5
Trabalhos Relacionados

Este capítulo busca apresentar os trabalhos diretamente relacionados com a presente dissertação e as subseções seguintes apresentam os trabalhos focados em ferramentas (Seção 5.1) e estudos empíricos (Seção 5.2).

5.1.Ferramentas para análise estática

Atualmente, existem algumas ferramentas que tem por finalidade abordar os problemas relacionados à manipulação de exceção no código de programação em várias linguagens. Contudo, o escopo desta dissertação está direcionado à linguagem Java e, por isso, analisaremos algumas soluções que foram propostas para análise estática a fim de apoiar o raciocínio sobre o fluxo de exceções em programas Java.

Robillard e Murphy (2003) desenvolveram a ferramenta Jex, que analisa o fluxo de exceções em programas Java e utiliza o código fonte do programa para realizar a análise estática, com o objetivo de encontrar os fluxos de exceção, tanto de exceções checadas quanto de não-checadas, e eventuais falhas que exceções que não foram capturadas e exceções capturadas por subsunção podem representar. A ferramenta *Jex* ainda utiliza o *Class Hierarchy Algorithm* (CHA) para construir o gráfico de chamadas do programa percorrido durante a análise produzindo um grafo de chamadas menos preciso do que as análises sensíveis ao contexto, como utilizamos em nossa ferramenta. Em contrapartida, Jo *et alii* (2004) desenvolveu uma ferramenta para analisar programas de Java com o objetivo de detectar a lista de exceções checadas associados a cada método, e os manipuladores de exceção desnecessários. No entanto, esta abordagem não considera as exceções não-checadas.

Fu e Ryder (2005) propuseram uma ferramenta de análise estática para a linguagem Java construída a partir do *framework* de análise de *bytecode* SOOT e do *framework* para construção de grafos de chamadas, o SPARK. Esta ferramenta

gera os caminhos de exceção para exceções específicas (por exemplo, `java.io.IOException`) que podem ocorrer no sistema.

Em Fu et al. (2007) foi realizada a extensão da ferramenta proposta em (FU *et alii*, 2005; FU e RYDER, 2005) a fim de calcular cadeias de exceções, isto é, uma sequência de caminhos de exceção semanticamente relacionados. Assim, a cadeia de exceções é capaz de representar, em um único fluxo de exceção, o caminho de uma exceção que é encapsulada em outra ou que tenha sido relançada – em vez de representar apenas segmentos de cada exceção. Desta forma, foi construído um algoritmo de análise estática que analisa o corpo que manipula exceção (cláusula *catch*) ligando cada exceção à sua causa e, após isso, os caminhos exceção são calculados. Analisar o encadeamento de exceções é relevante, pois este tipo de análise captura a natureza do tratamento de exceção em aplicações estruturadas em camadas e componentes, uma vez que as exceções que são lançadas podem ser capturadas e encapsuladas em outras a fim de manter a separação de camadas e o desacoplamento de componentes. A abordagem de encasular exceções também é, frequentemente, usada em implementações do padrão *Adapter* (GOF, 1995) a fim de manter a aplicação independente do componente adaptado. Já na nossa ferramenta, o encadeamento de exceções é uma das abordagens que adotamos para a análise do fluxo de exceções.

Em Coelho (2008) é introduzida uma nova forma para analisar fluxos de exceção, ela consiste em analisar o fluxo das exceções tanto em código orientado a objetos quanto em código orientado a aspectos, neste caso a implementação do framework de orientação a aspectos usada foi AspectJ (ver detalhes na Seção 2.4).

Apesar de nossa ferramenta utilizar como base a ferramenta *SAFE*, adotamos algumas decisões de projeto distintas em relação a *SAFE*, sendo (i) a análise de programas que utilizam o *JBoss AOP* ao invés de *AspectJ;* (ii) a análise a aplicação de forma semelhante à sua execução (ver Seção 3.1.1); (iii) a análise da cadeia de exceção (ver Seção 3.1.3); e (iv) a utilização de técnicas de programação dinâmica (ver Seção 3.1.4), a fim de evitar repetidas análises dos mesmos métodos em um fluxo de exceção, permitindo, assim, analisar a aplicação e suas bibliotecas.

5.2.Estudos Empíricos

No contexto de desenvolvimento de software orientada a aspectos, alguns estudos empíricos foram realizados para investigar a utilização de aspectos que visavam modularizar o código de manipulação de exceção (LIPPERT E LOPES, 2000; FILHO *et alii*, 2006; FILHO *et alii*, 2007).

Lippert e Lopes (2000) realizaram um estudo para investigar o uso de construções orientadas a aspectos a fim de modularizar a manipulação de exceção do código orientado a objeto no *framework JWAM* e tinha como objetivo avaliar a utilidade de aspectos para separar o código de manipulação de exceção a partir do código orientado a objetos. Além disso, os autores observaram que, quando o aplicativo adota uma política de tratamento de erro, o uso de aspectos para modularizar a detecção e tratamento de exceção traz vários benefícios, tais como a melhoria na reutilização de código e uma consequente diminuição do número de linhas de código. Juntamente a essas observações, obtiveram, também, uma grande redução na quantidade de código para tratamento de exceção na aplicação, passando de 11% do código da versão orientada a objetos a 2,9% na versão orientada a aspectos.

Castor Filho *et alii* (2006) realizaram um estudo similar, cujo objetivo era compreender os benefícios e limitações do uso de aspectos para modularizar o código de tratamento de exceções em cenários realistas. Nesse estudo, os autores modificaram as aplicações analisadas para que o tratamento de exceção fosse realizado por aspectos. Concomitante a isso, o estudo em destaque revelou que o processo de transformação do tratamento de exceção em aspectos e reutilização de manipuladores de exceção não é tão simples e depende de um conjunto de fatores que engloba (i) o tipo de exceções a ser tratada; (ii) o que o manipulador faz; (iii) a quantidade de informações contextuais necessárias; (iv) o que o método que levanta a exceção retorna; e (v) o que a cláusula *throws* especifica.

Coelho (2008), por sua vez, realizou um estudo empírico de três aplicações – sendo que duas das três foram analisadas em mais de uma versão – que analisam as consequências das exceções lançadas por aspectos e a propensão ao erro de construções AspectJ para tratamento de exceções. A autora ainda chegou à conclusão de que a adição de aspectos a aplicações tem uma tendência a apresentar mais cenários sujeitos a falhas, pois a mesma observou um aumento

substancial nas exceções não-capturadas e nas capturadas por subsunção nas versões orientadas a aspectos das aplicações analisadas.

O nosso estudo é semelhante ao realizado por Coelho (2008), de forma que foram analisadas três aplicações que utilizam *JBoss AOP* através do *framework EJB* ao invés do AspectJ, e além disso adicionamos ao estudo as bibliotecas utilizadas tanto pela aplicação quanto pelos aspectos do *EJB*. Nosso estudo também leva em consideração as cadeias de exceções que podem se formar durante o fluxo de uma exceção. Além disto, vale salientar algumas diferenças importantes com relação aos estudos:

Desenvolvimento dos Aspectos **–** nas aplicações estudadas por Coelho (2008), os aspectos foram desenhados para interceptar determinados pontos da aplicação e o código que o aspecto adiciona foi construido especificamente para as respectivas aplicações. Enquanto nos nossos experimentos, os aspectos foram desenvolvidos para um framework e as aplicações se beneficiam de tal framework para implementar, por exemplo, o gerenciamento de transações. Isto impacta diretamente em como os códigos dos aspectos se interlaçam com os códigos da aplicação. Nos estudos realizados por Coelho, os desenvolvedores do código objeto não tinham conhecimento de quais e nem onde os aspectos inteceptariam o código. Nos nossos estudos, os desenvolvedores da aplicação sabem com precisão quais são os pontos que os aspectos irão interceptar, mesmo que o desenvolvedor não conheça com precisão quais serão os aspectos que interceptam o código. Desta forma, eles podem adicionar códigos para o tratamento de exceções nos pontos onde a aplicação invoca métodos que serão interceptados pelos aspectos do *framework EJB*.

Joinpoints Interceptados **–** enquanto nas aplicações que nós estudamos os joinpoints são pontos bem definidos em relação à arquitura da aplicação (ver Seção 4.1.1), nas aplicações analisadas por Coelho (2008) os aspectos interceptam componentes de várias camadas. Desta forma, a adição de novas exceções oriundas de aspectos impacta a arquitetura da aplicação em vários pontos, e assim, tornando o tratamento de exceção relativo aos aspectos mais difícil.

Tratamento das Exceções pela Aplicação – na maioria das aplicações analisadas por Coelho (2008) as exceções foram capturadas por um bloco de tratamento de exceção específico. Enquanto, nas aplicações que analisamos 99% dos fluxos de exceção capturadas são realizadas por subsunção (ver Seção 4.3.1).

Interesses Transversais – outro aspecto importante para entendermos a relação do estudo de Coelho (2008) com o nosso são os interesses transversais, os quais têm papel determinante nos fluxos de exceção. Sendo assim, na Tabela 18 é apresentada os interesses transversais estudados por Coelho enquanto os interesses que nós estudamos estão na Tabela 9.

Tabela 18 – Interesses transversais das aplicações analisadas por Coelho (2008).

Aplicação	Interesses
Health Watcher	Persistência, concorrência, gerenciamento de transações, tratamento de exceções;
Móbile Photo	Tratamento de exceções, envio de SMS, ordenação de listas e selecionar favoritos;
AJHotDraw	Persistência, políticas de design e a funcionalidade desfazer;

5.3.Resumo

Neste capítulo, foram apresentados os trabalhos diretamente relacionados à presente dissertação, além de terem sido apresentadas, também, várias ferramentas e técnicas de análise estática que tem por objetivo analisar o tratamento de exceção. Em seguida, foi apontado um conjunto de estudos empíricos realizados com o intuito de mensurar o impacto que o fluxo de exceção exerce em uma aplicação, sendo que alguns deles se concentram na interação com os aspectos e suas consequências trazendo tanto vantagens quanto desvantagens.