à extração de LPS e localização de características.

Projeto Experimental: nenhum estudo estatístico será aplicado sobre os resultados.

3.2 Seleção de Fontes de Pesquisa

As fontes de publicação deveriam adequar-se aos critérios como segue. As fontes deveriam estar disponíveis para consulta web, devendo haver a presença de mecanismos de busca por meio de palavras-chave. Também foram consideradas as fontes sugeridas por

Tabela 3.1. Lista de fontes de pesquisa.

Fonte	Link
ACM Digital Library	*http://dl.acm.org/*
IEEE Xplore	*http://ieeexplore.ieee.org/*
ScienceDirect	*http://www.sciencedirect.com/*

profissionais com experiência na área, a saber, o orientador e co-orientador deste trabalho. Visto isso, as fontes selecionadas foram "ACM Digital Library", "IEEE Xplore" e "ScienceDirect". O endereço de cada uma delas é apresentado na Tabela 3.1.

As publicações deveriam ser do tipo "revista" ou "conferência" e somente trabalhos escritos em língua inglesa foram considerados. Neste caso, as palavras-chave, seus sinônimos e o acrônimo definidos foram traduzidos. Além disso, foram consideradas somente publicações com estudo completo e com número de páginas superior a 8.

3.3 Seleção de Estudos

Uma vez que as fontes foram identificadas, a busca das publicação foi feita através de máquinas de busca web. Foram consideradas inicialmente todas as publicações das revistas e conferências publicadas a partir de Janeiro de 2000 até Dezembro de 2010 por acreditar-se que a visibilidade granha pela engenharia de LPS nos últimos dez anos produzia maior concentração de trabalhos neste período. No entanto, trabalhos publicados após o ano de 2010 ou anteriormente ao ano de 2000, que eventualmente fossem identificados e se adequassem aos critérios descritos a seguir foram adicionados individualmente na sumarização de resultados (Seção 3.4).

Os estudos selecionados deveriam apresentar alguma iniciativa de migração de produtos de software para a abordagem de LPS e/ou localização de características em software. O levantamento não selecionou estudos com método ou processos de desenvolvimento de LPS desde o início, somente iniciativas de migração ou extração para esta abordagem foram consideradas. Todos os tipos de estudo foram selecionados, excetuando-se trabalhos de apresentação de lições aprendidas, artigos curtos, resumos, relatórios técnicos e trabalhos duplicados. No caso de trabalhos provenientes do mesmo estudo, considerou-se o mais completo.

A partir da busca com as palavras-chave definidas na Seção 3.1 foram selecionados 363 artigos. Destes, após a leitura do resumo de cada um, restou um conjunto de 60 artigos. Em seguida, procedeu-se leitura dinâmica para identificação dos trabalhos relacionados. Os trabalhos resultantes são apresentados na Seção 3.4.

3.4 Sumarização de Resultados

Nesta seção são apresentados os trabalhos relacionados ao tema desta pesquisa. Os trabalhos serão organizados em seções de acordo como o tipo de problema que eles atacam. Na Seção 3.4.1 são apresentados trabalhos relacionados a manutenção e evolução de LPS. Na Seção 3.4.2 tratamos alguns trabalhos sobre testes no contexto de LPS. Na Seção 3.4.3 alguns trabalhos direcionados à localização ou identificação de características são discutidos e finalmente a Seção 3.4.4 apresetna trabalhos sobre migração de software para a abordagem LPS.

3.4.1 Manutenção de LPS

Apel e Beyer [Apel & Beyer, 2011] conduziram um trabalho exploratório de análise da coesão de características em LPS. Eles utilizaram 40 linhas de produtos, entre elas sistemas diferentes e variantes de um mesmo sistema. Nesse trabalho, além do uso de métricas tradicionais foram definidas novas métricas baseadas no agrupamento visual para a análise do grau de coesão das características. Em seguida, foi analisada a existência de correlação entre *tamanho* (medido em linhas de código) e o *processo de desenvolvimento* (considerando LPS construídas desde o início e LPS refatoradas) com a coesão das características de uma LPS. A análise confirmou a correlação entre o tamanho da característica e sua coesão, bem como, a correlação entre o processo de desenvolvimento e sua coesão.

Greevy e Ducasse [Greevy & Ducasse, 2005] propuseram uma abordagem dirigida por características baseadas em análise dinâmica para produzir um mapeamento explícito entre características e unidades computacionais (mais precisamente, classes em sistemas orientado a objetos) de uma aplicação. Esse trabalho utiliza da execução de cenários para a criação do mapeamento. Na avaliação de sua abordagem, é feita a caracterização do relacionamento existente entre classes e características, classificando as características em "completamente relacionadas" (quando não foram encontradas classes específicas para uma dada característica) e "fortemente relacionadas" (quando foram encontradas poucas classes específicas).

Em adição, o trabalho de White e outros [White et al., 2008] apresentou uma técnica para a transformação de um modelo de características defeituoso em um problema de satisfação de restrições. Além disso, eles mostraram como derivar um conjunto mínimo de características a fim de tornar válida uma configuração inválida através de experimentos para a avaliação de sua técnica. Para a execução de experimentos eles construíram um gerador de modelos de características para criar aleatoriamente mo-

delos de características com as ramificações e restrições desejadas. Os experimentos foram feitos com modelos de até 5000 características apresentando diagnósticos entre 45 segundos e 4 minutos.

Os trabalhos apresentados nesta seção apresentam diferenças significativas em relação ao presente trabalho. Todos eles apresentam claro interesse na evolução de LPS. Diferentemente, este trabalho busca prover a apoio à construção de uma linha. Em outras palavras, as abordagens ou técnicas apresentadas são aplicadas após a construção da LPS, enquanto este trabalho apresenta um método aplicável anteriormente à sua construção.

3.4.2 Testes e LPS

No tocante à testes de software relacionados a abordagem de LPS, dois estudos semelhantes foram conduzidos recentemente na forma de mapeamento sistemático da pesquisa de testes de LPS. Silveira Neto e outros [Silveira Neto et al., 2011] e Engström e Runeson [Engström & Runeson, 2011] chegaram a resultados semelhantes sobre a pesquisa na área. No entanto, nenhum deles apresentou trabalhos relacionados com a localização de características através do reuso de teste de software. Entre os temas dos trabalhos enumerados estão *(i)* estratégias de teste de LPS, *(ii)* níveis de teste em LPS, *(iii)* teste de regressão em LPS, *(iv)* teste de comunalidade e variabilidade, *(v)* redução de esforço de teste em LPS.

Ghanam e Maurer [Ghanam & Maurer, 2010] propuseram anexar testes de aceitação ao modelo de características utilizando para isto o que eles chamaram de testes de aceitação executáveis (EAT). Cada EAT foi adicionado ao nível mais baixo do modelo de características como características folha. O objetivo desse trabalho foi prover a possibilidade de rastreamento entre o modelo de características e artefatos de código-fonte. A avaliação deste trabalho é subjetiva e não apresenta dados de experimentos reais. Este trabalho, apesar de não ter o objetivo de localizar características em software desenvolvido como produto único, serviu como inspiração na definição do método aqui proposto e apresentado no Capítulo 4.

Os trabalhos apresentados nesta seção apresentam diferenças significativas em relação ao presente trabalho. Enquanto o trabalho de Silveira Neto e outros [Silveira Neto et al., 2011] e Engström e Runeson [Engström & Runeson, 2011] enumeram e classificam iniciativas de verificação e validação de LPS através de teste automatizado, o trabalho de Ghanam e Maurer [Ghanam & Maurer, 2010] associa testes de aceitação às características de nível mais baixo no modelo de características. Este trabalho, por outro lado, busca reutilizar testes na extração de uma LPS através da localização de características.

3.4.3 Localização de Características

O problema de localização de características é o foco principal deste trabalho. Vários trabalhos na literatura [Eisenbarth et al., 2003, Antoniol & Guéhéneuc, 2005, Poshyvanyk et al., 2007, Walkinshaw et al., 2007] atacam o mesmo problema.

Eisenbarth e outros [Eisenbarth et al., 2003] apresentam em seu trabalho uma técnica semi-automática para reconstruir o mapeamento de características (que possuem comportamento observável quando acionadas pelo usuário) para o código-fonte. O objetivo principal era o entendimento de programas para facilitar a tarefa de manutenção do software. Mais especificamente, encontrar as unidades computacionais que implementam uma característica e o conjunto de unidades computacionais que conjuntamente ou distintamente são necessárias para um conjunto de características. Nesse contexto, unidade computacional é entendida como uma parte executável do sistema e compreende desde instruções até módulos e subsistemas.

Esse trabalho combinou técnicas de análise dinâmica e estática. Como técnica dinâmica eles utilizaram o conceito de cenários (sequências de entradas de usuário que ativam ações de um sistema com resultados observáveis) para reunir as unidades computacionais para as características relevantes. Em seguida, a análise estática utiliza essas unidades computacionais para identificar unidades computacionais adicionais específicas de cada característica através do grafo de dependências. Resumidamente, a ideia apresenta certa fragilidade ao depender de diversos indivíduos para a execução da abordagem, além de diversas iterações e consequentemente quantidade de tempo considerável.

Na abordagem três atores humanos são definidos: o *analista*, que faz a interpretação da *látice conceitual*[1] e para a execução da análise estática; o *especialista no domínio da aplicação*, que planeja os cenários e enumera as características que serão invocadas em cada cenário; e o *usuário*, que utilizará do sistema sob avaliação de acordo com cenários selecionados. A abordagem apresentou problemas de escalabilidade no momento da utilização de um sistema de grande porte na avaliação devido ao crescimento da complexidade do sistema e a necessidade de tratamento para o relacionamento entre características.

O processo definido por Antoniol e Guéhéneuc [Antoniol & Guéhéneuc, 2005] de identificação de características tem como objetivo assistir à tarefa de entendimento de programas multitarefa orientados a objeto. Para isso, eles identificaram microar-

[1] Tradução pessoal para "concept lattice". Uma "lattice" é conceito matemático que representa um conjunto parcialmente ordenado que contém um maior limite máximo e um menor limite mínimo. Já um "concept", de maneira simplificada, compreende um par (O, A), onde O é um objeto do conjunto e A é um atributo.

quiteturas que implementam algumas características de interesse, destacando classes, atributos e métodos ativados. A abordagem é aplicada através da comparação do código executado em diferentes cenários. O entendimento do programa acontece com a visualização da diferença entre um *cenário A* e um *cenário B* de execução do programa. O cenários diferem entre si pela adição de tarefas adicionais no segundo.

A avaliação é feita no tocante à utilidade da abordagem na identificação de classes, métodos e campos que implementam a funcionalidade de salvar uma página visitada no navegador Web *Mozilla*. É feita uma comparação com uma abordagem ingênua de busca baseada em ferramentas de casamento de strings e com a abordagem de análise conceitual apresentada por Eisenbarth e outros [Eisenbarth et al., 2003]. Segundo os autores, sua abordagem reduz a quantidade de dados a serem processados e não tem problemas de escalabilidade. Ela produz uma lista de métodos e classes que participam de uma característica e pode ser estendida ao estudo da evolução desta característica através das microarquiteturas.

Poshyvanyk e outros [Poshyvanyk et al., 2007] propuseram a técnica chamada de Ranqueamento Probabilístico de Métodos baseado na Execução de Cenários e na Recuperação de Informação (PROMESIR). PROMESIR utiliza duas técnicas para a localização de características: *(i)* a baseada em recuperação de informação que usa indexação de semântica latente (LSI) [Deerwester et al., 1990] (utiliza técnicas matemáticas para identificar padrões e relacionamento entre os termos e conceitos contidos em uma coleção de texto não estruturado) e *(ii)* a de ranqueamento probabilístico baseado em cenários (SPR) (técnica utilizada pelo trabalho de Antoniol e Guéhéneuc [Antoniol & Guéhéneuc, 2005]). A avaliação da técnica PROMESIR é feita replicando os estudos de caso feitos por Antoniol e Guéhéneuc [Antoniol & Guéhéneuc, 2005] e comparando os resultados das técnicas SPR, LSI e PROMESIR. Com isso, eles argumentam que aumentam significativamente a efetividade da localização se comparadas a técnica PROMESIR com as duas técnicas utilizadas separadamente: LSI e SPR.

O trabalho de Walkinshaw e outros [Walkinshaw et al., 2007] apresenta uma abordagem baseada em um grafo de chamadas dos métodos que utiliza "slicing" para redução do tamanho do grafo e permite encontrar parte do código que implementa uma característica funcional. Sua abordagem introduz dois conceitos: *método referência*[2], o qual se imagina que seja peça chave na implementação da característica são os pontos de partida; e *método obstáculo*, o qual se imagina que seja irrelevante para a implementação da característica e deste modo limita a explosão do grafo de chamadas do sistema.

[2]Tradução própria para: landmark method

Com o grafo que contém todos os caminhos diretos entre cada par dos métodos de referência é possível identificar todos os caminhos do grafo original que influenciam ou que podem ser influenciados por ele. Nesse trabalho, apesar de terem sido alcançados significativos valores de cobertura, os valores referentes à precisão deixavam a desejar. Portanto, eles passaram a eliminar as seções irrelevantes do grafo através da identificação dos métodos obstáculos.

A abordagem apresentada por Walkinshaw e outros é dependente do conhecimento do usuário no domínio da aplicação para o julgamento quanto à classificação dos métodos como de obstáculo ou de referência. Além disso, a abordagem foi proposta para a localização de código fonte no contexto da execução de tarefas de manutenção de software. As métricas de precisão e cobertura são calculadas com relação aos métodos identificados no grafo de chamadas final.

Eisenberg e De Volder [Eisenberg & De Volder, 2005] também atacam o problema de localização de característica. Sua técnica apresenta o conceito de Rastro Dinâmico de Característica (DFT) e tenta localizar métodos relacionados a uma característica baseado no grafo de chamadas de suítes de testes Java constuídas com uso do JUnit [Massol & Husted, 2003] com suporte de algumas heurísticas. Segundo os autores, a principal contribuição do trabalho é a aplicação de heurísticas para a construção de um ranqueamento para determinar a relevância de um elemento de código para uma dada característica. Em sua abordagem é necessário que os desenvolvedores alterem as suítes de testes para possibilitar o uso da técnica desenvolvida. Além disso, como não há a preocupação em extrair o código para a utilização em LPS, sua avaliação é feita de forma qualitativa para avaliar a utilidade em comparação com abordagens anteriores.

Os trabalhos apresentados nesta seção apresentam diferenças sutis em relação a este trabalho. Embora todos tenham como objetivo a localização de características, este trabalho é inserido em um contexto distinto: a extração de linha de produtos de software. Diferentemente, os trabalhos descritos nesta seção buscam facilitar a manutenção e evolução de software. Embora, extrair uma LPS possa ser considerado evolução de software, o paradigma de engenharia de LPS exige tarefas que não existem quando da solicitação de ajustes e melhorias num dado sistema de software.

3.4.4 Migração de Software para Linha de Produtos de Software

Alguns trabalhos foram conduzidos com o objetivo de apoio à migração de software legado para a abordagem de LPS [Alves et al., 2005, Valente et al., 2012, Couto et al., 2011].

Alves e outros [Alves et al., 2005] apresentaram a extração e evolução de linhas de produtos manualmente usando *Programação Orientada à Aspectos* com *AspectJ*. Alves e seus colegas utilizaram uma abordagem reativa, onde um ou mais produtos existentes são refatorados com o objetivo de expor a variabilidade existente. Em seguida, são feitas refatorações para acomodar o(s) produtos existentes e novas variantes da linha. Nesse trabalho são utilizados jogos para celular desenvolvidos com Java, Edição Micro (J2ME).

Recentemente, Valente e outros [Valente et al., 2012] propuseram uma abordagem semi-automática para migração de código legado para a abordagem de linha de produtos de software com o auxílio da ferramenta CIDE+ [De Oliveira et al., 2010]. A ferramenta CIDE+ foi construída com base na ferramenta CIDE [Kästner et al., 2008]. Ela auxilia na extração através da coloração das linhas de código. Um método é proposto para expandir a coloração por todo o código com interferência de um usuário que autoriza ou não a expansão em casos onde a ferramenta não consegue inferir automaticamente. Além destes, Couto e outros [Couto et al., 2011] apresentaram um estudo de caso de extração manual de uma LPS utilizando de compilação condicional.

Os experimentos no trabalho de Valente e outros foram conduzidos com três sistemas classificados como de médio-grande porte: Prevayler[3], JFreeChart[4] e ArgoUML[5]. Foram feitas extrações manuais de cada um dos sistemas e comparados os resultados com os obtidos com a abordagem proposta. Os resultados de precisão e cobertura foram medidos em relação aos bytes anotados e foram, em média, próximos a 90%. Nesse estudo não fica claro como foi feita a verificação de bytes anotados resultantes das extrações manual e semi-automática através do algoritmo proposto. De forma que, não há como garantir que os bytes considerados iguais correspondem exatamente às linhas de código anotadas.

Os trabalhos apresentados nesta seção apresentam algumas diferenças em relação ao presente trabalho. Os trabalhos estão relacionados à extração de LPS, onde é apresentada duas iniciativas de extração manual, uma com compilação condicional e outra através do uso de programação orientada à aspectos. O terceiro, uma abordagem semi-automático de localização de características através do grafo de chamadas. Diferentemente desses, este trabalho, não apresenta a extração final da LPS. Além disto, nenhuma delas apresenta um método fim-a-fim, que leva o desenvolvedor desde a modelagem do modelo de características até a localização de características.

[3]www.prevayler.org, versão 2.3.
[4]www.jfree.org/jfreechart, versão 1.0.13.
[5]argouml.tigris.org, versão 0.28.

3.5 Trabalhos Relacionados: em poucas palavras

A Tabela 3.2 apresenta de forma sucinta, uma comparação deste trabalho com seus relacionados identificados com a revisão de literatura. O símbolo "•" é utilizado para informar que um dado item de contribuição (linhas da tabela com sombreamento alternado em cinza e branco) está presente em um dado trabalho (colunas). Enquanto, o símbolo "○" indica a ausência e o símbolo "-" foi utilizado para indicar um item não se aplica a um dado trabalho. O capítulo seguinte apresenta o método proposto neste trabalho para localização de características através do reuso de teste de software.

A Tabela 3.2 apresentada informações de cada trabalho detalhado nas seções anteriores de acordo com sua contribuição. Foram definidas três linhas para a observação: o **objetivo** principal do trabalho, a **técnica** utilizada e a **granularidade** tomada em consideração para a análise dos resultados. Cada uma destas foram encontrados grupos, enumerados como segue:

- Quanto ao *objetivo*, foram detalhados, quatro grupos: *(i)* Manutenção/Entendimento de Software; *(ii)* Extração de LPS; *(iii)* Evolução de LPS; e *(iv)* Revisão de Literatura.

- Quanto à *técnica*, foram detalhados, oito grupos: *(i)* Grafo de Chamadas; *(ii)* Execução de Testes; *(iii)* Ranqueamento Probabilístico e Indexação de Semântica Latente; *(iv)* Modelos; *(v)* Execução de Cenários; *(vi)* Agrupamento Visual; *(vii)* Aspectos; e *(viii)* Manual.

- Quanto à *granularidade*, foram detalhados, três grupos: *(i)* Classe; *(ii)* Método; e *(iii)* Linha de código.

Foram encontrados seis trabalhos relacionados à manutenção/entendimento de software [Ghanam & Maurer, 2010, Eisenbarth et al., 2003, Antoniol & Guéhéneuc, 2005, Poshyvanyk et al., 2007, Walkinshaw et al., 2007, Eisenberg & De Volder, 2005]. Diferentes técnicas para a localização de características são apresentadas nestes trabalhos. Cada uma delas atinge granularidade semelhante: a de método. Com exceção de Ghanam e outros [Ghanam & Maurer, 2010] com granularidade de classe e Antoniol e Guéhéneuc [Antoniol & Guéhéneuc, 2005] com granularidade de linha de código.

Foram encontrados três trabalhos relacionados à extração de LPS [Alves et al., 2005, Valente et al., 2012, Couto et al., 2011]. Destes, somente o trabalho de Valente e outros [Valente et al., 2012] propõem uma abordagem semi-automática, as outras duas, uma abordagem de extração manual utilizando desenvolvimento em

aspectos [Alves et al., 2005] e a outra apresenta um estudo de caso de extração manual com compilação condicional [Couto et al., 2011]. A abordagem de Valente e outros depende do conhecimento, por parte do usuário, da maioria do código do sistema.

Foram encontrados quatro trabalhos relacionados à evolução de LPS [Apel & Beyer, 2011, Greevy & Ducasse, 2005, White et al., 2008, Alves et al., 2005]. Enquanto Apel e Beyer [Apel & Beyer, 2011] analisaram a coesão de características na evolução de LPS, Greevy e Ducasse [Greevy & Ducasse, 2005] apresentaram um mapeamento de características para unidades computacionais e White e outros [White et al., 2008] a validação de modelos de características através da solução de problemas de satisfação de restrições. O trabalho de Alves e outros [Alves et al., 2005] além da extração orientada à aspectos também se aplica a evolução de LPS.

Foram encontrados dois trabalhos relacionados à revisão de literatura [Silveira Neto et al., 2011, Engström & Runeson, 2011]. Dois trabalhos semelhantes conduzidos na forma de mapeamento sistemático da pesquisa na área de teste de LPS. Nenhum dos trablahos apontados nestes trabalhos aperesentava uma técnica baseada em testes para a localização e extração de características em software desenvolvido como produtos únicos.

A organização dos trabalho discutidos no capítulo na Tabela 3.2 possibilita perceber que muitos trabalhos [Eisenbarth et al., 2003, Antoniol & Guéhéneuc, 2005, Poshyvanyk et al., 2007, Walkinshaw et al., 2007, Eisenberg & De Volder, 2005] relacionados à localização da implementação de características em software têm sido voltados unicamente para a manutenção de software deixando de fora a migração para a abordagem de LPS. Mesmo com foco na manuteção de software não existe concenso quanto a melhor técnica para a localização da implementação de características. Detectou-se ainda que poucas iniciativas abordaram este problema [Alves et al., 2005, Valente et al., 2012] de forma direta. Isso configura um ambiente propício para o surgimento de novas abordagens: híbridas, com a junção de pontos fortes de algumas das técnicas apresentadas; ou ainda, de caráter inovador como o método a que será proposto no capítulo seguinte.

3.6 Resumo

Este capítulo apresentou uma revisão da literatura relacionado ao tema de localização de características e de extração de LPS inspirada nos trabalhos de revisão [Kitchenham et al., 2009] e mapeamento [Petersen et al., 2008] sistemáticos em engenharia de software. Inicialmente os trabalhos foram classificados em 4 principais cate-

Tabela 3.2. Comparativo entre os trabalhos relacionados.

	Contribuição	[Apel & Beyer, 2011]	[Greevy & Ducasse, 2005]	[White et al., 2008]	[Silveira Neto et al., 2011]	[Engström & Runeson, 2011]	[Ghanam & Maurer, 2010]	[Eisenbarth et al., 2003]	[Antoniol & Guéhéneuc, 2005]	[Poshyvanyk et al., 2007]	[Walkinshaw et al., 2007]	[Eisenberg & De Volder, 2005]	[Alves et al., 2005]	[Valente et al., 2012]	[Couto et al., 2011]	Proposta
Objetivo	Manutenção de Software	○	○	○	○	○	●	●	●	●	●	●	○	○	○	●
	Extração de LPS	○	○	○	○	○	○	○	○	○	○	○	●	●	●	●
	Evolução de LPS	●	●	●	○	○	○	○	○	○	○	○	●	○	○	○
	Revisão de Literatura	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○
Técnica	Grafo de Chamadas	○	○	○	-	-	○	○	○	○	●	●	○	●	○	○
	Execução de Testes	○	○	○	-	-	○	○	○	○	○	●	○	○	○	●
	Ranqueamento Probabilístico e Indexação de Semântica Latente	○	○	○	-	-	○	○	○	●	○	○	○	○	○	○
	Modelos	○	○	●	-	-	●	○	○	○	○	○	○	○	○	○
	Execução de Cenários	○	●	○	-	-	○	●	●	○	○	○	○	○	○	○
	Agrupamento Visual	●	○	○	-	-	○	○	○	○	○	○	○	○	○	○
	Aspectos	○	○	○	-	-	○	○	○	○	○	○	●	○	○	○
	Manual	○	○	○	-	-	○	○	○	○	○	○	●	○	●	○
Granularidade	Classe	-	●	●	-	-	●	●	●	●	●	●	●	●	●	●
	Método	-	○	○	-	-	○	●	●	●	●	●	●	●	●	●
	Linha de Código	-	○	○	-	-	○	○	●	○	○	○	●	●	○	●

gorias na Seção 3.4, a saber: *(i)* manutenção de LPS; *(ii)* testes e LPS; *(iii)* localização de características; e *(iv)* migração de software para LPS.

Por fim, na Seção 3.5 a Tabela 3.2 apresentou um sumário dos trabalhos identificados apontando o objetivo principal de cada estudo, a técnica utilizada e também a granularidade alcançada em cada um deles. O capítulo seguinte apresenta o método para extração de LPS.

Capítulo 4

Método Proposto

Este capítulo apresenta um método para extração de Linha de Produtos de Software (LPS) com foco na localização de código-fonte que implementa uma dada característica em sistemas previamente desenvolvidos como produtos únicos. Como visto no Capítulo 3, ainda existem iniciativas de extração manual de LPS. A extração manual de LPS é uma tarefa onerosa e requer muito tempo. Na tentativa de minimizar o esforço necessário para esta atividade, resolveu-se pela proposta de uma nova alternativa apresentada neste capítulo. O restante do capítulo está organizado da seguinte maneira. Inicialmente é apresentada uma visão geral do método (Seção 4.1). Em seguida, cada um dos passos que compõem o método é detalhado. Passo 1: a construção do modelo de características é apresentada na Seção 4.2. Passo 2: o mapeamento de requisitos para características é apresentado na Seção 4.3). Passo 3: o mapeamento de testes para características é apresentado na Seção 4.4. Passo 4: a localização das características no código-fonte é apresentada na Seção 4.5. Concluindo este capítulo, um breve resumo é apresentado na Seção 4.6.

4.1 Visão Geral do Método

Esta seção apresenta uma visão geral do método aqui proposto. O método em questão é baseado nos artefatos do sistema alvo que foram produzidos ainda na fase de desenvolvimento. Precisamente, é proposto não somente a extração de um modelo de características parcial com base nos documentos de projeto e requisitos, mas também a localização do código-fonte das características relevantes. Uma das novidades apresentadas é o reuso de testes de integração, que foram produzidos para o sistema alvo, no processo de localização das características.

O método consiste em quatro passos, ilustrados sequencialmente na Figura 4.1. As setas de cor cinza que apontam da esquerda para a direita indicam os sucessivos passos do método. A saída de cada passo é apresentada nos quadros identificados por (b), (c), (d) e (e). O quadro (a) ilustra os artefatos que são utilizados como entrada para a execução do método. As saídas de cada passo são consideradas como entrada para o passo imediatamente seguinte. Por exemplo, de posse dos artefatos de documentação do sistema alvo, como produto, o *Passo 1* constrói um modelo de características para o *Passo 2*. Assim segue até o final do método. Os Passos 1, 2 e 3, idealmente, são executados, uma única vez, enquanto que o Passo 4 é executado sempre que necessário localizar uma nova característica.

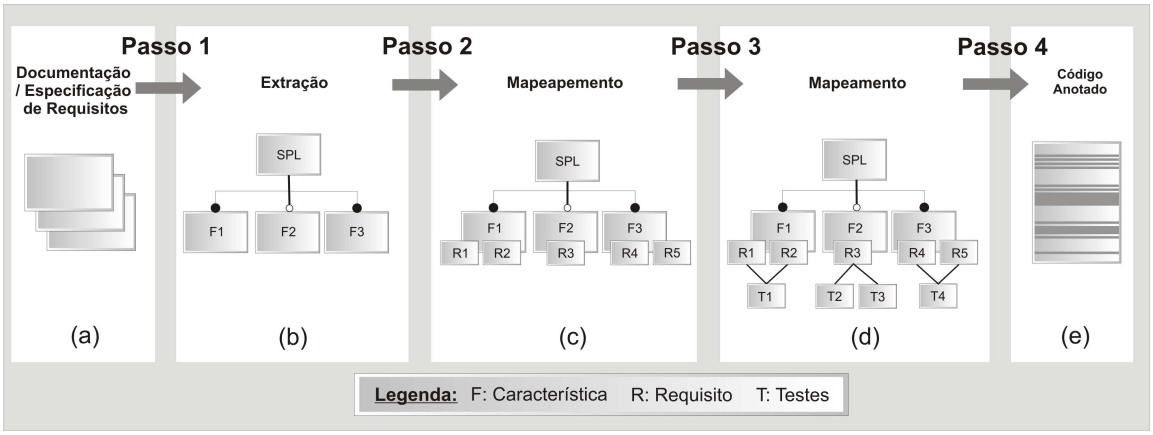

Figura 4.1. Ilustração do método proposto para localização de características.

O ponto de partida do método está na reutilização dos artefatos do sistema alvo. Os artefatos que podem ser utilizados no passo inicial são *(i)* documentos de especificações de requisitos, *(ii)* descrições de casos de uso e *(iii)* documentação de usuário e não necessariamente tem foco em variabilidade. Esses artefatos são comumente produzidos no processo de desenvolvimento de software e, portanto, eles permitem que o método proposto possa ser aplicado a um grande número de sistemas já desenvolvidos, independentemente do processo de desenvolvimento adotado. Cada um dos documentos disponíveis contribui para a melhor identificação das características através da adaptação do procedimento a cada um deles.

Tomando os passos do método é possível fazer analogia entre o método proposto com a engenharia de LPS. Os Passos 1, 2 e 3, por exemplo, correspondentes à construção dos mapeamentos entre artefatos da documentação e da implementação do sistema correspondem a atividades do *processo de engenharia de domínio*. Neste processo, a variabilidade e o reuso dos artefatos são planejados. Além disso, o Passo 4 procura a

reutilização dos artefatos para realizar a variabilidade da linha de produtos, reuso este que é o objetivo do *processo de engenharia de aplicação*.

Cada um dos quatro passos do método proposto será apresentado individualmente em uma seção. A Seção 4.2 inicia apresentando o *Passo 1*, referente à *Construção do Modelo de Características*; a Seção 4.3 descreve o *Passo 2*, referente ao *Mapeamento de Requisitos para Características*; em seguida, o *Passo 3*, referente ao *Mapeamento de Testes para Características* é apresentado na 4.4; e finalmente, na Seção 4.5 descreve-se o *Passo 4*, referente à *Localização das Características no Código-fonte*.

4.2 Construção do Modelo de Características

Esta seção descreve a extração do modelo de características para a LPS resultante do sistema alvo ao final da aplicação deste método. Ainda que o modelo gerado neste passo não represente uma versão final do mesmo, ele irá descrever a linha de produtos baseado na documentação existente. Assim, para a construção de um modelo de características que possa ser fidedigno à linha de produtos a ser extraída é importante que a documentação em questão esteja atualizada e consistente. Sabendo que este é um panorama ideal e não real, acredita-se que documentos com pequeno período de desatualização ainda sejam úteis na identificação de características.

A extração do modelo de características corresponde ao *Passo 1* do método exibido na Figura 4.1. A extração é feita a partir da documentação existente com o uso de uma série de heurísticas que guiam este processo. Estas heurísticas serão descritas à diante. Embora estas heurísticas possam ser adaptadas e aplicadas a diferentes conjuntos de artefatos do sistema. Aqui serão exibidos exemplos da aplicação destas com descrições de casos de uso. A Figura 4.2 exibe um modelo de descrição de casos de uso que detalha a informação que pode ser utilizada neste passo.

Cada uma das heurísticas a serem utilizadas têm como objetivo, além de identificar novas características do sistema alvo, identificar a natureza delas, classificando-as como "opcionais", "mandatórias", "alternativas inclusivas" (do tipo OR) ou "alternativas exclusivas" (do tipo XOR). Não serão apresentados muitos detalhes sobre cada uma delas, uma vez que elas foram definidas no trabalho de Varela e outros [Varela et al., 2011] e não são uma contribuição deste trabalho. Estas heurísticas foram incorporadas ao método pois *(i)* atendem adequadamente a estes passo, *(ii)* podem ser adaptadas e extendidas e *(iii)* foram avaliadas em estudos anteriores e se mostraram úteis.

As heurísticas são enumeradas abaixo. Elas foram nomeadas de $H1$ a $H10$. Os exemplos de características foram retirados de um dos sistemas utilizados na avaliação

Nome	Registrar usuário
Fontes	Conhecimento do sistema de biblioteca Web, conjunto de requisitos iniciais, partes interessadas.
Atores	Usuário Administrador
Descrição	O usuário administrador adiciona o registro de um novo usuário da biblioteca.
Decomposição	<Nenhum>
Lista de requisitos	
1. O usuário administrador deve estar autenticado no sistema; 2. A tela exibe a página de administração de usuários; 3. A página solicita a seleção de uma das opções de escolha: "Adicionar Novo" ou "Atualizar Antigo"; 4. Após selecionar "Adicionar Novo", a tela exibe a página de registro de novo usuário; 5. A página solicita (caso desejado) a seleção do papel a ser exercido pelo novo usuário (administrador, leitor ou bibliotecário); 6. A página solicita os dados necessários para o registro do usuário; 7. A página solicita a seleção de pelo menos uma das notificações que o usuário deseja receber (nenhuma, novo exemplar, aproximação de data de devolução, etc.); 8. Após o preenchimento do formulário, a página solicita a seleção de uma das opções de escolha: "Salvar" ou "Cancelar";	
Lista de casos de uso de extensão	
"Validar Dados de Usuário"	
Lista de casos de uso de inclusão	

Figura 4.2. Modelo de descrição de caso de uso que pode ser utilizada para extração de informações para o *Passo 1*.

do método, o *JBook* – um sistema de gerenciamento de bibliotecas apresentado em mais detalhes no Capítulo 6.

H1: define a *identificação da característica raiz*. A característica raiz será o nome do sistema;

H2: define a *identificação de características* baseada no nome de casos de uso. Por exemplo, "Registrar Usuário", "Cancelar Empréstimo" originam características com estes mesmos nomes;

H3: define a *identificação de características agrupáveis*, ou seja, que podem ser agrupadas ou ainda que podem se tornar sub-características de uma outra com significado mais geral. Por exemplo, considerando as características, "Registrar Usuário" e "Atualizar Usuário", é possível adicionar uma característica pai chamada "Processar Usuário" ou mesmo "Usuário". Tornando as duas primeiras sub-características da terceira;

H4: define a *identificação de características* baseada nas entradas da lista de requisitos que exercem papel importante no sistema. Por exemplo, "Registrar Usuário" tem a sub-característica "Escolher Papel do Usuário";

H5: define a *identificação de variabilidade opcional* baseada na descrição de casos de uso. A "Descrição" identifica uma característica opcional se um verbo modal expressar a sua não obrigatoriedade, por exemplo, se aparecer o verbo "pode" na descrição. Esta variabilidade é relacionada às características extraída da heurística $H2$. Por exemplo, "Notificar usuário" tem a descrição "O sistema pode notificar os usuários da biblioteca sobre a aquisição de um novo exemplar". Assim a característica "Notificar Usuário" é opcional;

H6: define a *identificação de variabilidade opcional* baseada na busca de expressões, como "caso desejado", "se quiser", "se possível". Estas expressões, quando adicionadas em algum requisito, classificam as características extraídas como opcionais. Por exemplo, o requisito "A página solicita (caso desejado) a seleção de somente um, entre os papéis existentes (administrador, leitor, bibliotecário), a ser exercido pelo novo usuário;" faz com que a característica "Escolher Papel do Usuário" obtida na $H4$ seja classificada como opcional;

H7: define a *identificação de variabilidade alternativa exclusiva* baseada na busca de expressões, como "somente uma das alternativas". Por exemplo, as características "E-mail com HTML" e "E-mail com Texto Puro" são sub-características de "Notificar Usuário" provendo uma alternativa exclusiva no modelo de características;

H8: define a *identificação de variabilidade alternativa inclusiva* baseada na busca de expressões, como "pelo menos uma das alternativas". Por exemplo, "A página solicita a seleção de pelo menos uma das notificações que o usuário deseja receber (nenhuma, novo exemplar, aproximação de data de devolução, etc.)" permite a identificação de características "Notificar Novo Exemplar", "Notificar Data de Devolução" de natureza alternativa inclusiva;

H9: define a *identificação de dependência entre características do tipo "A requer B"*, baseado na lista de extensão de casos de uso. A "Lista de casos de uso de extensão" origina relacionamentos de dependência do tipo *requer*. Por exemplo, o caso de uso "Validar Dados de Usuário" origina uma relação de dependência: "Registrar Usuário" *requer* "Validar Dados de Usuário";

H10: define a *identificação de dependência entre características do tipo "A exclui B"*, baseado na busca de requisitos contendo a expressão "excluindo a possibilidade X". Este tipo de expressão origina um relacionamento de dependência de exclusão, onde X identifica a outra característica presente na ligação.

Vale ressaltar que, apesar de algumas heurísticas referirem a descrição do caso de uso como fonte para extração de uma característica (heurística $H5$), é possível que em um caso de uso mais de uma característica seja identificada no mesmo. Da mesma forma, é possível que um requisito esteja associado a mais de uma característica.

Os casos em que as heurísticas não definem a natureza da característica, elas são consideradas mandatórias. Além disso, a classificação fornecida pelas heurísticas deve ser refinada e revisada por um engenheiro de domínio. Em adição às heurísticas apresentadas, técnicas adicionais podem ser utilizadas para a derivação da variabilidade da LPS como a abordagem de Extração de Similaridade e Variabilidade (CAVE) proposta por John [John, 2010].

4.3 Mapeamento de Requisitos para Características

Requisito é um conceito amplo e que pode fazer referência tanto às funcionalidades do sistema capturadas por casos de uso (coisas que o sistema deve fazer) – *requisito funcional* – quanto a qualidade do sistema (segurança, desempenho, confiabilidade) – *requisito não-funcional*. Neste trabalho, tanto em exemplos, quanto nos estudos de caso apresentados na avaliação do método aqui proposto (Capítulo 6) foram considerados somente requisitos funcionais. Os requisitos não-funcionais não são verificados através de testes de integração mas por outros tipos de teste, por exemplo os testes de desempenho e estresse.

O segundo passo do método proposto é o mapeamento dos requisitos do sistema para as características identificadas no Passo 1 detalhado na Seção 4.2. As heurísticas apresentadas naquela seção, não somente apóiam a identificação de características para a extração do modelo de características, como também, apóiam o mapeamento proposto neste passo. Isto acontece pelo fato de que as descrições de caso de uso capturam os requisitos, possibilitando, portanto, a associação destes às características identificadas.

A Tabela 4.1 apresenta alguns exemplos de relacionamentos entre requisitos e características bem como as heurísticas através das quais as características foram identificadas no passo anterior. Percebe-se em alguns casos, que mais de uma heurística pode incidir sobre uma característica. Por exemplo, os requisitos de "E-mail com HTML" e "E-mail com Texto Puro" através das heurísticas $H4$ e $H7$ produzem características com o mesmo nome. Geralmente, uma identifica a característica e as demais modificam a natureza da mesma, tornando-a opcional ou alternativa.

Tabela 4.1. Relacionamento entre Requisitos e Características.

Requisitos	**Heurísticas**	**Características**
Registrar usuário	$H2$	*(mesmo nome)*
Cancelar empréstimo	$H2$	*(mesmo nome)*
Notificar usuário	$H2$,$H5$	*(mesmo nome)*
Notificar registro de novo exemplar aos usuários	$H2$,$H8$	*Notificar Novo Exemplar*
Notificar aproximação de data de devolução aos usuários	$H2$,$H8$	*Notificar Data de Devolução*
E-mail com HTML	$H4$,$H7$	*(mesmo nome)*
E-mail com Texto Puro	$H4$,$H7$	*(mesmo nome)*
Escolher papel do usuário	$H4$	*(mesmo nome)*
Validar Dados de Usuário	$H9$	*(mesmo nome)*

4.4 Mapeamento de Testes para Características

Dado o conhecimento de quais requisitos foram associados a quais características (Passo 2 - Seção 4.3), é possível também associar indiretamente os artefatos de teste com às características. Em outras palavras, o mapeamento dos testes para as características pode ser inferido dado que:

- O teste de software tem como objetivo encontrar defeitos no código que implementa os requisitos [Abran et al., 2004] e, portanto, eles estão implicitamente ou explicitamente ligados aos requisitos;
- Os requisitos foram mapeados para as características no Passo 2 (Seção 4.3);
- Existem vários processos [Ramesh & Jarke, 2001] que prescrevem a criação de uma matriz de rastreabilidade que permite identificar a ligação entre requisitos, código e testes.

Além disso, embora os artefatos de teste de software não sejam explicitamente mapeados para requisitos nos sistemas desenvolvidos, este mapeamento pode ser facilmente recuperado pela especificação dos testes. A ligação existente entre os artefatos de teste e os requisitos, bem como, a ligação entre requisitos e características permitem o exercício do código que implementa uma dada característica através da execução dos testes.

A Figura 4.3 ilustra um mapeamento de algumas suítes de teste para as características do sistema JBook. As linhas tracejadas na cor cinza representam agrupamentos,

Figura 4.3. Mapeamento entre suítes de teste para características através dos requisitos.

sejam eles entre suítes de teste, casos de uso ou características. Estes agrupamentos facilitam a ilustração do mapeamento. As setas de cor preta indicam o relacionamento.

Os relacionamentos ocorrem sempre entre *(i)* um grupo de suítes de teste e um grupo de casos de uso ou *(ii)* de um grupo de casos de uso e um grupo de características. Além disso, a Figura 4.3 apresenta um mapeamento de alto nível, onde os requisitos foram representados com o nome dos casos de uso. Assim, os testes ilustrados não estão relacionados aos casos de uso, mas a algum requisito que este tenha capturado. O agrupamento de características é feito na ocasião do relacionamento entre características e suas respectivas sub-características. Estas, por sua vez, são representadas em níveis inferiores da árvore. O ator responsável pelos relacionamentos pode ser um especialista do domínio da aplicação que está sendo alvo da extração.

Como foi apresentado no Capítulo 2, existem diferentes tipos de teste de software. Neste trabalho, optou-se por utilizar somente testes de integração por serem comumente produzidos durante o desenvolvimento do software. Além disso, os testes de integração cobrem boa parte dos resquisitos elicitados para um sistema. A localização das características através do método aqui proposto é fortemente relacionada com o tipo de teste. Por exemplo, características relacionadas a requisitos funcionais devem ser identificadas através do uso de testes de integração, enquanto que, requisitos não-funcionais possivelmente através de testes de desempenho e estresse, dada a relação entre este tipo de teste e o tipo do requisito.

4.5 Localização das Características no Código-fonte

Uma vez que o passo detalhado na Seção 4.4 foi concluído e as suítes de teste foram devidamente associadas às características a serem identificadas, dar-se-á procedimento à *localização da implementação das características no código-fonte* do sistema alvo. Neste quarto passo, duas estratégias distintas são definidas para a localização de código-fonte. A primeira busca encontrar *parte* do código que implementa uma característica (Seção 4.5.2). A segunda busca encontrar *todo* o código que implementa uma característica (Seção 4.5.2). Diferentemente dos primeiros três passos que, idealmente, são executados uma única vez, esse passo deve ser executado sempre que o código de uma nova característica deva ser localizado.

Para que o objetivo seja alcançado em cada uma das estratégias, define-se quatro etapas a serem realizadas. A Figura 4.4 ilustra tais etapas. Estas etapas devem ser executadas sempre da necessidade da localização parcial ou completa do código que implementa uma determinada característica, doravante, *característica de interesse*. Cada uma das etapas é detalhada a seguir:

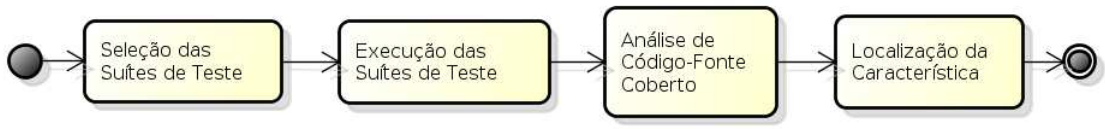

Figura 4.4. Processo de 4 etapas para localização do código de uma característica.

Etapa 1: nesta etapa é feita uma **seleção das suítes de teste** mapeadas para a características de interesse;

Etapa 2: nesta etapa é feita a **execução das suítes de teste** selecionadas na etapa anterior. Como resultado desta etapa tem-se uma porção do código do sistema executado, esta porção é dita *coberta* pelas suítes de teste. A Figura 4.5 ilustra como é feita a marcação da execução do código pelas suítes de teste no código-fonte;

Etapa 3: nesta etapa é feita uma **análise do código coberto** pelas suítes de teste executadas na etapa anterior. Esta análise é necessária para que seja possível a identificação do código que não pertence à característica de interesse, mas sim ao código necessário para que a implementação desta seja viável;

Etapa 4: por fim, nesta última etapa é feita a **localização do código** que implementa a característica de interesse. Esta etapa complementa o trabalho realizado na etapa anterior. No caso da localização parcial (Seção 4.5.1), ao final desta etapa, apenas parte do código da característica de interesse é identificado. No caso de localização completa (Seção 4.5.2), ao final desta etapa o código que implementa a característica de interesse poderá ser extraído do produto final.

```
public class Exemplar implements Serializable, Comparable<Exemplar> {

    private static final long serialVersionUID = 4448872540560235275L;

    public Exemplar() {
        this.dataCatalogacao = new Date();
        this.situacao = SituacaoExemplarEnum.DISPONIVEL.getValor();
        this.emprestimos = new ArrayList<Emprestimo>();
    }

    /** Atributo que identifica o exemplar */
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long idExemplar;

    /** Atributo que define a publicação que o exemplar pertence */
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "idPublicacao", nullable = false)
    private Publicacao publicacao;
```

Figura 4.5. Parte de código executado por uma suíte de teste.

Em seguida, cada uma das estratégias de localização é tratada em detalhes. Na Seção 4.5.1 é detalhada a localização parcial e na Seção 4.5.2 é detalhada a localização completa.

4.5.1 Localização parcial do código-fonte

A estratégia de identificação parcial do código-fonte consiste em utilizar todas as suítes de teste, relacionadas a uma característica, que foram executadas e o código por elas coberto. A porção do código localizada com esta estratégia é chamada de *semente* [Valente et al., 2012], pois ela permite a localização do restante do código-fonte. Para conseguir uma semente do código-fonte que implementa uma característica de interesse é aplicada a intersecção entre os conjuntos de código anotados pelas diferentes suítes de teste executadas.

A Figura 4.6 ilustra esta operação de intersecção. Dado que duas suítes de testes foram executadas para a localização do código da característica "Exemplar", "HashCodeExemplarTest" e "ValidacoesExemplarTest". A execução de cada uma delas produz

um conjunto de código executado. O código comum executado por ambas as suítes é utilizado para análise nas Etapas 3 e 4 e é ilustrado pela região delimitada pela área hachurada.

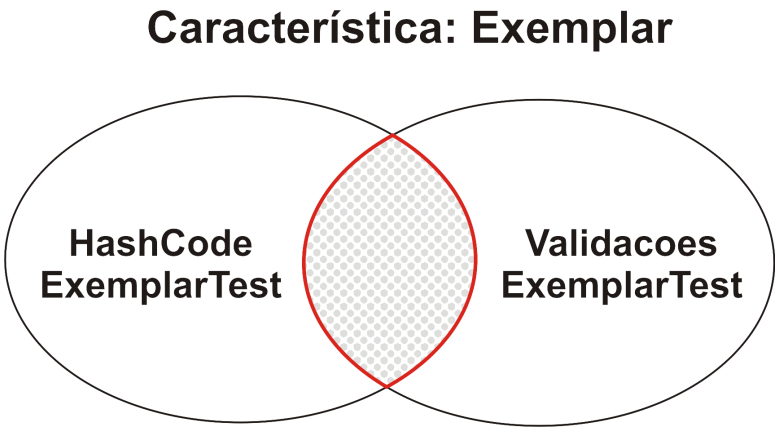

Figura 4.6. Interseção entre os conjuntos de código executados pelas suítes de teste.

Esta estratégia tem como objetivo identificar uma ou mais sementes para o código completo da característica. O uso desta estratégia é justificado quando uma semente é suficiente para realizar uma tarefa de manutenção demandada, ou ainda, para encontrar o restante código da característica de interesse, por exemplo, através de uma outra abordagem de localização baseada em expasão [Valente et al., 2012].

4.5.2 Localização completa do código-fonte

A estratégia localização completa do código-fonte da característica de interesse consiste no uso de todos os conjuntos de código executado pelas suítes de teste relacionadas a esta característica. Neste caso, é aplicada a união dos conjuntos de código executado.

A Figura 4.7 ilustra esta operação de união. Tomando o mesmo exemplo dado na estratégia parcial o código executado por ambas as suítes é utilizado para análise nas Etapas 3 e 4 e é ilustrado pela região delimitada pela área hachurada.

Esta estratégia tem como objetivo identificar todo ou a maior parte do código que implementa a característica de interesse e, portanto, deve ser utilizada quando uma semente não é suficiente para completar a tarefa demandada. A meta desta estratégia é facilitar a extração das características de interesse e consequentemente da LPS. Portanto, após executadas as quatro etapas deste passo é possível, utilizando esta estratégia, localizar o código que implementa a característica de interesse. A partir desta localização pode-se efetuar a extração da característica para a construção de uma LPS de um sistema inicialmente construído como um produto único.

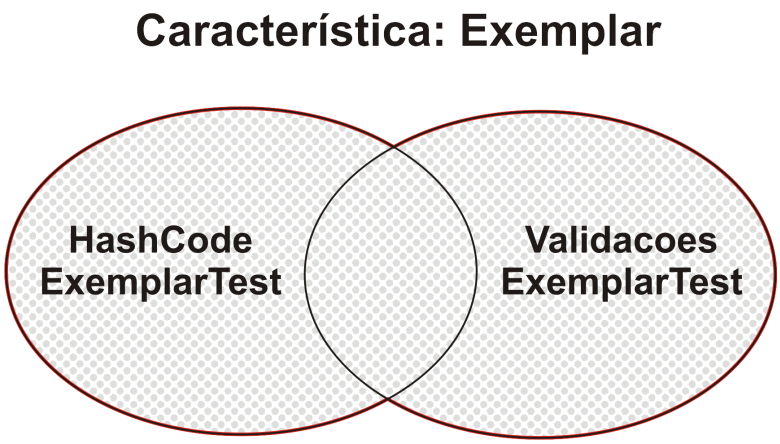

Figura 4.7. União entre os conjuntos de código executado pelas suítes de teste.

4.6 Resumo

Este capítulo apresentou o método proposto para extração de uma LPS em que a localização do código-fonte das características de interesse reutilizão dos artefatos de requisitos e teste de software. Assim, é possível construir uma LPS a partir de um sistema inicialmente construído como um produto único. O método é composto por 4 passos resumidos abaixo:

Passo 1: neste passo é feito a **derivação do modelo de características** da linha de produtos desejada;

Passo 2: neste passo é feita o **mapeamento dos requisitos do sistema para as características** identificadas no Passo 1;

Passo 3: neste passo é feito o **mapeamento de testes para as características** identificadas no Passo 1, através dos requisitos mapeados a elas no Passo 2;

Passo 4: por fim, neste último passo é feito a **localização do código** que implementa a característica de interesse.

Idealmente, os Passos 1, 2 e 3 são executados um única vez. Por outro lado, que o Passo 4 deve ser executado sempre da necessidade de localizar uma nova característica para a extração da LPS. O capítulo seguinte apresenta uma ferramenta desenvolvida para prover apoio automatizado à execução do método proposto.

Capítulo 5

TaBuLeTa

Este capítulo apresenta a Ferramenta de Localização de Características Baseado em Testes (TaBuLeTa)[1], desenvolvida com o intuito de automação parcial do método proposto no Capítulo 4. A ferramenta foi desenvolvida como um *plug-in* para o Ambiente de Desesenvolvimento Integrado (IDE) Eclipse. Escolheu-se esta IDE devido à sua grande popularidade tanto na comunidade acadêmica quanto industrial, bem como sua facilidade de extensão por meio de *plug-ins*. Além disso, é nosso objetivo integrar TaBuLeTa ao ambiente real de desenvolvimento para facilitar sua utilização por desenvolvedores.

Com a proposta do método de localização de características baseado em testes passa a ser igualmente importante prover apoio ferramental para que ele possa ser utilizado na prática. A automação, ainda que parcial, do método possibilita diminuir o tempo gasto na execução do mesmo e consequentemente na extração de uma Linha de Produtos de Software (LPS). Dos quatro passos do método, TaBuLeTa provê automação do terceiro e parte do quarto passo. Ou seja, o mapeamento de testes para características e a localização do código-fonte que implementa a característica de interessse. Para a automação destes passos, tomou-se como base ferramentas previamente desenvolvidas independentemente cujas finalidades serviam ao nosso propósito.

O restante do capítulo é organizado como segue. A arquitetura da ferramenta é apresentada na Seção 5.1. O projeto e detalhes de implementação aparecem na Seção 5.2. Explanações sobre o seu funcionamento são apresentados na Seção 5.3. Finalmente, um resumo do capítulo é feito na Seção 5.4.

[1]Disponível em: http://alcemirsantos.github.com/tabuleta

5.1 Arquitetura

TaBuLeTa é uma extensão da ConcernMapper [Robillard & Weigand-Warr, 2005] e utiliza a EclEmma[2] como apoio para encontrar o código coberto pela execução dos testes. A Figura 5.1 ilustra a arquitetura da ferramenta e seu relacionamento com o IDE Eclipse. Como é apresentado na figura, a TaBuLeTa incorpora todas as funcionalidades da ferramenta ConcernMapper. Além disso, ela utiliza funcionalidades do EclEmma. De fato, a diferença entre incorporar e usar as funcionalidades é feita por que TaBuLeTa reutiliza todo o modelo definido por ConcernMapper (por exemplo, ConcernNode, ConcernModel, etc.) e somente recupera algumas das meta-informações sobre a cobertura disponibilizada pela EclEmma.

Segundo os autores da ConcernMapper [Robillard & Weigand-Warr, 2005], a principal motivação por trás da criação desta ferramenta foi oferecer uma plataforma simples e extensível para experimentação com técnicas avançadas de separação de interesses. Não por acaso, uma dessas técnicas é a localização de características, a qual pode automaticamente produzir uma estimativa de implementação de uma características pela análise do rastro de execução de sistemas [Eisenbarth et al., 2003]. Isto motivou a decisão por elaborar TaBuLeTa como uma extensão da ConcernMapper uma vez que ela permite adicionar unidades computacionais, como classes, métodos e atributos à um dado interesse. No nosso caso, os interesses mapeados são características da linha de LPS.

EclEmma é um *plug-in* para o IDE Eclipse que provê meta-dados sobre a cobertura de código Java a partir da execução de testes. Essa ferramenta permite *(i)* um ciclo rápido de desenvolvimento/teste, onde, os botões de lançamento de testes, a exemplo do *JUnit* [Massol & Husted, 2003], facilitam a análise da cobertura do código; *(ii)* uma análise rica sobre a cobertura, onde, os resultados da cobertura são imediatamente sumarizados e sombreados no editor de código Java do Eclipse; e tudo disso, *(iii)* de forma não invasiva. EclEmma não requer a modificação de seus projetos ou a execução de quaisquer configuração adicional.

Para o mapemento de testes para características, utilizou-se as facilidades implementadas pelo *plug-in* Eclipse chamado *ConcernMapper*. Já a localização do código que implementa a característica de interesse utilizou-se as facilidades providas pela ferramenta *EclEmma*, outro *plug-in* para IDE Eclipse. Os pré-requisitos e o modo de instalação da ferramenta está disponível no Anexo A. TaBuLeTa é distribuída sob a licensa pública do Eclipse versão 1.0.

[2]EclEmma: Cobertura de código Java para Eclipse. Disponível em: *http://www.eclemma.org*. Acesso em 11 de Novembro de 2012.

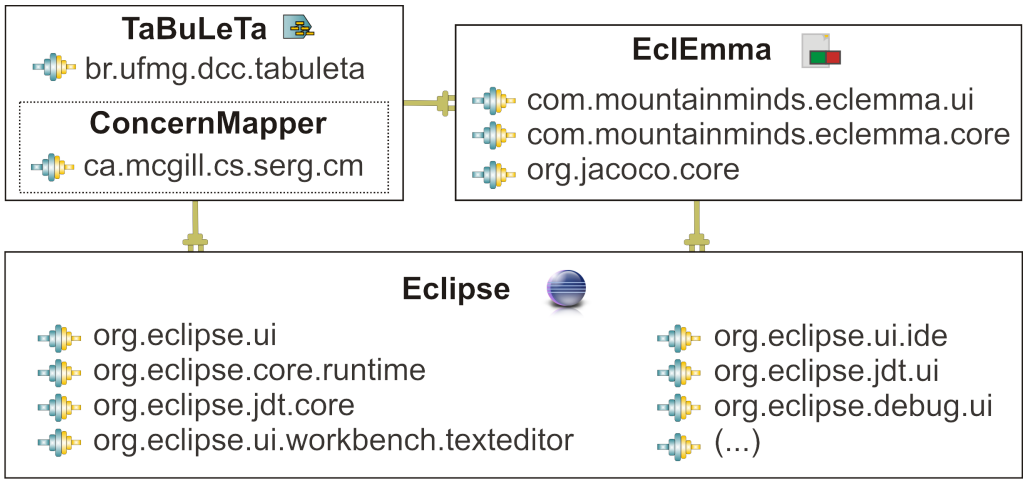

Figura 5.1. Relacionamento TaBuLeTa-ConcernMapper-EclEmma-Eclipse.

Além da saída visual do mapeamento de testes para características, TaBuLeTa permite ao usuário persistir o mapa em um arquivo com formato compatível com o utilizado pelo ConcernMapper (o arquivo com extensão *.cm*). Essa persistência é possibilitada pelo reuso do Modelo de Interesses definido pelo ConcernMapper. Esta integração é benéfica para o usuário do ConcernMapper que decidir utilizar TaBuLeTa.

TaBuLeTa pode produzir alguns arquivos durante sua utilização. Um destes arquivos é o mapa dos testes para as características (primeiro *.cm*) que permite a geração de uma suíte de testes para uma dada característica (arquivo *.java*). Após a execução desta suíte de testes em modo de cobertura com a EclEmma, TaBuLeTa pode gerar um novo mapa entre o código que fora coberto pela suíte de teste e a característica de interesse. Este novo mapa é gravado em um segundo arquivo *.cm*. TaBuLeTa ainda permite comparar o mapa do código coberto a um mapa gerado manualmente e para fins experimentais, calcular as métricas precisão e cobertura. Além disso, a ferramenta pode produzir uma intersecção entre n arquivos com mapa de cobertura referentes à característica de interesse. Esta interseção é gravado como saída da ferramenta em um terceiro arquivo *.cm*.

5.2 Projeto e Implementação

A ferramenta TaBuLeTa é escrita em linguagem Java e utiliza a Interface de Programação de Aplicativos (API) do Eclipse para implementar as funcionalidades necessárias. A API do Eclipse é construída baseada na composição de diversos *plug-in*. TaBuLeTa não utiliza todos os *plug-ins* do Eclipse, mas somente os listados na Figura 5.1. As reticências na imagem representam todos os demais *plug-in* que não foram utilizados.

Para a ferramenta TaBuLeTa, foi necessário o desenvolvimento de alguns componentes existentes na API do Eclipse. São eles:

Visões: são cada uma das abas existentes no IDE Eclipse. *Package Explorer*, *Tasks*, *Outline* são exemplos de visões existentes;

Menus de contexto: é o tipo de menu ativado ao clique do botão direito do *mouse* em diferentes contextos da IDE. No caso da TaBuLeTa foram construídos menus de contexto somente nas visões *Package Explorer* e *Test2Feature Mapping*, os quais serão discutidos na Seção 5.3;

Página de preferências: é um local do IDE Eclipse reservado para configuração do comportamento dos *plug-ins* instalados. Um exemplo bem conhecido é a página de preferências para a configuração do *Build Path* do Java. As páginas de preferências são acessíveis através da barra de menus no topo do IDE. Especificamente, ela está localizada no menu "Window" e opção "Preferences".

Quanto a interação com o usuário, eventualmente a TaBuLeTa escreve alguns arquivos, comentados na Seção 5.1. A saber, três arquivos de mapeamento com extensão *.cm* e suítes de teste com extensão *.java* – compatíveis com a versão 4 do JUnit [Massol & Husted, 2003]. Os arquivos *.cm* são escritos baseados no modelo de interesses definido como uma gramática pelo ConcernMapper na forma de Bakus-Naur [Aho et al., 1985] como mostra a Figura 5.2. Um modelo de interesse consiste de zero ou mais interesses, onde cada interesse mapeia os nomes dos elementos ponderados. Um elemento ponderado é uma associação entre um objeto de um tipo (classe, método, atributo) e um valor que indica o grau de relevância do objeto para o interesse.

$$< modelo >::=< interesse > *$$
$$< interesse >::=< nome >< elemento - ponderado > *$$
$$< elemento - ponderado >::=< objeto >< grau >$$

Figura 5.2. Definição do modelo de interesse do ConcernMapper na forma Bakus-Naur.

O modelo das suítes de testes compatíveis com JUnit versão 4 é apresentado na Figura 5.3. Neste exemplo "ATest.class" é uma classe que é adicionada à suíte e "import (...);" são os *imports* respectivos às classes adicionadas à suíte e necessários para o correto funcionamento da mesma. Podem ser adicionadas n classes à suíte.

A Tabela 5.1 apresenta dados de quantitativos da implementação da ferramenta TaBuLeTa. A ferramenta foi implementada em 2.712 linhas de código (LOC) em 53

```
package tabuleta.testsuites;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;
import (...);

@RunWith(Suite.class)
@SuiteClasses({
ATest.class
})
public class <Nome da Característica>TestSuite{
}
```

Figura 5.3. Modelo de uma suíte de testes gerada pela TaBuLeTa.

Tabela 5.1. Elementos de implementação da TaBuLeTa.

Elementos	Quantidade
Pacotes	9
Classes	53
Propriedades (.properties)	3
XML	1
LOC Java	2.712
Plugins EclEmma	3
Plugins Eclipse	7

classes Java distríbuídas em nove (9) pacotes com o apoio de um (1) arquivo XML e três (3) arquivos *.properties* para configuração do plugin.

5.3 TaBuLeTa em Ação

Nesta seção será descrito um fluxo de trabalho com a ferramenta TaBuLeTa. A Figura 5.4 ilustra o fluxo principal referente à visão "Test2Feature Mapping (T2FM)" (Figura 5.5). A Figura 5.6 mostra dois fluxos adicionais, "A" e "B", correspondentes às ações indicadas pelos números 2 e 3 na Figura 5.7, respectivamente.

Como fluxo principal tem-se a localização da característica através da execução dos testes. O produto final deste é o sobreamento do código coberto pelos testes e um arquivo de extensão *.cm* com a representação do código-fonte da característica de interesse. A Figura 5.4 apresenta como isso pode ser alcançado.

Primeiro o usuário deve mapear as classes de teste para as características (região

1 da Figura 5.5). Para isso, na visão "T2FM", o usuário adiciona as características as quais os testes devem ser mapeados. Em seguida, o usuário localiza as classes de teste na visão "Project Explorer" e faz o mapeamento para a respectiva característica. Concluído o mapeamento, o usuário aciona a geração da suíte de testes na visão "T2FM" para a característica de interesse (ação 2 da Figura 5.5). Então o usuário pode executar a suíte de testes com a EclEmma em modo de cobertura. Após a execução, o usuário pode acionar a geração do arquivo *.cm* que mapeia a cobertura dos testes para a característica selecionada (ação 3 da Figura 5.5). Ao seguir estes passos, o usuário estará executando a estratégia de localização completa do código-fonte apresentada na Seção 4.5.2 do Capítulo 4.

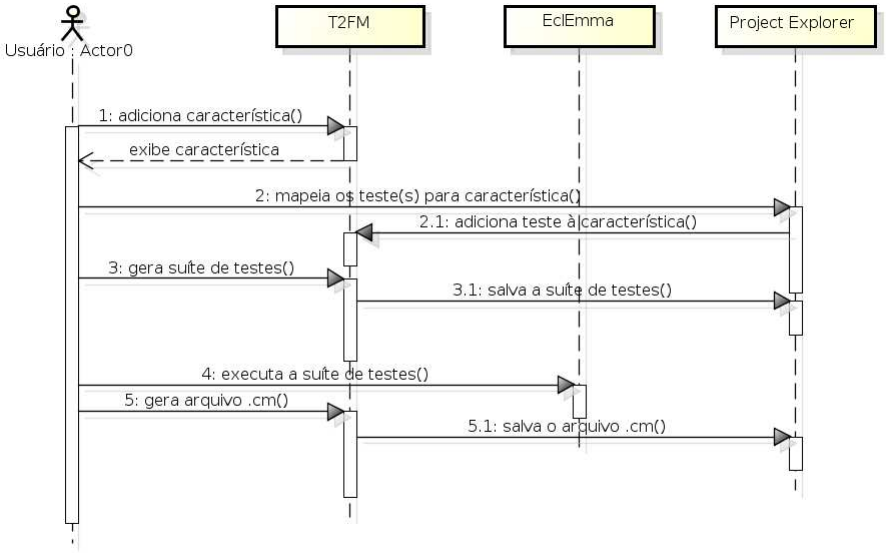

Figura 5.4. Interação com usuário da visão de mapeamento.

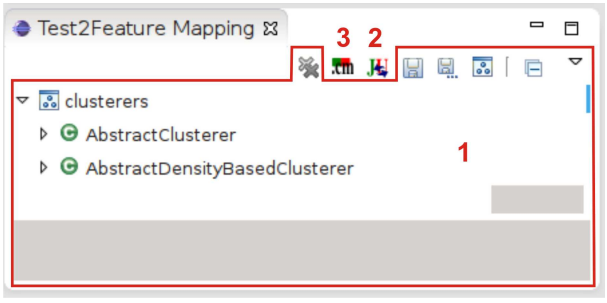

Figura 5.5. Visualização do Mapeamento de Testes para Características.

Especificamente, para o usuário mapear os testes para as características ele pode ir diretamente ao *Project Explorer* do Eclipse. Outra possibilidade é adicionar algumas

características, as quais serão alvo do mapeamento, na visão *Test2Feature Mapping* que provê o mapeamento de testes para características (Figura 5.5). No *Project Explorer* também é possível disparar o menu de contexto *Add to Feature* disponível por um clique com o botão direito do *mouse* em uma classe[3].

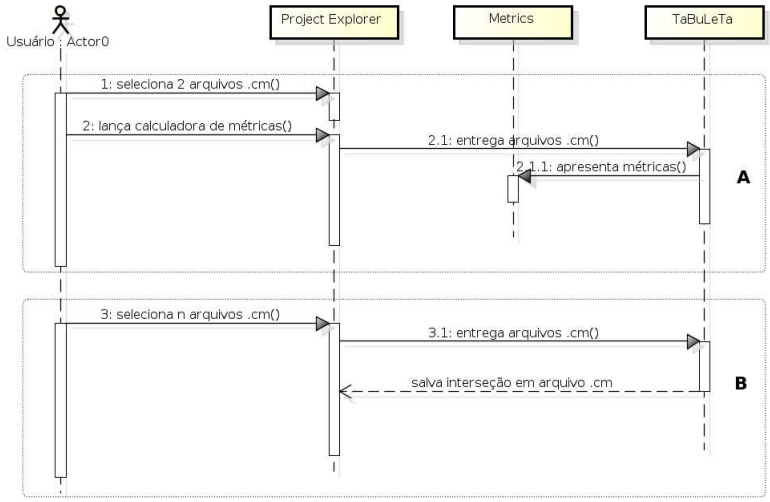

Figura 5.6. Interações do usuário para o lançamento de ações na TaBuLeTa.

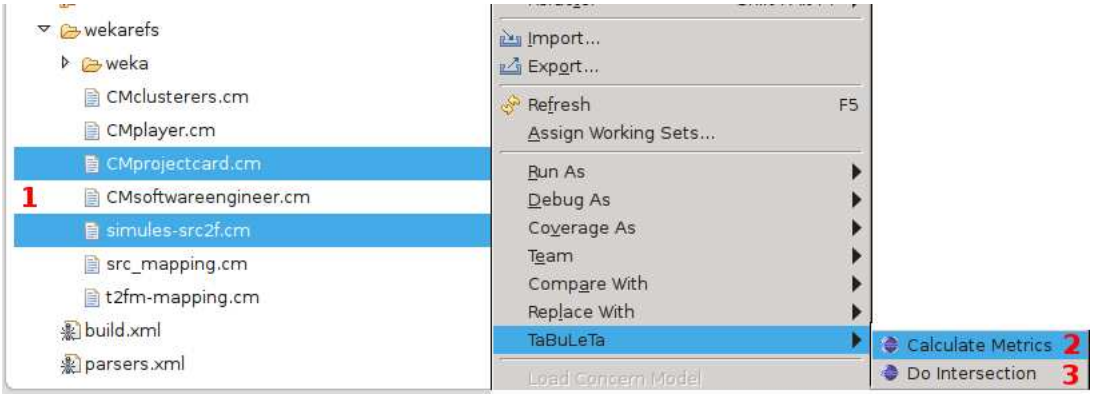

Figura 5.7. Visualização do menu de lançamento de ações da TaBuLeTa.

De forma complementar, TaBuLeTa apresenta dois fluxos adicionais iniciados a partir da seleção de dois ou mais arquivos *.cm* como indicado na posição 1 da Figura 5.7. O primeiro fluxo adicional é a intersecção de arquivos *.cm* (item 3 da Figura 5.7). Com a intersecção de arquivos é possivel criar um novo arquivo *.cm* com a intersecção de n outros arquivos de mapeamento. Isto permite ao usuário executar a

[3]Arquivos *.java* apesar de disparar o menu de contexto não serão adicionados ao mapeamento. Você deve expandir o arquivo no *package explorer* e repetitir esta ação com a classe correspondente ao arquivo

estratégia de localização parcial do código-fonte apresentada na Seção 4.5.1 do Capítulo 4. O segundo fluxo permite a comparação de um par de arquivos *.cm* para cálculo de métricas (item 2 da Figura 5.7). Desta forma, a ferramenta permite a exibição de métricas como *verdadeiros positivos*, *falsos positivos*, *falsos negativos*, *precisão*, *cobertura* e $f_1 - score$. As métricas são mostradas na visão *Metrics* que provê a visualização de métricas em uma tabela (Figura 5.8). Estas métricas são detalhadas na Seção 6.1 do Capítulo 6, exceto $f_1 - score$, que é uma média hamônica entre precisão e cobertura calculada de acordo com a Equação 5.1.

$$f_1 - score = \frac{2 * precisão * cobertura}{precisão + cobertura} \tag{5.1}$$

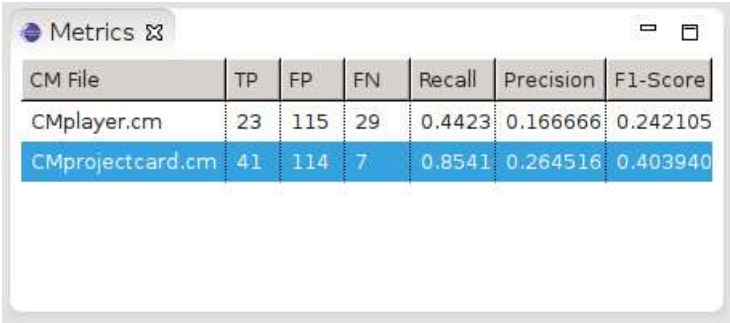

Figura 5.8. Visualização de Métricas.

5.4 Resumo

Este capítulo apresentou o protótipo de uma ferramenta entitulada TaBuLeTa desenvolvida para automatizar parte do método proposto no Capítulo 4. A ferramenta foi concebida como um *plug-in* para o IDE Eclipse e utiliza outros dois *plug-ins* EclEmma e ConcernMapper. TaBuLeTa provê ainda uma página de preferências onde são configurados alguns parâmetros necessários para a execução de suas ações. Além disso, ela possui um menu de contexto para permitir o lançamento de duas ações adicionais:

Intersecão de arquivos .cm: habilitada quando da seleção de dois ou mais arquivos *.cm*;

Cálculo de métricas: habilitada quando, e somente quando, da seleção de dois arquivos *.cm*.

A ferramenta TaBuLeTa é composta por duas visões:

Test2Feature Mapping: que provê o mapeamento de testes para as características de interesse;

Metrics: que provê a exibição das métricas resultantes da comparação de um par de arquivos *.cm*.

O capítulo seguinte apresenta a avaliação do método proposto, bem como a utilização da ferramenta TaBuLeTa.

Capítulo 6

Avaliação

Este capítulo é dedicado à avaliação do método proposto no Capítulo 4. O estudo experimental foi concebido, estruturado e conduzido na forma de estudos de caso. Tomou-se como base os conceitos apresentados por Wohlin e outros [Wohlin et al., 2012] para este tipo de avaliação em engenharia de software. O foco deste trabalho é o uso dos testes como apoio à localização de características. Portanto, o foco da avaliação será o último passo do método; ou seja, localização de características baseado em testes. Os demais passos são avaliados indiretamente e uma avaliação sistemática de cada um deles vai além deste trabalho.

O restante do capítulo está organizado como segue. Na Seção 6.1 apresenta-se a definição da avaliação (objetivo, questões de pesquisa e métricas). O planejamento é feito na Seção 6.2 que traz o processo de avaliação, bem como, o método de coleta de dados utilizando os sistemas alvo (apresentados na Seção 6.3). A partir daí são apresentados os dados coletados. A construção do modelo de características é exibida na Seção 6.5. Em seguida são aprsentados os resultados da localização de características parcial (Seção 6.6) e completa (Seção 6.7), além da avaliação de escalabilidade (Seção 6.8). Finalmente, as questões de pesquisa discutidas na Seção 6.9, as limitações e ameaças à validade são discutidas na Seção 6.10 e a Seção 6.11 resume o capítulo.

6.1 Definição

Esta seção consiste na definição do objetivo e questões de pesquisa para avaliação do método proposto no Capítulo 4. Assim, dado o objetivo do estudo, foram definidas duas questões a serem respondidas e três métricas para auxiliar na resposta à estas perguntas. O objetivo, questões e métricas são detalhados em seguida.

Objetivo: O objetivo deste estudo experimental é avaliar o método proposto neste trabalho com respeito a sua *viabilidade*, *efetividade* e *escalabilidade*.

Questões: Para a condução da avaliação definiu-se as seguintes questões de pesquisa:

Q_0**:** Os testes de integração são efetivos para a localização de uma semente do código-fonte que implementa uma característica de interesse?

Q_1**:** Os testes de integração são capazes de localizar o código-fonte que implementa uma característica de interesse?

Métricas: Uma vez que as questões foram definidas é preciso utilizar-se de medidas para ajudar a respondê-las. Para isto foram utilizadas as métricas *precisão*, *cobertura* e *acurácia*. Estas três métricas são definidas em função de quatro medidas auxiliares descritas abaixo:

Verdadeiros positivos (TP): mede a quantidade de linhas de código relevantes que foram executadas;

Verdadeiros negativos (TN): mede a quantidade de linhas de código não relevantes que não foram executadas;

Falsos positivos (FP): mede a quantidade de linhas de código não relevantes que foram executadas;

Falsos negativos (FN): mede a quantidade de linhas de código relevantes que não foram executadas;

O verdadeiros (positivos e negativos) representam os acertos do método. Por outro lado, os falsos (positivos e negativos) indicam os erros. As métricas *precisão*, *cobertura* e *acurácia* são definidas nas equações 6.1, 6.2 e 6.3, respectivamente. A Figura 6.1 ilustra a definição de cada uma delas. O conjunto U representa todas as linhas de código do sistema avaliado. O conjunto **A** representa as linhas de código que foram executadas e o conjunto **B** representa as linhas de código que pertencem a uma dada característica e, consequentemente, consideradas relevantes para o nosso propósito.

Precisão (*p*): (do inglês *precision*) esta métrica mede a fração de linhas de código que foram recuperadas e são consideradas relevantes para o nosso propósito;

$$p = \frac{TP}{TP + FP} \tag{6.1}$$

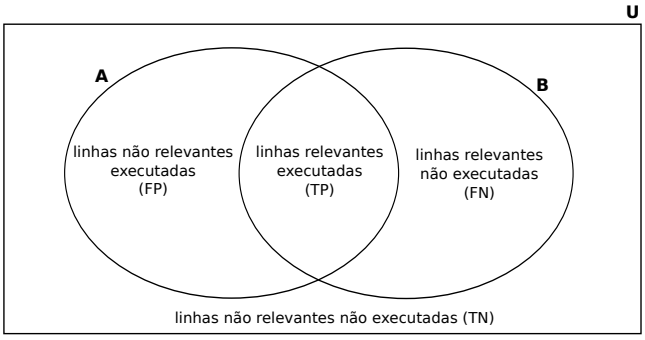

Figura 6.1. Ilustração das métricas utilizadas.

Cobertura (r): (do inglês *recall*) esta métrica mede a fração de linhas de código consideradas relevantes que foram recuperadas;

$$r = \frac{TP}{TP + FN} \tag{6.2}$$

Acurácia (a): (do inglês *accuracy*) esta métrica mede a fração de acertos obtidos com a utilização do método.

$$a = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.3}$$

6.2 Planejamento

Dado o caráter multidisciplinar da engenharia de software, muitas das questões de pesquisa são adequadas ao "estudo de caso" [Wohlin et al., 2012]. Por restrição de tempo e recursos, este estudo investiga a aplicação do método proposto em um pequeno número de instâncias, o que por definição é um "estudo de caso". Optou-se, então, por esta estratégia de avaliação para que fosse possível a coleta de dados para o nosso propósito específico. De acordo com Wohlin e outros [Wohlin et al., 2012], conduzir um estudo de caso requer *(i)* planejamento, *(ii)* preparação da coleta de dados, *(iii)* coleta de dados, *(iv)* análise dos dados coletados e *(v)* relatório. Portanto, procedeu-se a escolha dos sistemas, construção dos oráculos, localização de características de acordo com o método proposto, comparação da localização com o oráculo e o cálculo das métricas.

Planejamento. A escolha dos sistemas foi feita levando-se em consideração as condições de avaliação. Primeiro, existe a necessidade da existência de testes automatizados e de alguma documentação dos sistemas avaliados. Segundo, a seleção dos sistemas de tamanho aceitável para viabilizar o estudo. Num primeiro momento, a avaliação foi conduzida manualmente e, portanto, selecionou-se dois sistemas de pequeno

porte. Sistemas de grande porte dificultariam a coleta de dados e prejudicariam a análise posterior. No segundo momento, selecionou-se um sistema de grande porte para avaliação da escalabilidade do método com a utilização da ferramenta Ferramenta de Localização de Características Baseado em Testes (TaBuLeTa). Mais detalhes sobre os sistemas alvo são descritos na Seção 6.3.

Preparação e coleta de dados. Para verificar a corretude dos dados resultantes da aplicação do método proposto, foram produzidos documentos de referência contendo o resultado esperado, que chamamos "oráculo". Após a construção dos "oráculos" procedeu-se a aplicação do método de localização, o que produziu novos artefatos. Estes artefatos permitiram a comparação da localização das características feita com o método em relação aos "oráculos". Por fim, esta comparação permitiu o cálculo das métricas descritas na Seção 6.1. Mais detalhes sobre a preparação e coleta de dados são apresentados na Seção 6.4.

Análise dos dados coletados. A Seção 6.5, 6.6, 6.7 e 6.8 apresentam os resultados da coleta de dados. Além da apresentação dos dados é feita uma análise dos resultados obtidos com a utilização de ambas as estratégias definidas na Seção 4.5, a saber, localização parcial e localização total de código-fonte. Em seguida, na Seção 6.9 é feita uma discussão sobre as questões de pesquisa à luz dos dados coletados e questões de pesquisa.

Relatório. Por fim, o relatório inclui a produção desta dissertação para apresentação do método e este capítulo, especificamente, para o relato da experiência. Em adição, um artigo foi publicado com a parte conduzida manualmente deste estudo de caso no "28^{th} Symposium on Applied Computing" promovido pela Associação para Maquinaria de Computação (ACM) no ano de 2013.

6.3 Sistemas Alvo

Todos os sistemas utilizados na avaliação deste estudo foram desenvolvidos em linguagem Java e proveem os artefatos necessários para a execução do estudo. Foram gastos em torno de 3 semanas nos estudos de caso manuais com o JBook (Seção 6.3.1 e WebStore (6.3.2) e 2 dias no estudo de caso com a ferramenta utilizando o sistema Ambiente para Análise de Conhecimento de Waikato (WEKA) (Seção 6.3.3). Este tempo é uma estimativa e pode ter variado para mais ou para menos, uma vez que o "tempo de execução" não foi auferido. Em seguida, os três sistemas utilizados na avaliação são descritos, a saber, JBook, WebStore e WEKA.

6.3.1 JBook

JBook é um sistema de gerenciamento de biblioteca desenvolvido em aproximadamente *1 Mil Linhas de Código (KLOC)*. O código deste sistema está disponível na Internet[1]. Este é um sistema real, utilizado para o gerenciamento de uma biblioteca, desenvolvido por uma empresa de software brasileira. No sistema JBook, os usuários podem ser classificados com três papéis diferentes: *leitor*, *bibliotecário* ou *administrador*.

Enquanto leitores e bibliotecários têm acesso restrito à algumas funcionalidades, os administradores têm permissão para fazer todas as ações disponíveis no JBook. Este sistema permite usuários, dependendo de seus papéis dentro do sistema, executar: *(i)* registro de novas publicações e seus exemplares; *(ii)* gerenciar os empréstimos; e *(iii)* notificar a aquisição de novos exemplares e a aproximação de datas de devolução de livros emprestados através do envio de e-mails.

6.3.2 WebStore

WebStore é um sistema de simulação de comércio eletrônico desenvolvido em aproximadamente *1 KLOC*. O código deste sistema está disponível na Internet[2]. O WebStore foi implementado para fins acadêmicos com foco na maioria das características de uma loja Web real [Ferreira et al., 2011] a partir do "Java PetStore" (uma aplicação de referência Java, Edição Empresarial (J2EE))[Filho et al., 2006].

Este sistema permite que os usuários executem: *(i)* a inserção de novos produtos; *(ii)* a visualização dos produtos inseridos por categoria e por data; *(iii)* ordenar compra de produtos cadastrados; além de *(iv)* finalizar o pedido com diferentes meios de pagamento, como o pagamento padrão, *paypal* e boleto bancário.

6.3.3 WEKA

O WEKA [Hall et al., 2009] é um sistema grande aceitação tanto na comunidade acadêmica quanto na indústria. Ele tem uma comunidade ativa e já foram baixadas mais de 1.4 milhões cópias através da Internet[3]. O projeto WEKA tem como objetivo prover uma abrangente coleção de algoritmos de aprendizado de máquina e ferramentas de processamento de dados tanto para pesquisadores quanto para a comunidade industrial [Hall et al., 2009].

WEKA possui diversas interfaces gráficas de usuário, as quais possibilitam carregar dados, transformá-los, classificá-los, agrupá-los, entre outras atividades. O sistema

[1]Disponível em: http://sourceforge.net/projects/jbookweb/
[2]Disponível em: http://sourceforge.net/projects/webstorespl/
[3]Disponível em: http://www.cs.waikato.ac.nz/ml/weka/downloading.html

implementado em Java foi escrito em mais de *144 KLOC* e permite que usuários implementem funcionalidades que, eventualmente, não tenham sido disponibilizadas.

6.4 Coleta de Dados

Com os sistemas alvo escolhidos, foi possível a construção dos "oráculos". Para cada um dos sistemas foram escolhidos seis características. Para a escolha das características levou-se em consideração sua granularidade em termos de linhas de código, características mais gerais e/ou mais específicas e sua importância dentro do sistema. Exceto para o sistema WEKA, a caracterização da importância de cada características foi determinada subjetivamente de acordo com o conhecimento prévio do pesquisador e dada a quantidade de testes automatizados existentes. Quanto maior a quantidade de testes para uma característica, mais importante era considerada esta característica para o sistema. No caso do sistema WEKA, após breve explanação do trabalho, um pesquisador com experiência no sistema apontou as características que poderiam ser alvo do estudo.

Os oráculos foram construídos por desenvolvedores com experiência nos sistemas alvo e revisados em conjunto com o autor deste trabalho posteriormente. A construção dos oráculos nos estudos de caso conduzidos manualmente foi realizada com o uso da técnica de sombreamento [Figueiredo et al., 2008]. A Tabela 6.1 apresenta as características de cada um dos sistemas alvo que foram analisadas e, portanto, tiveram seu oráculo construído.

Tabela 6.1. Características analisadas em cada um dos sistemas alvo.

JBook	**WebStore**	**WEKA**
Empréstimo	Pagamento	Filtros
Relatório de Empréstimo	Paypal	Árvore de Decisão[a]
Solicitação de Empréstimo	Boleto Bancário	Algoritmos de Agrupamento
Publicação	Exibição	Cobweb[b]
Exemplar	Fechamento de Pedido	Seleção de Atributos
Usuário	Gerenciamento de Conteúdo	Meta-algoritmo de Classificação

[a] Árvore de decisão é um tipo específico de algoritmo de classificação.
[b] Cobweb é um tipo específico de algoritmo de agrupamento.

A coleta de dados foi feita como segue. Nos estudos de caso dos sistemas JBook e WebStore (manuais) após a execução dos passos 1, 2, 3 e 4 do método obteve-se

um conjunto de código anotado de acordo com a cobertura das suítes de testes para cada característica de interesse. Portanto, nos estudos de caso manuais, obteve-se doze (12) conjuntos de código anotado para cada sistema (JBook e WebStore). Seis (6) deles referentes à estratégia de localização parcial de código-fonte e os outros seis (6) referentes à estratégia de localização completa de código-fonte.

Depois da anotação do código foi feita uma comparação entre pares de conjuntos de código anotado. Os pares foram formados por um oráculo da característica de interesse e um conjunto resultante da estratégia de localização parcial ou completa. Esta comparação foi feita para cada característica em ambas as estratégias, o que resultou em novos conjuntos de código anotado. Porém, desta vez foram anotados para o cálculo de métricas, e cada linha foi marcada como *verdadeiro positivo*, *verdadeiro negativo*, *falso positivo* ou *falso negativo* de acordo com o apresentado na Seção 6.1.

Para o estudo de caso automatizado, os artefatos utilizados para comparação foram os arquivos *.cm* gerados pela TaBuLeTa. De forma análoga, solicitou-se a um pesquisador com conhecimentos prévios do sistema WEKA para construir um arquivo *.cm* para representar o oráculo das características de interesse. Assim, com a utilização da ferramenta foi possível gerar um novo arquivo *.cm* com o código coberto e consequentemente fazer a comparação pareada e calcular as métricas também com a TaBuLeTa.

6.5 Construção do Modelo de Características

Como dito anteriormente, o foco dos estudos de caso é a localização de características com o uso dos testes e, portanto, manteve-se foco na avaliação do quarto passo do método. Os demais passos são avaliados indiretamente. A Tabela 6.2 mostra exemplos de características extraídas da documentação dos sistemas alvo com o uso das heurísticas apresentadas na Seção 4.2. Com os termos em mão, é possível analisar e organizá-los logicamente em um modelo de características. As Figuras 6.2, 6.3 e 6.4 ilustram os modelos de características parciais dos sistemas alvo, respectivamente, JBook, WebStore e WEKA. Os modelos derivados com ajuda das heurísticas são parciais vez que não procurou-se exaustivamente todas as características dos sistemas avaliados, mas acredita-se que eles refletem a linha de produtos final.

Em seguida, são apresentados os dados coletados com a aplicação do método proposto. Os sistemas foram avaliados através das estratégias de localização parcial (Seção 6.6) e completa (Seção 6.7) do código-fonte das características de interesse. A Seção 6.8 apresenta o estudo de escalabilidade realizado com a ferramenta TaBuLeTa.

Tabela 6.2. Exemplos Características identificadas pelas heurísticas.

Heurística	Característica
$H1$	JBook, WebStore, WEKA
$H2$	**JBook:** Registrar usuário, Atualizar usuário, Registrar exemplar, Cancelar empréstimo **WebStore:** Inserir produto **WEKA:** Seleção de atributos, Adicionar filtros, Meta-algoritmo de classificação, Algoritmo de agrupamento Cobweb
$H3$	**JBook:** Registrar usuário e Atualizar usuário podem ser agrupados por Usuário **WebStore:** Paypal e Boleto bancário podem ser agrupados por Pagamento **WEKA:** Algoritmos de Classificação, Algoritmos de Agrupamento,
$H4$	**JBook:** Escolher Papel do Usuário; Adicionar, Atualizar. **WebStore:** *[nenhum exemplo encontrado]* **WEKA:** *[nenhum exemplo encontrado]*
$H5$	**JBook:** Notificar Usuário **WebStore:** *[nenhum exemplo encontrado]* **WEKA:** *[nenhum exemplo encontrado]*
$H6$	**JBook:** Escolher Papel do Usuário **WebStore:** Exibir por Categoria e Exibir o mais Novo **WEKA:** *[nenhum exemplo encontrado]*
$H7$	**JBook:** E-mail com HTML e E-mail com Texto Puro **WebStore:** *[nenhum exemplo encontrado]* **WEKA:** *[nenhum exemplo encontrado]*
$H8$	**JBook:** Notificar Novo Exemplar, Notificar Data de Devolução **WebStore:** Paypal, Boleto Bancário **WEKA:** *[nenhum exemplo encontrado]*
$H9$	**JBook:** Validar Dados do Usuário **WebStore:** *[nenhum exemplo encontrado]* **WEKA:** *[nenhum exemplo encontrado]*
$H10$	Não encontrou-se nenhum exemplo.

6.6 Localização Parcial do Código-fonte

Esta seção apresenta o resultado da localização de características utilizando a estratégia de localização parcial do código-fonte. Inicialmente, são apresentados os resultados colhidos para o sistema JBook e, em seguida, para o sistema WebStore.

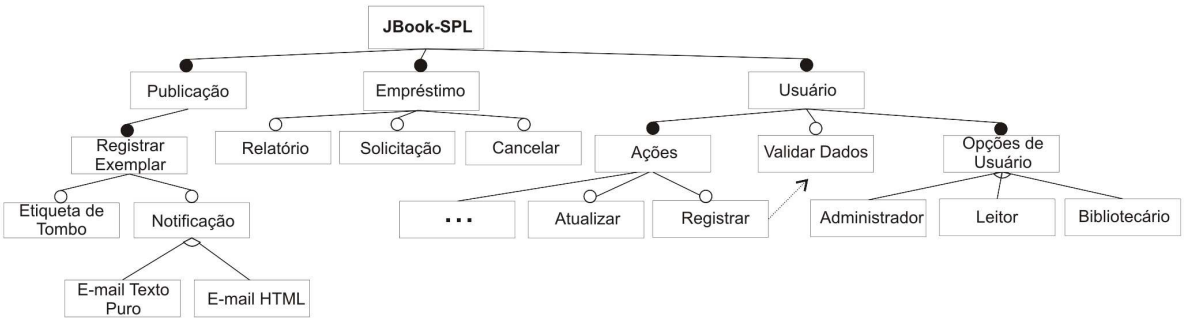

Figura 6.2. Modelo de características do JBook.

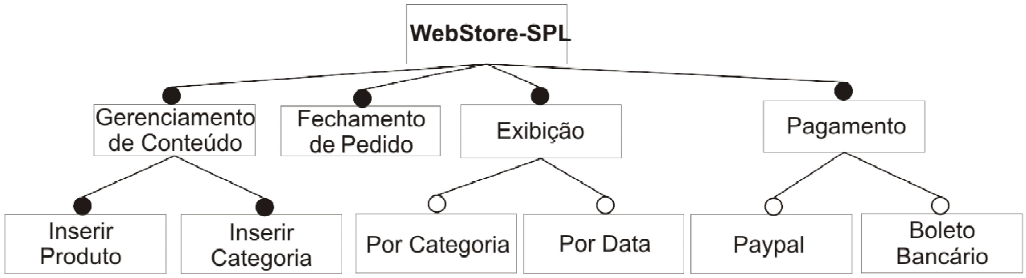

Figura 6.3. Modelo de características do WebStore.

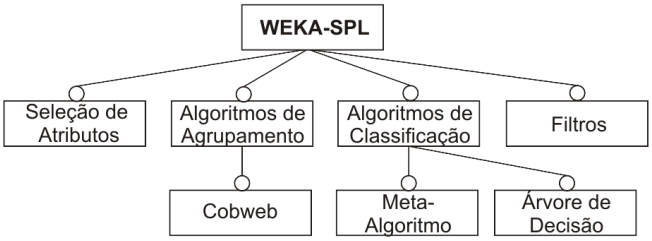

Figura 6.4. Modelo de características do WEKA.

6.6.1 JBook

As Figuras 6.5 e 6.6 mostram os resultados para as métricas de precisão e acurácia para o sistema JBook referentes à localização parcial do código fonte da característica. As características "Empréstimo", "Exemplar", "Publicação" e "Usuário" alcançaram taxas de precisão acima de 85% (Figura 6.5). Estes resultados indicam que o código anotado é quase completamente dedicado à implementação da característica em questão, isto é, poucos falsos positivos. No caso das características "Solicitação de empréstimo" e "Relatório de empréstimo", os resultados não foram tão bons quanto o esperado. No entanto, pensa-se que estes valores são relacionados ao fato de que ambas as características são refinamentos da característica "Empréstimo" e as suítes de testes específicas

de cada característica terminam por executar código alheio exatamente pela existência do relacionamento com a característica "Empréstimo".

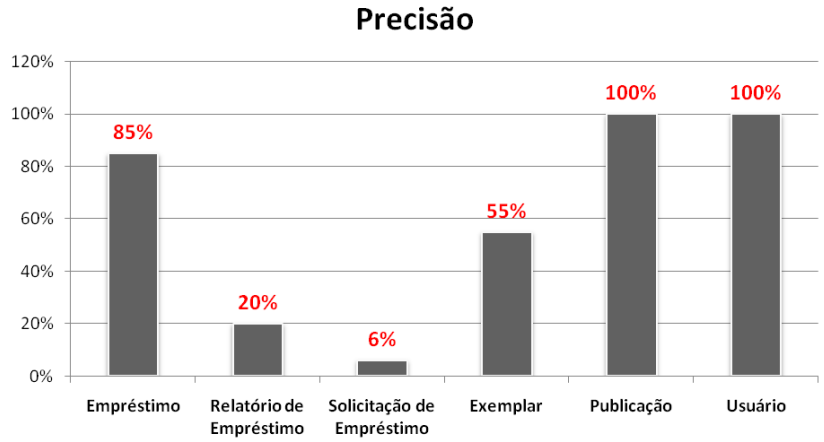

Figura 6.5. "Precisão" para o JBook na localização parcial do código-fonte.

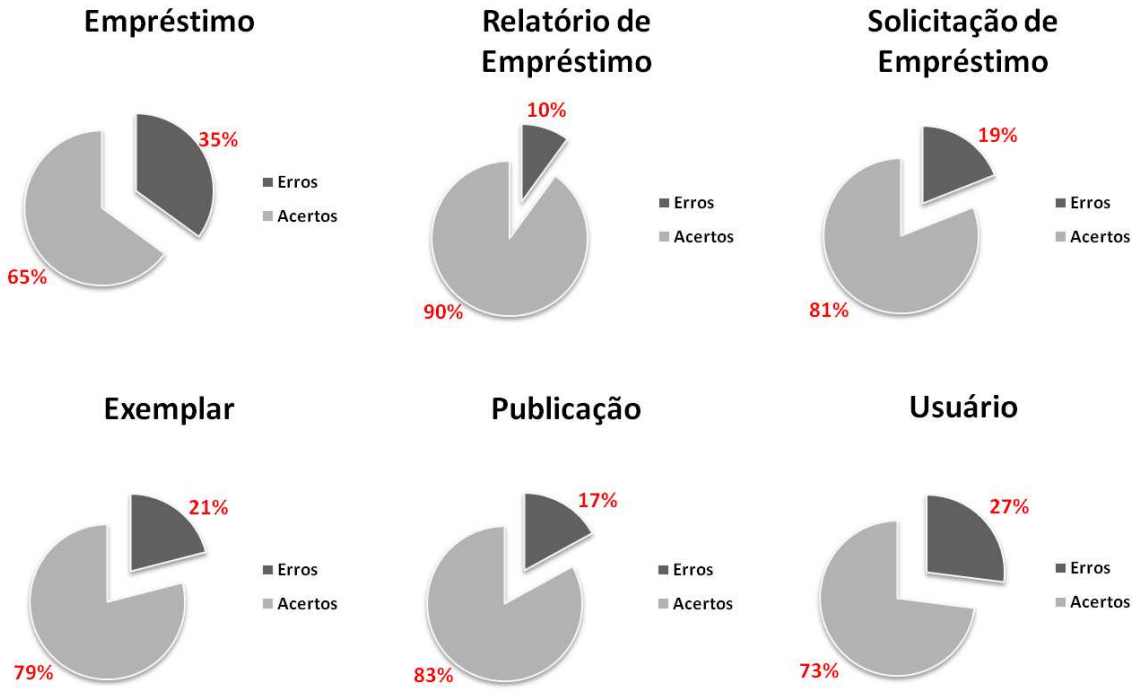

Figura 6.6. "Acurácia" para o JBook na localização parcial do código-fonte.

"Solicitação de empréstimo" é uma característica com baixos valores para precisão. A investigação deste caso particular permitiu identificar dois fatores que impactaram em resultados fracos. Primeiro, poucos casos de teste têm requisitos associados a

esta característica, ou seja, baixa cobertura de testes. Por isso, a maioria do código da característica nunca é executado, o que leva a baixos valores de cobertura. Em adição, "Solicitação de empréstimo" é uma característica muito específica implementada por algumas poucas linhas de código. Assim, observou-se que não se consegue bons níveis de precisão para este tipo de característica (específicas e pequenas) com a localização de características baseada em testes.

O gráfico de acurácia (Figura 6.6) mostra a porcentagem de acertos e erros na anotação para cada uma das características analisadas no sistema JBook sob a estratégia de localização parcial do código-fonte. As porcentagens de erros são baixas, variando de 10% (característica 'Relatório de Empréstimo') a 35% (característica 'Empréstimo') de erro na localização do código que implementa cada característica. No caso da característica Empréstimo, acredita-se que o aumento da quantidade de erro foi proporcional ao crescimento da característica, em outras palavras, o fato da característica "Empréstimo" ser uma das maiores do sistema dificultou ainda mais a máxima cobertura de testes. Além disso, os erros podem ter sido agravados pelo descarte de linhas marcadas corretamente, quando da interseção entre os conjuntos de código executados pelas diferentes suítes de teste. Ainda assim, os 85% de precisão obtidos para esta característica, de fato permitem que o esforço para a localização de uma boa semente seja reduzido.

6.6.2 WebStore

As Figuras 6.7 e 6.8 mostram os resultados para as métricas de precisão e acurácia para o sistema WebStore referentes à localização parcial do código fonte da característica. Quando da aplicação da estratégia de localização parcial, uma baixa quantidade de testes foi identificada para três das seis características a serem analisadas. Isto, portanto, impossibilitou a interseção entre os conjuntos de código executados pelos testes destas características. Assim, serão apresentados os dados coletados para as características cuja análise foi viabilizada: "Fechamento de Pedido", "Exibição" e "Gerenciamento de Conteúdo".

Mais uma vez, baixos valores de precisão para as duas características de menor granularidade: "Fechamento de Pedido" e "Exibição". Apesar de ser uma característica pequena, "Exibição" é uma característica que apresenta um maior espalhamento em relação à "Fechamento de Pedido", o que refletiu na grande diferença entre os valores de precisão obtidos em cada uma. Por ser uma característica maior, obteve-se uma maior precisão na localização de uma semente de "Gerenciamento de Conteúdo".

Como semente de uma característica é um conceito bastante subjetivo, e qualquer parte do código pode ser considerado uma semente, optou-se por não limitar o

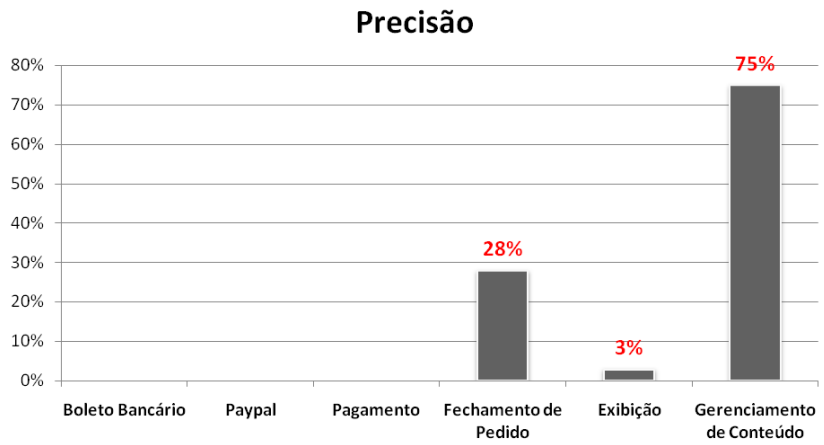

Figura 6.7. "Precisão" para o WebStore na localização parcial do código-fonte.

oráculo a determinadas partes do código da característica. Sendo assim, as análises de precisão para esta estratégia são úteis para a detecção de padrões de comportamento na localização de características baseada em teste.

Figura 6.8. "Acurácia" para o WebStore na localização parcial do código-fonte.

O gráfico de acurácia (Figura 6.8) mostra a porcentagem de acertos e erros na anotação para cada uma das características analisadas no sistema WebStore sob a estratégia de localização parcial do código-fonte. As porcentagens de erros são baixas, variando de 8% a no máximo 28% das linhas da característica. Assim como no sistema JBook, os erros podem ter sido agravados pelo descarte de linhas marcadas corretamente e a grande quantidade de acertos facilita a identificação de uma semente para às características.

6.7 Localização Completa do Código-fonte

Esta seção apresenta os resultado da localização de características utilizando a estratégia de localização completa do código-fonte. Igualmente à seção anterior, primeiro são apresentados os resultados colhidos para o sistema JBook (Seção 6.7.1) e em seguida para o sistema WebStore (Seção 6.7.2).

6.7.1 JBook

As Figuras 6.9, 6.10 e 6.11 mostram os resultados para as métricas de precisão, cobertura e acurácia para o sistema JBook referentes à localização completa do código fonte da característica. Sempre que há a execução de um caso de teste de integração partes do código dos módulos que estão sendo integrados são executadas. Portanto, pode acontecer deste código não esteja diretamente relacionado à característica a qual o teste foi mapeado o que leva a muitos falsos positivos. Assim, espera-se que os valores de precisão obtidos nos resultados sejam baixos comparados à estratégia de localização parcial (Seção 6.6). Exatamente por este motivo, os valores mais altos de precisão ocorreram para as maiores características do sistema: "Empréstimo", "Publicação" e "Usuário". Nestas características a probabilidade de falsos positivos é menor.

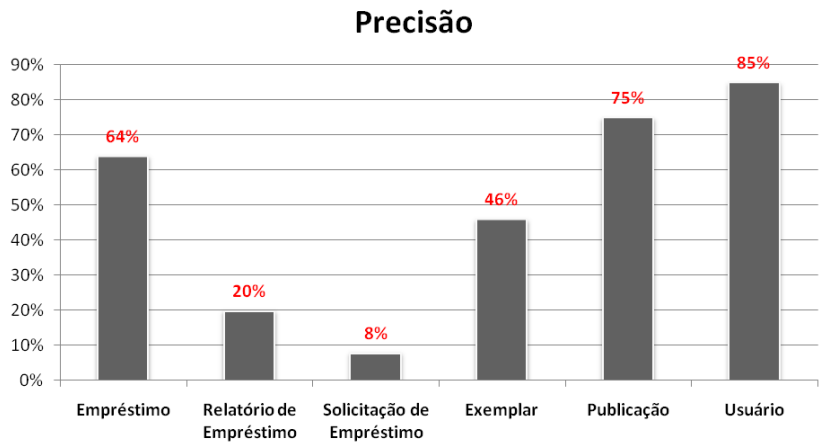

Figura 6.9. "Precisão" para o JBook na localização completa do código-fonte.

Os valores mais baixos de precisão (Figura 6.9) foram para as características "Exemplar", "Relatório de Empréstimo" e "Solicitação de Empréstimo". A característica "Exemplar" possui grande proximidade semântica com "Publicação". Por outro lado, as características "Relatório de Empréstimo" e "Solicitação de Empréstimo" possuem grande proximidade com "Empréstimo". Tais semelhanças entre características

justificam a grande quantidade de falsos positivos. De fato, estes valores refletem o entrelaçamento de código entre essas características devido sua proximidade.

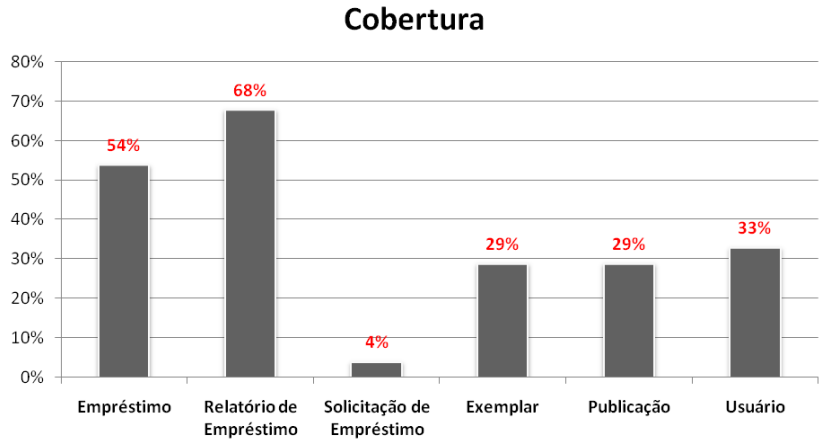

Figura 6.10. "Cobertura" para o JBook na localização completa do código-fonte.

Os resultados obtidos com o sistema JBook na análise da cobertura (Figura 6.10) não foram satisfatórios. No caso da característica "Solicitação de Empréstimo" a causa predominante foi a falta de casos de teste. Para as características "Exemplar", "Publicação" e "Usuário" acredita-se que apesar da grande quantidade de testes existentes, a cobertura deles não fora suficiente para a localização completa das características. Além disso a forma como a arquitetura do sistema foi concebida, baseada em um *framework* próprio da empresa que o desenvolveu, foi um agravante dos baixos valores de cobertura. Este fator, no entanto, não interferiu tanto nos casos das características "Empréstimo" e "Relatório de Empréstimo", onde havia maior quantidade de testes disponível.

Os erros na localização de características baseada em testes se dão, basicamente, pela ausência de marcação em linhas de interesse (falsos negativos) e pela marcação de código desnecessário para a característica (falsos positivos). As causas principais para estes erros são a ausência de testes para a execução do código de interesse e o entrelaçamento de código relacionado a diferentes características, respectivamente.

O gráfico de acurácia (Figura 6.11) mostra a porcentagem de acertos e erros na anotação para cada uma das características analisadas no sistema JBook sob a estratégia de localização completa do código-fonte. A porcentagem de acertos ficou em torno de 72% a 90%.

Para a extração de uma linha de produtos após a anotação feita pelos testes o desenvolvedor teria o trabalho adicional de juiz. Ele deveria julgar cada linha como marcada corretamente (TP) ou não (FP), além de julgar se as linhas não marcadas

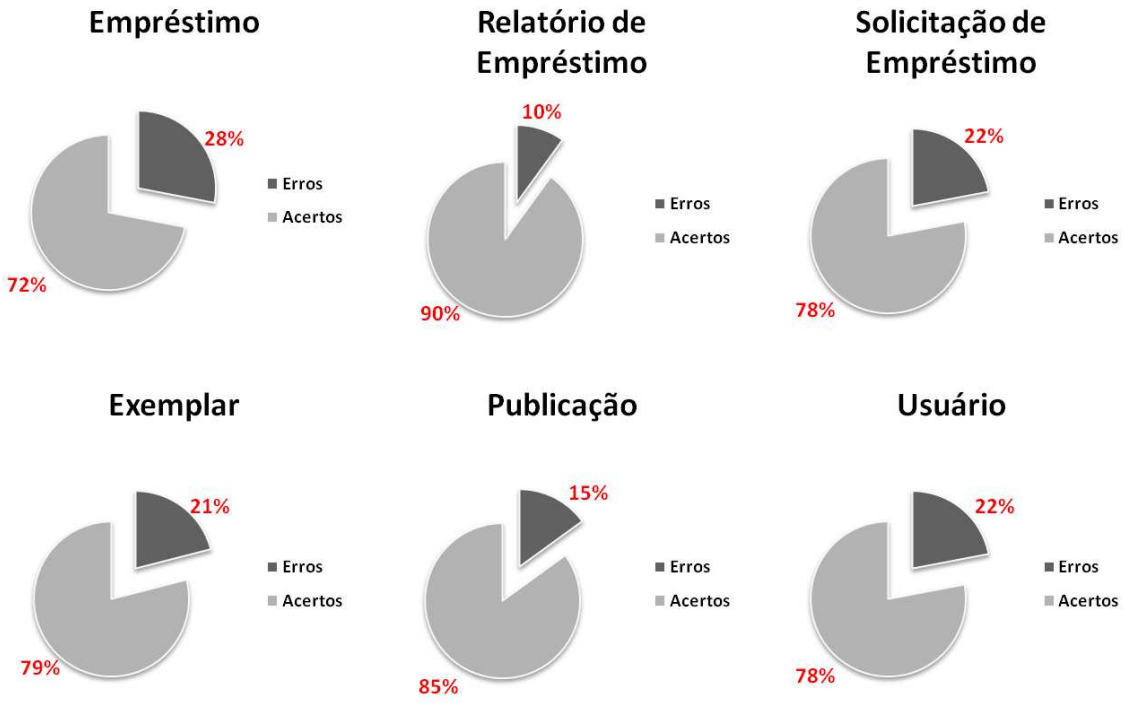

Figura 6.11. "Acurácia" para o JBook na localização completa do código-fonte.

Tabela 6.3. Acertos e erros na anotação das linhas de código no JBook.

Característica	TP	FP	FN	TN
Empréstimo	19.58%	11.17%	16.53%	52.72%
Relatório de Empréstimo	2.31%	9.14%	1.11%	87.44%
Solicitação de Empréstimo	0.65%	7.39%	14.13%	77.84%
Exemplar	6.00%	6.93%	14.40%	72.67%
Publicação	5.45%	1.85%	13.20%	79.50%
Usuário	9.79%	1.66%	19.85%	68.70%

não foram marcadas por não pertencerem às características (TN) ou por falta de testes para a sua marcação (FN). A Tabela 6.3 apresenta o restultado deste julgamento para o sistema JBook. As porcentagens são a relação entre a quantidade de linhas classificadas (como TP, FP, FN ou TN) e o total de linhas do sistema. A quantidade de linhas corretamente descartadas (TN) pela localização baseada em testes é significativa.

Em caso de não haverem falsos negativos no caso do JBook, 80% de verdadeiros negativos (TN) equivalem, aproximadamente, a ter que analisar de 200 linhas de código ao invés das 1000 que compõem o sistema. Se transpusermos estes valores a um sistema de grande porte que possui, geralmente, acima de 1 milhão de linhas de código, a redução de código a ser analisado manualmente é considerável. A existência de falsos

negativos se dá pela ausência de testes, portanto, é importante que haja quantidade suficiente de testes para eliminar os falsos negativos e reduzir consideravelmente o espaço de busca para as linhas anotadas pelos testes.

6.7.2 WebStore

As Figuras 6.12, 6.13 e 6.14 mostram os resultados para as métricas de precisão, cobertura e acurácia para o sistema WebStore referentes à localização completa do código fonte da característica. Os maiores valores de precisão (Figura 6.12), "Pagamento" e "Gerenciamento de conteúdo", são exatamente as maiores características analisadas. Embora os testes executem muito código que não pertencem a estas características, seus tamanhos contribuem para aumentar os valores de precisão.

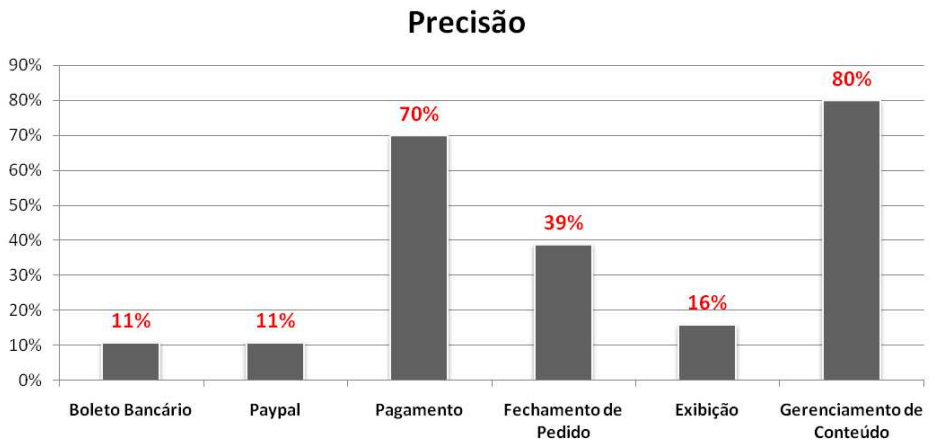

Figura 6.12. "Precisão" para o WebStore na localização completa do código-fonte.

Olhando para os valores de cobertura (Figura 6.13), todas as características atingiram mais que 60%. Os piores valores de cobertura foram observados nos casos de "Pagamento" e "Gerenciamento de conteúdo". Analisando estas duas características, percebeu-se que os baixos valores de cobertura são principalmente devido ao tamanho delas. Neste caso, características menores são mais facilmente encontradas pela abordagem baseada em testes visto que exigem uma quantidade menor de testes para serem totalmente anotadas.

Problema similar foi observado em outras características devido à pequena quantidade casos de teste para localizar o código da característica de interesse. Engenheiros de teste estabelecem prioridades para os requisitos que merecem maior atenção

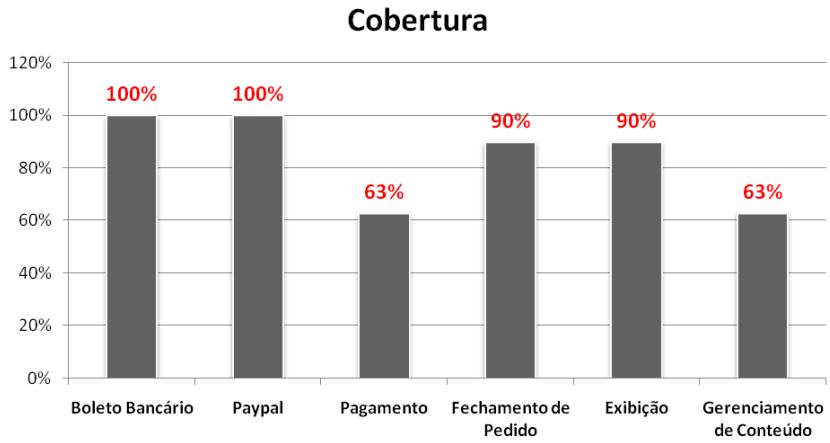

Figura 6.13. "Cobertura" para o WebStore na localização completa do código-fonte.

[Tassey, 2002, Binder, 1999]. De fato, quanto maior a quantidade de casos de teste, melhores serão os resultados de cobertura com o método proposto.

As características "Pagamento" e "Gerenciamento de Conteúdo" foram as que tiveram os maiores valores de precisão e contrastantemente, os menores valores de cobertura. Isso pode acontecer pela não cobertura de todos o código da característica pelos testes disponíveis. Visto que ambas correspondem exatamente às maiores características analisadas no sistema WebStore.

Tabela 6.4. Acertos e erros na anotação das linhas de código no WebStore.

Característica	TP	FP	FN	TN
Boleto Bancário	0.41%	3.26%	0.00%	96.34%
Paypal	0.41%	3.26%	0.00%	96.34%
Pagamento	8.95%	3.87%	5.19%	81.99%
Pechamento de Pedido	6.10%	9.36%	0.71%	83.83%
Exibição	1.93%	9.87%	0.20%	88.00%
Gerenciamento de Conteúdo	19.74%	4.88%	11.60%	63.78%

O gráfico de acurácia (Figura 6.14) mostra a porcentagem de acertos e erros na anotação para cada uma das características analisadas no sistema WebStore sob a estratégia de localização completa do código-fonte. Este foi o caso em que se percebeu as maiores valores de acertos, em que os valores obtidos estão na faixa de 84% a 97%. A Tabela 6.4 apresenta o restultado do julgamento da correção da anotação das linhas de código para o sistema WebStore. Assim como no sistema JBook, os valores de verdadeiros negativos (TN) são bastante altos e o de falsos negativos (FN) bem mais

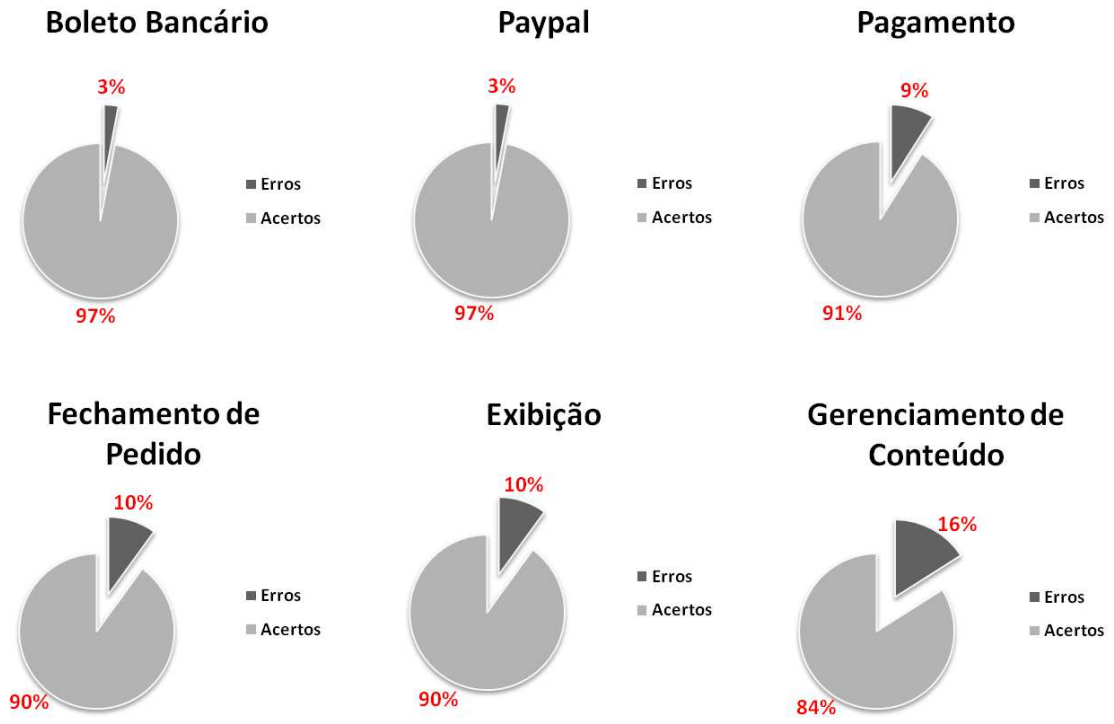

Figura 6.14. "Acurácia" para o WebStore na localização completa do código-fonte.

baixos. Com uma quantidade adicional de testes seria possível reduzir o espaço de busca em, no mínimo, 52.72% do código fonte no caso da característica "Empréstimo" do JBook e de 63.78% para a característica "Gerenciamento de Conteúdo" no caso do WebStore.

6.8 Avaliação de Escalabilidade: Estudo de Caso WEKA

O procedimento utilizado para a coleta dos dados neste estudo de caso foi semelhante ao utilizado nos estudos de caso anteriores. Inicialmente, solicitou-se a um pesquisador com conhecimento do sistema WEKA a seleção das características a serem analisadas no estudo de caso. Em seguida, solicitou-se ao mesmo pesquisador a construção do arquivo *.cm* a ser usado como "oráculo". Por outro lado, construiu-se o mapeamento dos testes para as características de tal forma que a ferramenta TaBuLeTa pudesse ser utilizada.

Tanto o "oráculo", quanto o mapeamento dos testes para características foi feito com a TaBuLeTa. Além disto, a localização da implementação das características através de ambas as estratégias e o cálculo das métricas também foram feitos com o auxílio da ferramenta. O cálculo das métricas "precisão" e "cobertura" não foi feito em relação às linhas de código, mas em relação aos itens de $< elemento{-}ponderado >$ dos arquivos *.cm* (Figura 5.2). Mais detalhes sobre o modelo de interesse foi apresentado na Seção 5.2. Infelizmente, o modelo de interesse inviabiliza o cálculo da métrica "acurácia", vez que não é possível contabilizar os valores de verdadeiros negativos (TN).

Mantendo a organização das seções anteriores, esta seção apresenta, primeiro os resultados obtidos com a localização parcial e em seguida os da localização completa da implementação de características utilizando a ferramenta TaBuLeTa. A Figura 6.15 exibe um gráfico de precisão com o resultado da estratégia de localização parcial, da mesma forma que as Figuras 6.16 e 6.17 exibem gráficos de precisão e cobertura com os resultados da localização completa.

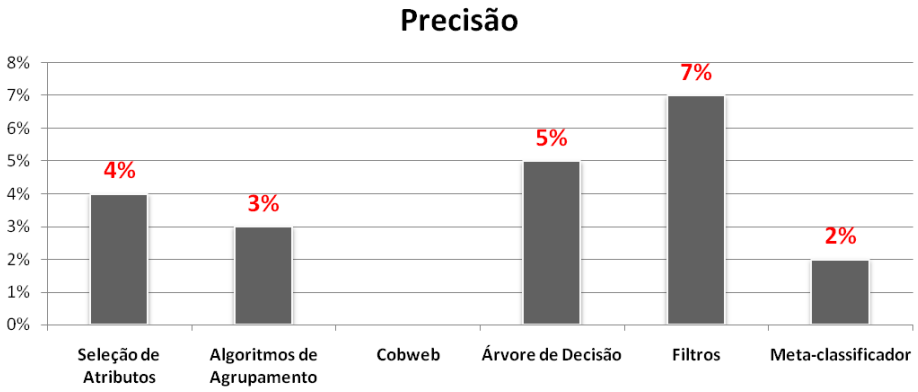

Figura 6.15. "Precisão" para o WEKA na localização parcial do código-fonte com a TaBuLeTa.

Na aplicação da estratégia de localização parcial aconteceu um caso semelhante ao enfrentado no sistema WebStore quanto à falta de casos de teste para executar a interseção entre conjuntos. O problema foi encontrado para a característica "Cobweb" e, portanto, a Figura 6.15 não apresenta valores para esta característica.

Os valores obtidos para a métrica *precisão* em ambas as estratégias (Figuras 6.15 e 6.16) foram ruins. Além das causas identificadas nos estudos de caso manuais, a adição de "elementos-ponderados" para todos os atributos das classes que foram cobertas de alguma forma pelos testes no modelo de interesse causou imenso número de falsos positivos. Isto foi feito porquê a EclEmma não provê informações de cobertura sobre os atributos, somente para métodos e classes. Então, optou-se por adicionar todos,

pois os falsos positivos impactam menos na cobertura. Embora seja necessário buscar alternativas para melhorar a construção do modelo de interesse a fim de melhorar os índices de precisão, acredita-se que atributos não adicionam um grande volume de trabalho extra e são facilmente identificados como pertencentes ou não a uma dada característica.

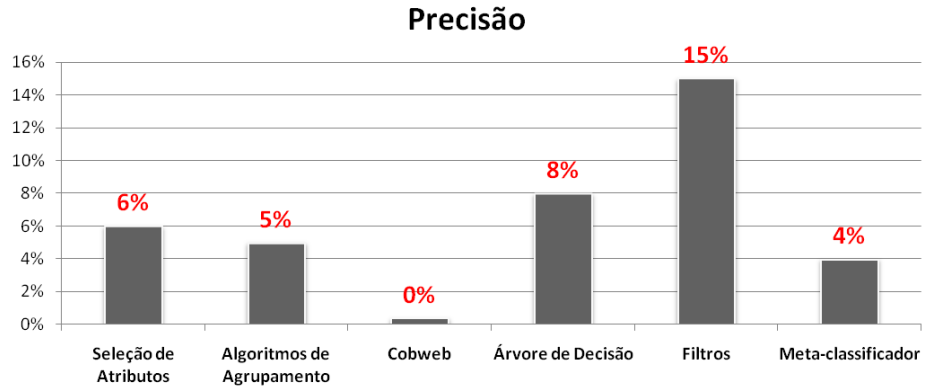

Figura 6.16. "Precisão" para o WEKA na localização completa do código-fonte com a TaBuLeTa.

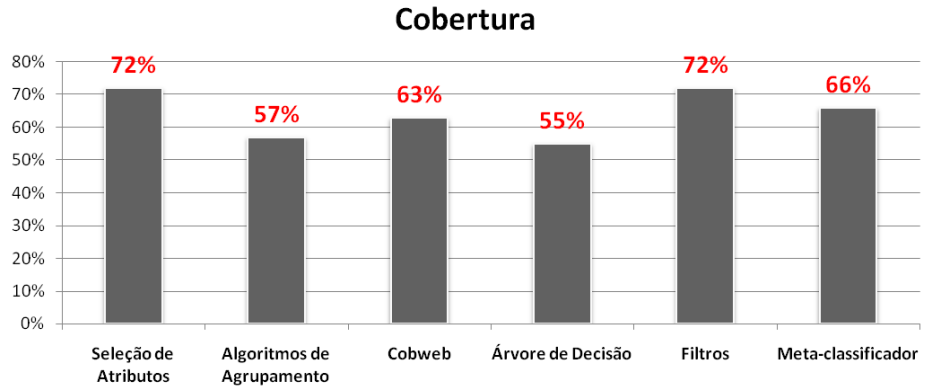

Figura 6.17. "Cobertura" para o WEKA na localização completa do código-fonte com a TaBuLeTa.

Os resultados obtidos para cobertura para a estratégia de localização completa (Figura 6.17) foram melhores que os de precisão. Em média, os valores de cobertura ficaram em 64% nesta estratégia. Estes valores representam quantidade significativa da implementação da característica de interesse.

Na seção seguinte é feita uma discussão sobre os resultados obtidos nesta avaliação. A discussão busca respostas para as questões de pesquisa enumeradas no início deste capítulo.

6.9 Discussão das Questões de Pesquisa

Após a apresentação dos resultados é possível responder às questões de pesquisa. Cada uma das seções seguinte trata de uma delas. A Seção 6.9.1 discute a localização de uma semente de uma característica e a Seção 6.9.2 a localização do código da característica por completo.

6.9.1 Localização de Semente

A resposta à questão Q_0 que questiona se os testes de integração são efetivos para a localização de uma semente do código-fonte que implementa uma característica de interesse é sim, de forma parcial, através da utilização da estratégia apresentada na Seção 4.5.1. Em outras palavras, os testes de integração podem localizar uma semente caso o conjunto de intersecção entre todo o código executado pelos testes de uma característica seja considerado para esta atividade.

Foram alcançados taxas de precisão acima de 55% para maioria das características analisadas com a estratégia de localização de sementes de características. Assim, em ambos os casos JBook e WebStore fica claro que a localização de uma "semente" para a característica de interesse é viável. Observa-se que esta estratégia apresenta um caminho confiável para o início do processo de uma extração de Linha de Produtos de Software (LPS). Em alguns casos a precisão tenha ficado abaixo do esperado acredita-se que é possível obter melhores resultados em sistemas com menor espalhamento de código.

6.9.2 Localização do Código da Característica

A resposta para a questão Q_1 que questiona se os testes de integração são capazes de localizar o código-fonte que implementa uma característica de interesse também é sim, de forma parcial. É possível extrair características utilizando testes sob algumas condições, tal como, alta cobertura dos testes. Adicionalmente, é requerido algum esforço adicional para finalizar a extração da LPS. Os resultados mostraram que a cobertura de teste tem grande poder de localização de características com valores de cobertura acima de 60% na maioria das características analisadas, sendo que em alguns casos obteve-se 100% de cobertura.

Quando o objetivo desta estratégia é identificar o código completo da característica de interesse, altos valores de cobertura significam que a estratégia atingiu o seu objetivo. Ou seja, a maioria das linhas de código que pertenciam às características foram anotadas pelos testes. De fato, esta estratégia deveria ser utilizada para a redução

do espaço de busca pelo código que implementa a característica. Os resultados mostraram que pelo menos 70% do código da aplicação foi corretamente classificado pelos testes como pertencente ou não à característica a qual foram mapeados. Portanto, o desenvolvedor terá que buscar o restante do código da característica em menos de 30% do código da aplicação.

O uso da ferramenta TaBuLeTa reduziu consideravelmente o tempo de execução do estudo de caso e conseguiu resultados semelhantes aos obtidos manualmente. A redução fica ainda mais clara com a comparação do tamanho dos sistemas, o sistema WEKA é em torno de 144 vezes maior que os sistemas JBook e WebStore. Vale ressaltar que apesar do caráter de protótipo, a ferramenta apresenta uma boa automação do método de localização de características aqui proposto.

6.10 Limitações e Ameaças à Validade

A validade de um estudo denota a confiabilidade dos resultados e para quais contextos os resultados são considerados verdadeiros e não tendenciosos através do ponto de vista subjetivo dos pesquisadores [Wohlin et al., 2012]. Nesta seção são tratadas as ameaças às validade de acordo com sua importância para experimentos aplicados, ou seja, validade interna, validade externa, validade de construção e validade de conclusão [Wohlin et al., 2012].

6.10.1 Validade Interna

A validade interna define se o relacionamento observado entre o tratamento e o resultado é causal e não é o resultado da influência de outro fator que não é controlado ou mesmo não foi medido [Wohlin et al., 2012]. Assim, três fatores principais podem ser apontados como ameaças à validade interna *(i)* o tamanho e a natureza dos sistemas utilizados nos estudos de caso manuais, *(ii)* a cobertura dos testes de integração existentes e *(iii)* o espalhamento e entrelaçamento de código nos sistemas.

Foram utilizados nos estudos de caso conduzidos manualmente dois sistemas de informação (SI) desenvolvidos para a plataforma Web, que são muito comuns, mas pequenos. Um grande sistema de informação nada mais é que um sistema com muito mais funções. Ou seja, o tamanho do sistema de informação não deve fugir ao comportamento de um sistema de pequeno porte. Quanto a natureza, talvez nossas conclusões so sejam validas para SI e não para qualquer outro tipo. As vezes um jogo, uma aplicação de tempo real possuem comportamentos diferentes e que poderiam nos ajudar ou beneficiar. Quanto ao tamanho dos sistemas, utilizar sistemas de grande porte nos

estudos de caso manuais seria um impedimento para a construção dos "oráculos" utilizados na comparação dos resultados obtidos com o método. Quanto à natureza dos sistemas, infelizmente neste tipo de estudo existe uma dificuldade inerente da identificação de sistemas de diferentes plataformas com código aberto e que pudessem ser utilizados. Portanto, optou-se por utilizar sistemas já de conhecimento dos pesquisadores. Além disso, no estudo de caso automatizado utilizou-se um sistema de grande porte desenvolvido para uma plataforma diferente a fim de minimizar às ameaças à validade interna.

Diferentes níveis de cobertura de testes foram utilizadas a fim de que a influência desta nos resultados fosse minimizada. Desta forma, ao final do estudo generalizações puderam ser feitas em respeito ao uso de testes com maior grau de confiança. Por exemplo, o sistema JBook possui em torno de 60 a 70 por cento de cobertura de linhas de código. Em contraste, o WebStore e o WEKA possuem em torno de 50 e 60 por cento de cobertura, respectivamente. A abordagem baseada em testes é dependente da quantidade de testes existente no sistema. A atividade de teste é cara, mas aos poucos tem ganho espaço na comunidade industrial. É sabido ainda que cobrir 100% dos casos e teste é uma dificuldade inerente ao teste de software, mas que a comunidade industrial tem se esforçado para cobrir o máximo possível em busca de produtos de qualidade.

Sobre o entrelaçamento e o espalhamento de código, tentou-se utilizar-se de sistemas com diferentes graus de modularização entre os módulos independentes dos sistemas. No entanto, nenhuma medida foi tomada em relação a isto além da constatação de sua existência durante a construção dos "oráculos" e não se pode afirmar o quanto um sistema alvo dos estudos de caso é mais modularizado que os demais.

6.10.2 Validade Externa

A validade externa define as condições que limitam a habilidade de generalizar os resultados de um experimento na prática industrial [Wohlin et al., 2012]. Alguns pontos foram identificados como ameaças à validade externa, são eles *(i)* o tamanho da amostra, *(ii)* as plataformas utilizadas no estudo, e *(iii)* configuração usual de extração de LPS.

O tamanho da amostra certamente é uma ameaça à validade externa deste estudo. Considerando que a cada dia estão sendo construídos novos sistemas com novas tecnologias para diferentes plataformas, a amostra aqui utilizada não é significativa. Porém, dada esta mesma dinamicidade do contexto, torna-se inviável atacar uma amostra que seja de fato significativa, dadas as limitações de tempo e recursos.

Ao tempo em que se fala de validade externa uma pergunta mais que pertinente

é quais contextos as conclusões apresentadas neste estudo são verdadeiras? Mesmo com a pequena amostra utilizada no estudo, acredita-se que ao menos em sistemas desenvolvidos com linguagem de programação Java, sistemas de informação para as plataformas Web e de estações de trabalho os resultados tendem a ser replicáveis.

Além disso, a configuração de extração de LPS neste estudo não é a usual, pois foram utilizados apenas produtos únicos. Geralmente, a extração acontece quando se possui um conjunto de produtos construídos à base de duplicação de código e sua manutenção e evolução já estão comprometidas devido o crescimento cada vez maior dos custos. No entanto, quando da replicação de código para construção de novos produtos, as suítes de teste do código replicado devem estar disponíveis para todos os diferentes produtos em que o código foi duplicado. Assim, o uso dos testes podem representar um bom caminho para a identificação dos componentes compartilhados, bem como das características variáveis da linha de produtos de software, seja em um produto único ou múltiplos produtos construídos à partir da replicação de código.

Para os casos em que exista mais de um produto e que o interesse seja eliminar o código replicado nos diversos produtos, acredita-se que seja possível adaptar este método para abordar os diferentes produtos. Como ocorre a duplicação de código-fonte na construção de múltiplos produtos é possível a utilização de casos de teste em diferentes produtos para executar a localização de características no produto com código fonte replicado. Além disso, os novos casos de teste que são adicionados ao produto criado a partir da replicação ajudam na diferenciação entre o que foi replicado e a variabilidade adicionada.

6.10.3 Validade de Construção

Durante a avaliação da validade de construção os aspectos relevantes ao projeto do experimento e os fatores humanos devem ser considerados [Wohlin et al., 2012]. Isto é, o comportamento humano pode ser incorreto, tanto do lado do experimentador quanto dos participantes. O trabalho manual para construir os "oráculos" é de certo mais uma das ameaças à validade, especificamente, à validade de construção. Apoiou-se em indivíduos com experiência, os quais deviam entender a maior parte os sistemas para a construção dos "oráculos". Mesmo tendo sido posteriormente revisado por um grupo, este trabalho pode conter erros. Além disso, fazer as devidas comparações e cálculo de métricas foi feito unicamente pelo pesquisador e também pode conter erros. Para amenizá-los, o pesquisador tentou alternar tarefas durante longos tempos de trabalho.

6.10.4 Validade de Conclusão

A validade de conclusão é relacionada à habilidade de chegar a uma conclusão correta a respeito dos relacionamentos entre o tratamento e o resultado do experimento [Wohlin et al., 2012]. Aqui a fragilidade encontrada é exatamente a inexistência do tratamento estatístico aos dados coletados. Não houve análise estatística. Isto aconteceu dado a natureza da avaliação escolhida: o estudo de caso. Além disso, o tamanho não significativo da amostra faria com que este tipo de análise fosse considerada insuficiente. Ainda assim, acredita-se que as métricas utilizadas são confiáveis e apresentam uma boa representação do que se está investigando. Este tipo de estudo é largamente aceito e utilizado pela comunidade acadêmica de Engenharia de Software.

6.11 Resumo

Este capítulo apresentou uma avaliação do método de localização de características aqui proposto na forma de estudos de caso com sistemas reais de domínios diferentes. Dois sistemas foram desenvolvidos para o ambiente Web (JBook, WebStore) e um para manipulação de dados através de algoritmos de mineração de dados (WEKA).

Os resultados mostraram que a cobertura de teste tem grande poder de localização de características com valores de cobertura acima de 60% na maioria das características analisadas, sendo que em alguns casos obteve-se 100% de cobertura. Além disso, alguns pontos importantes foram identificados com o estudo experimental apresentado neste capítulo:

Cobertura de testes: uma alta cobertura de testes é benefica ao uso da abordagem proposta, pois quanto maior ela for, meno a quantidade de falsos negativos e consequentemente maior a redução do espaço de busca;

Tamanho das características: características de maior granularidade são localizadas com maior precisão, no entanto, exigem maior quantidade de testes para obter-se altos índices de cobertura. Por outro lado, características de menor granularidade são mais facilmente localizadas;

Entrelaçamento de características: este fenômeno acontece pela falta de modularidade do sistema e favorece a redução dos índices de precisão no uso da técnica de localização de características baseada em testes;

Semelhança entre características: a proximidade semântica de características favorece o fenômeno do entrelaçamento de características e consequentemente a baixar os índices de precisão.

Em suma, os índices de cobertura estão associados à quantidade de casos de testes disponíveis, equanto que os índices de precisão à modularidade da implementação de cada característica.

O capítulo seguinte apresenta as conclusões obtidas com o presente estudo, bem como, algumas indicações para trabalhos futuros.

Capítulo 7

Conclusão

Este trabalho propôs um método para a extração de Linha de Produtos de Software (LPS) que aborda desde os estágios iniciais na engenharia de domínio, quando o modelo de características é criado, até a localização do código-fonte que implementa cada característica variável na engenharia de aplicação. Especificamente, o método proposto define novas estratégias para a localização e anotação do código fonte que implementa cada característica variável da LPS.

Além disso, o trabalho apresentou um estudo exploratório sobre o uso de testes como apoio a extração de LPS a partir de sistemas de software desenvolvidos inicialmente como produtos únicos. Foram utilizados três sistemas como estudos de caso, chamados Jbook[1], WebStore [Ferreira et al., 2011] e WEKA [Hall et al., 2009]. Apresentou-se os resultados relacionados a localização de características para cada um dos estudos de caso em termos de *precisão*, *cobertura* e *acurácia*.

O restante do capítulo está organizado da seguinte maneira: a Seção 7.1 apresenta as considerações finais do trabalho e a Seção 7.2 apresenta direções de trabalhos futuros.

7.1 Considerações Finais

Os resultados apresentados neste estudo foram interessantes e a técnica baseada em testes pode ser considerada no contexto de extração de LPS. Adicionalmente, baseado nas questões de pesquisa respondidas (Seção 6.9), conclui-se que a técnica baseada em testes pode ajudar na extração de LPS através da indicação de *(i)* uma boa semente ou *(ii)* um superconjunto de código que implementa uma característica. Além disso, muitos dos sistemas de software desenvolvidos hoje em dia possuem suítes de teste

[1]Disponível em: http://sourceforge.net/projects/jbookweb/

e este fato reduz os custos da abordagem proposta. Com os resultados apresentados através dos estudos de caso manuais, ficou claro que a abordagem não é livre de falhas e, portanto, também apresenta limitações (Seção 6.10).

Adicionalmente à proposta do método, desenvolveu-se uma Ferramenta de Localização de Características Baseado em Testes (TaBuLeTa) (Capítulo 5) para a automação parcial do método. Com o uso da ferramenta foi possível realizar um novo estudo de caso a fim de avaliar a escalabilidade do método utilizando-se para isto um sistema de grande porte, chamado WEKA. Apesar dos valores de precisão abaixo do esperado, considerou-se o estudo de caso bem sucedido. A partir dos resultados ficou claro que o método proposto é escalável, no entanto, identificou-se a necessidade de melhorias futuras na ferramenta desenvolvida.

7.2 Trabalhos Futuros

Este trabalho pode ser visto como uma estudo inicial em relação ao uso dos testes na localização de características para apoiar à extração de LPS. Sendo assim, algumas direções interessantes ficam por ser atacadas a fim de melhorar o que foi iniciado aqui e novas rotas podem ser exploradas no futuro. Os seguintes pontos podem ser investigados em trabalhos futuros:

Esforço adicional de extração de LPS: Como nesta dissertação não foi conduzido nenhum estudo com a extração real de uma LPS, uma possível direção para trabalho futuro seria conduzir novos experimentos utilizando a técnica proposta de forma a analisar o esforço adicional necessário na tarefa de extração de LPS;

Tipos de requisitos: O método só foi estudado com requisitos funcionais: requisitos não-funcionais de caráter transversais como "tratamento de exceções" ou "registro de atividades"[2] podem apresentar comportamento diferente dos analisados. Portanto, um estudo experimental inserindo requisitos desta natureza configuram trabalho futuro;

Paradigmas de programação: Outros paradigmas de programação podem ser analisados em futuros desdobramentos deste trabalho. Por exemplo, orientação a aspectos é uma técnica que vem sendo investigada para extração de LPS [Alves et al., 2005]. Com um grau diferente de modularização das características é possível investigar se o entrelaçamento entre as características é de fato um empecilho para a localização de características baseada em testes;

[2]Do inglês: logging.

Além de trabalhos de investigação científica sobram também trabalhos de natureza prática, como incrementos na TaBuLeTa. São alguns pontos que podem merecer alguma atenção futura:

- Permitir a execução de testes diretamente da visão de mapeamento, eliminando a geração da suíte de testes. A geração da suíte de testes foi um passo adicionado devido a dificuldes encontradas com a tecnologia Eclipse por falta de experiência e tempo limitado;
- Permitir o cálculo do espalhamento e entrelaçamento entre o código coberto pelos testes;
- Melhorar a construção do modelo de interesse baseado na cobertura dos testes a fim de reduzir a quantidade de falsos positivos apresentados em estudos automatizados.

Referências Bibliográficas

[Abran et al., 2004] Abran, A.; Bourque, P.; Dupuis, R. & Moore, J. W., editores (2004). *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA.

[Adams et al., 2010] Adams, B.; Jiang, Z. M. & Hassan, A. E. (2010). Identifying crosscutting concerns using historical code changes. Em *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pp. 305--314, New York, NY, USA. ACM.

[Aho et al., 1985] Aho, A.; Sethi, R. & Ullman, J. (1985). *Compilers. Principles Techniques and Tools.* Addison-Wesley Publishing Company, Reading, MA.

[Alves et al., 2005] Alves, V.; Jr., P. M.; Cole, L.; Borba, P. & Ramalho, G. (2005). Extracting and evolving mobile games product lines. Em *Proceedings of 9th Software Product Line Conference*, pp. 70--81.

[Antoniol & Guéhéneuc, 2005] Antoniol, G. & Guéhéneuc, Y.-G. (2005). Feature identification: A novel approach and a case study. Em *Proceedings of the 21st IEEE International Conference on Software Maintenance*, pp. 357--366, Washington, DC, USA. IEEE Computer Society.

[Apel & Beyer, 2011] Apel, S. & Beyer, D. (2011). Feature cohesion in software product lines: an exploratory study. Em *Proceeding of the 33rd International Conference on Software Engineering*, ICSE'11, pp. 421--430, New York, NY, USA. ACM.

[Bass et al., 1998] Bass, L.; Clements, P. & Kazman, R. (1998). *Software Architeture in Practice.* Addison-Wesley.

[Batory, 2005] Batory, D. (2005). Feature models, grammars, and propositional formulas. Em Obbink, H. & Pohl, K., editores, *Software Product Lines*, volume 3714 of *Lecture Notes in Computer Science*, pp. 7–20. Springer Berlin Heidelberg.

[Bergmans & Aksit, 2001] Bergmans, L. & Aksit, M. (2001). Composing crosscutting concerns using composition filters. *Commun. ACM*, 44(10):51--57.

[Binder, 1999] Binder, R. (1999). *Testing object-oriented Systems – Models, Patterns and Tools*. Addison Wesley Reading.

[Clements & Northrop, 2001] Clements, P. & Northrop, L. M. (2001). *Software Product Lines: Practices and Patterns*. Addison-Wesley.

[Couto et al., 2011] Couto, M. V.; Valente, M. T. & Figueiredo, E. (2011). Extracting software product lines: A case study using conditional compilation. Em *Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering*, CSMR'11, pp. 191--200, Washington, DC, USA. IEEE Computer Society.

[De Oliveira et al., 2010] De Oliveira, V. B.; Garcia, R. & Valente, M. T. (2010). CIDE+: A semi-automatic approach for extracting software product lines. Em *Proceedings of Brazilian Conference on Software: Theory and Practice – Tools Session*, CBSoft-Tools'10, pp. 73--78, Bahia-Brazil.

[Deerwester et al., 1990] Deerwester, S.; Dumais, S. T.; Furnas, G. W.; Landauer, T. K. & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391--407.

[Duvall et al., 2007] Duvall, P.; Matyas, S. & Glover, A. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional.

[Eisenbarth et al., 2003] Eisenbarth, T.; Koschke, R. & Simon, D. (2003). Locating features in source code. *IEEE Transactions on Software Engineering*, 29:210--224.

[Eisenberg & De Volder, 2005] Eisenberg, A. D. & De Volder, K. (2005). Dynamic feature traces: Finding features in unfamiliar code. Em *Proceedings of the 21st IEEE International Conference on Software Maintenance*, pp. 337--346, Washington, DC, USA. IEEE Computer Society.

[Engström & Runeson, 2011] Engström, E. & Runeson, P. (2011). Software product line testing - a systematic mapping study. *Inf. Softw. Technol.*, 53:2--13.

[Ferreira et al., 2011] Ferreira, G.; Gaia, F.; Figueiredo, E. & Maia, M. (2011). On the use of feature-oriented programming for evolving software product lines: A comparative study. Em *Anais do XV Simpósio Brasileiro de Linguagens de Programação (SBLP)*, São Paulo, Brazil.

[Figueiredo et al., 2008] Figueiredo, E.; Cacho, N.; Sant'Anna, C.; Monteiro, M.; Kulesza, U.; Garcia, A.; Soares, S.; Ferrari, F.; Khan, S.; Castor Filho, F. & Dantas, F. (2008). Evolving software product lines with aspects: an empirical study on design stability. Em *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pp. 261--270, New York, NY, USA. ACM.

[Filho et al., 2006] Filho, F. C.; Cacho, N.; Figueiredo, E.; Maranhão, R.; Garcia, A. & Rubira, C. M. F. (2006). Exceptions and aspects: the devil is in the details. Em *SIGSOFT FSE*, pp. 152–162.

[Fé et al., 2010] Fé, I. S.; Santos, A. R.; Santos, I. S.; Santos Neto, P. & Britto, R. S. (2010). Geração de dados para testes de desempenho e estresse a partir de testes funcionais. Em *Anais do IX Simpósio Brasileiro de Qualidade Software*, pp. 1--12, Belém, Brasil.

[Ghanam & Maurer, 2010] Ghanam, Y. & Maurer, F. (2010). Linking feature models to code artifacts using executable acceptance tests. Em *Proceedings of the 14th international conference on Software product lines: going beyond*, SPLC'10, pp. 211--225, Berlin, Heidelberg. Springer-Verlag.

[Greevy & Ducasse, 2005] Greevy, O. & Ducasse, S. (2005). Correlating features and code using a compact two-sided trace analysis approach. Em *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, CSMR'05, pp. 314--323, Manchester, UK. IEEE Computer Society.

[Hall et al., 2009] Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P. & Witten, I. H. (2009). The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10--18.

[John, 2010] John, I. (2010). Using documentation for product line scoping. *Software, IEEE*, 27(3):42 –47.

[Kiczales et al., 1997] Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.-M. & Irwin, J. (1997). Aspect-oriented programming. Em Akşit, M. & Matsuoka, S., editores, *ECOOP'97 — Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pp. 220–242. Springer Berlin Heidelberg.

[Kitchenham et al., 2009] Kitchenham, B.; Pearl Brereton, O.; Budgen, D.; Turner, M.; Bailey, J. & Linkman, S. (2009). Systematic literature reviews in software engineering - a systematic literature review. *Inf. Softw. Technol.*, 51(1):7--15.

[Kästner et al., 2008] Kästner, C.; Apel, S. & Kuhlemann, M. (2008). Granularity in software product lines. Em *Proceedings of the 30th International Conference on Software Engineering*, ICSE'08, pp. 311--320, New York, NY, USA. ACM.

[Kästner et al., 2011] Kästner, C.; Dreiling, A. & Ostermann, K. (2011). Variability mining with leadt. Relatório técnico, Philipps University Marburg.

[Massol & Husted, 2003] Massol, V. & Husted, T. (2003). *JUnit in Action*. Manning Publications Co., Greenwich, CT, USA.

[Myers, 2004] Myers, G. J. (2004). *The Art of Software Testing*. John Wiley and Sons.

[Petersen et al., 2008] Petersen, K.; Feldt, R.; Mujtaba, S. & Mattsson, M. (2008). Systematic mapping studies in software engineering. Em *Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering*, EASE'08, pp. 68--77, Swinton, UK, UK. British Computer Society.

[Pohl et al., 2005] Pohl, K.; Böckle, G. & Linden, F. J. v. d. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Poshyvanyk et al., 2007] Poshyvanyk, D.; Gueheneuc, Y.-G.; Marcus, A.; Antoniol, G. & Rajlich, V. (2007). Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering*, 33:420--432.

[Ramesh & Jarke, 2001] Ramesh, B. & Jarke, M. (2001). Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58--93.

[Robillard & Weigand-Warr, 2005] Robillard, M. P. & Weigand-Warr, F. (2005). Concernmapper: Simple view-based separation of scattered concerns. Em *Proceedings of the Eclipse Technology Exchange at Object-Oriented Programming, Systems, Languages and Applications*, OOPSLA'05.

[Santos et al., 2011] Santos, I. S.; Santos, A. R. & Santos Neto, P. (2011). Reusing functional testing in order to decrease performance and stress testing costs. Em *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering*, pp. 1--5, Miami, USA.

[Santos et al., 2010] Santos, I. S.; Santos Neto, P.; Moura, R. S. & Soares, A. C. B. (2010). Documentação dirigida por testes. Em *Anais do IX Simpósio Brasileiro de Qualidade de Software*, pp. 25--40, Belém, Brasil.

[Silveira Neto et al., 2011] Silveira Neto, P. A. d. M.; Carmo Machado, I. d.; McGregor, J. D.; de Almeida, E. S. & de Lemos Meira, S. R. (2011). A systematic mapping study of software product lines testing. *Inf. Softw. Technol.*, 53:407--423.

[Sommerville, 2006] Sommerville, I. (2006). *Software Engineering*. Pearson Education.

[Tassey, 2002] Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing. Relatório técnico, National Institute of Standards & Technology (NIST).

[Valente et al., 2012] Valente, M. T.; Borges, V. & Passos, L. (2012). A semi-automatic approach for extracting software product lines. *IEEE Transactions on Software Engineering*, 38(4):737–754.

[van der Linden et al., 2007] van der Linden, F.; Schmid, K. & Rommes, E. (2007). *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, Berlin.

[Varela et al., 2011] Varela, P.; Araújo, J. a.; Brito, I. & Moreira, A. (2011). Aspect-oriented analysis for software product lines requirements engineering. Em *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pp. 667--674, New York, NY, USA. ACM.

[Walkinshaw et al., 2007] Walkinshaw, N.; Roper, M. & Wood, M. (2007). Feature location and extraction using landmarks and barriers. Em *Proceedings of the 23st IEEE International Conference on Software Maintenance*, ICSM'07, pp. 54 – 63. IEEE Computer Society.

[White et al., 2008] White, J.; Schmidt, D. C.; Benavides, D.; Trinidad, P. & Ruiz-Cortés, A. (2008). Automated diagnosis of product-line configuration errors in feature models. Em *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC'08, pp. 225--234, Limerick, Ireland. IEEE Computer Society.

[Wohlin et al., 2012] Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C. & Regnell, B. (2012). *Experimentation in Software Engineering*. Springer.

Anexo A

Como usar a ferramenta TaBuLeTa

Este anexo apresenta uma tutorial de como instalar (Seção A.2) e utilizar (Seção A.3) a ferramenta TaBuLeTa que serve de apoio ao método proposto neste trabalho. Além disso, são enumerados os requisitos de sistema (Seção A.1) necessários para a instalação da ferramenta.

A.1 Requisitos Para Instalação

Antes de poder instalar TaBuLeTa você deve verifiar se o seu sistema obedece os seguintes requisistos:

- Sistema Operacional (SO): qualquer sistema que possa executar uma instância da Ambiente de Desesenvolvimento Integrado (IDE) Eclipse;
- Versão do IDE Eclipse: somente foi testada na distribuição SDK do Eclise Juno. Contudo, uma versão mais recente, embora não possamos garantir, pode funcionar também;

A.2 Instalação

Para instalar o *plug-in* da TaBuLeTa no se Eclipse, siga os passos abixo:

1. Abra o sua IDE Eclipse JUNO;
2. Expanda o menu *Help* e clique em *Install New Software...*;
3. Adicione o seguinte endereço para que o Eclipse possa encontrar os arquivos da ferramenta: *http//alcemirsantos.github.com/tabuleta/update*

4. Selecione TaBuLeTa e prossiga até finalizar o procedimento;

OBSERVAÇÃO: O eclipse irá lhe informar que o conteúdo não está assinado e pode não ser seguro, continue assim mesmo caso deseje.

A.3 Como utilizar

Dado que a ferramenta já está instalada para utilizá-la o usuário de proceder da seguinte maneira:

Passo 0 Abra o sua IDE Eclipse;

Passo 1 Importe o projeto eclipse que contém o código do sistema a ser utilizado como alvo no processo de localização de características;

Passo 2 Abra a visão *Test2Feature Mapping*;

Passo 3 Na visão *Test2Feature Mapping*, utilize o botão 1 (Figura A.1) adicionar as características que devem ser localizadas;

Passo 4 Abra a visão *Package Explorer*;

Passo 5 Na visão *Package Explorer* clique com o botão direito numa classe de teste, vá até o menu "Add to Feature" escolha uma das características adicionadas no **Passo 3**. Só é possivel adicionar a classe, o arquivo java não é aceito;

Passo 6 Repita os passos 3 e 5 quantas vezes forem necessárias para o mapeamento de testes para características. Após o término, utilize os botões 2 ou 3 (Figura A.1) para salvar o arquivo *.cm* do mapeamento de testes para características;

Passo 7 Gere uma suíte de testes para a característica de seu interesse com o botão 4 (Figura A.1). Para gerar a suíte é necessário indicar a pasta que contém os testes na página de preferências da ferramenta. A página de preferências pode ser encontrada através do menu "Window" na opção "Preferences". Na janela que abrirá procure por TaBuLeTa;

Passo 8 Execute em modo de cobertura com a EclEmma, a suíte que foi gerada no pacote *tabuleta.testsuites* no arquivo de nome $< NomeDaSuaCaracteristica > TestSuite$;

Passo 9 Selecione a característica cuja suíte de teste foi a última a ser executada com a EclEmma e gere o arquivo *.cm* com a cobertura dos testes utilizando o botão 5 (Figura A.1);

Paso 10 Importe o arquivo *.cm* com a cobertura dos testes na visão *Test2Feature Mapping*. Você pode fazer isso com a função "arraste e solte", arrastando o arquivo *.cm* da visão *Package Explorer* para a visão *Test2Feature Mapping*.

Utilize o botão 6 (Figura A.1) para excluir os dados da visão *Test2Feature Mapping*.

Figura A.1. Botões na visão *Text2Feature Mapping*.