Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Teste de Software Orientado a Aspectos: Uma Revisão Sistemática

Fabiano Cutigi Ferrari
José Carlos Maldonado

Nº 291

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Janeiro/2007

Teste de Software Orientado a Aspectos: Uma Revisão Sistemática*

Fabiano Cutigi Ferrari
José Carlos Maldonado

Departamento de Ciências de Computação e Estatística
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo – Campus de São Carlos
Laboratório de Engenharia de Software
Caixa Postal 668, 13560-970 – São Carlos, SP, Brasil
e-mail: {ferrari,jcmaldon}@icmc.usp.br

Resumo: A Programação Orientada a Aspectos trouxe benefícios para o desenvolvimento de software e, como toda nova metodologia de desenvolvimento, novos desafios para a atividade de teste. Neste relatório são apresentados detalhes da condução e dos resultados de uma revisão sistemática cujo objetivo foi identificar os trabalhos que abordam a aplicação de técnicas e critérios de teste no contexto de software Orientado a Aspectos. Os resultados mostram que diversos trabalhos têm enfatizado a definição de critérios e a caracterização de tipos de defeitos específicos a esse tipo de software. Além disso, pode-se observar que existe pouca validação dos trabalhos propostos. Os resultados servirão de base para a condução de novos trabalhos relacionados, incluindo a definição e avaliação de critérios de teste, ferramentas de apoio automatizado e estudos experimentais.

Palavras-chave: teste de software, técnicas de teste, critérios de teste, software Orientado a Aspectos, revisão sistemática.

Abstract: Aspect-Oriented Programming has brought several benefits to the software development process. However, as well as other development methodologies, it has also brought new challenges to the testing activity. This paper describes details of the process and results of a systematic review performed aiming at identifying works regarding software testing techniques and criteria applied to aspect-oriented software. The results show that a variety of works have been defining testing criteria and characterising specific fault types. Moreover, we could also observe that most of these approaches lack validation. The systematic review will serve as a basis for new research related to aspect-oriented software testing, including the definition and evaluation of testing criteria, automated support tools and empirical studies.

Keywords: software testing, testing techniques, testing criteria, Aspect-Oriented software, systematic review.

*__Agradecimento:__ Ao apoio financeiro da FAPESP e do CNPq.

Sumário

Capítulo

1

Introdução

A POA (Programação Orientada a Aspectos) surgiu no final da década de 90 como uma possível solução para algumas dificuldades enfrentadas no desenvolvimento de software, principalmente relacionadas à separação de interesses envolvidos no software. A idéia principal é possibilitar que interesses que se encontram espalhados ou entrelaçados sejam implementados tão separadamente quanto possível dos demais (Kiczales et al., 1997).

Os esforços iniciais dedicados à pesquisa envolvendo POA concentraram-se no estabelecimento dos conceitos e em como implementá-los nas tecnologias de apoio. Entretanto, pode-se observar que existe uma preocupação com a garantia da qualidade de software OA (Orientado a Aspectos), mais especificamente com a atividade de teste. A despeito dos benefícios trazidos pela POA para o desenvolvimento de software, sua simples adoção não evita que defeitos sejam introduzidos ao longo do processo de desenvolvimento (Alexander et al., 2004). Dependências implícitas e explícitas entre aspectos e módulos tradicionais (classes, no caso de programas orientados a objetos) resultam em sistemas com novos desafios para a atividade de teste, incluindo novas fontes e novos tipos de defeitos.

Técnicas e critérios tradicionalmente aplicados em software desenvolvido sob os paradigmas procedimental e OO (Orientado a Objetos) têm sido investigados no contexto de software OA, e adaptações têm sido propostas. Além disso, novos critérios baseados em técnicas tradicionais como, por exemplo, teste estrutural e baseado em modelos de estados, têm sido propostos para tratar de características específicas de software OA.

Este relatório apresenta o processo de condução de uma revisão sistemática cujo objetivo principal foi identificar trabalhos que apresentam um ou mais itens de interesse relacionados ao teste de software OA. Uma revisão sistemática consiste em um meio de identificação, avaliação e interpretação de todos os trabalhos de pesquisa relevantes e disponíveis sobre uma questão de pesquisa, tópico ou fenômeno de interesse (Kitchenham, 2004). Apesar de demandar maior esforço do que uma revisão tradicional, uma revisão sistemática tem como premissa evitar viés por parte dos revisores, apresentando também outras vantagens como a possibilidade de ser auditada e replicada.

O principal objetivo da revisão foi a identificação de trabalhos que abordam a aplicação de técnicas e critérios de teste tradicionais ou novos no contexto de software OA. Além disso, a revisão visou também à identificação dos tipos de defeitos OA[1] caracterizados nesses trabalhos, e quais desses trabalhos têm apresentado estudos experimentais para demonstrar a efetividade da abordagem proposta.

Além dos resultados obtidos ao final da revisão, este relatório também inclui o detalhamento das atividades intermediárias realizadas, sendo elas o planejamento da revisão, a estratégia adotada para utilizar as máquinas de busca e a seleção de trabalhos.

Ressalta-se que não foram identificados outros trabalhos na literatura que abordem revisões formais com o mesmo objetivo da revisão apresentada neste relatório, fato que pôde ser confirmado durante a condução da revisão sistemática realizada. Naqvi et al. (2006) avaliam três abordagens de teste propostas na literatura em relação à capacidade de detectar tipos de defeitos caracterizados em uma taxonomia de defeitos candidata para software OA (Alexander et al., 2004). Entretanto, Naqvi et al. limitam-se a essas três abordagens, embora diversas outras já haviam sido propostas até então. Em geral, os demais trabalhos envolvendo teste de software OA apresentam revisões limitadas a poucos trabalhos relacionados, sendo que as seleções desses trabalhos aparentemente não passaram por um processo planejado na mesma linha de uma revisão sistemática.

Resultados parciais desta revisão sistemática foram apresentados em um artigo publicado nos anais do 3^o Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos (WASP'2006) (Ferrari e Maldonado, 2006). Nesse artigo foram enfatizados os resultados relativos às técnicas e critérios de teste abordados nos trabalhos selecionados.

O restante deste relatório está assim organizado: no Capítulo 2, é apresentado o plano elaborado para a condução da revisão, de acordo com um modelo de protocolo adotado. Apresenta-se também nesse capítulo o detalhamento do processo de condução da revisão, incluindo as estratégias de busca adotadas e os resultados obtidos em cada máquina de busca. No Capítulo 3, é apresentada a análise dos resultados obtidos com a condução

[1]Neste relatório, "defeito OA" é utilizado como sinônimo de "defeito característico de software OA".

da revisão sistemática, de acordo com os objetivos e questões de pesquisa propostos. No Capítulo 4, são apresentadas as conclusões deste relatório, em relação tanto à experiência adquirida na condução da revisão sistemática quanto aos resultados obtidos após a análise dos trabalhos recuperados. Apresentam-se também as possibilidades de trabalhos futuros a partir da revisão conduzida. Por fim, no Apêndice A, é apresentado um exemplo de formulário de coleta de informações, que foi preenchido durante a condução da revisão.

Capítulo

2

Planejamento e Condução da Revisão

Diferentemente de revisões tradicionais, uma revisão sistemática é conduzida de acordo com um planejamento (ou protocolo) previamente definido (Biolchini et al., 2005; Kitchenham, 2004). O planejamento é o ponto de partida para a revisão, cujos pontos principais são a definição de uma ou mais questões de pesquisa e dos métodos que serão empregados para conduzir a revisão, incluindo seleção de fontes para buscas e estratégias de busca. Neste capítulo, as etapas de planejamento e condução da revisão são apresentadas em detalhes.

2.1 Planejamento

O planejamento da revisão sistemática foi realizado de acordo com o modelo de protocolo apresentado por Biolchini et al. (2005). Nesta seção, são apresentados os principais pontos do plano elaborado. O plano foi revisado por um especialista, e as sugestões de ajuste foram discutidas com os revisores e implementadas no plano final.

2.1.1 Objetivos da Pesquisa

- Identificar e analisar as técnicas e critérios de teste que têm sido propostos e aplicados para o teste de software OA;

- Identificar os tipos de defeitos específicos de software OA que têm sido descritos nesses trabalhos;
- Observar os estudos experimentais que têm sido realizados como forma de validar as abordagens de teste propostas.

2.1.2 Questões de Pesquisa

Uma questão de pesquisa principal e três questões secundárias foram elaboradas para atender aos objetivos propostos, cada uma com critérios próprios de inclusão e exclusão de trabalhos:

Questão Primária: Quais técnicas e critérios de teste de software têm sido investigados para o teste de software OA?

Questão Secundária 1: Dentre as técnicas e critérios de teste investigados no contexto de teste de software OA, quais são específicas para esse tipo de software?

Questão Secundária 2: Quais tipos de defeitos específicos a software OA têm sido identificados nos trabalhos relacionados com o teste de software OA, incluindo taxonomias e modelos de defeitos?

Questão Secundária 3: Quais tipos de estudos experimentais vêm sendo realizados nos trabalhos relacionados com o teste de software OA como forma de validação das abordagens propostas?

Itens relacionados ao escopo (*range*) ou especificidades (*specificity*) das questões:

- *Intervenção*: Técnicas e critérios de teste de software.
- *Controle*: Coleção de artigos e outros trabalhos levantados e relacionados em revisões bibliográficas de monografias de qualificação, dissertações de mestrado e teses de doutorado relacionadas ao teste de software OA redigidas pelo grupo.
- *População*: Pesquisadores e desenvolvedores de software que, respectivamente, utilizam recursos e tecnologias OA em sua pesquisa ou em seu processo de desenvolvimento.
- *Resultados*: Técnicas e critérios de teste de software aplicados em software OA.
- *Aplicação*: Projetos de desenvolvimento de software que envolvam utilização de recursos e tecnologias OA.

2.1.3 Estratégia de Busca para Seleção de Estudos Primários

A estratégia de busca e seleção dos estudos primários foi definida de acordo com as fontes de trabalhos e as línguas de redação selecionadas, e de acordo com as palavras-chave para a revisão:

- **Fontes**: bases de dados eletrônicas indexadas (IEEE, ACM e Springer), busca em máquinas de busca eletrônica (Scirus e Google), anais de eventos relacionados e consultas a especialistas.
- **Língua dos trabalhos**: inglês e português. Inglês por essa ser a língua internacionalmente aceita para a redação de trabalhos científicos. Português porque sua exclusão automaticamente excluiria trabalhos relevantes de autoria de pesquisadores brasileiros. Esses trabalhos envolvem tanto a definição de critérios de teste de software OA quanto a implementação de ferramentas de apoio automatizado.
- **Palavras chave**: *"aspect-oriented"* e *"software testing"*, com os seguintes termos e frases relacionadas:
  - *Aspect-oriented*: aspect-oriented software, aspect-oriented application, aspect-oriented app, aspect-oriented program, AOP.
  - *software testing*: testing, testing criterion, testing technique, fault, defect, error, incorrect, fault model, failure

2.1.4 Critérios e Procedimento para Seleção dos Estudos

Critérios de inclusão

Os seguintes critérios de inclusão de trabalhos foram definidos para atender a cada uma das questões de pesquisa:

- Questão Primária: Aplicação de técnica/critério de teste em software OA.
- Questão secundária 1: Proposição de novas técnicas e critérios específicos para teste de software OA.
- Questão secundária 2: Caracterização de tipos de defeitos específicos de software OA.
- Questão secundária 3: Realização de estudos experimentais para validar a abordagem proposta.

Critérios de exclusão

Os seguintes critérios de exclusão de trabalhos foram definidos para atender a cada uma das questões de pesquisa:

- Questão Primária: Aplicação de técnica/critério de teste em software desenvolvidos sob outros paradigmas (por exemplo, procedimental ou orientado a objetos).
- Questão secundária 1: Aplicação de técnica/critério de teste em software OA adaptadas de outros paradigmas.
- Questão secundária 2: Trabalhos que não caracterização tipos de defeitos específicos de software OA.
- Questão secundária 3: Trabalhos que não apresentam estudos experimentais para validar a abordagem proposta.

Processo de seleção preliminar

Serão construídas *strings* de busca formadas pela combinação dos sinônimos das palavras-chave identificados. Essas *strings* serão submetidas às máquinas de busca relacionadas. Em seguida, será realizada a leitura dos resumos dos trabalhos recuperados por ambos os revisores. Constatando-se a relevância de um trabalho, já destacada no resumo, e havendo um consenso entre os revisores, ele será pré-selecionado para ser lido na íntegra. Ocorrendo falta de consenso, esse trabalho será colocado em espera, e sua inclusão ou exclusão será definida em reuniões entre os revisores.

Processo de seleção final

A leitura completa dos trabalhos selecionados na etapa de seleção preliminar será realizada por pelo menos um dos revisores, que se encarregará de fazer um resumo, destacando a abordagem de teste apresentada no trabalho e os conceitos subjacentes envolvidos.

2.1.5 Extração dos Resultados

Após a leitura completa dos trabalhos selecionados, será elaborado um relatório de forma a sintetizar as análises críticas elaboradas pelos revisores. O relatório poderá conter tanto sínteses discursivas quanto tabuladas, de forma a comparar as diferentes técnicas e critérios aplicados no contexto de teste de software OA. Para apoiar a comparação entre os trabalhos, a síntese poderá incluir uma lista de verificação (*checklist*) de itens que devem ser observados em relação às técnicas e critérios identificados, como, por exemplo,

uma lista de características e tipos defeitos específicos de software OA que são enfatizados pelas técnicas e critérios.

2.2 Condução

A revisão sistemática foi conduzida por um período de três meses (Maio/2006 a Julho/2006), de acordo com o planejamento apresentado nas seções anteriores. Ao todo, foram recuperados 260 trabalhos, que foram submetidos para as etapas de seleção preliminar, seleção final e extração de resultados. Nas próximas seções são apresentados mais detalhes das atividades realizadas, incluindo a estratégia adotada para construção das *strings* de busca e os resultados das buscas para cada uma das fontes selecionadas.

2.2.1 Seleção Preliminar

A seleção preliminar foi conduzida em três etapas:

1. Construção das *strings* de busca;
2. Realização das buscas;
3. Seleção preliminar de trabalhos.

Essas etapas são detalhadas nas próximas seções. Vale observar que as buscas em repositórios indexados (IEEE, ACM e Springer) foram realizadas no início da condução da revisão e foram replicadas no final, após a leitura dos artigos pré-selecionados. A replicação foi realizada como uma forma de verificação dos resultados obtidos na primeira busca, sendo que foram utilizadas as mesmas *strings* de busca. Optou-se pela replicação das buscas somente nos repositórios indexados devido à consistência de resultados recuperados para buscas replicadas, o que nem sempre é possível em máquinas de busca convencionais, conforme poderá ser observado na Seção 2.2.1.2.

2.2.1.1 Construção das Strings de Busca

Para a construção das *strings* de busca, foram utilizados os sinônimos das palavras-chave identificados na Seção 2.1.4. Uma opção seria construir uma *string* completa utilizando os operadores lógicos "e" (*AND*) e "ou" (OR), como a apresentada a seguir:

```
(aspect-oriented software OR aspect-oriented application OR aspect-
oriented app OR aspect-oriented program OR AOP) AND (testing OR
testing criterion OR testing technique OR fault OR defect OR error
OR incorrect OR fault model OR failure)
```

Entretanto, como poderá ser observado no detalhamento das buscas realizadas, a entrada dessa *string* completa em algumas das máquinas de busca não retorna os resultados esperados. Nesse caso, foram adotadas estratégias particulares para a construção de uma ou mais *strings* para essas máquinas, que retornassem resultados relevantes para a busca.

2.2.1.2 Buscas Realizadas

A seguir são apresentadas em mais detalhes as buscas realizadas em cada uma das fontes selecionadas. Vale observar que como as buscas foram realizadas em seqüência, trabalhos retornados que já haviam aparecido em buscas anteriores não foram novamente analisados. Isso ocorria somente após o revisor constatar que o respectivo trabalho havia sido devidamente analisado em uma busca anterior.

Busca no IEEE

A *string* de busca completa (Seção 2.2.1.1) foi ligeiramente modificada para ser submetida à máquina no modo "pesquisa avançada". As modificações consistiram em uma adaptação para o padrão de composição de *strings* utilizando os operadores lógicos `<or>` (*OR*) e `<and>` (*AND*). A busca foi restringida aos títulos e resumos das publicações, conforme planejado para a atividade de seleção preliminar (Seção 2.1.4).

A *string* a seguir retornou um total de 36 trabalhos:

```
((<or>(aspect-oriented software, aspect-oriented application, aspect-
oriented app, aspect-oriented program, AOP)<and><or>(testing, testing
criterion, testing technique,fault, defect, error, incorrect, fault
model, failure))<in>(ab,ti))
```

Algumas observações:

- A busca foi inicialmente realizada em 17/05/2006 e replicada em 31/07/2006, na página de busca avançada (`http://ieeexplore.ieee.org/search/advsearch.jsp`).

- A string foi inserida no campo de entrada da Opção "2" de busca. O restante das opções busca ficaram preenchidas conforme o padrão (*default*) da página de busca.

Busca na ACM

Algumas dificuldades surgiram para realizar a busca na máquina da biblioteca digital da ACM (`http://portal.acm.org/advsearch.cfm`). Não foi encontrado um modo de definir uma *string* de busca completa como foi feito na busca pela máquina do IEEE. Para amenizar as dificuldades e manter a ACM como um dos repositórios pesquisados, foi utilizada a seguinte estratégia:

- Foram construídas *strings* considerando-se a combinação dos termos mais abrangentes para as questões de pesquisa;
- A busca foi realizada em 19/05/2006 na página de busca tradicional da *ACM Digital Library* (`http://portal.acm.org/dl.cfm`) e replicada em 04/08/2006. A página de busca avançada foi utilizada somente para alguns testes de construção de *strings*. As *strings* foram então submetidas na página de busca tradicional, sendo inseridas no campo de entrada adequado.

As *strings* a seguir foram submetidas à máquina de busca da ACM. O total de trabalhos retornados para cada uma delas está representado entre parênteses logo após cada uma delas.

```
+abstract:"aspect-oriented" +abstract:testing (13)
+abstract:"aspect-oriented" +abstract:fault   (8)
+abstract:"aspect-oriented" +abstract:defect  (0)
+abstract:"aspect-oriented" +abstract:"fault model" (1)
+abstract:"aspect-oriented" +"fault model" (4)
+abstract:"aspect-oriented" +abstract:failure  (1)
```

Vale ressaltar que nos casos em que ocorreram replicações de trabalhos recuperados para diferentes *strings* de busca, as repetições foram descartadas durante a análise dos resultados das buscas.

Busca na Springer (Kluwer)

As buscas por publicações até então realizadas no repositório da Kluwer passaram a ser redirecionadas para serem realizadas na máquina de busca da Springer (SpringerLink - `http://www.springerlink.com/`), conforme informativo na página da Springer[1].

Da mesma forma que na busca pela máquina da ACM, algumas dificuldades[2] surgiram durante a execução das buscas na máquina da biblioteca digital da Springer, sendo elas:

- A opção de "Pesquisa Avançada" permitia a criação de *strings* de busca utilizando operadores lógicos para combinar os termos desejados. Entretanto, existiam limitações para a formação das *strings*, sendo que *string* extensas (por exemplo, como a utilizada na busca da máquina do IEEE) não retornavam os resultados desejados.

- A procura por termos compostos (por exemplo, *"aspect-oriented"*) não funcionava como desejado. A busca retornava publicações que possuem todas as partes do termo composto, mas não necessariamente juntos.

- A princípio, a máquina permitia restringir a busca aos títulos e resumos das publicações, da mesma forma que a máquina do IEEE. Entretanto, observou-se com algumas tentativas de busca que a restrição não era atendida e a busca era feita no corpo do texto, retornando uma grande quantidade de publicações sendo que diversas delas não eram relevantes para a revisão.

- A seleção dos periódicos/publicações de interesse era dificultada pois não existe uma grande área (por exemplo, Ciência da Computação) para ser selecionada. Uma extensa lista de periódicos/publicações era exibida como possibilidades de refinamento da busca.

Para amenizar as dificuldades e manter a Springer como um dos repositórios pesquisados, foi adotada uma estratégia semelhante à utilizada para a busca na máquina da ACM:

- Foram construídas *strings* considerando a combinação dos termos mais abrangentes para as questões de pesquisa. Alguns testes de busca foram realizados e, de acordo com os resultados, um subconjunto de possíveis *strings* foi utilizado.

[1] `http://springer.lib.tsinghua.edu.cn/(gwwxu0bgx0am1ari2xzrug55)/app/home/main.asp?referrer=default`

[2] Algumas dificuldades não estão mais presentes em uma versão mais recente da máquina de busca da Springer, fato que pôde ser notado durante a condução de outra revisão sistemática em Novembro/2006.

- Foram selecionados todos os periódicos/publicações para busca. Dessa forma, algumas publicações retornadas na busca poderiam não ser da área de interesse desta revisão e, quando isso foi constatado, foram descartadas.
- A busca foi realizada utilizando-se as opções de pesquisa da aba *"Article by text"* da página de busca avançada. Os campos de pesquisa foram preenchidos da seguinte forma:
  - No campo *"Search for"* foram inseridas as strings de busca;
  - Em *"Using"*, foi selecionada a opção *"All Words"*;
  - Em *"Within"* foi marcada a opção *"Abstract (Includes Title)"*;
  - Os demais campos ficaram preenchidos conforme o padrão (*default*).

As buscas foram realizadas em 23/05/2006 e replicadas em 31/07/2006. As *strings* submetidas à máquina de busca da Springer são apresentadas a seguir. O total de trabalhos retornados para cada uma delas está representado entre parênteses logo após cada uma delas.

```
"aspect oriented" software testing (17)
"aspect oriented" software fault (7)
```

Também vale ressaltar que nos casos em que ocorreram replicações de trabalhos recuperados para as diferentes *strings* de busca, as repetições foram descartadas durante a análise dos resultado das buscas.

Busca na Scirus (Elsevier)

As buscas realizadas na máquina de busca da Elsevier[3] retornam somente publicações completas como, por exemplo, livros e periódicos, não dando acesso aos trabalhos individuais presentes nessas ou em outras publicações. Entretanto, trabalhos individuais (artigos e relatórios, entre outros) podem ser recuperados a partir de uma máquina de busca associada, a Scirus[4]. Dessa forma, adotou-se a máquina de busca Scirus como substituta à da Elsevier. A busca foi realizada a partir da página de busca avançada (http://www.scirus.com/srsapp/advanced/index.jsp).

Em relação às *string* de busca utilizadas, não foi possível a construção imediata de uma *string* completa, semelhante à submetida à máquina do IEEE. A *string* completa foi

[3] `http://www.elsevier.com`
[4] `http://www.scirus.com/srsapp/`

particionada em duas *substrings*, cada uma com o conjunto de sinônimos relacionados, procedendo-se da seguinte forma:

- No primeiro campo de entrada foram inseridos os termos relacionados à POA e selecionada a opção *"any of these words"*. Essa *substring* ficou assim construída:

```
"aspect oriented software" OR "aspect oriented application" OR
"aspect oriented app" OR "aspect oriented program" OR aop
```

- No segundo campo de entrada foram inseridos os termos relacionados ao teste de software e selecionada a opção *"any of these words"*. Essa *substring* ficou assim construída:

```
testing OR "testing criterion" OR "testing technique" OR fault
OR defect OR error OR incorrect OR "fault model" OR failure
```

- O operador *"AND"* foi selecionado para formar a composição das duas *substrings*.
- Em *"Information types"*, foram marcadas as opções *"Articles"* e *"Books"*;
- Em *"File formats"* foram marcadas as opções "PDF" e "HTML";
- Em *"Subject areas"* foi marcada a opção *"Computer Science"*;
- Os demais campos ficaram preenchidos conforme o padrão (*default*).

A string final formada no procedimento acima, listada a seguir, aparece na página de resultados da busca. Um total de 129 trabalhos foi recuperado.

```
("aspect oriented software" OR "aspect oriented application" OR
"aspect oriented app" OR "aspect oriented program" OR aop) AND
(testing OR "testing criterion" OR "testing technique" OR fault
OR defect OR error OR incorrect OR "fault model" OR failure)
```

Algumas observações que merecem destaque:

- A busca foi realizada em 19/05/2006;

- A tentativa de repetição de uma mesma consulta em momentos distintos nem sempre retornava os mesmos resultados nessa máquina de busca. Uma possível causa para essa diferença poderia ser a indisponibilidade de fontes no momento da consulta, pois aparentemente a Scirus não tem uma base de dados de publicações própria, realizando suas buscas em outros repositórios da *Web*;

- As informações sobre os trabalhos recuperados na busca não eram exibidos de forma estruturada, como acontece com as buscas realizadas em outras máquinas que trabalham com bases de dados próprias como, por exemplo, o IEEE e a ACM. Para se obter as informações relevantes sobre os trabalhos, era necessário navegar pelos *links* retornados.

Busca no Google

A busca por informações específicas no Google não é trivial, principalmente devido ao grande número de resultados geralmente retornados em buscas simples (por exemplo, buscas por alguns termos-chave). Mesmo assim, é relevante incluir a pesquisa no Google em uma revisão sistemática, visto que diversos trabalhos não disponíveis nos repositórios tradicionais (por exemplo, IEEE e ACM) podem ser recuperados no Google. Como exemplo, podem-se citar relatórios técnicos e artigos não incluídos em bases indexadas.

Após várias tentativas de compor uma *string* de busca que retornasse uma quantidade viável de publicações, chegou-se à *string* seguinte, que retornou 506 publicações:

```
"aspect-oriented software" OR "aspect-oriented application" OR
"aspect-oriented app" OR "aspect-oriented program" testing fault
filetype:pdf
```

Algumas observações:

- A busca foi realizada em 06/06/2006 na página de pesquisa tradicional do Google (`http://www.google.com`), tendo sido filtrada por tipo de arquivo (somente arquivos em formato PDF) e pela língua inglesa (somente termos em inglês foram utilizados para construir a *string* de busca).

- Após uma observação dos resultados da busca, constatou-se que existe uma recorrência das publicações. Dessa forma, observou-se que acessar as primeiras páginas de resultados seria suficiente para recuperar as publicações relevantes.

Consulta a Especialistas

Conforme observado na Seção 2.1.3, a opção de consulta a especialistas poderia ser considerada como um viés para a revisão sistemática, visto que os especialistas consultados poderiam indicar trabalhos próprios para serem incluídos na revisão. Entretanto, a opção de não consultar os especialistas poderia excluir trabalhos relevantes como, por exemplo, relatórios técnicos e artigos não presentes em bases indexadas.

Dessa forma, a estratégia adotada para consulta a especialistas foi a seguinte:

- Antes de consultar os especialistas, foram realizadas as buscas nas demais fontes selecionadas (IEEE, ACM, Springer, Scirus e Google);
- Após a seleção preliminar dos artigos, uma lista de trabalhos pré-selecionados foi elaborada para ser apresentada aos especialistas;
- Após uma avaliação da lista, o especialistas recomendam a inclusão de outros trabalhos não presentes na lista na revisão sistemática.

A consulta aos especialistas proporcionou a identificação de eventos relacionados ao desenvolvimento e teste de software OA que possivelmente conteriam trabalhos relevantes para a revisão pretendida. Pode-se observar que nem todos os eventos tinham seus trabalhos incluídos nas bases indexadas selecionada como fontes de busca.

Dentre os eventos identificados, estava a segunda edição do WTAOP(*Workshop on Testing Aspect-Oriented Programs*), direcionado para trabalhos específicos de teste de software OA, e alguns dos trabalhos aceitos foram pré-selecionados para leitura completa. Observa-se que embora os trabalhos aceitos nessa segunda edição do WTAOP tenham sido disponibilizados na biblioteca digital da ACM, nem todos foram recuperados com as *strings* de busca construídas, devido à estratégia adotada de restringir a busca aos resumos dos trabalhos. Como um dos especialistas consultados teve um trabalho aceito no evento, ele recebeu uma cópia prévia dos demais trabalhos aceitos.

Na consulta a especialistas, quatro trabalhos foram identificados, sendo duas indicações diretas dos especialistas e duas publicação da segunda edição do WTAOP que não haviam sido recuperadas na busca realizada na biblioteca digital da ACM.

2.2.1.3 Seleção Preliminar de Trabalhos

Os resumos dos trabalhos recuperados foram lidos logo ao término das buscas nos repositórios indexados e nas máquinas de busca convencionais. Após a leitura, foi elaborada

uma lista de referências de trabalhos pré-selecionados, que foi encaminhada para os especialistas, conforme descrito na Seção 2.2.1.2. Os resumos dos trabalhos indicados pelos especialistas foram lidos na seqüência.

Considerando todos os trabalhos, constatou-se que não foi possível definir a relevância de alguns deles tendo como base somente a leitura dos resumos. Nesses casos, outras seções do texto (por exemplo, introdução e conclusão) foram analisadas e a decisão sobre a pré-seleção do trabalho foi tomada. Observa-se que não foi necessária a realização de reunião entre os revisores para a decisão de inclusão ou exclusão de trabalhos.

2.2.2 Seleção Final e Extração dos Resultados

A leitura completa de cada um dos trabalhos pré-selecionados foi realizada por pelo menos um dos revisores. Um formulário foi elaborado para possibilitar a coleta das informações relevantes de cada artigo, de acordo com as questões de pesquisa definidas. A própria estrutura do formulário de coleta serviu como lista de verificação de informações que deveriam ser identificadas nos trabalhos lidos. Um exemplo de formulário preenchido é apresentado no Anexo A.

A leitura completa dos trabalhos possibilitou a identificação de 25 trabalhos relevantes para os objetivos da revisão, sendo que os critérios de inclusão e exclusão puderam ser aplicados conforme planejado. Foram selecionados trabalhos que têm como foco principal a proposição ou aplicação de técnicas e critérios de teste em software OA, ou que explicitamente tratam de tipos de defeitos OA. Trabalhos cujo foco principal diferem desse foco não foram incluídos. Dentre esses, estão trabalhos que apresentam tecnologias de apoio ao teste ou outras atividades relacionadas ao teste como, por exemplo, teste de regressão, depuração ou geração de casos de teste sem o apoio de uma técnica ou critério específico. As informações de interesse para as questões de pesquisa foram identificadas na etapa de extração de resultados. Para cada trabalho selecionado foi preenchido um formulário de coleta de informações. A análise dos resultados obtidos, de acordo com as informações coletadas, é apresentada no próximo capítulo.

Capítulo

3

Análise dos Resultados

Neste capítulo são apresentados os resultados obtidos com a condução da revisão sistemática, de acordo com os objetivos e questões de pesquisa propostos. Ressalta-se que a revisão teve como objetivo identificar os trabalhos que envolvem a aplicação de técnicas e critérios e outras questões relacionadas. Entretanto, fatores de qualidade desses trabalhos não foram considerados, podendo ser avaliados em trabalhos futuros.

Inicialmente, um sumário dos trabalhos selecionados é organizado na Tabela 3.1. Algumas colunas da tabela foram incluídas para facilitar o mapeamento dos trabalhos de acordo com as questões de pesquisa.

A primeira coluna da Tabela 3.1 contém uma numeração seqüencial para os trabalhos. As colunas intituladas "Trabalho" e "Fonte" contêm a citação do trabalho e a fonte a partir da qual o trabalho foi recuperado. A coluna "Técnica Explorada" informa a(s) técnica(s) de teste enfatizada(s) no trabalho. A coluna "Define Critério" informa se no trabalho é proposto algum critério específico para o teste de software OA. A coluna "Estudo Experim." indica quais trabalhos apresentam algum tipo de estudo experimental para validar a abordagem proposta. Por fim, a coluna "Defeitos OA" contém a indicação de trabalhos que caracterizam tipos de defeitos OA.

De acordo com a Tabela 3.1, pode-se observar que alguns trabalhos abordam mais de uma técnica de teste em conjunto. Outros, por sua vez, não enfatizam uma técnica específica, concentrando-se em propor estratégias de teste e sugerindo técnicas e critérios

Tabela 3.1: Sumário dos trabalhos selecionados.

#	Trabalho	Fonte	Técnica Explorada	Define Critério	Estudo Experim.	Caracteriza Defeitos OA
1	(Mortensen e Alexander, 2004)	Google	estrutural e baseada em defeitos	sim	estudo de caso	não
2	(Mortensen e Alexander, 2005)	Google	estrutural e baseada em defeitos	sim	estudo de caso	não
3	(Lemos et al., 2006)	Especialista/ ACM	estrutural e baseada em defeitos	não	não	sim
4	(van Deursen et al., 2005)	Google	funcional e estrutural	não	estudo de caso	sim
5	(Zhao, 2003)	IEEE	estrutural	não	não	não
6	(Zhao, 2002)	Scirus	estrutural	não	não	não
7	(Lemos et al., 2005)	Google	estrutural	sim	não	não
8	(Lemos et al., 2004b)	Especialista	estrutural	sim	não	não
9	(Lemos et al., 2004a)	Especialista	estrutural	sim	não	não
10	(Xie e Zhao, 2006)	ACM	estrutural e baseada em modelos de estados	sim	estudo de caso	não
11	(Xie et al., 2005)	Scirus	estrutural e baseada em modelos de estados	não	não	não
12	(Xu et al., 2004)	Google	estrutural e baseada em modelos de estados	não	não	não
13	(Xu et al., 2005)	Google	baseada em modelos de estados	não	não	não
14	(Xu e Xu, 2006a)	ACM	baseada em modelos de estados	sim	não	sim
15	(Xu e Xu, 2006b)	ACM	baseada em modelos de estados	não	estudo de caso	sim
16	(Badri et al., 2005)	IEEE	baseada em modelos de estados	sim	não	não
17	(Xu e Xu, 2005)	Google	baseada em modelos UML	não	não	não
18	(Massicotte et al., 2005)	IEEE	baseada em modelos UML	sim	não	não
19	(Massicotte et al., 2006)	Springer	baseada em modelos UML	sim	não	sim
20	(Lesiecki, 2005)	Google	não enfatizam	não	não	não
21	(Bækken e Alexander, 2006)	Especialista/ ACM	não enfatizam	não	não	sim
22	(Alexander et al., 2004)	Google	não enfatizam	sim	não	sim
23	(McEachen e Alexander, 2005)	ACM	não enfatizam	não	não	sim
24	(Ceccato et al., 2005)	Google	não enfatizam	sim	não	sim
25	(Zhou et al., 2004)	Google	não enfatizam	sim	estudo de caso	não

que sejam convenientes para as etapas de teste que compõem a estratégia. Ainda, alguns trabalhos discutem questões envolvidas no teste de software OA e caracterizam defeitos específicos. Observa-se também que todas as técnicas de teste exploradas nesses trabalhos já vinham sendo empregadas no teste de software desenvolvido sob outros paradigmas. Dessa forma, não foi necessário incluir na tabela uma coluna exclusiva para informar se o trabalho aborda alguma técnica específica para software OA.

Todos os trabalhos incluídos na Tabela 3.1 são sucintamente apresentados na próxima seção. A Seção 3.2, com suas respectivas subseções, está organizada de acordo com as questões de pesquisa definidas para a revisão sistemática.

3.1 Síntese dos Trabalhos Selecionados

Os trabalhos apresentados nesta seção estão agrupados de acordo com a técnica de teste abordada. Pode-se observar que alguns trabalhos abordam a aplicação de mais de uma técnica em conjunto. Cada trabalho ou conjunto de trabalhos relacionados é apresentado em um parágrafo.

Teste Estrutural e Baseado em Defeitos

Mortensen e Alexander apresentam uma abordagem mista de teste estrutural e baseado em defeitos para programas OA escritos em AspectJ (Mortensen e Alexander, 2004, 2005). Os autores tentam adequar os tipos de defeitos que podem ser encontrados à taxonomia de defeitos candidata para software OA (Alexander et al., 2004). Nesses trabalhos, definem um conjunto de critérios estruturais para atuarem em diferentes níveis de granularidade, desde cobertura de comandos e pares definição-uso de variáveis até a cobertura de execução de todos as inserções de adendos e todas as introduções realizadas por um aspecto. No contexto de teste baseado em defeitos, definem um conjunto de três operadores de mutação para atuar nas definições de escopo dos conjuntos de junção e nas definições de precedência de aspectos.

Lemos et al. apresentam uma abordagem mista de teste estrutural e baseado em defeitos para o teste de descritores de conjuntos de junção (PCD - *Pointcut Descriptor*) (Lemos et al., 2006). A estratégia proposta é dividida em dois passos: (i) identificar pontos de junção selecionados não desejados, com apoio de teste estrutural baseado em fluxo de controle; e (ii) identificar pontos de junção negligenciados com apoio de teste baseado em Análise de Mutantes. Grafos de fluxo de controle compostos pelos grafos do método e dos adendos que o afetam são utilizados no primeiro passo, e operadores de mutação para ampliar o escopo de conjuntos de junção são utilizados no segundo passo.

Teste Estrutural e Funcional

van Deursen et al. apresentam uma estratégia para refatoração de software OO com o apoio da POA (van Deursen et al., 2005). Nessa estratégia está incluída uma estratégia de teste para garantir o correto comportamento do software independentemente da refatoração. A estratégia, intitulada *BETTAR* (*Better Evolvability Through Tested Aspect Refactorings*), consiste de cinco passos, podendo-se destacar o projeto do conjunto de testes que seja sensível a exercitar as porções de código refatoradas. Uma taxonomia

de defeitos de defeitos também é proposta no trabalho, incluindo defeitos relacionados a declarações inter-tipos, definição de conjuntos de junção e definição e implementação de adendos.

Teste Estrutural

Zhao apresenta uma abordagem de teste estrutural de unidade baseado em fluxo de dados (Zhao, 2002, 2003). A abordagem é baseada na implementação da linguagem AspectJ corrente. As unidades consideradas são as classes e os aspectos que compõem a aplicação completa. Os métodos e adendos são considerados os módulos que compõem as unidades, a partir dos quais o autor define três níveis de teste: (i) intra-módulo; (ii) inter-módulo; e (iii) intra-classe ou intra-aspecto. O autor explora duas perspectivas de teste. A primeira considera um aspecto em conjunto com os métodos que podem ser afetados pelos seus adendos, e a segunda considera uma classe em conjunto com os adendos que podem afetar seu comportamento. Um conjunto de grafos de fluxo de controle acrescido de informações sobre o fluxo de dados é definido, a partir dos quais são derivados pares definição-uso (def-uso) para cada um dos níveis de teste abordados.

Lemos et al. apresentam uma abordagem de teste estrutural de unidade que permite o teste dos aspectos separadamente das classes base da aplicação (Lemos et al., 2005, 2004b). A abordagem é focada para o teste de programas AspectJ. Esse trabalho tem como base o trabalho de Vincenzi (2004), que propõe uma abordagem de teste estrutural de unidade OO a partir de informações extraídas dos *bytecodes* Java. Para Vincenzi, a unidade considerada é um método. Para Lemos et al., as unidades podem ser métodos ou adendos. Lemos et al. (2004b) definem o grafo de fluxo de controle AOCFG (*Aspect-Oriented Control Flow Graph*). Além de nós convencionais e de tratadores de exceções, o AOCFG inclui nós transversais, que representam os pontos de junção capturados pelos aspectos que interceptam as classes da aplicação e os adendos que atuam nesses pontos. O AOCFG é estendido para incluir informações de definições e usos de variáveis, resultando no AODU (*Aspect-Oriented Def-Use*) (Lemos et al., 2005). Um conjunto de critérios baseados em fluxo de controle e fluxo de dados é definido, e uma ferramenta para apoiar a aplicação dos critérios é sucintamente apresentada.

Lemos et al. também apresentam uma abordagem de teste estrutural de integração de programas OA (Lemos et al., 2004a), tendo como base a abordagem de teste de unidade apresentada em seus outros trabalhos (Lemos et al., 2005, 2004b). O grafo AODU é estendido, resultando no grafo MADU (*Method-Advice Def-Use*), que consiste no grafo AODU de um método adicionado dos grafos AODU dos adendos que alteram o compor-

tamento desse método. Baseado nas diferentes interações de dados que podem ocorrer entre métodos e adendos, quatro critérios envolvendo pares definição-uso de variáveis são definidos.

Teste Estrutural e Baseado em Modelos de Estados

Xie et al. e Xie e Zhao apresentam abordagens de teste estrutural e baseado em modelos de estados com o apoio de um *framework* intitulado Aspectra (Xie e Zhao, 2006; Xie et al., 2005). O Aspectra é utilizado para gerar classes empacotadoras (*wrappers*) para os aspectos e para as classes afetadas por eles. Essas classes empacotadoras são então submetidas para duas ferramentas que geram casos de teste considerando cobertura estrutural e de estados. Os casos de teste são avaliados segundo dois critérios definidos no trabalho. O Aspectra apóia a instrumentação e execução dos testes, calculando a cobertura obtida. A principal diferença entre os dois trabalhos está nas partes de código enfatizadas nas abordagens. Enquanto em um dos trabalhos o foco está no teste de métodos afetados por adendos, adendos e métodos inter-tipo declarados (Xie et al., 2005), no outro o foco está no teste de comportamentos aspectuais (implementados em adendos, métodos inter-tipo declarados ou métodos de aspectos) (Xie e Zhao, 2006).

Xu et al. apresentam uma abordagem de teste baseado em modelos de estados e teste estrutural (Xu et al., 2004, 2005). Uma extensão para o modelo de estados tradicional é proposta, intitulada ASM (*Aspectual State Model*). O ASM contém notações para representar as construções básicas de POA e os novos estados resultantes da atuação dos aspectos. Uma árvore de transições de estados é derivada do modelo de estados. Essa árvore é estendida de forma a incluir os grafos de fluxo de controle das operações que geram as transições de estados (Xu et al., 2004). Critérios baseados em fluxo de controle podem ser empregados em conjunto com os critérios de cobertura de transições de estados.

Teste Baseado em Modelos de Estados

Xu e Xu estendem os trabalhos de Xu et al. (2004, 2005), definindo formalmente um modelo de estados que inclui estados referentes tanto a uma classe base quanto resultantes da atuação dos aspectos sobre essa classe (Xu e Xu, 2006a). A partir desse modelo, é proposto um procedimento para a geração de testes. Parte-se do modelo de estados considerando somente as classes base e evolui-se para o modelo incluindo os estados inseridos ou modificados pelos aspectos. Para ambos os casos, os testes são gerados de acordo com

um critério de cobertura de ramos de uma árvore de transições de estados que é obtida a partir do modelo.

Xu e Xu apresentam também uma abordagem para teste baseada em modelos de estados cujo foco é no teste de aspectos que fazem a integração de uma classe base com classes utilizadas, por exemplo, em declarações inter-tipos (Xu e Xu, 2006b). Essas últimas são classificadas como classes integradas, e o aspecto que realiza a introdução é classificado como aspecto de integração. O modelo de estados apresentado por Xu e Xu (2006a) é estendido de forma a representar os estados de mais de uma classe simultaneamente. Um novo modelo de estados é definido para representar como as classes integradas são utilizadas em conjunto com as classes base. A mesma estratégia de teste incremental proposta por Xu e Xu (2006a) é utilizada tendo como base os modelos de estados combinados.

Badri et al. apresentam uma abordagem de teste baseado em modelos de estados cujo foco está no teste de uma classe que tem seu comportamento afetado por um ou mais aspectos (Badri et al., 2005). O modelo de estados OO é estendido para incluir os estados resultantes da atuação dos aspectos, contendo também novas notações para representar os pontos de atuação dos aspectos. Partindo-se do teste de uma classe isolada, adiciona-se os estados resultantes da atuação de um aspecto por vez. A geração dos casos de teste tem como base uma árvore de transições de estados obtida a partir do modelo.

Teste Baseado em Modelos UML

Apesar do modelo de estados fazer parte do conjunto de modelos da UML, alguns trabalhos têm utilizado outros modelos da UML como base para a derivação de requisitos de teste. Dentre os trabalhos selecionados, três deles utilizam diagramas de interação da UML e são apresentados a seguir.

Xu e Xu apresentam uma abordagem de teste baseada em modelos da UML (Xu e Xu, 2005). Os autores estendem a UML para permitir representar os pontos onde os aspectos modificam o comportamento das classes base. A partir do diagrama de seqüência estendido, um grafo de mensagens é construído. Uma árvore de transições é derivada do grafo de mensagens, e os casos de teste são gerados para percorrer caminhos dessa árvore. Dois critérios são abordados, envolvendo a cobertura de ramos da árvore e a cobertura de mensagens polimórficas.

Massicotte et al. apresentam uma abordagem de teste baseado em diagramas de colaboração (Massicotte et al., 2006, 2005). Essa abordagem semelhante à de teste baseado em estados proposta por Badri et al. (2005). Entretanto, o foco está no teste da integração de um ou mais aspectos com um grupo de objetos que colaboram entre si. A abordagem

é iterativa, iniciando com o teste dos cenários de colaboração sem considerar os aspectos. Na segunda etapa, um a um os aspectos são analisados e o diagrama de colaboração é modificado para representar as novas seqüências resultantes da combinação do aspecto. Para a geração dos casos de teste, uma árvore de mensagens é derivada do diagrama de colaboração, representando as possíveis seqüências de operações. A partir dessa árvore, os casos de teste são gerados de acordo com os critérios definidos nesses trabalhos.

Outros Trabalhos

Além dos trabalhos já apresentados, outros seis trabalhos foram selecionados por estarem relacionados a pelo menos uma das questões de pesquisa definidas. Alguns propõem estratégias de teste nas quais as técnicas tradicionais podem ser empregadas. Outros, por sua vez, têm como foco principal a caracterização de tipos de defeitos OA.

Lesiecki apresenta uma abordagem prática para o teste de unidade de software OA, considerando como unidade uma classe ou um aspecto (Lesiecki, 2005). A abordagem é baseada na linguagem AspectJ. Entretanto, o autor não emprega técnicas ou critérios de teste específicos. Um conjunto de pequenos padrões de teste é apresentado, envolvendo teste funcional, uso de ferramentas de visualização, transferência de lógica dos adendos para outras classes e uso de objetos que simulam as classes base, chamados objetos *mock*. De acordo com os padrões apresentados, a abordagem é direcionada para o teste de adendos e conjuntos de junção.

Alexander et al. discutem questões envolvidas no teste de software OA, apresentando quatro potenciais fontes de defeitos em um programa OA que já passou pelo processo de combinação (Alexander et al., 2004). Baseado nessas fontes de defeitos, os autores propõem uma taxonomia de defeitos candidata para programas OA, considerando também características específicas da linguagem AspectJ. Além disso, apresentam um critério de teste definido em alto nível, destinado a apoiar a identificação dos tipos de defeitos apresentados.

McEachen e Alexander, por sua vez, discutem possíveis tipos de defeitos decorrentes da combinação de aspectos em código reutilizado que já possui aspectos combinados, chamados aspectos estrangeiros (McEachen e Alexander, 2005). Consideram características específicas da linguagem AspectJ que permite, por exemplo, a combinação de aspectos com aplicações já compiladas. Os tipos de defeitos destacados estão principalmente relacionados à definição dos conjuntos de junção e à precedência de aspectos. Os autores fornecem algumas diretrizes sobre como proceder no caso de reutilização de código que possui aspectos estrangeiros.

Ainda no contexto de modelos de defeitos OA, Bækken e Alexander apresentam o início de um trabalho que visa a elaborar um modelo de defeitos para programas AspectJ (Bækken e Alexander, 2006). Nesse trabalho, o foco é na classificação de tipos de defeitos relacionados a conjuntos de junção. Quatro categorias de defeitos são identificadas. Além disso, um exemplo de tipo de defeito é descrito de acordo com um formato sugerido pelos autores, destacando-se as possíveis formas sintáticas nas quais o defeito pode aparecer em um programa e o impacto semântico que o defeito pode causar na execução do programa.

Ceccato et al. propõem algumas adaptações aos critérios de teste tradicionais de forma a cobrir os novos requisitos de teste introduzidos pela POA (Ceccato et al., 2005). Estendem a taxonomia de defeitos de Alexander et al. (2004), incluindo outros três tipos de defeitos, e avaliam o emprego dos critérios de teste no apoio à sua identificação. Propõem também outros dois critérios específicos para tratar de defeitos relacionados ao fluxo de controle de aplicações OA e precedência e aspectos. Além disso, apresentam uma estratégia incremental de teste similar a outras já observadas, iniciando-se pelo teste da aplicação base e incluindo-se incrementalmente os aspectos.

Zhou et al. apresentam uma abordagem de teste incremental, iniciando-se pelo teste da aplicação base sem aspectos, e testando-se os aspectos combinados incrementalmente até chegar ao teste da aplicação completa (Zhou et al., 2004). Um algoritmo para a identificação de testes relevantes para executar os aspectos é proposto. Definem ainda um critério de cobertura de teste para programas OA que requer a execução de todos os métodos interceptados por um determinado aspecto. Uma ferramenta que implementa o algoritmo de seleção de casos de teste e calcula a cobertura do teste também é sucintamente descrita.

3.2 Análise em Relação às Questões de Pesquisa

3.2.1 Técnicas e Critérios de Teste Explorados

Pode-se observar que todas as técnicas de teste abordadas nos trabalhos apresentados na Seção 3.1 são técnicas que já vinham sendo exploradas no contexto de outros paradigmas de desenvolvimento. Entretanto, diversos critérios de teste que envolvem características específicas de software OA são propostos. Nesta seção, cada um desses critérios é sucintamente apresentado, sendo organizados de acordo com o trabalho no qual são propostos.

Observa-se que boa parte dos critérios é derivada de critérios tradicionais, de acordo com a técnica de teste empregada. Esses critérios têm como base os artefatos e modelos subjacentes das referidas técnicas, e são definidos em diferentes níveis de granularidade.

Por exemplo, critérios baseados em fluxo de controle e fluxo de dados são derivados de grafos de fluxo de dados estendidos, que incorporam características específicas de programas OA. Outros critérios, por sua vez, são definidos em alto-nível, requerendo, por exemplo, uma execução de cada um dos adendos de um determinado aspecto para alcançar a cobertura de teste desejada.

Na Figura 3.1 são apresentados os trabalhos que definem critérios para o teste de software OA, de acordo com uma linha de tempo. A figura possibilita ao leitor identificar os primeiros critérios (e respectivas técnicas) investigados no contexto de software AO.

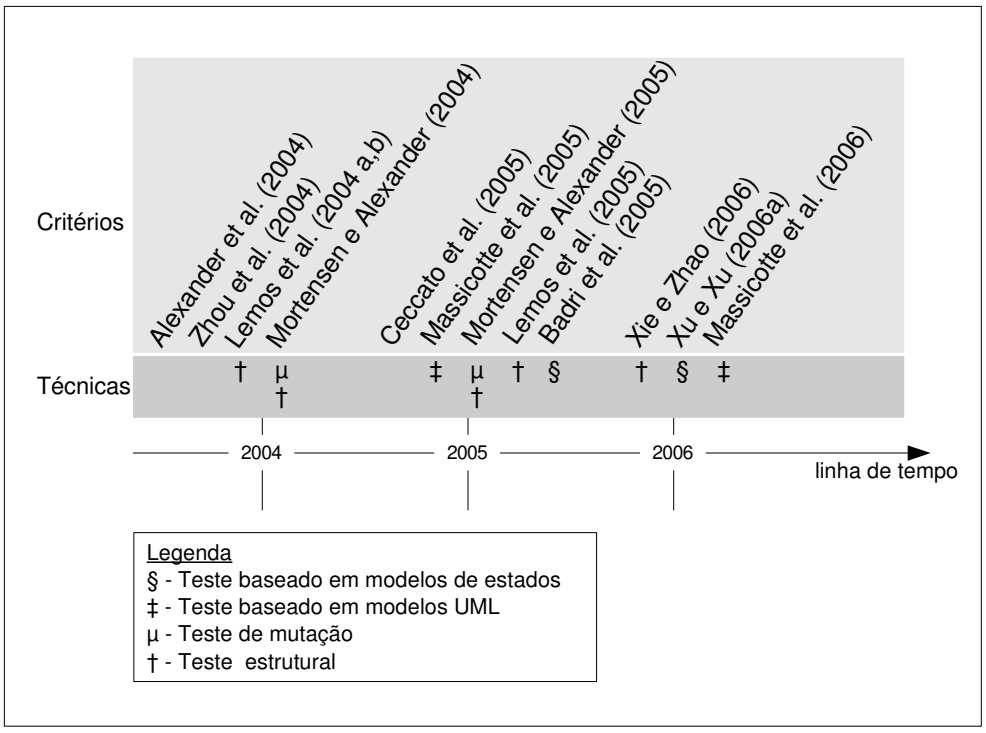

Figura 3.1: Linha de Tempo para as Propostas de Critérios de Teste de Software OA.

Observa-se ainda que a maior parte dos critérios não é formalmente definida, sendo que a descrição textual de um critério pode levar a interpretações imprecisas ou confusas. Nas apresentações a seguir, dentro do possível, procurou-se ser fiel às definições apresentadas pelos autores.

Critérios de Mortensen e Alexander (2004, 2005) para Teste Estrutural e de Mutação

Os autores definem três critérios para teste estrutural e adaptam o critério Análise de Mutantes para o contexto de software OA, apresentando um conjunto de três operadores de mutação.

Os seguintes critérios para teste estrutural foram definidos:

- Cobertura de comandos (*statement coverage*): similar ao critério *todos-nós* (Myers et al., 2004), requer que todos os comandos de um fragmento de código de um aspecto (por exemplo, um adendo ou um método introduzido) seja executado pelo menos uma vez.

- Cobertura de inserção (*insertion coverage*): requer o teste de cada fragmento de aspecto em todos os locais em que ele é combinado no código base. Por exemplo, um adendo devem ser testado junto com os todos os possíveis pontos de junção que ele pode afetar, e um atributo introduzido deve ser testado com cada método que o acesse ou modifique.

- Cobertura de contexto (*context coverage*): requer o teste de cada fragmento de aspecto em todos os locais em que ele é utilizado. Por exemplo, para um adendo que afeta um método, esse adendo deve ser testado em todos os locais onde o método é chamado.

- Cobertura de definições e usos (*def-use coverage*): similar aos critérios de teste intra e inter-módulo de fluxo de dados, requer o teste de fluxo de dados decorrente de definições e usos de atributos e variáveis que ocorrem intra ou inter adendos e métodos.

Os seguintes operadores de mutação foram definidos:

- PCS (*Pointcut Strengthening*): realiza alterações nos descritores de conjuntos de junção para reduzir a quantidade de pontos de junção capturados. Alguns exemplos de alterações realizadas são: a substituição de um padrão de casamento para nomes de métodos por um dos métodos em específico; e a substituição de um padrão de casamento de subtipos por um dos subtipos em específico.

- PCW (*Pointcut Weakening*): realiza alterações nos descritores de conjuntos de junção para para aumentar a quantidade de pontos de junção capturados. Um exemplo é a adição de caracteres curinga aos descritores de conjuntos de junção.

- PRC (*Precedence Changing*): realiza alterações nas declarações de ordem de precedência entre aspectos.

Critérios de Lemos et al. (2004a, 2005, 2004b) para Teste Estrutural

A partir dos grafos AOCFG (Lemos et al., 2004b), AODU (Lemos et al., 2005) e MADU (Lemos et al., 2004a), os autores definem um conjunto de critérios estruturais baseados em fluxo de controle e fluxo de dados para teste de unidade e de integração. Ressalta-se que os mesmos conceitos de pares definição-uso de variáveis apresentados por Rapps e Weyuker (1982), incluindo caminhos livres de definição, são válidos para esses critérios.

Para teste de unidade, os seguintes critérios foram definidos:

- Todos-nós-de-interceptação: Requer que cada nó transversal, e conseqüentemente cada execução de adendo que ocorre para a unidade afetada, seja exercitado pelo menos uma vez.

- Todas-arestas-de-interceptação: Requer que cada aresta do AOCFG que tem um nó transversal como início ou destino seja exercitada pelo menos uma vez.

- Todos-usos-transversais (*all-crosscutting-uses*): Requer que cada par def-uso no qual o uso está em um nó transversal seja exercitado pelo menos uma vez. A caracterização do uso de uma variável em um nó transversal dá-se pela passagem (e conseqüente captura) dessa variável para o adendo representado nesse nó.

Para teste de integração, os seguintes critérios foram definidos:

- Todos usos componente-aspecto (*all component-aspect uses*): para uma variável definida no escopo de um método, requer que todos os usos dessa variável que ocorrem no escopo dos adendos que afetam esse método sejam exercitados.

- Todos usos aspecto-componente (*all aspect-component uses*): para uma variável definida no escopo de um adendo, requer que todos os usos dessa variável que ocorrem no escopo do método, após o controle do fluxo de execução ter retornado para esse método, sejam exercitados.

- Todos usos aspecto-aspecto (*all aspect-aspect uses*): para uma variável definida no escopo de um adendo, requer que todos os usos dessa variável que ocorrem no escopo de outro adendo, após o controle do fluxo de execução ter sido transferido para esse outro adendo, sejam exercitados.

- Todos usos componente-componente (*all component-component uses*): para uma variável definida no escopo de um método, requer que todos os usos dessa variável que ocorrem no escopo desse método sejam exercitados.

Critérios de Xie e Zhao (2006) para Teste Estrutural

Os autores definem dois critérios para teste estrutural, sendo um mais refinado para cobrir certos caminhos do fluxo de controle de um método ou adendo e outro para cobrir possíveis chamadas de métodos:

- Cobertura de ramos aspectuais (*aspectual branch coverage*): é semelhante ao critério estrutural todas-arestas (Myers et al., 2004) e visa a executar todas as ramificações de fluxo de controle identificadas nos comportamentos implementados nos aspectos.

- Cobertura de interação (*interaction coverage*): requer o exercício das possíveis interações que ocorrem entre métodos afetados por algum adendo, chamados *advised methods*, e métodos dos aspectos, chamados *aspect methods*, e as interações entre métodos dos aspectos. Pode ser comparado a algum critério que gera requisitos que consistem em cobrir todos os nós de chamada de um grafo de chamadas de métodos.

Critério de Xu e Xu (2006a) para Teste Baseado em Modelos de Estados

Os autores definem um critério baseado em árvores de transições de estados, que são obtidas a partir dos modelos de estados que incluem os estados de um objeto somados aos estados resultantes da atuação dos aspectos. A definição do critério não é explícita no trabalho, mas o critério é adotado no exemplos apresentados.

- Cobertura de ramos da árvore de transições de estados derivada do modelo: requer que todos os ramos da árvore de transições de estados, tanto para seqüências de transições válidas quanto inválidas, sejam percorridos por algum caso de teste.

Critérios de Badri et al. (2005) para Teste Baseado em Modelos de Estados

Tendo como base o modelo de estados proposto no trabalho e a respectiva árvore de transições de estados, os autores definem um conjunto de critérios baseados em seqüências de transições. Para a aplicação desses critérios, os autores consideram que a aplicação

base já foi devidamente testada. Nesse sentido, alguns dos critérios requerem que determinadas seqüências de transições de estados sejam exercitadas novamente, conforme pode ser observado na descrição dos critérios a seguir:

- Critério de transição (*transition criterion*): requer que cada transição entre estados que é afetada por um aspecto seja exercitada no mínimo uma vez.
- Critério de seqüência (*sequence criterion*): requer que todas as seqüências de transições que são afetadas por um ou mais aspectos sejam exercitadas novamente.
- Critério de execução de adendo (*advice execution criterion*): quando uma transição é afetada por um adendo, requer que o conjunto das possíveis execuções do adendo em cada seqüência contendo essa transição seja exercitada no mínimo uma vez. Esse critério considera que informações de contexto podem influenciar nas possíveis execuções de um adendo.
- Critério de integração multi-aspecto (*multi-aspect integration criterion*): nos casos em que um método é afetado por vários aspectos, requer que as seqüências contendo chamadas a esse método sejam exercitadas novamente no mínimo uma vez, considerando todas as possíveis permutações de precedência entre os aspectos que interceptam as classes envolvidas.

Critérios de Massicotte et al. (2006, 2005) para Teste Baseado em Modelos UML

Os autores definem um conjunto de critérios de teste tendo como base os diagramas de colaborações entre objetos, considerando as influências dos aspectos nas colaborações. Os critérios são semelhantes aos apresentados por Badri et al. (2005). Os primeiros critérios são aplicados para o teste da aplicação sem considerar os aspectos que a afetam. Os demais, por sua vez, devem ser aplicados após a aplicação ter sido testada com os primeiros critérios.

Os seguintes critérios para teste somente da aplicação base foram definidos:

- Critério de cobertura de transição (*transition coverage criterion*): requer que cada transição do digrama seja exercitada no mínimo um vez.
- Critério de cobertura de seqüências (*sequence coverage criterion*): requer que todas as seqüências de transições válidas do diagrama sejam exercitadas no mínimo uma vez.

Os seguintes critérios para teste considerando a influência dos aspectos sobre a aplicação base foram definidos:

- Critério de cobertura de seqüências modificadas (*modified sequences coverage criterion*): requer que todas as seqüências que foram modificadas por algum aspecto sejam exercitadas novamente.
- Critério de cobertura para integração simples (*simple integration coverage criterion*): no caso em que um método afetado por um adendo é invocado em alguma colaboração do diagrama, requer que todas as seqüências que usam esse método sejam exercitadas novamente.
- Critério de cobertura para integração multi-aspectos (*multi-aspects integration coverage criterion*): no caso em que um método afetado por diversos adendos é invocado em alguma colaboração do diagrama, requer que todas as seqüências que usam esse método sejam exercitadas novamente, considerando todas as possíveis permutações de precedência entre os aspectos que interceptam as classes envolvidas.

Critérios de Alexander et al. (2004)

Os autores apresentam definições em alto nível de quatro critérios de teste, sem abordar uma técnica de teste em particular. Cada um dos critérios visa a tratar de tipos específicos de defeitos OA que foram caracterizados no trabalho.

- A identificação de defeitos relacionados ao escopo de conjuntos de junção requer o teste do aspecto em si.
- Para tratar de defeitos relacionados à precedência de aspectos, deve-se testar todas as possíveis ordens de combinação.
- Defeitos relacionados à violação de pós-condições, à violação de de invariantes e às dependências de controle requerem a reexecução dos testes que foram aplicados à aplicação base antes da combinação com os aspectos.
- Defeitos relacionados ao foco no fluxo de controle requerem o teste de cobertura de condições dos descritores de conjuntos de junção.

Critérios de Ceccato et al. (2005)

Os autores definem dois critérios de teste direcionados a tratar alguns tipos de defeitos OA caracterizados por Alexander et al. (2004). Os tipos de defeitos enfatizados estão relacionados à precedência entre aspectos e foco incorreto em fluxo de controle. Os critérios não são relacionados a alguma técnica de teste em particular, sendo assim definidos:

- Cobertura de descritores (*designator coverage*): requer no mínimo uma execução de todos os pontos de junção dinâmicos (dependentes de informação de contexto).
- Cobertura de composição (*composition coverage*): requer o teste de todas as ordens de precedência de aspectos que quando alteradas resultam em diferentes dependências de dados.

Critério de Zhou et al. (2004)

Os autores definem o seguinte critério, sem abordar uma técnica de teste em particular:

- Para um determinado aspecto, esse critério requer que todos os métodos das classes da aplicação que são afetados por algum adendo desse aspecto sejam executados pelo menos uma vez por algum caso de teste.

3.2.2 Estudos Experimentais Conduzidos

Estudos experimentais são importantes para a engenharia de software quando se busca obter resultados objetivos e significativos em relação à compreensão, controle, prognóstico e melhoria nos processos de software (Wohlin et al., 2000). No contexto da atividade de teste de software, estudos experimentais são importantes para comprovar a efetividade das abordagens em alcançar os objetivos propostos, ou seja, revelar defeitos presentes no software.

Segundo Wohlin et al., os estudos experimentais podem ser classificados em três tipos: (i) execução de pesquisa de opinião (do inglês, *survey*), na qual os dados qualitativos ou quantitativos são obtidos por meio da aplicação de questionários ou entrevistas a uma amostragem da população a ser estudada; (ii) estudos de caso, que são utilizados para monitorar projetos, atividades ou tarefas, por meio da observação de atributos específicos e suas relações, nos quais informações detalhadas são coletadas durante um período contínuo

de tempo de atividade; e (iii) experimentos controlados, que são conduzidos quando um controle da situação é necessário, podendo ser diretamente manipulados e sistematizados.

Os experimentos controlados são apropriados para investigar diferentes aspectos como, por exemplo, a confirmação de teorias, o teste de concepções e a avaliação da precisão de modelos. Dentre os três tipos de estudos experimentais, os experimentos controlados são os mais rigorosos, embora possam ser de alto custo e requerer grande esforço.

No contexto da revisão sistemática apresentada neste relatório, de acordo com as informações coletadas, nota-se que em poucos trabalhos foi realizado algum tipo de estudo experimental, todos limitados a estudos de caso. Os demais trabalhos utilizam exemplos para demonstrar a aplicação das abordagens, sem conduzir estudos mais detalhados para comprovar sua efetividade em revelar os tipos de defeitos OA caracterizados. Os estudos de caso conduzidos são sucintamente apresentados a seguir.

Estudo de caso de Xu e Xu (2006b) para Teste Baseado em Modelos de Estados

Apresentam um estudo de caso para ilustrar o emprego de sua abordagem para teste de aspectos de integração em uma aplicação AspectJ de simulação de telefonia, extraída do trabalho do AspectJ Team (2003). O estudo de caso envolveu o teste de um aspecto que faz a integração de uma classe base e uma classe integrada. Em particular, o aspecto insere um cronômetro na classe que representa uma conexão entre dois clientes, para fins de controle de tempo e faturamento. Os modelos de estados e as respectivas seqüências de transições de estados foram derivados do código das classes e do aspecto. A partir desses artefatos, os autores discutem como alguns tipos de defeitos OA podem ser revelados com o emprego da abordagem proposta.

Estudo de caso de Xie e Zhao (2006) para Teste Estrutural e Baseado em Modelos de Estados

Apresentam um estudo de caso com 12 programas AspectJ, obtidos a partir de fontes variadas. De acordo com critérios de cobertura definidos pelos autores, o estudo foi conduzido em três etapas, objetivando a comparação da cobertura de testes com e sem a utilização do *framework* de automatização apresentado no trabalho. Na primeira etapa, desconsiderou-se o uso das classes empacotadoras geradas pelo *framework*. As aplicações foram compiladas com o compilador ajc (AspectJ Team, 2005), e as classes combinadas foram submetidas para as ferramentas de geração de testes do *framework*. Na segunda etapa, foram empregadas as classes empacotadoras, e os demais passos da primeira etapa

foram repetidos. Por fim, na terceira etapa, aplicaram-se algumas diretrizes propostas para melhorar a cobertura obtida, com as classes empacotadoras novamente utilizadas. As coberturas obtidas nas três etapas do estudo foram relatadas para fins de comparação.

Estudo de caso de Mortensen e Alexander (2004, 2005) para Teste Estrutural e de Mutação

Apresentam um estudo de caso com quatro programas AspectJ, obtidos a partir de fontes variadas. Os autores aplicaram a abordagem de teste estrutural e de mutação proposta, derivando os requisitos de teste e implementando casos de teste para cobrir esses requisitos. Para cada um dos exemplos, foi fornecido um conjunto de testes inicial para exercitar as funcionalidades implementadas nas classes. Os autores mostram que esses testes iniciais não são suficientes para cobrir todos os requisitos derivados a partir dos critérios propostos nos trabalhos. Novos casos de teste foram criados para alcançar a cobertura esperada.

Estudo de caso de van Deursen et al. (2005) para Teste Estrutural e Funcional

Apresentam um estudo de caso de aplicação de sua estratégia de refatoração e teste em um *framework* para aplicações Java para desenhos 2D. O *framework*, intitulado JHotDraw (Gamma e Eggenschwiler, 2004), foi refatorado como o emprego de AspectJ, resultando no AJHotDraw. O JHotDraw é originalmente fornecido com um conjunto de classes de testes vazias, geradas automaticamente por uma ferramenta em particular. Os autores preencheram essas classes com testes funcionais, que foram também aplicados no AJHotDraw. Para cada um dos módulos refatorados, os autores criaram novos casos de teste para cobrir os requisitos de teste derivados de acordo com a estratégia proposta.

Estudo de caso de Zhou et al. (2004)

Apresentam um estudo de caso que utiliza uma aplicação AspectJ de processamento de contas bancárias, extraída do trabalho de Laddad (2003). Os autores apresentam alguns detalhes da aplicação, e em seguida aplicam a abordagem de teste incremental proposta no trabalho. Inicialmente criaram um conjunto de testes para as classes base da aplicação. Em seguida, com o apoio de uma ferramenta, evoluíram incrementalmente para o teste dos aspectos combinados com as classes-base, até testar a aplicação completa.

3.2.3 Tipos de Defeitos OA Caracterizados

A caracterização de tipos de defeitos específicos a uma abordagem ou tecnologia de desenvolvimento de software é um ponto fundamental para a definição ou adaptação de técnicas e critérios de teste. Conforme sugerido por Binder (1999) e Alexander et al. (2004), critérios e estratégias de teste devem ser investigados em termos de defeitos que reflitam as características comportamentais e estruturais dos programas envolvidos. Nesse contexto, uma das questões de pesquisa investigadas na revisão sistemática abordava a identificação dos tipos de defeitos OA caracterizados nos trabalhos selecionados.

Dentre os 25 trabalhos, em nove deles houve a preocupação de se caracterizar tipos de defeitos OA. Esses tipos de defeitos são apresentados nesta seção. Alguns deles são caracterizados em mais de um trabalho, embora a descrição apresentada possa variar de um trabalho para outro. Sendo assim, os tipos de defeitos estão agrupados de acordo com as características de software OA às quais eles estão relacionados. Para cada um deles, são citados os trabalhos que os caracterizam.

Defeitos Relacionados a Descritores de Conjuntos de Junção

Podem ser decorrentes, por exemplo, do uso incorreto de caracteres "coringa" (*wildcards*) ou da definição incorreta do padrão de casamento do descritor do conjunto de junção (por exemplo, na especificação do padrão do tipo, método, atributo ou do construtor). Os seguintes tipos de defeitos foram caracterizados:

- Seleção de um superconjunto de pontos de junção (Alexander et al., 2004; Bækken e Alexander, 2006; Lemos et al., 2006; McEachen e Alexander, 2005; van Deursen et al., 2005; Xu e Xu, 2006a,b);
- Seleção de um subconjunto de pontos de junção (Alexander et al., 2004; Bækken e Alexander, 2006; Lemos et al., 2006; van Deursen et al., 2005);
- Seleção de um conjunto incorreto de pontos de junção, contendo parte dos pontos de junção desejados e outros indesejados (Bækken e Alexander, 2006; Lemos et al., 2006; van Deursen et al., 2005);
- Seleção de um conjunto incorreto de pontos de junção, contendo somente pontos de junção indesejados (Bækken e Alexander, 2006; Lemos et al., 2006; van Deursen et al., 2005);

- Uso incorreto de descritor primitivo de conjunto de junção (por exemplo, trocar uma chamada pela execução de um método) (Bækken e Alexander, 2006; van Deursen et al., 2005);
- Lógica incorreta de composição de conjuntos de junção (Bækken e Alexander, 2006; Lemos et al., 2006; van Deursen et al., 2005);
- Implementação incorreta de conjuntos de junção que capturam lançamento de exceções (Bækken e Alexander, 2006; Massicotte et al., 2006).
- Padrão de casamento baseado em circunstâncias dinâmicas incorreto (por exemplo, padrões que dependem de informações referentes a fluxo de execução ou valores de argumentos) (Alexander et al., 2004; Bækken e Alexander, 2006).

Defeitos Relacionados a Declarações Inter-tipos ou outras Declarações

Em geral, são decorrentes de erros na especificação dos locais (pacotes e classes) nos quais métodos e atributos serão inseridos ou da falta de conhecimento do desenvolvedor a respeito da hierarquia de classes da aplicação base. Podem decorrer também de declarações que alteram a severidade de exceções lançadas ou de precedência entre aspectos. Os seguintes tipos de defeitos foram caracterizados:

- Introdução de método com nome incorreto, resultando em sobrescrita de método não prevista ou não resultando em uma sobreposição esperada (van Deursen et al., 2005);
- Introdução de membro (método ou atributo) em classe incorreta, resultando em introdução em um local incorreto na hierarquia de classes (van Deursen et al., 2005);
- Alteração incorreta da hierarquia de classes, resultando em superclasse indesejada para uma dada classe (Ceccato et al., 2005; van Deursen et al., 2005);
- Introdução incorreta de método que sobrescreve outro método (Ceccato et al., 2005; van Deursen et al., 2005);
- Omissão de declaração de implementação de interface, resultando em um método que opera por si só (van Deursen et al., 2005);
- Mudanças incorretas em fluxo de controle dependente de exceção, decorrentes de cláusulas que alteram a severidade de uma exceção (por exemplo, cláusula `declare soft` de AspectJ) (Ceccato et al., 2005).

- Declaração incorreta ou omissão de declaração de precedência de aspectos (Alexander et al., 2004; McEachen e Alexander, 2005; van Deursen et al., 2005)

Defeitos Relacionados à Definição e Implementação de Adendos

Podem decorrer de interpretação incorreta de um ou mais requisitos ou de erros de definição do tipo do adendo que será executado nos pontos de junção alcançados. Os seguintes tipos de defeitos foram caracterizados:

- Especificação do tipo adendo incorreta (por exemplo, um adendo que executa antes de um ponto de junção ao invés de executar após o ponto de junção) (van Deursen et al., 2005; Xu e Xu, 2006a,b)

- Alteração incorreta do fluxo de controle da aplicação devido à invocação incorreta do comando que aciona a execução do ponto de junção corrente (por exemplo, comando `proceed` de AspectJ) (Alexander et al., 2004; van Deursen et al., 2005; Xu e Xu, 2006a,b)

- Lógica do adendo em desacordo com a especificação, resultando em violação de invariantes ou pós-condições (Alexander et al., 2004; Massicotte et al., 2006; van Deursen et al., 2005; Xu e Xu, 2006a,b)

Defeitos Relacionados ao Código Base da Aplicação

Estão relacionados à falta de conhecimento sobre a implementação dos aspectos que serão reutilizados para serem combinados em uma nova aplicação. O seguinte tipo de defeitos foi caracterizado:

- Código base não oferece os pontos de junção nos quais um ou mais aspectos estrangeiros deveriam atuar quando combinados com esse código base (McEachen e Alexander, 2005);

Nota-se que alguns tipos de defeitos estão diretamente relacionados. Por exemplo, defeitos resultantes de um erro na definição da lógica de composição de conjuntos de junção podem resultar em superconjuntos ou subconjuntos de pontos de junção capturados. Entretanto, para simplificar a descrição de cada tipo de defeitos caracterizado, optou-se por apresentá-los separadamente.

Capítulo

4

Conclusão e Trabalhos Futuros

As conclusões obtidas com a realização da revisão sistemática apresentada neste relatório podem ser exploradas sob duas perspectivas. A primeira em relação à condução da revisão em si e as peculiaridades dos mecanismos de busca, incluindo as dificuldades encontradas para realizar as buscas e a estratégia adotada para contornar essas dificuldades. A segunda perspectiva refere-se aos resultados obtidos após a leitura dos trabalhos selecionados e a extração das informações de interesse para as questões de pesquisa.

A principal dificuldade enfrentada durante a condução da revisão está relacionada aos mecanismos de busca atualmente disponíveis. As opções de busca de cada máquina de busca variam significativamente e o modo como as *strings* devem ser construídas é particular para cada máquina. Os tipos de publicações são restritos nos repositórios indexados, e diversos trabalhos muito citados (relatórios técnicos e artigos apresentados em *workshops*, por exemplo) não aparecem nessas bases. Dessa forma, máquinas de busca convencionais tiveram de ser incluídas entre os repositórios para possibilitar a recuperação desses trabalhos.

Em relação à estratégia de busca adotada, as citações observadas nos trabalhos selecionados constituem indícios de que as buscas tiveram a amplitude desejada. Todas as referências citadas nos trabalhos selecionados foram recuperadas com as *strings* de busca construídas. Observou-se também que alguns trabalhos são citados somente pelo próprio grupo de pesquisas que os publicaram. Esses trabalhos, classificados como relevantes para

os objetivos da revisão apresentada neste relatório, não são citados em boa parte dos demais, indicando que a existência de critérios sistemáticos para a inclusão e exclusão de trabalhos é efetivamente importante para evitar viés na revisão.

Ainda na perspectiva da condução da revisão e das peculiaridades dos mecanismos de busca, pôde-se observar que o trabalho de Alexander et al. (2004) foi um dos mais citados dentre os trabalhos selecionados. Nota-se, portanto, que a decisão de incluir repositórios não indexados pode ser importante para uma revisão sistemática, na qual pretende-se recuperar trabalhos de reconhecida relevância para uma pesquisa específica. Trata-se de um relatório técnico que não foi publicado em repositórios indexados como IEEE e ACM, mas que após disponibilizado tem sido citado por quase todos os trabalhos relacionados ao teste de software OA.

Em relação aos resultados obtidos na revisão, observa-se que todos os trabalhos analisados que abordam a aplicação de técnicas de teste procuram adaptar as técnicas tradicionais para o contexto de software OA. Nesse sentido, diversos critérios têm sido propostos para tratar de características particulares de software OA. Outros trabalhos, entretanto, concentram-se em propor estratégias de teste na qual o testador pode adotar a técnica de teste mais conveniente.

Embora em diversos trabalhos houve a preocupação com a caracterização de tipos de defeitos OA, os estudos de caso sucintamente apresentados na Seção 3.2.2 indicam que poucos trabalhos realizam algum tipo de estudo experimental para comprovar a efetividade das técnicas em revelar esses tipos de defeitos. Os demais trabalhos utilizam exemplos para demonstrar a aplicação das abordagens, sem conduzir estudos mais detalhados para comprovar sua efetividade.

Em relação às características particulares de software OA enfatizadas nas abordagens analisadas, observa-se que as propostas em geral são restritas a um subconjunto de características. Por exemplo, enquanto algumas são direcionadas ao teste da integração de adendos e métodos, outras são direcionadas para o teste de descritores de conjuntos de junção ou de comportamentos que concorrem por pontos de junção em comum.

Em relação às tecnologias de implementação, a maioria das abordagens de teste de software OA é baseada em características da linguagem AspectJ. Algumas abordagens utilizam os *bytecodes* Java obtidos após a combinação dos aspectos com a aplicação base utilizando um compilador AspectJ. Outras tratam de elementos particulares da linguagem como, por exemplo, contextos de fluxo de execução ou a exposição de informações de contexto possibilitada pela linguagem. Entretanto, pode-se observar que a maior parte dos conceitos presentes nessas abordagens pode ser aplicada no teste de software OA em geral, independentemente da tecnologia de implementação adotada. Essa possibilidade é

motivada por duas razões principais: (i) boa parte das iniciativas de implementação de tecnologias de apoio à POA é baseada na características da linguagem AspectJ, podendo-se citar como exemplo as linguagens AspectC (Coady et al., 2001) e AspectC++ (Coady et al., 2001); e (ii) de forma geral, as tecnologias de apoio implementam os elementos básicos da POA, fornecendo um meio de modularizar comportamentos transversais e implementando um mecanismo de quantificação, que permite definir os pontos da execução do software em que esses comportamentos serão executados (Filman e Friedman, 2000).

4.1 Trabalhos Futuros

As lições aprendidas com a condução da revisão sistemática apresentada neste relatório, incluindo os desafios e dificuldades para planejar uma atividade desse tipo, podem ser aproveitadas para novas revisões sistemáticas que venham a ser conduzidas, visto que essa é uma tendência para atividades de pesquisa bibliográfica para trabalhos acadêmicos e científicos na área de Engenharia de Software.

Dentre as principais direções para pesquisas na área de teste de software, Harrold (2000) destaca a importância da condução de estudos teóricos e experimentais para demonstrar a eficácia das técnicas de teste, e também a importância do desenvolvimento de processos, métodos e ferramentas para que os resultados obtidos possam ser empregados na prática. Além disso, ferramentas de apoio também são de fundamental importância para a condução de estudos experimentais que permitam a avaliação de custo e esforço para a aplicação das abordagens propostas.

Nesse contexto, os resultados obtidos nesta revisão, de acordo com os objetivos e questões de pesquisa propostos, servirão como base para a realização de novos trabalhos relacionados ao teste de software OA. Esses trabalhos incluem a definição de critérios de teste, a implementação de mecanismos de apoio automatizado à aplicação desses critérios, e a realização de estudos experimentais para verificar a eficácia dos critérios investigados em revelar defeitos.

Referências

Alexander, R. T.; Bieman, J. M.; Andrews, A. A. *Towards the systematic testing of aspect-oriented programs.* Tech. Report CS-04-105, Dept. of Computer Science, Colorado State University, Fort Collins/Colorado - USA, 2004.

AspectJ Team The AspectJ programming guide. Online, disponível em `http://www.eclipse.org/aspectj/doc/released/progguide/index.html` - último acesso em 20/01/2006, 2003.

AspectJ Team The AspectJ development environment guide. Online, disponível em `http://www.eclipse.org/aspectj/doc/released/devguide/index.html` - último acesso em 27/09/2006, 2005.

Badri, M.; Badri, L.; Bourque-Fortin, M. Generating unit test sequences for aspect-oriented programs: Towards a formal approach using uml state diagrams. In: *Enabling Technologies for the New Knowledge Society: Proceedings of the ITI 3rd International Conference on Information & Communications Technology (ICICT'2005)*, Cairo - Egypt: IEEE Computer Society, 2005, p. 237–253.

Bækken, J. S.; Alexander, R. T. Towards a fault model for aspectj programs: Step 1 pointcut faults. In: *Proceedings of the 2nd Workshop on Testing Aspect Oriented Programs (WTAOP'2006) – held in conjunction with ISSTA'2006*, 2006, p. 1–6.

Binder, R. V. *Testing object-oriented systems: Models, patterns, and tools.* 1st. ed. Addison Wesley, 1999.

Biolchini, J.; Mian, P. G.; Natali, A. C. C.; Travassos, G. H. *Systematic review in software engineering.* Tech. Report RT-ES 679/05, Systems Engineering and Computer Science Dept., COPPE/UFRJ, Rio de Janeiro/RJ - Brazil, 2005.

Ceccato, M.; Tonella, P.; Ricca, F. Is AOP code easier or harder to test than OOP code? In: *Proceedings of the 1st Workshop on Testing Aspect Oriented Programs (WTAOP'2005) – held in conjunction with AOSD'2005*, Chicago/IL - USA, 2005.

Coady, Y.; Kiczales, G.; Feeley, M.; Smolyn, G. Using AspectC to improve the modularity of path-specific customization in operating system code. In: *Proceedings of 9th ACM SIGSOFT Internation Symposium on the Foundations of Software Engineering (FSE-9)*, 2001.

Ferrari, F. C.; Maldonado, J. C. Uma revisão sistemática sobre teste de software orientado a aspectos. In: *Anais do 3º Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos (WASP'2006)*, Florianópolis/SC - Brasil, 2006, p. 101–110.

Filman, R.; Friedman, D. Aspect-oriented programming is quantification and obliviousness. In: *Workshop on Advanced Separation of Concerns, OOPSLA 2000*, Minneapolis - USA, 2000, p. 21–35.

Gamma, E.; Eggenschwiler, T. JHotDraw as open-source project. Online, disponível em `http://www.jhotdraw.org/` - último acesso em 28/09/2006, 2004.

Harrold, M. J. Testing: A roadmap. In: *22th International Conference on Software Engineering - Future of SE Track*, ACM Press, 2000, p. 61–72.

Kiczales, G.; Irwin, J.; Lamping, J.; Loingtier, J.-M.; Lopes, C.; Maeda, C.; Menhdhekar, A. Aspect-oriented programming. In: Akşit, M.; Matsuoka, S., eds. *Proceedings of the 11th European Conference on Object-Oriented Programming*, Jyväskylä - Finland: Springer-Verlag, 1997, p. 220–242 (Lectures Notes in Computer Science, v.1241).

Kitchenham, B. *Procedures for performing systematic reviews.* Joint Technical Report TR/SE-0401 (Keele) – 0400011T.1 (NICTA), Software Engineering Group - Department of Computer Science - Keele University and Empirical Software Engineering - National ICT Australia Ltd, Keele/Staffs-UK and Eversleigh-Australia, 2004.

Laddad, R. *AspectJ in action.* Manning Publications, 2003.

Lemos, O. A. L.; Ferrari, F. C.; Masiero, P. C.; Lopes, C. V. Testing aspect-oriented programming pointcut descriptors. In: *Proceedings of the 2nd Workshop on Testing*

Aspect Oriented Programs (WTAOP'2006) – held in conjunction with ISSTA'2006, Portland/Maine - USA: ACM Press, 2006, p. 33–38.

Lemos, O. A. L.; Maldonado, J. C.; Masiero, P. C. Data flow integration testing criteria for aspect-oriented programs. In: *Anais do 1º Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos (WASP'2004)*, Brasília/DF - Brasil, 2004a.

Lemos, O. A. L.; Maldonado, J. C.; Masiero, P. C. Structural unit testing of AspectJ programs. In: *Proceedings of the 1st Workshop on Testing Aspect Oriented Programs (WTAOP'2005) – held in conjunction with AOSD'2005*, Chicago/IL - USA, 2005.

Lemos, O. A. L.; Vincenzi, A. M. R.; Maldonado, J. C.; Masiero, P. C. Teste de unidade de programas orientados a aspectos. In: *Anais do 18th Simpósio Brasileiro de Engenharia de Software (SBES'2004)*, Brasília/DF - Brasil, 2004b, p. 55–70.

Lesiecki, N. Unit test your aspects. *IBM developerWorks Homepage*, disponível em `http://www-128.ibm.com/developerworks/java/library/j-aopwork11/` - último acesso em 03/08/2006, 2005.

Massicotte, P.; Badri, L.; Badri, M. Aspects-classes integration testing strategy: An incremental approach. In: *Proceedings of the 2nd International Workshop on Rapid Integration of Software Engineering Techniques (RISE'2005) - Revised Selected Paper*, Heraklion/Crete - Greece: Springer Berlin, 2006, p. 158–173 (Lectures Notes in Computer Science, v.3943).

Massicotte, P.; Badri, M.; Badri, L. Generating aspects-classes integration testing sequences: A collaboration diagram based strategy. In: *Proceedings of the 3rd International Conference on Software Engineering Research, Management and Applications (ACIS'2005)*, Mt. Pleasant/MI - USA: IEEE Computer Society, 2005, p. 30–37.

McEachen, N.; Alexander, R. T. Distributing classes with woven concerns: An exploration of potential fault scenarios. In: *Proceedings of the 4th International Conference on Aspect-oriented Software Development (AOSD'2005)*, Chicago/IL - USA: ACM Press, 2005, p. 192–200.

Mortensen, M.; Alexander, R. T. *Adequate testing of aspect-oriented programs.* Technical Report CS 01-110, Department of Computer Science, Colorado State University, Fort Collins/Colorado - USA, 2004.

Mortensen, M.; Alexander, R. T. An approach for adequate testing of aspectj programs. In: *Proceedings of the 1st Workshop on Testing Aspect Oriented Programs (WTAOP'2005) – held in conjunction with AOSD'2005*, Chicago/IL - USA, 2005.

Myers, G. J.; Sandler, C.; Badgett, T.; Thomas, T. M. *The art of software testing.* 2nd. ed. John Wiley & Sons, 2004.

Naqvi, S. A. A.; Ali, S.; Khan, M. U. An evaluation of aspect oriented testing techniques. In: *Proceedings of the IEEE Symposium on Emerging Technologies*, Islamabad - Pakistan: ACM Press, 2006, p. 461–466.

Rapps, S.; Weyuker, E. J. Data flow analysis techniques for program test data selection. In: *Proceedings of the 6th International Conference on Software Engineering (ICSE'1982)*, Tokio - Japan: IEEE Computer Society Press, 1982, p. 272–278.

van Deursen, A.; Marin, M.; Moonen, L. *A systematic aspect-oriented refactoring and testing strategy, and its application to JHotDraw.* Tech.Report SEN-R0507, Stichting Centrum voor Wiskundeen Informatica, Amsterdam - The Netherlands, 2005.

Vincenzi, A. M. R. *Orientação a objeto: Definição, implementação e análise de recursos de teste e validação.* Tese de Doutoramento, ICMC/USP, São Carlos, SP - Brasil, 2004.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B.; Wesslén, A. *Experimentation in Software Engineering: an Introduction.* Kluwer Academic Publishers, 2000.

Xie, T.; Zhao, J. A framework and tool supports for generating test inputs of AspectJ programs. In: *Proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD'2006)*, Bonn - Germany: ACM Press, 2006, p. 190–201.

Xie, T.; Zhao, J.; Marinov, D.; Notkin, D. Automated test generation for AspectJ programs. In: *Proceedings of the 1st Workshop on Testing Aspect Oriented Programs (WTAOP'2005) – held in conjunction with AOSD'2005*, Chicago/IL - USA, 2005.

Xu, D.; Xu, W. State-based incremental testing of aspect-oriented programs. In: *Proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD'2006)*, Bonn - Germany: ACM Press, 2006a, p. 180–189.

Xu, D.; Xu, W.; Nygard, K. *A state-based approach to testing aspect-oriented programs.* Technical Report NDSU-CS-TR04-XU03, Department of Computer Science, North Dakota State University, Fargo/ND - USA, 2004.

Xu, D.; Xu, W.; Nygard, K. A state-based approach to testing aspect-oriented programs. In: *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'2005)*, Taiwan, 2005.

Xu, W.; Xu, D. A model-based approach to test generation for aspect-oriented programs. In: *Proceedings of the 1st Workshop on Testing Aspect Oriented Programs (WTAOP'2005) – held in conjunction with AOSD'2005*, Chicago/IL - USA, 2005.

Xu, W.; Xu, D. State-based testing of integration aspects. In: *Proceedings of the 2nd Workshop on Testing Aspect Oriented Programs (WTAOP'2006) – held in conjunction with ISSTA'2006*, Portland/Maine - USA: ACM Press, 2006b, p. 7–14.

Zhao, J. Tool support for unit testing of aspect-oriented software. In: *Workshop on Tools for Aspect-Oriented Software Development - held in conjunction with OOPSLA'2002*, Seattle/WA - USA, 2002.

Zhao, J. Data-flow-based unit testing of aspect-oriented programs. In: *Proceedings of the 27th Annual IEEE International Computer Software and Applications Conference (COMPSAC'2003)*, Dallas/Texas - USA: IEEE Computer Society, 2003, p. 188–197.

Zhou, Y.; Richardson, D. J.; Ziv, H. Towards a practical approach to test aspect-oriented software. In: *Proceedings of the Net.ObjectiveDays 2004 Workshop on Testing Component-based Systems (TECOS'2004)*, Germany, 2004, p. 1–16.

Apêndice

A

Exemplo de Formulário de Extração de Informações

Título do Trabalho	
Fonte	

Entrada BibTeX	

Resumo do Revisor

Técnica de Teste

Critérios de Teste

Estudo de Caso

Defeitos OA

Figura A.1: Exemplo de formulário de extração em branco.

Título do Trabalho	State-Based Testing of Integration Aspects
Fonte: ACM	

Entrada BibTeX

```
@inproceedings{Xu2006b,
  author =       {Weifeng Xu and Dianxiang Xu},
  title =        {State-Based Testing of Integration Aspects},
  booktitle =    {Proceedings of the $2^{nd}$ Workshop on Testing Aspect Oriented Programs
(WTAOP'2006) -- held in conjunction with the International Symposium on Software Testing and
Analysis (ISSTA'2006)},
  year =         {2006},
  pages =        {7--14},
  address =      {Portland/Maine - USA},
  publisher =    {ACM Press}
 }
```

Resumo do Revisor

Apresentam uma abordagem para teste baseada em modelos de estado cujo foco é no teste de aspectos que fazem a integração de uma classe base com classes utilizadas, por exemplo, em declarações inter-tipos. Essas últimas são classificadas como classes integradas, e o aspecto que realiza a introdução é classificado como aspecto de integração. O modelo de estados apresentado por Xu e Xu (2006a) é estendido de forma a representar os estados de mais de uma classe simultaneamente.
Um novo modelo de estados, o modelo de aspectos, é definido para representar como as classes integradas são utilizadas em conjunto com as classes base. Um modelo de aspectos é formado por um ou mais modelos de adendos, que representam os estados e os eventos das classes integradas.
A combinação dos modelos de estados dos aspectos e da classe base resulta em um modelo que representa todas as transições de estado envolvendo a classe base e as classes integradas.
Tendo como base esses modelos, uma estratégia de teste incremental é proposta, consistindo inicialmente em testar a classe base sem considerar os aspectos. Em uma segunda etapa, faz-se a combinação dos modelos (classe base e aspectos) e em seguida pode-se criar os testes considerando o modelo completo.
Para a criação dos testes, uma árvore de transições de estados é derivada do modelo de estados. Critérios de cobertura de caminhos da árvore (por exemplo, cobertura de ramos) podem ser adotados para derivar os requisitos de teste.

Técnica de Teste

Teste baseado em modelos de estados.

Critérios de Teste

Não definem.

Estudo de Caso

Apresentam um estudo de caso, no qual aplicam a estratégia proposta em uma aplicação de simulação de telefonia [The AspectJ Team 2003]. Nesse caso, uma classe base e uma classe integrada são utilizadas por um aspecto de integração.

Defeitos OA

Enfatizam quatro tipos de defeitos:

- Conjunto de junção selecionando pontos de junção a mais;
- Definição incorreta do conjunto de junção. Por exemplo, o padrão (pattern) do conjunto de junção não possibilita a captura do conjunto correto;
- Tipo de adendo incorreto. Por exemplo, um adendo que deveria executar antes de um ponto de junção é executado após esse ponto;
- Lógica do adendo incorreta (defeito intra-adendo).

Figura A.2: Exemplo de formulário de extração preenchido.