

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

**INVESTIGAÇÃO DE OPERADORES
ESSENCIAIS DE MUTAÇÃO PARA
PROGRAMAS ORIENTADOS A ASPECTOS**

JÉBUS THIAGO SOUSA LACERDA

ORIENTADOR: PROF. DR. FABIANO CUTIGI FERRARI

São Carlos – SP

Outubro/2014

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

**INVESTIGAÇÃO DE OPERADORES
ESSENCIAIS DE MUTAÇÃO PARA
PROGRAMAS ORIENTADOS A ASPECTOS**

JÉBUS THIAGO SOUSA LACERDA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software

Orientador: Prof. Dr. Fabiano Cutigi Ferrari

São Carlos – SP

Outubro/2014

**Ficha catalográfica elaborada pelo DePT da
Biblioteca Comunitária da UFSCar**

L131io

Lacerda, Jésus Thiago Sousa.

Investigação de operadores essenciais de mutação para programas orientados a aspectos / Jésus Thiago Sousa Lacerda. -- São Carlos : UFSCar, 2014.
120 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2014.

1. Software - testes. 2. Testes de mutação. 3. Controle de custo. 4. Orientação a aspectos. 5. Validação, verificação e teste. I. Título.

CDD: 005.14 (20ª)

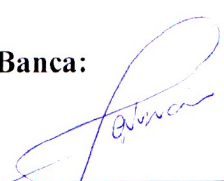
Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

**“Investigação de Operadores Essenciais de
Mutação para Programas Orientados a Aspectos”**

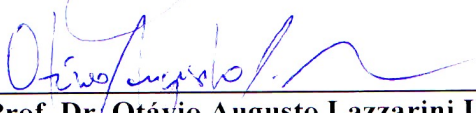
Jésus Thiago Sousa Lacerda

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

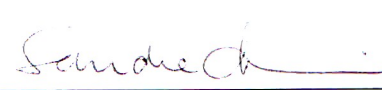
Membros da Banca:



Prof. Dr. Fabiano Cutigi Ferrari
(Orientador - DC/UFSCar)



Prof. Dr. Otávio Augusto Lazzarini Lemos
(UNIFESP)



Profa. Dra. Sandra Camargo P. Ferraz Fabbri
(DC/UFSCar)

São Carlos
Outubro/2014

Aos meus pais.

AGRADECIMENTOS

A Deus, pelas bênçãos derramadas em minha vida e pela sua infinita misericórdia.

Ao meu orientador, Prof. Dr. Fabiano Cutigi Ferrari, pela orientação, incentivo, dedicação demonstrados durante todo o trabalho e, sobretudo, pela amizade.

Aos meus pais, que apesar das dificuldades sempre me apoiaram em todas as fases da minha vida, principalmente na minha mudança para São Carlos.

À Amanda pelo amor, companheirismo, paciência e apoio dados durante esse tempo.

Aos meus familiares, agradecimento especial aos meus tios Lacerda, Socorro, Carmem e Reginaldo pelo suporte aqui em São Paulo e pelos finais de semanas acolhedores em um ambiente familiar, às minhas avós Edith e Rita pelo imenso carinho e ao meu avô Joaquim pelo exemplo de vida.

Ao pessoal do LaPES: Matheus, Odair, Gaspar, Elis, Ivan, Gastaldi, Kamilla, Anderson, Erik, André, Fábio, Fabiano, Augusto, Sandra, Cleitinho, Bento, Abade e Ana pelos momentos divertidos compartilhados no laboratório e nos *happy hours*. Aos demais amigos dos outros laboratórios: Vitinho, Mirela, Bruno, Jãum, Guido, Sanches, Maranhão, Tomé, Stéfano, Porto, Baiano e Danilo pela amizade.

Ao pessoal da “Rep. BigTable”: Guilherme Carvalho (Pequeno pônei), Odair Moreira (Oda, Foi), Afonso Takayoshi (Japa), Rafael Nuernberg (Animal) e Rodrigo Muñoz (Colômbia) pela amizade, convivência, campeonatos de futebol e pelos churrascos.

À CAPES e Carlos Augusto Costa Lacerda pelo apoio financeiro.

“Demore o tempo que for para você ver o que você quer da vida e depois que decidiu não recue ante nenhum pretexto, porque o mundo tentará te dissuadir”

Profeta Zaratrusta de Nietzsche

RESUMO

Contexto: A literatura de teste de software relata a aplicação do critério Análise de Mutantes – ou teste de mutação – em programas orientados a aspectos (OA) como uma forma promissora para revelar defeitos. Entretanto, esse critério é reconhecidamente de alto custo devido ao grande número de mutantes usualmente gerados e ao esforço para detectar os mutantes equivalentes. Ressalta-se que as iniciativas de aplicação de teste de mutação nesse contexto apresentam pouco enfoque em estratégias de redução de custo.

Objetivo: este trabalho tem como objetivo investigar a redução de custo de teste de mutação para programas OA. Em específico, este trabalho objetiva reduzir o custo do teste de mutação por meio da identificação de um conjunto reduzido de operadores de mutação que mantenham a efetividade do critério em garantir a qualidade dos conjuntos de teste produzidos.

Metodologia: para atingir o objetivo proposto, aplicou-se uma abordagem intitulada Procedimento Essencial, a qual resulta em conjuntos de operadores essenciais de mutação. Os testes adequados para os mutantes produzidos com esses operadores são capazes de revelar a maioria dos defeitos simulados em um conjunto completo de mutantes.

Resultados: por meio da aplicação do Procedimento Essencial, foi possível obter reduções de custo substanciais para três conjuntos de programas OA. As reduções obtidas nos experimentos variam de 52% a 62%. Os escores de mutação finais alcançados pelos testes adequados aos mutantes produzidos com os operadores essenciais variam de 92% a 94%.

Conclusão: com os resultados alcançados neste trabalho pode-se afirmar que é possível reduzir o custo do teste de mutação em programas OA sem perdas significativas na capacidade de revelar tipos de defeitos pré-definidos. O Procedimento Essencial mostrou-se eficaz na redução de custo e na manutenção da efetividade do critério.

Palavras-chave: Teste de software, teste de mutação, redução de custo, programas orientados a aspectos, verificação e validação de software

ABSTRACT

Context: The literature on software testing reports on the application of the Mutation Analysis criterion – or mutation testing – as a promising approach for revealing faults in aspect-oriented (AO) programs. However, it is widely known that this criterion is highly costly due to the large number of generated mutants and the effort required to identify equivalent mutants. We highlight that little existing research on mutation testing for AO programs focuses on cost reduction strategies.

Objective: this work aims at investigating the cost reduction of mutation testing for AO programs. In particular, we intend to reduce the cost of mutation testing by identifying a reduced set of mutation operators that are capable of keeping the effectiveness in guaranteeing the quality of the designed test sets.

Method: to achieve the goals, we applied an approach called Sufficient Procedure. Such approach yields sufficient (sets of) mutation operators. Test sets that are adequate with respect to mutants produced by sufficient operators are able to reveal the majority of faults simulated by a whole set of mutants.

Results: by applying the Sufficient Procedure, we obtained substantial cost reductions for three groups of AO programs. The cost reduction in the experiments range from 52% to 62%. The final mutation scores yielded by the test sets that are adequate to mutants produced by the sufficient operators range from 92% to 94%.

Conclusion: with the achieved results, we conclude that it is possible to reduce the cost of mutation testing applied to AO programs without significant losses with respect to the capacity of revealing prespecified fault types. The Sufficient Procedure has shown to be able to support cost reduction and to maintain the effectiveness of the criterion.

Keywords: software testing, mutation testing, cost reduction, aspect-oriented programs, verification and validation

LISTA DE FIGURAS

2.1	Relação de Inclusão entre os critérios estruturais (adaptada do trabalho de Delamaro (1997)).	30
2.2	Elementos Básicos a um sistema POA	32
2.3	Principais funcionalidades da ferramenta <i>Proteum/AJ</i> , adaptado do trabalho de Ferrari (2010).	37
3.1	Processo para a realização do Mapeamento Sistemático, adaptado do trabalho de Ferrari e Maldonado (2008).	41
3.2	Gráfico quantitativo das linguagens-alvo dos estudos primários.	46
3.3	Gráfico quantitativo das abordagens utilizadas nos estudos	48
3.4	Gráfico quantitativo dos tipos de objetos abordados	50
3.5	Gráfico de bolhas para as diferentes linguagens usadas em cada abordagem identificadas na revisão sistemática	50
3.6	Gráfico <i>escore</i> x <i>cost</i> (em escala logarítmica) para cada operador de mutação. (Figura adaptada de (MRESA; BOTTACI, 1999))	58
4.1	Exemplo de caso de teste - método <i>authenticateUser</i> da Classe <i>Log</i>	81
4.2	Diagrama comparativo dos dois conjuntos essenciais	91
5.1	Diagrama comparativo dos operadores essenciais identificados para os conjuntos <i>Set1</i> , <i>Set2</i> e <i>SetAll</i> de aplicações	101
5.2	Evolução dos conjuntos CE_{pre} para cada conjunto de aplicações (<i>Set1</i> , <i>Set2</i> e <i>SetAll</i>)	104
5.3	Gráfico comparativo com a redução de custo obtida nesse estudo e nos estudos de Offutt et al. (1996), Barbosa (1998)	106

5.4	Gráfico comparativo com o escore de mutação obtido com o conjunto essencial formado nesse estudo e nos estudos de Offutt et al. (1996), Barbosa (1998) . . .	107
-----	--	-----

LISTA DE TABELAS

2.1	Descrição dos operadores de mutação para AspectJ, tabela adaptada de (FER- RARI; MALDONADO; RASHID, 2008)	36
2.2	Resultados do estudo de Wedyan e Ghosh (2011).	38
3.1	Quantidade de estudos retornados por cada base de dados	43
3.2	Quantidade de estudos aceitos, rejeitados e duplicados na etapa de pré-seleção .	44
3.3	Quantidade de estudos Extraídos para revisão bibliográfica	44
3.4	Descrição dos estudos retornados	45
3.5	Quantidade de estudos por Linguagem-alvo	46
3.6	Quantidade de estudos por abordagem	48
3.7	Tabela comparativa com custo das cinco estratégias. (Tabela adaptada de Mresa e Bottaci (1999))	59
3.8	Tabela comparativa com os escores das cinco estratégias. (Tabela adaptada de Mresa e Bottaci (1999))	60
3.9	Tabela descritiva dos programas objetos	61
3.10	Tabela Resultante do modelo CBLARS	62
4.1	Descrição das aplicações do Experimento	67
4.2	Mutantes gerados a partir das 12 aplicações do conjunto Set1 (adaptada de Lacerda e Ferrari (2014))	69
4.3	Resultado do teste de mutação para as 12 aplicações do conjunto Set1 (adap- tada de Lacerda e Ferrari (2014))	70
4.4	Escore de mutação por operador do conjunto Set1	72
4.5	Escore de Mutação, strength e custo dos operadores.	73

4.6	Relação de inclusão empírica entre os operadores do conjunto Set1	74
4.7	Relação de inclusão empírica entre os operadores do conjunto Set1 após a exclusão do operador ABHA	74
4.8	Relação de inclusão empírica entre os operadores do conjunto Set1 após a exclusão do operador ABAR	75
4.9	Custo de cada operador essencial do conjunto Set1	77
4.10	Descrição das aplicações do Experimento	78
4.11	Exemplo de Tabela de Plano de Testes com classes de equivalência e valores limites	80
4.12	Mutantes gerados a partir das seis aplicações do conjunto Set2)	80
4.13	Resultado do teste de mutação para as seis aplicações do conjunto Set2	82
4.14	Escore de mutação por operador do conjunto Set2	82
4.15	Escore de Mutação, strength e custo dos operadores.	83
4.16	Relação de inclusão empírica entre os operadores do conjunto Set2	84
4.17	Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-1} após a exclusão do operador PCCE	85
4.18	Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-1} após a exclusão do operador POPL	85
4.19	Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-1} após a exclusão do operador ABAR	85
4.20	Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-1} após a exclusão do operador POAC	86
4.21	Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-2} após a exclusão do operador ABHA	86
4.22	Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-2} após a exclusão do operador POPL	87
4.23	Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-2} após a exclusão do operador ABAR	87
4.24	Custo de cada operador essencial do conjunto Set2	90

5.1	Descrição das aplicações do conjunto União dos Experimentos Set1 e Set2 . . .	94
5.2	Mutantes gerados a partir das 18 aplicações do conjunto união Set1 U Set2 . . .	95
5.3	Resultado do teste de mutação para as 18 aplicações do conjunto SetAll	96
5.4	Escore de mutação por operador do conjunto Set-All	96
5.5	Escore de Mutação, strength e custo dos operadores.	97
5.6	Relação de inclusão empírica entre os operadores do conjunto Set-All	97
5.7	Relação de inclusão empírica entre os operadores do conjunto Set-All	98
5.8	Relação de inclusão empírica entre os operadores do conjunto Set-All	98
5.9	Custo de cada operador essencial do conjunto Set-All	100
5.10	Tabela com a inserção gradual dos operadores dos Conjuntos Essenciais	102
5.11	Tabelas comparativa com as reduções de custos relatadas por outros estudos . . .	105
5.12	Tabelas comparativa com os escores de mutação alcançados por outros estudos . .	106

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO	17
1.1 Motivação e Definição do Problema	18
1.2 Objetivo	19
1.3 Metodologia	19
1.4 Organização do Documento	20
CAPÍTULO 2 – FUNDAMENTOS DE TESTE DE SOFTWARE E PROGRAMAÇÃO ORIENTADA A ASPECTOS	21
2.1 Considerações Iniciais	21
2.2 Conceitos	21
2.3 Técnicas e Critérios de Teste	23
2.3.1 Teste Funcional	23
2.3.2 Teste Estrutural	24
2.3.3 Teste Baseado em Defeitos	25
2.4 Automatização de Testes	28
2.5 Experimentação em Testes de Software	29
2.6 Teste de Mutação de Programas Orientados a Aspectos	30
2.6.1 Visão Geral da POA	31
2.6.2 Abordagens de Teste de Mutação para Programas OA	33
2.7 Considerações Finais	39

CAPÍTULO 3 – ESTRATÉGIAS DE REDUÇÃO DE CUSTO DO TESTE DE MUTAÇÃO 40

3.1	Considerações Iniciais	40
3.2	Visão Geral do Protocolo e dos Processos Empregados	41
3.2.1	Critérios de Inclusão e Exclusão	42
3.2.2	String de Busca	42
3.2.3	Fontes de Estudos Seleccionadas e Resultados	43
3.3	Sumarização dos Resultados	44
3.3.1	Análise Individual das Facetas	44
3.3.2	Análise Cruzada das Facetas	49
3.4	Estratégias Baseadas em Operadores Essenciais	51
3.4.1	A Investigação de Offutt et al. (1996)	52
3.4.2	A Investigação de Barbosa (1998)	54
3.4.3	A Investigação de Mresa e Bottaci (1999)	56
3.4.4	A Investigação de Namin, Andrews e Murdoch (2008)	60
3.5	Considerações Finais	63

CAPÍTULO 4 – PREPARAÇÃO DO ESTUDO, EXECUÇÃO E ANÁLISE DOS RESULTADOS 64

4.1	Considerações Iniciais	64
4.2	Informações Preliminares	64
4.2.1	Procedimento Essencial	65
4.2.2	Infraestrutura de Apoio	66
4.3	Aplicação do Procedimento Essencial no Conjunto 1	67
4.3.1	Programas do conjunto Set1	67
4.3.2	Geração dos Conjuntos AM-Adequados	69
4.3.3	Preparação dos Dados	71
4.3.4	Coleta de Dados	72

4.3.5	Análise dos Dados - Conjunto Set1	77
4.4	Aplicação do Procedimento Essencial no Conjunto 2	77
4.4.1	Seleção dos Programas	78
4.4.2	Geração dos Conjuntos AM-Adequados	79
4.4.3	Preparação dos Dados	82
4.4.4	Aplicação Passo-a-Passo do Procedimento Essencial	83
4.4.5	Análise dos Dados - Conjunto Set2	90
4.5	Comparação entre os conjuntos Set1 e Set2	91
4.6	Considerações Finais	92

CAPÍTULO 5 – UM ESTUDO COMPLEMENTAR - O PROCEDIMENTO ESSENCIAL APLICADO AO CONJUNTO SETALL 93

5.1	Considerações Iniciais	93
5.2	Aplicação do Procedimento Essencial no conjunto SetAll	93
5.2.1	Seleção dos Programas	94
5.2.2	Geração dos Conjuntos AM-adequados	94
5.2.3	Preparação dos Dados	95
5.2.4	Aplicação Passo-a-Passo do Procedimento Essencial	96
5.2.5	Análise dos Dados - Conjunto SetAll	100
5.3	Comparação com os Conjuntos Set1 e Set2	100
5.4	Comparação com Outros Conjuntos Essenciais	103
5.5	Considerações Finais	107

CAPÍTULO 6 – CONCLUSÃO E TRABALHOS FUTUROS 108

6.1	Contribuições	110
6.2	Limitações	111
6.3	Trabalhos Futuros	111

REFERÊNCIAS

113

GLOSSÁRIO

121

Capítulo 1

INTRODUÇÃO

As atividades de Verificação e Validação (V&V) de software tem o objetivo de ajudar a aumentar a qualidade do software que está sendo desenvolvido. A qualidade é um dos principais requisitos a serem considerados durante o desenvolvimento, e para que o software atinja um nível de qualidade aceitável, atividades de V&V são introduzidas ao longo do processo de desenvolvimento. Uma dessas atividades é o teste de software, cujo objetivo é revelar defeitos presentes nos artefatos produzidos (MYERS et al., 2004). Ressalta-se que apesar do uso de técnicas avançadas de engenharia e ferramentas de apoio, defeitos ainda persistem. Sendo assim, o teste de software é empregado como uma forma de revelar esses defeitos que, em momento posterior, serão analisados e corrigidos pela equipe de desenvolvimento.

Ainda em relação à qualidade de um software, uma possível maneira de melhorá-la está relacionada à aplicação de técnicas contemporâneas de desenvolvimento. Essas abordagens visam lidar com a crescente complexidade dos produtos de software. Pode-se citar como exemplo dessas abordagens a Programação Orientada a Objetos (POO), a qual se beneficia da decomposição de problemas baseada em objetos (BOOCH, 1994).

Para propor soluções adequadas para problemas mais complexos, nos quais nem a programação procedimental nem a POO poderiam resolver, a Programação Orientada a Aspectos (POA) foi proposta por Kiczales et al. (1997). A POA propõe-se a melhorar a modularização de um software em face às dificuldades enfrentadas pelas técnicas existentes, dentre elas a POO. Essas dificuldades referem-se aos interesses de software que geralmente encontram-se espalhados por diversos módulos do sistema ou entrelaçados aos demais interesses. Tais interesses são conhecidos como transversais - do inglês, *crosscutting concerns* (KICZALES et al., 1997) - e podem representar tanto requisitos funcionais como requisitos não-

funcionais do software.

Embora a POA consiga resolver problemas relacionados à separação de interesses transversais envolvidos no software, a aplicação dessa técnica (ou paradigma) pode introduzir novos tipos de defeitos de software. Sendo assim, reforça-se a importância do teste de software para garantir a qualidade do software que utiliza esse paradigma.

Dentre os critérios de teste investigados para software orientado a aspectos (OA), a Análise de Mutantes (DEMILLO; LIPTON; SAYWARD, 1978), também conhecida como *teste de mutação*, tem se mostrado uma importante alternativa na avaliação tanto do software em si, quanto do conjunto de testes utilizado para testá-lo. O teste de mutação consiste em inserir defeitos simples no software, com o intuito de mostrar que os testes projetados para esse software são capazes de revelar esses defeitos. Ressalta-se que os defeitos inseridos no software visam a simular defeitos comumente introduzidos durante o desenvolvimento de software.

1.1 Motivação e Definição do Problema

No contexto de software OA, Mortensen e Alexander (2005) foram os primeiros pesquisadores a proporem uma abordagem de teste de mutação para ser aplicada a programas escritos na linguagem AspectJ (KICZALES et al., 2001). Os autores definem três operadores de mutação que atuam em elementos básicos de POA como, por exemplo, *pointcuts* e expressões de precedência entre aspectos. Em um trabalho subsequente, utilizando dois dos operadores definidos por Mortensen e Alexander, Anbalagan e Xie (2008) propuseram um *framework* para a geração de mutantes e para a detecção automática de mutantes equivalentes ao programa original, como foco no teste de *pointcuts*, que consiste em um dos principais mecanismos da POA. Os trabalhos de Ferrari et al. (2008, 2010) e de Delamare, Baudry e Le Traon (2009) também propuseram diferentes abordagens e ferramentas para apoiar o teste de mutação em programas OA, escritos na linguagem AspectJ (KICZALES et al., 2001).

Os trabalhos mencionados têm mostrado a relevância do critério Análise de Mutantes para programas OA, porém, apesar da eficácia do critério em revelar defeitos, a aplicação do mesmo é reconhecidamente de alto custo. Dessa forma, faz-se necessário a investigação de técnicas de redução de custo no contexto de programas OA.

Alguns estudos têm proposto técnicas para redução de custo em teste de mutação, alguns desses estudos inclusive têm aplicado técnicas de redução em programas OA. Uma das formas de reduzir o custo do teste de mutação consiste na aplicação de apenas um subconjunto relevante de operadores de mutação, chamados *operadores essenciais* – do inglês, *sufficient opera-*

tors (OFFUTT et al., 1996). Os casos de teste projetados para revelar os defeitos modelados pelos operadores essenciais serão capazes de revelar a maioria dos defeitos simulados pelo conjunto completo de operadores, sendo que essa propriedade é demonstrada empiricamente (OFFUTT et al., 1996; MRESA; BOTTACI, 1999; BARBOSA, 1998; NAMIN; ANDREWS; MURDOCH, 2008).

Tendo em vista a possibilidade de se identificar em operadores essenciais como uma forma de reduzir o custo de aplicação desse critério de teste em programas OA, define-se na próxima seção o objetivo deste trabalho.

1.2 Objetivo

Este trabalho tem por objetivo a investigação de técnicas de redução de custos de teste de mutação, especificamente a redução de custo em teste de mutação para o paradigma orientado a aspectos. Ressalta-se que, apesar de existirem na literatura alguns estudos relatando técnicas de redução de custo neste paradigma, não há até o momento, nenhum outro estudo propondo a redução de custo por meio da determinação de um conjunto essencial de operadores no paradigma OA. Este trabalho visa reduzir o custo de teste de mutação no paradigma OA, por meio da identificação de um conjunto essencial de operadores, porém sem diminuir a eficiência e eficácia do critério.

1.3 Metodologia

O objetivo do trabalho foi alcançado por meio da realização de três experimentos, no quais foi aplicado o Procedimento Essencial (BARBOSA, 1998) em três conjuntos de programas orientados a aspectos, desenvolvidos em AspectJ. Cada um desses conjuntos é composto por pequenas aplicações escritas em AspectJ, algumas delas já utilizadas em outros estudos.

Para cada um dos conjuntos de programas, foi aplicado o critério análise de mutantes utilizando os operadores definidos por Ferrari, Maldonado e Rashid (2008). Posteriormente foram criados casos de teste adequados para os dados de teste gerados pelo teste de mutação. A partir desses dados de teste e casos de teste adequados foram realizados cruzamentos entre os operadores e coleta dos dados necessários para aplicação do Procedimento Essencial. Posteriormente, foram realizadas análises das reduções de custos dos conjuntos obtidos com o procedimento, e foram feitas também comparações entre os operadores de mutação de cada conjunto.

Finalmente, após as comparações entre os conjuntos, foram realizadas comparações dos resultados obtidos com os experimentos realizados neste trabalho, com os resultados relatados

em outros estudos que também determinaram conjuntos essenciais para redução de custo em teste de mutação.

1.4 Organização do Documento

Neste capítulo foram apresentados o contexto no qual este trabalho está inserido, a motivação e definição do problema que este trabalho propõe resolver. O restante do documento está disposto da seguinte forma:

- No *Capítulo 2* apresenta-se a fundamentação teórica para possibilitar a compreensão do trabalho que foi realizado.
- No *Capítulo 3* é descrito um Mapeamento Sistemático realizado, cujo objetivo foi caracterizar a pesquisa no tópico de redução do custo do teste de mutação. O capítulo apresenta o desenvolvimento do mapeamento e também uma síntese e análise dos resultados.
- No *Capítulo 4* são apresentados a infraestrutura de apoio para aplicação do critério análise de mutantes, a aplicação de abordagem para obtenção de um conjunto reduzido de operadores de mutação, bem como os resultados obtidos com a abordagem.
- No *Capítulo 5* é apresentado um estudo complementar que envolveu um conjunto mais amplo de aplicações no qual a estratégia de redução de custo foi aplicada. Os resultados desse estudo complementar são detalhados e também confrontados com os resultados descritos no Capítulo 4.
- Por fim, no *Capítulo 6* são apresentadas as conclusões, limitações e possibilidades de trabalhos futuros.

Capítulo 2

FUNDAMENTOS DE TESTE DE SOFTWARE E PROGRAMAÇÃO ORIENTADA A ASPECTOS

2.1 Considerações Iniciais

Neste capítulo serão apresentados alguns conceitos sobre a atividade de teste de software, sobre a importância dessa atividade no processo de desenvolvimento do software, e as principais técnicas e critérios aplicados durante essa atividade. Atenção especial é dedicada ao critério de Análises de Mutantes da técnica de teste baseada em defeitos, por tratar-se do critério que será explorado na presente proposta de trabalho de Mestrado.

2.2 Conceitos

Durante o processo de desenvolvimento do software, podem ser utilizadas diversas técnicas e ferramentas com o propósito de reduzir ao máximo a quantidade de defeitos em um software. Contudo, ainda não é possível garantir que o software está livre de defeitos. Os defeitos e as falhas associadas podem estar ligadas a diversas causas como, por exemplo, requisitos mal compreendidos, especificação incorreta, e também erros de codificação.

O papel do teste de software é detectar a presença de defeitos no software (MYERS et al., 2004). Um defeito é detectado quando uma ou mais falhas são observadas com o software em operação, seja em ambiente simulado ou real. Tais falhas surgem devido à execução de algum trecho do software que contém um ou mais defeitos. Nesse contexto, o teste de soft-

ware exerce um papel muito importante no processo de garantia de qualidade do software, impactando seus atributos qualitativos como, por exemplo, confiabilidade, usabilidade, funcionalidade, eficiência, portabilidade e manutenibilidade.

Uma estratégia de teste bastante difundida pelas equipes de teste de software consiste na aplicação incremental dos testes. Essa estratégia incremental parte do nível de unidade, no qual os testes do sistema são realizados considerando suas menores partes, depois avança para a fase da integração dessas unidades, e finaliza com o teste do sistema completamente integrado. Essas etapas são descritas a seguir:

- **Teste de Unidade:** nessa fase o teste tem por objetivo garantir que as menores unidades do sistema funcionam corretamente, ou seja, identificar defeitos de lógica e de implementação das unidades em teste. O foco nesse nível está na lógica interna do processamento e nas estruturas de dados dentro dos limites de uma unidade do software (PRESSMAN, 2010)
- **Teste de Integração:** essa fase tem por objetivo testar unidades integradas de software, que foram individualmente testadas na fase anterior. Parte-se do pressuposto que a afirmativa “*se todos funcionam individualmente, então irão funcionar se colocados em conjunto*” não é válida, pois quando colocados juntos as unidades podem produzir resultados inesperados ou contrários aquilo que foi especificado. O teste de integração é uma técnica sistemática utilizada para construir a arquitetura do software, ao mesmo tempo em que conduz testes para descobrir erros associados às interfaces. (MALDONADO et al., 2000)
- **Teste de Validação:** Na fase de validação o software está montado e integrado, e neste momento os requisitos que foram determinados na análise de requisitos são validados em contraste com o que acabou de ser construído. Essa etapa de validação é baseada em teste *caixa-preta*, também chamado *teste funcional*, que são testes que focam nos requisitos funcionais do software (PRESSMAN, 2010).
- **Teste de Sistema:** Nessa última fase, a de teste de sistema, o software é incorporado aos outros elementos, como por exemplo: hardware, pessoal e informações. A partir desse ponto vários testes são realizados com o intuito exercitar e validar o sistema no ambiente de produção. Nota-se que para essa validação “completa” do sistema, são realizados outros tipos de testes de sistema como, por exemplo, teste de recuperação, teste de segurança, teste de estresse e teste de desempenho.

2.3 Técnicas e Critérios de Teste

Diferentes técnicas e critérios de testes podem ser empregados durante o processo de desenvolvimento de software. As técnicas e os respectivos critérios diferem entre si principalmente pelo tipo de artefato (ou abstração de software) utilizado para criar um modelo que representa o software em teste. Exemplos desses artefatos são uma especificação de requisitos, um grafo de fluxo de controle, um diagrama de sequência da UML, e uma taxonomia de defeitos. A partir dessa abstração, critérios de teste guiam a derivação dos requisitos de teste, que representam os elementos do software que devem ser exercitados pelos testes. Dentre as técnicas mais empregadas pela indústria de software e mais investigadas pela comunidade científica, encontram-se o *teste funcional*, o *teste estrutural* e o *teste baseado em defeitos*.

A seguir serão descritas cada uma dessas técnicas e os critérios utilizados para a avaliação e construção dos casos de teste. Ressalta-se que um critério de teste pode ser visto como uma diretriz para reduzir o domínio de entradas que o software pode receber, visando a tornar o teste viável em termos de custo de aplicação (FRANKL; WEYUKER, 2000).

2.3.1 Teste Funcional

O teste funcional, também conhecido como teste caixa-preta (MYERS et al., 2004), é destinado à avaliação do comportamento externo do componente ou da unidade de software. No teste funcional, os valores de entrada são fornecidos para o software, os testes são processados, e o resultado desse processamento é comparado com as saídas esperadas, sem que se considerem detalhes internos de implementação do software. O teste funcional baseia-se nos requisitos funcionais do software (PRESSMAN, 2010), ou seja, tem por objetivo certificar que as funcionalidades do sistema estão em acordo com o que foi elicitado nos requisitos.

Os critérios de testes funcionais mais empregados são:

- **Particionamento de Equivalência:** consiste na divisão do conjunto de elementos do domínios de entrada em um número finito de classes de equivalência, de forma que um teste de um elemento para uma determinada classe seja equivalente ao teste de outro elemento da mesma classe, diminuindo o número de casos de teste necessários. A ideia é que se um caso de teste para um determinado elemento de uma classe revela um defeito, por equivalência todos os outros casos de teste para os outros elementos da mesma classe revelariam o mesmo defeito.
- **Análise de Valor Limite:** visa a testar os limites superior e inferior de cada classe de

equivalência. Esse critério é um complemento do critério do Particionamento de Equivalência, e é motivado pelo grande número de defeitos que tende a ocorrer nesses limites de entrada (PRESSMAN, 2010).

- **Grafo de Causa-efeito:** é uma representação da relação entre objetos de dados e objetos do programa, o que permite a derivação dos casos de teste que verificam se todos os objetos têm a relação esperada uns com os outros (BEIZER, 1995). Em resumo, são levantadas as condições de entrada (causas) e suas saídas (efeitos) e a partir desses levantamentos, é então montado um grafo relacionando as causas e seus possíveis efeitos. Depois esse grafo é convertido em uma tabela de decisão, do qual serão gerados os casos de teste (MALDONADO et al., 2000).

2.3.2 Teste Estrutural

O teste estrutural, ou teste *caixa-branca* (MYERS et al., 2004; PRESSMAN, 2010), é complementar ao teste funcional, e baseia-se na estrutura interna do software para derivar os casos de teste. Com o teste estrutural é possível derivar casos de teste que, por exemplo: (i) garantam que todos os caminhos independentes de um módulo tenham sido exercitados pelo menos uma vez; (ii) exercitem todas as decisões lógicas; (iii) executam todos os ciclos dentro de seus limites operacionais; e (iv) exercitem todas as possíveis estruturas de dados internas (PRESSMAN, 2010).

Na técnica estrutural, para auxiliar na definição dos requisitos de teste, utiliza-se uma representação do código fonte denominada Grafo de Fluxo de Controle (GFC). Nesse grafo, o programa é decomposto em blocos de instruções sequenciais. Cada bloco de instrução é um trecho de programa que é representado por um nó do grafo, e a correspondência entre os blocos indicando possíveis fluxos de controle de execução são representados por arestas. O GFC pode ser definido então como um grafo orientado com apenas um nó de entrada, podendo ter vários nós de saída, onde cada nó é um bloco indivisível de código e cada aresta representa um possível desvio para algum outro bloco, ou para o mesmo que o gerou (MYERS et al., 2004).

Os critérios de teste estrutural baseiam-se em diferentes tipos de elementos para determinar quais partes do programa devem ser executadas. Os critérios são classificados em dois tipos: *Baseados em Fluxo De Controle* (MYERS et al., 2004) e *Critérios Baseados em Fluxo De Dados* (RAPPS; WEYUKER, 1982).

Crítérios Baseados em Fluxo de Controle:

- **Todos-Nós:** requer que cada comando do programa seja executado pelo menos uma vez, ou seja, cada nó do grafo de fluxo de controle deve ser coberto por pelo menos um caso de teste.
- **Todos-Arcos:** requer que cada desvio do fluxo de controle seja executado ao menos uma vez, ou seja, cada arco (aresta) do nó no grafo de fluxo de controle deve ser coberto por pelo menos um caso de teste.
- **Todos-Caminhos:** requer que todos os caminhos possíveis no grafo de fluxo de controle sejam executados. Esse é considerado o critério mais exigente pois, mesmo em programas simples, podem existir infinitos caminhos.

Crítérios Baseados em Fluxo De Dados:

- **Todas-Definições:** requer que cada variável definida em cada nó do grafo seja exercitada em algum uso predicativo (p-uso) ou algum uso computacional (c-uso) (RAPPS; WEYUKER, 1982). Em outras palavras, para cada variável definida, deve ser construído um caso de teste que exercite o caminho entre a definição da variável e algum uso dessa variável. A ideia é que se um valor é atribuído a essa variável, então essa variável deve ser utilizada em algum momento do programa; caso contrário, não existe a necessidade da atribuição a essa variável.
- **Todos-Usos:** requer que cada variável definida, em cada nó, seja exercitada ao longo do seu uso antes de ser redefinida. Esse critério possui ainda algumas variantes como, por exemplo, Todos-P-Usos e Todos-C-Usos (RAPPS; WEYUKER, 1982).
- **Todos-Potenciais-Usos:** requer que os caminhos entre cada variável definida em cada nó do grafo e todos os nós alcançáveis do grafo (nós nos quais possa ocorrer um uso dessa variável) sejam exercitados, antes que a variável seja redefinida (MALDONADO, 1991)

2.3.3 Teste Baseado em Defeitos

É uma técnica de teste que utiliza informações sobre os erros mais comuns cometidos no processo de desenvolvimento de software, os quais resultam na introdução de defeitos no mesmo (MORELL, 1990). Parte-se da premissa que esses defeitos comumente inseridos no software são os tipos de defeitos que se deseja revelar com os testes, então guiando o projeto dos casos de teste.

Dois critérios utilizados no teste baseado em defeitos são: *Semeadura de Erros* e *Análise de Mutantes* (DEMILLO; LIPTON; SAYWARD, 1978).

- **Semeadura de Erros:** consiste na inserção de uma quantidade determinada de defeitos no software, com o propósito de derivar um conjunto de casos de teste que sejam capazes de detectar esses defeitos inseridos e obter uma taxa real de defeitos inseridos ou os defeitos reais presentes no software.
- **Análise de Mutantes:** consiste na geração de várias versões modificadas (mutantes) de um software, com o objetivo de revelar defeitos comumente inseridos durante do desenvolvimento. Nesse critério de teste, os casos de teste também são avaliados quanto à sua capacidade de detectar esses defeitos.

Esse critério é parte fundamental para realização do trabalho proposto neste documento, que tratará de técnicas de redução de custo dos testes de mutação. Sendo assim, mais detalhes desse critério são apresentados na próxima seção.

Análise de Mutantes

A ideia do critério *Análise de Mutantes* – também conhecido como *teste mutação* – foi descrita pela primeira vez por DeMillo, Lipton e Sayward (1978), em um artigo que apresenta uma hipótese, denominada *Hipótese do Programador Competente*. Essa hipótese consiste na ideia de que programadores experientes escrevem programas corretos ou muito próximos do correto. Sendo assim, defeitos são introduzidos em programas por meios de pequenos desvios sintáticos, que podem alterar a semântica do programa e consequentemente causando erros de comportamento.

Outra hipótese explorada por DeMillo, Lipton e Sayward (1978) é o *Efeito de Acoplamento* (do inglês, *coupling effect*), a qual afirma que os defeitos complexos estão relacionados a defeitos simples. Sendo assim, espera-se que os conjuntos de casos de teste capazes de revelar defeitos simples são também capazes de revelar os defeitos complexos.

Tendo conhecimento da hipótese do programador competente e do efeito do acoplamento, o teste de mutação funciona da seguinte forma: geram-se várias versões de um programa P em teste ligeiramente modificadas (por exemplo, P', P'', ...), denominadas *mutantes*. Cada modificação representa um defeito inserido em P, sendo que tradicionalmente cada mutante possui apenas um defeito simples, tendo em vista as hipóteses subjacentes. Os testes são executados no programa original e em cada mutante. Se o resultado diferir em pelo menos um caso de teste, conclui-se que os testes são suficientemente sensíveis para detectar o defeito contido

naquele mutante, e o mutante é considerado *morto*. Caso contrário, o mutante é dito *vivo* e mais testes podem ser necessários para revelar o defeito do mutante, ou o mutante analisado é classificado como equivalente ao programa original. Para inserir essas modificações – ou seja, para gerar os mutantes – são utilizados *operadores de mutação* que, de modo geral, são regras que definem os tipos de modificações a serem realizadas no programa original.

Duas possibilidades devem ser analisadas quando um mutante permanece vivo: (i) o mutante é equivalente ao programa original e deve ser descartado, pois todos os testes terão o mesmo resultado tanto para o mutante quanto para o programa; ou (ii) é necessário melhorar o conjunto de testes de forma a incluir um ou mais casos de teste que resultem em uma saída diferente para o mutante.

Uma maneira de medir o nível de confiança da adequação dos casos de teste é através do *escore de mutação*, que é calculado da seguinte forma:

$$ms(P,T) = \frac{MK}{M(P) - EM(P)} \quad (2.1)$$

Onde:

- $ms(P,T)$: escore de mutação (do inglês, *ms – mutation score*) de um programa P para um conjunto T de casos de teste;
- MK: número de mutantes mortos (do inglês, *mutants killed*);
- $M(P)$: número de mutantes gerados a partir do programa P; e
- $EM(P)$: número de mutantes equivalentes (do inglês, *equivalent mutants*) gerados a partir do programa P.

A vantagem de se utilizar o teste de mutação, além de garantir a qualidade do produto, é certificar que o conjunto de casos de teste é adequado para revelar os defeitos inseridos no programa. Eventualmente, um mutante pode auxiliar na identificação de um defeito no programa original; por exemplo, quando a modificação realizada pelo operador de mutação na verdade gerou uma versão correta do programa (MATHUR, 2007).

2.4 Automatização de Testes

O teste manual de software permite a detecção de diversos defeitos em um software. Entretanto, esse processo é um trabalho desgastante e requer um grande demanda de tempo e recursos, além de ser propenso a erros (VINCENZI, 2004). Dessa forma, o uso de ferramentas para a realização de testes se faz necessário para aumentar a confiança de que o software desenvolvido desempenha as funções especificadas e para evidenciar características mínimas do ponto de vista da qualidade do software (HARROLD, 2000; WEYUKER, 1996).

A aplicação de um critério de teste está fortemente condicionada à sua automatização, sendo então de fundamental importância o desenvolvimento de ferramentas de teste. Sem a utilização de ferramentas que automatizam a aplicação das técnicas e critérios de teste, a atividade torna-se onerosa, propensa a erros e limitada a programas muito simples (VINCENZI, 2004).

Muitos métodos e ferramentas são desenvolvidos com intuito de auxiliar engenheiros de software a produzir software com qualidade. Conforme enfatizado por Harrold (2000), pesquisas são necessárias para viabilizar a criação desses métodos e ferramentas, e também para possibilitar a realização de estudos experimentais para transferir essa tecnologia para a indústria.

Diversas ferramentas têm sido desenvolvidas o propósito de auxiliar na automatização dos testes. Exemplos são as ferramentas Proteum e Proteum/IM (DELAMARO, 1993; DELAMARO; MALDONADO, 1996; DELAMARO; MALDONADO; MATHUR, 2001) que implementam o critério Análise de Mutantes para teste de unidade e integração, respectivamente, para programas escritos em linguagem C; a Proteum/SML (YANO, 2004), que apoia o critério Análise de Mutantes para o teste de unidade de programas escritos em SML; a família de ferramentas Poke-Tool, que apoia o teste estrutural de unidade para programas escritos nas linguagens C (CHAIM, 1991; CHAIM; MALDONADO; JIHO, 1997), Fortran (FONSECA, 1993), Cobol (Leitão Jr., 1992) e Clipper (BORGES et al., 1995); a JaBUTi (VINCENZI et al., 2003) e a sua extensão JaBUTi/AJ (LEMONS; MALDONADO; MASIERO, 2005), que apoiam o teste estrutural intra-unidade de programas OO escritos em Java e de programas OA escritos em AspectJ, respectivamente; e a ferramenta Proteum/AJ (FERRARI et al., 2010), que apoia testes de mutação em programas OA escritos em AspectJ.

Além das ferramentas Proteum, Proteum/IM, Proteum/SML e Proteum/AJ, em um extenso levantamento da literatura conduzido por Jia e Harman (2010), diversas ferramentas de teste de mutação foram identificadas. Dentre elas destacam-se, por exemplo: a ferramenta Mothra (DEMILLO; LIPTON; SAYWARD, 1978), desenvolvida para a linguagem FORTRAN; a ferramenta JavaMut (CHEVALLEY; Thévenod-Fosse, 2003), voltada para testes de mutação na linguagem Java;

a ferramenta MuJava (MA; OFFUTT; KWON, 2005), também desenvolvida para linguagem Java, e o seu *plugin* MuClipse (SMITH; WILLIAMS, 2007; SMITH, 2012) para o ambiente de desenvolvimento Eclipse; a ferramenta MUSIC (SHAHRIAR; ZULKERNINE, 2008), que implementa uma variação do teste de mutação para instruções escritas em SQL; e a ferramenta AjMutator (DELAMARE; BAUDRY; Le Traon, 2009), desenvolvida para o teste de programas OA escritos na linguagem AspectJ, e que será abordada com mais detalhes nos próximos tópicos.

2.5 Experimentação em Testes de Software

A realização de experimentos é importante para analisar e determinar os melhores métodos e técnicas para as atividades de Engenharia de Software. No contexto de teste de software, os experimentos auxiliam na identificação das abordagens (que empregam técnicas e critérios de teste específicos) que representam uma melhor relação custo/benefício na sua aplicação.

Na comparação entre os critérios de teste, sejam eles associados a uma mesma técnica ou a diferentes técnicas, Wong (1993) cita que três aspectos podem ser considerados nos estudos teóricos e experimentais: (i) *custo*, que refere-se ao esforço necessário na utilização de um critério e pode ser medido pelo número de requisitos gerados, pelo número de casos de teste necessário para cobrir os requisitos, e também por outros fatores como, por exemplo, o tempo necessário para executar os mutantes ou tempo para identificar os mutantes equivalentes; (ii) *efetividade*, que refere-se à capacidade do critério em revelar um maior número de defeitos em relação a outro; (iii) *dificuldade de satisfação* (do inglês, *strength*), que refere-se à probabilidade de satisfazer um critério C2 tendo sido satisfeito um critério C1.

A partir desses aspectos citados por Wong, diversos estudos empíricos e teóricos têm sido desenvolvidos com o objetivo de encontrar formas econômicas e produtivas para a realização de testes. Para estimar o custo de um critério, é utilizada como medida a complexidade de um critério. A complexidade é calculada pelo número de casos de teste necessários para satisfazer o critério. No teste de mutação, essa complexidade pode ser medida pelo número de mutantes gerados, que obviamente vai depender do conjunto de operadores de mutação utilizado. Para determinar a força de um critério, utiliza-se a relação de inclusão de um critério de teste com outros critérios de teste. Por exemplo, dados os critérios C e C' , é dito que C inclui C' quando um conjunto de casos de teste adequado para o critério C é também adequado para o critério C' .

Rapps e Weyuker (1982) e, posteriormente, Maldonado (1991), conduziram estudos teóricos que resultaram na relação de inclusão de critérios. Essa relação pode ser observada na Figura 2.1, a qual envolve critério baseados em fluxo de controle e fluxo de dados apresentados

na Subseção 2.3.2.

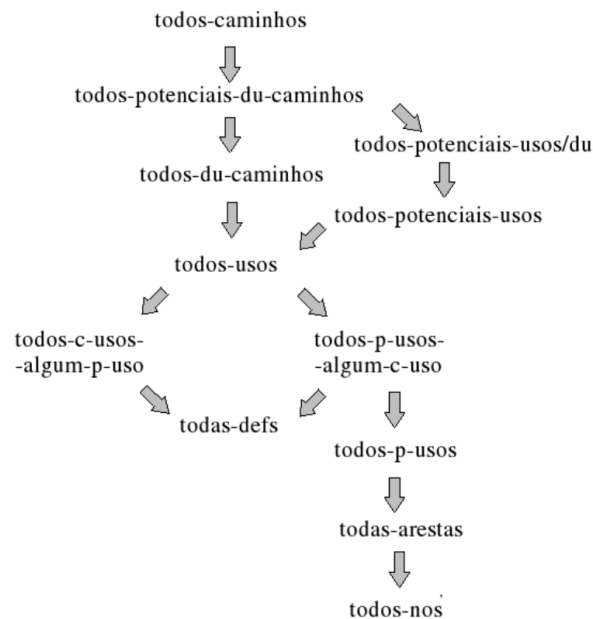


Figura 2.1: Relação de Inclusão entre os critérios estruturais (adaptada do trabalho de Delamaro (1997)).

Wong (1993), por sua vez, demonstra que a comparação entre critérios de naturezas diferentes, geralmente, não é uma tarefa simples. Em seu estudo, Wong demonstra que não é possível, por exemplo, realizar comparações do critério Análise de Mutantes com os critérios baseados em fluxo de dados (Todos-usos, Todos-p-usos e Todos-c-usos), pois o critério Análise de Mutantes não requer explicitamente a execução de sub-caminhos do programa em teste, como é realizado nos critérios baseados em fluxo de dados.

De acordo com Wong (1993) a partir de três propriedades é possível empiricamente e teoricamente comparar diferentes critérios: (i) *custo*, que pode por exemplo ser calculado pelo número de casos de teste necessários para cobrir os requisitos de teste; (ii) *eficácia*, que está relacionada a capacidade de detecção de falhas de um critério; (iii) *força*, está relacionado a capacidade de um determinado critério ser adequado a um outro critério.

2.6 Teste de Mutação de Programas Orientados a Aspectos

O teste de mutação, como visto na Seção 2.3.3, é importante para detectar defeitos comumente introduzidos no software durante o seu desenvolvimento. Esse critério de teste tem sido amplamente investigado no contexto de programas OO e procedimentais, para os quais têm-se desenvolvido novas estratégias. Grande parte dessas estratégias foi identificada na pesquisa realizada por Jia e Harman (2010). Mais recentemente, com o surgimento da POA, tem-se in-

vestigado o teste de mutação também nesse contexto. Considerando que o trabalho proposto neste documento insere-se nesse contexto, esta seção é dedicada ao teste de mutação de programas OA. Para possibilitar uma melhor compreensão dos trabalhos descritos nesta seção, inicialmente apresenta-se uma visão geral da POA.

2.6.1 Visão Geral da POA

O paradigma de Programação Orientada a Objetos (POO) tem sido importante para auxiliar na engenharia do software, pois o modelo de objeto subjacente oferece um melhor ajuste com problemas de domínio reais. Contudo, apesar das vantagens da POO, alguns problemas relacionados principalmente à modularização de software não podem ser adequadamente resolvidos com esse paradigma de programação. Em suma, as técnicas de POO não são suficientes para capturar claramente todas as decisões importantes que o software deve implementar, dessa forma resultando em uma deficiente separação dos interesses implementados no software (KICZALES et al., 1997).

A ideia central da Programação Orientada a Aspectos (POA) é que enquanto os mecanismos de modularização hierárquico das linguagens OO são extremamente úteis, eles são incapazes de modularizar todos os interesses em sistemas complexos. Ao invés disso, a POA baseia-se no fato de que na implementação de qualquer sistema complexo sempre existirão interesses que podem ser modularizados, evitando que esses fiquem entrelaçados com outros interesses ou espalhados por diversas partes do código (KICZALES et al., 1997). A POA permite ao desenvolvedor a separação e organização do código de acordo com os princípios de Separação de Interesses (do inglês, *Separation of Concerns*) (DIJKSTRA, 1976). A separação de interesses é importante para melhorar a qualidade do código fonte produzido e facilitar sua manutenção.

As linguagens OO conseguem capturar os comportamentos dos objetos, porém as falhas modeladas por esse paradigma são falhas nos comportamentos secundários, estruturais e invariantes. Tais comportamentos não estão isolados em um só local; eles estão dispersos ao longo dos módulos do sistema, ou seja, são transversais aos módulos. E o espalhamento do código leva à perda de modularidade e uma possível perda de abstração. A POA surgiu para combater essas fragilidades; ela fornece uma nova forma de modularização que permite a implementação de cada aspecto do sistema separadamente na sua forma natural.

Na POA, existem alguns conceitos intrínsecos ao entendimento desse paradigma de programação. Um deles é a expressão *aplicação base*, que representa o conjunto de funcionalidades (interesses) que podem ser modularizados de acordo com os paradigmas tradicionais. Outra expressão também utilizada na POA é *interesse transversal*, que refere-se a um interesse

que se encontra espalhado ou entrelaçado aos demais interesses do software.

A POA torna possível a modularização de interesses transversais, e a entidade responsável por fazer essa modularização é o *aspecto*. O processo utilizado para fazer com que os aspectos e a aplicação base interajam corretamente é denominado combinação (*weaving*), que é realizada por meio de um combinador, o *aspect weaver*.

Os elementos citados (*aspecto*, *weaving*, *aspect weaver*) são componentes básicos para dar apoio à POA. O funcionamento deles pode ser observado na Figura 2.2. A funcionalidade básica do sistema e os aspectos podem ser desenvolvidos separadamente, independentemente da linguagem imposta em cada um. O *combinador* (do inglês, *weaver*) é o elemento encarregado por produzir o código em formato executável.

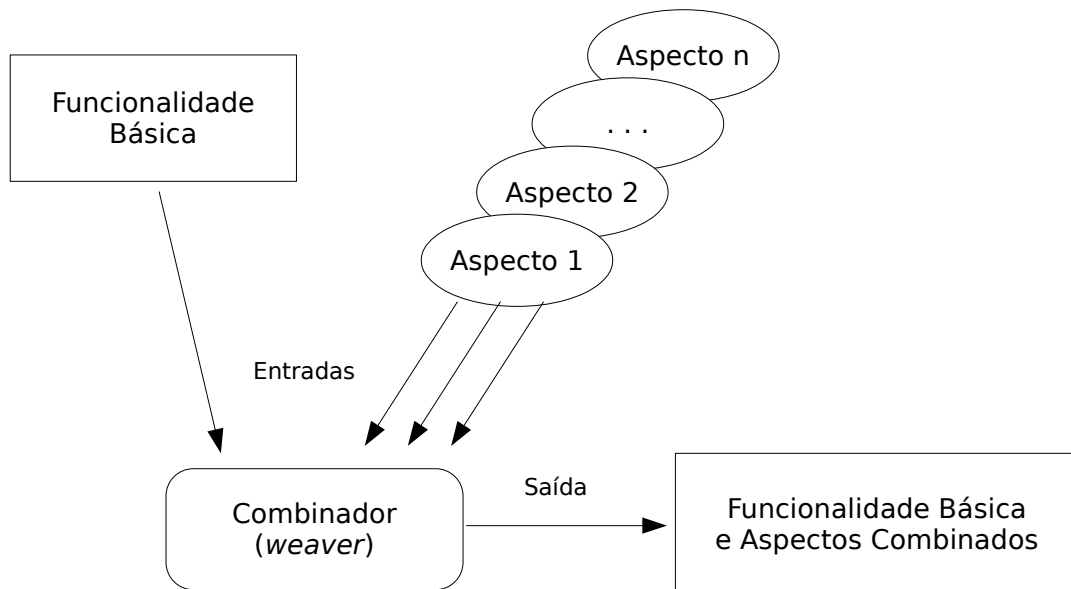


Figura 2.2: Elementos Básicos a um sistema POA

Os aspectos, diferentemente de uma classe da POO, podem alterar o comportamento dos elementos existentes no software sem que esses elementos tenham conhecimento da atuação dos aspectos. Os aspectos encapsulam os interesses transversais, que de outra forma ficariam entrelaçados ou espalhados em outros módulos do software. Em um aspecto, o comportamento relacionado a um interesse transversal é implementado com o conceito de *advice* (do inglês, *advice*), e em momento oportuno é acionado durante a execução do software. O momento de execução de um *advice* é definido por um ponto de junção (do inglês, *join point*), que se refere

a um ponto bem definido da estrutura estática ou dinâmica de um software (por exemplo, a chamada a uma função, a definição de uma variável ou a instanciação de um objeto).

Levando isso em consideração, tem-se uma propriedade inerente a POA, a *inconsciência* (FILMAN; FRIEDMAN, 2004), que permite ao programador desenvolver o software tendo conhecimento apenas das funcionalidades básicas, sem a necessidade de ter conhecimento de todo o domínio. Nessa abordagem, outros interesses podem ser desenvolvidos separadamente e depois combinados à aplicação base. Outra propriedade, a *quantificação*, permite ao programador a execução de uma determinada ação em um programa, a qualquer momento da aplicação, dado que uma determinada condição seja satisfeita. É importante ainda ressaltar que para o restante do estudo serão utilizados os termos originais dos elementos.

2.6.2 Abordagens de Teste de Mutação para Programas OA

A Abordagem de Mortensen e Alexander (2005)

O teste de mutação tem sido amplamente pesquisado e estudado. Na literatura de teste de software, a primeira iniciativa para o aplicar o teste de mutação em programas OA foi descrita por Mortensen e Alexander (2005).

A abordagem proposta pelos autores combina os teste estrutural e o teste de mutação e tem como referência a linguagem AspectJ. Em relação ao teste de mutação, são definidos três operadores. Dois deles têm como alvo os descritores de conjuntos de junção (DCJs), que consistem em expressões predicativas que identificam os pontos de junção em um programa, e o terceiro foca expressões que definem a precedência entre aspectos que podem atuar em pontos de junção coincidentes. A descrição dos operadores é apresentada seguir:

- (i) *Pointcut strengthenig (PCS)*: diminui o número de pontos de junção capturados e são feitos de diversas maneiras para os operadores de classe, por exemplo, é alterado o nome do tipo pai para um tipo filho, ou para nome de ponto de corte que coincide com vários métodos;
- (ii) *Pointcut weakening (PCW)*: faz o contrário do operador *Pointcut strengthenig*, para os operadores de classe muda-se o nome do tipo para o nome do tipo acima na hierarquia;
- (iii) *Precedence changing (PRC)* ao modificar a precedência do aspecto, é possível determinar a sensibilidade dos conjuntos de testes para ordenar os comportamentos dos aspectos, a fim de maximizar os benefícios, o teste de mutação normalmente aplica seletivamente operadores para apenas algumas partes do sistema. Mortensen e Alexander afirmam que o teste de mutação, aplicado com os operadores propostos, podem ser utilizados para revelar

defeitos relacionados ao escopo de conjuntos de junção e precedência de aspectos. Além disso os critérios estruturais podem ser empregados com o objetivo de encontrar os demais tipos de defeitos.

A Abordagem de Anbalagan e Xie (2008)

Outra abordagem para o teste de mutação de programas OA foi definida por Anbalagan e Xie (2008). Os autores propõem um *framework* para a geração de mutantes e para detecção automática de mutantes equivalentes ao programa original. Essa abordagem foca exclusivamente o teste de descritores de conjuntos de junção (DCJs). Os autores baseiam-se na afirmativa de que DCJs mutantes são efetivos quando variam pouco sintáticos e semanticamente quando comparado ao DCJs original.

O *framework* proposto utiliza a ferramenta AJTE (YAMAZAKI et al., 2005) para identificar todos os possíveis módulos-base da aplicação. O AJTE fornece apoio ao teste de programas OA sem necessidade de combinar os aspectos com as classes. Para isso, o AJTE fornece uma API que permite representar DCJs e pontos de junção como objetos. Após esse processo de identificação, os DCJs são gerados. O *framework* proposto por Anbalagan e Xie (2008) produz mutantes de DCJs baseados em dois operadores de mutação: *pointcut strengthenig* e *pointcut weakening*, que têm como objetivo diminuir ou aumentar o número de pontos de junção selecionados pelo DCJ, criando assim mutantes a partir do DCJ original.

No caso de mais de um DCJ mutante capturar o mesmo conjunto de pontos de junção, o DCJ que contiver a string mais longa, que em tese é menos genérica do que as demais e menos propensa a apresentar defeitos, será escolhido para apoiar a identificação e seleção do “melhor mutante“. Tanto o DCJ mutante como o DCJ original são submetidos a um classificador de mutantes, que os classificam como *neutro* (mutantes equivalentes), *fraco* (DCJ original é um subconjunto do DCJ mutante) e *forte* (DCJ mutante é um subconjunto do DCJ original). Essa classificação permite ao testador escolher o melhor DCJ, que é o mutante mais semelhante ao DCJ original.

A Abordagem de Ferrari, Maldonado e Rashid (2008)

Ferrari, Maldonado e Rashid (2008), baseados nos tipos de defeitos agrupados em uma taxonomia de defeitos, definiram um conjunto de operadores de mutação para testes em programas OA. Esses tipos de defeitos e seus respectivos operadores foram agrupados de acordo com os principais conceitos da POA sendo eles: (F1) expressões de *pointcut* (ou seja, DCJs); (F2)

Declarações inter-tipos e outras declarações; (F3) definição e implementação de *advices*; e (F4) a aplicação base.

Baseados nesses tipos de defeitos, e seguindo as construções sintáticas possíveis permitidas pela linguagem AspectJ e linguagens semelhantes, os autores definiram os três grupos de operadores que são apresentados a seguir, e os operadores referentes a cada um desses grupos são apresentados com mais detalhes na Tabela 2.1:

- **Grupo 1 - Operadores para DCJs:** esse grupo contém 15 operadores de mutação, relacionados aos tipos de defeitos possíveis em DCJs. Esse grupo foi subdividido em 4 categorias:
 - **Operadores *Pointcut Weakening*:** é composto pelos operadores PWSR, PWIW e PWAR; os mutantes resultantes possibilitam o aumento do número de pontos de junção selecionados se comparado ao DCJ original.
 - **Operadores *Pointcut Strengthening*:** é composto pelos operadores PSSR, PSWR e PSDR; esse grupo de operadores realizam modificações para reduzir o número de pontos de junção selecionados se comparado ao DCJ original.
 - **Operadores *Pointcut Weakening or Pointcut Strengthening*:** Os operadores desse grupo, POPL, POAC e POEC, produzem mutantes que podem aumentar ou reduzir o número de pontos de junção selecionados se comparado ao DCJ original.
 - **Operadores para modificação de *Pointcuts*:** operadores desse grupo, PCTT, PCGS, PCCR e PCLO produzem uma variedade de modificações em DCJs.
- **Grupo 2 - Operadores para declarações gerais:** Contém 5 operadores baseados nos tipos de defeitos relacionado às declarações gerais do AspectJ. Esses defeitos levam a execuções de fluxo de controle e a estados de objeto/aspectos indesejáveis. Os operadores definidos nesse grupo são: DAPC, DAPO, DSSR, DEWC e DAIC.
- **Grupo 3 - Operadores para definição e implementação de *advices*:** Grupo de 6 operadores relacionados a defeitos associados à definição e implementação de *advices*. Os operadores desse grupo modelam defeitos relacionados ao tipo de *advices* (ABAR), lógica incorreta do *advices* (APSR, APER e AJSC), e execução incorreta do *advices* (ABHA, ABPR).

Posteriormente, Ferrari et al. (2010) propuseram a ferramenta *Proteum/AJ* para automatizar o teste de mutação tendo com base os operadores de mutação propostos. A *Proteum/AJ* realiza

Tabela 2.1: Descrição dos operadores de mutação para AspectJ, tabela adaptada de (FERRARI; MALDONADO; RASHID, 2008)

Operador	Descrição
PWSR	Modifica Pointcut por meio da substituição de um tipo por seu supertipo imediato.
PWIW	Modifica Pointcut por meio da inserção de caracteres coringas dentro dele.
PWAR	Modifica Pointcut por meio da remoção marcações de tipo, campo, método e padrões de construtor.
PSSR	Modifica Pointcut por meio da substituição de um tipo por seu subtipo imediato.
PSWR	Modifica Pointcut por meio da remoção de caracteres coringas dentro dele.
PSDR	Modifica Pointcut por meio da remoção de declarações "declare @" do código do aspecto.
POPL	Modifica Pointcut por meio da alteração das listas de parâmetros do Pointcuts primitivos.
POAC	Modifica Pointcut por meio da alteração das cláusulas advices "after [retuning throwing]".
POEC	Modifica Pointcut por meio da alteração das cláusulas exception throwing.
PCTT	Modifica Pointcut por meio da substituição de "this" por "target" e vice versa.
PCCE	Modifica o contexto pelas substituições "call/execution/initialization/ preinitialization" nos Pointcuts
PCGS	Modifica Pointcut pela substituição de "get" por "set" e vice versa.
PCCR	Modifica Pointcut pela substituição de partes individuais da própria composição do Pointcut.
PCLO	Modifica Pointcut modificando os operadores lógicos presentes no tipo e composição dos Pointcuts.
PCCC	Modifica Pointcut pela substituição de "cflow" por "cflowbelow" e vice versa.
DAPC	Modifica precedência do Aspecto por meio da alteração da ordem das expressões <code>declare precedence</code> .
DAPO	Modifica um aspecto por meio de uma remoção arbitrária de expressão "declare precedence".
DSSR	Modifica tratamento de exceção por meio da remoção da expressão "declare soft".
DEWC	Modifica controle de fluxo de execução por meio de alteração de expressão "declare error/warning".
DAIC	Modifica instanciação do aspecto pela alteração das cláusulas "perthis/pertarget/percflow/percflowbelow".
ABAR	Modifica tipo de Advice por meio da substituição da cláusula <code>before</code> por uma <code>after [retuning throwing]</code> e vice versa.
APSR	Modifica lógica do Advice por meio da remoção de invocações para declarações "proceed".
APER	Modifica lógica do Advice por meio da remoção das condições de guarda que circundam declarações "proceed".
AJSC	Modifica origem de informação estática por meio da substituição de referência "thisJoinPoint-StaticPart" por uma "thisEnclosingJoinPointStaticPart" e vice versa.
ABHA	Modifica comportamento por meio da remoção de advices.
ABPR	Modifica vínculo Advice-Pointcuts por meio da substituição de Pointcuts que são limitados a advices.

um conjunto de requisitos necessários a uma ferramenta de teste de mutação, de acordo com a proposta de Nakagawa et al. (2007) para uma arquitetura de referência para ferramentas de testes denominada *RefTEST*. Além disso, a *Proteum/AJ* supera limitações que foram identificadas nas ferramentas anteriores para programas como, por exemplo, o *framework* proposto por Anbalagan e Xie (2008) e a ferramenta AjMutator, que é descrita na próxima seção.

A *Proteum/AJ* apoia os quatro principais passos para teste de mutação, descritos originalmente por (DEMILLO; LIPTON; SAYWARD, 1978): (i) o programa original é executado em um conjunto de testes e o resultado é armazenado; (ii) os mutantes são criados baseados em uma seleção de operadores de mutação que podem evoluir em uma nova interação do ciclo de teste; (iii) os mutantes podem ser executados de uma única vez ou individualmente, assim como o conjunto de casos de teste pode ser aumentado ou reduzido baseado na estratégia a ser utilizada; e (iv) os resultados são avaliados a fim de classificar os mutantes remanescentes como

equivalentes ou decidir pela evolução do conjunto de testes.

Na Figura 2.3 são mostrados com mais detalhes os quatro passos básicos descritos anteriormente e as principais funcionalidades da ferramenta *Proteum/AJ*.

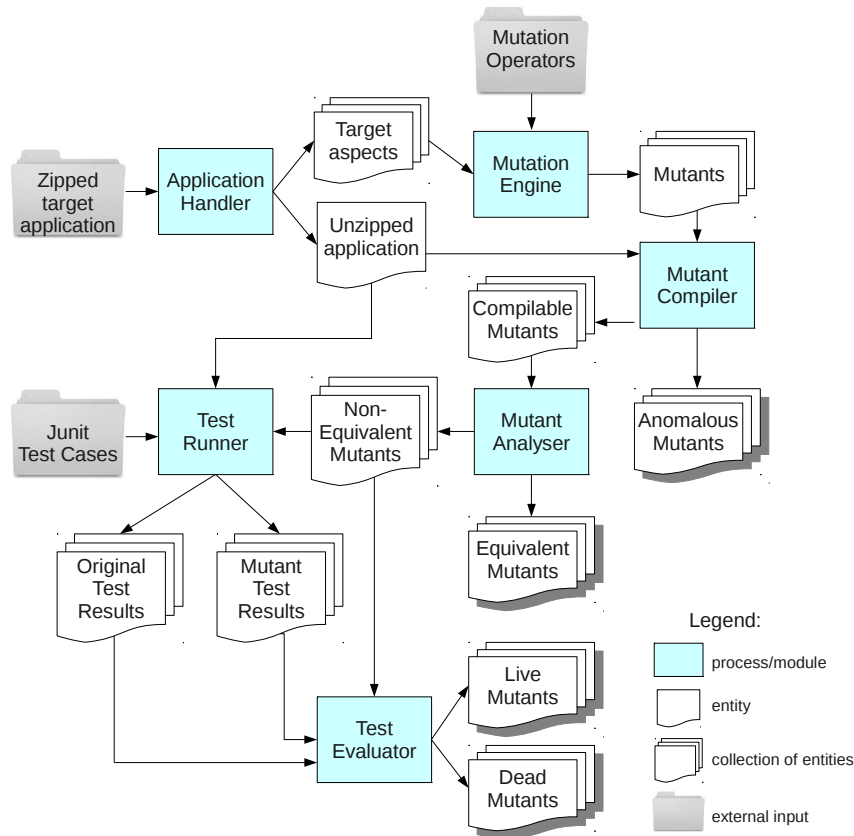


Figura 2.3: Principais funcionalidades da ferramenta *Proteum/AJ*, adaptado do trabalho de Ferrari (2010).

A Ferramenta AjMutator (DELAMARE; BAUDRY; Le Traon, 2009)

Delamare, Baudry e Le Traon (2009) apresentaram a AjMutator, uma ferramenta para auxílio na análise de mutação dos DCJs. Os operadores de mutação implementados são parte do conjunto de operadores para AspectJ proposto por Ferrari, Maldonado e Rashid (2008), todos relacionados a DCJs. Depois de gerados os mutantes, eles são compilados e classificados automaticamente. Para essa compilação, a ferramenta conta com o auxílio do compilador *abc* (abc Development Team, 2009), um compilador alternativo para AspectJ. O compilador produz um arquivo '.jar' para cada mutante, que contém informações como, por exemplo, sobre os pontos de junção capturados pelos DCJs, e essas informações são passadas para a AjMutator fazer a classificação dos mutantes.

A classificação automática é realizada separando os mutantes em classes (classe 1, classe

2, classe 3) de acordo com as características de cada um, caso o mutante não se enquadre em nenhuma dessas classes então ele é considerado um mutante equivalente e não é selecionado para análise de mutação. A precisão desse processo vai depender se o Pointcut dos mutantes é estático ou dinâmico.

A Investigação de Wedyan e Ghosh (2011)

Em um trabalho mais recente, Wedyan e Ghosh (2011) realizaram um estudo em que eles propuseram o uso de uma abordagem de análise simples dos objetos dos programas para evitar a geração de mutantes equivalentes para alguns operadores de mutação. Foram abordados três tipos de problemas relacionados ao teste de mutação de programas OA, sendo eles: (i) a identificação e a remoção de mutantes equivalentes são difíceis e demoradas; (ii) os mutantes gerados precisam cobrir vários tipos de defeitos descritos na literatura de modelos de defeitos para programas em AspectJ; e (iii) a dificuldade em matar os mutantes gerados.

Nesse estudo foram utilizadas três ferramentas de teste de mutação, sendo duas específicas para o teste de mutação de programas OA (Proteum/AJ e AjMutator) e uma terceira ferramenta, a *MuJava* (MA; OFFUTT; KWON, 2005), criada para o teste de mutação de programas OO escritos em Java.

Com essa nova proposta de abordagem de análise, os autores mostram que consideráveis reduções no número de mutantes equivalentes podem ser obtidas. Os resultados são sumarizados na Tabela 2.2, que foi extraída e adaptada do trabalho original Wedyan e Ghosh (2011).

Tabela 2.2: Resultados do estudo de Wedyan e Ghosh (2011).

Ferramenta	#M	Operadores	%Antes	%Depois	%Redução
<i>AjMutator</i>	249	PWIW, POPL	77.5	32.6	67.4
<i>Proteum/AJ</i>	423	PWIW, POPL, PSWR	50.1	24.1	79.9
<i>MuJava</i>	1262	Class, AOIS	21.2	7.5	92.5
Todas	1934		34.8	19.9	80.1

Na Tabela 2.2 na coluna #M têm-se a quantidade total de mutantes gerados pelo conjunto total de operadores para cada ferramenta, a coluna *Operadores* mostra os operadores que podem ser utilizados nas abordagens apresentadas, e nas três últimas colunas são mostradas as porcentagens de mutantes equivalentes antes e depois da aplicação da abordagem proposta no estudo e de redução obtida por meio da aplicação dessa abordagem.

Em termos gerais, é possível observar que a abordagem obteve uma taxa de redução em média de 80%, comprovando assim a eficácia da mesma.

2.7 Considerações Finais

Neste capítulo foi apresentada uma revisão bibliográfica sobre teste de software. Foram abordados os principais conceitos e a importância do teste para garantir a qualidade dos produtos criados. Apresentou-se também uma visão geral das principais técnicas e critérios de teste, a necessidade de automatizar essa atividade, e como os diferentes critérios podem ser avaliados teórica e/ou experimentalmente.

Deu-se ênfase ao critério de Análise de Mutantes, também conhecido como *teste de mutação*, pois esse é o contexto o qual este trabalho está inserido. Abordagens de teste de mutação de programas OA foram descritas, como forma de embasar a Proposta de Trabalho apresentada no Capítulo 1 deste documento.

No próximo capítulo são apresentados os resultados de um mapeamento sistemático cujo objetivo foi caracterizar as estratégias de redução de custo do teste de mutação, que consiste no objetivo para o trabalho a ser realizado.

Capítulo 3

ESTRATÉGIAS DE REDUÇÃO DE CUSTO DO TESTE DE MUTAÇÃO

3.1 Considerações Iniciais

O teste de mutação tem sido amplamente investigado, porém, ainda é considerado como critério de testes bastante caro devido à demanda computacional necessária para criar, executar e analisar os mutantes criados. Visando tornar o teste de mutação uma prática mais difundida nas equipes de teste, muitos estudos têm sido realizados acerca da redução de custos para os teste de mutação.

Com o propósito de caracterizar o estado da arte das abordagens de teste de mutação voltadas para redução de custo, foi realizado um Mapeamento Sistemático da literatura. O Mapeamento Sistemático tem como objetivo fornecer uma visão geral da área de pesquisa, identificando e quantificando as pesquisas relacionadas bem como seus resultados obtidos (PETERSEN et al., 2008).

Os resultados do Mapeamento Sistemático realizado são relatados neste capítulo. Inicialmente, apresenta-se uma visão geral do protocolo definido para o Mapeamento Sistemático realizado (Seção 3.2), e após é realizado a sumarização e análise dos resultados (Seção 3.3).

3.2 Visão Geral do Protocolo e dos Processos Empregados

Para a realização do mapeamento sistemático descrito neste capítulo, seguiu-se o processo proposto por Ferrari e Maldonado (2008) em seu estudo, que é a adaptação de um processo proposto inicialmente por Biolchini et al. (2005). O processo original de Biolchini et al. (2005), especificamente as fases de execução e síntese dos resultados, pode ser descrito nos seguintes passos: realiza-se uma seleção preliminar que visita partes específicas dos estudos primários - por exemplo título, resumo e palavras-chave - com o objetivo de verificar se esses estudos recuperados pelas buscas nos repositórios são potencialmente relevantes para serem analisados na seleção final. Na seleção final o pesquisador realiza uma leitura completa do estudo primário e decide, de acordo com os critérios de inclusão e exclusão, pela inserção ou não do estudo para extração dos dados. Finalizada a seleção final é realizada então a extração dos dados dos estudos, que são armazenados em formulários específicos para cada pesquisa. Durante todo esse processo é realizada a atividade de documentação para permitir uma futura auditabilidade e replicabilidade do processo.

No processo adaptado por Ferrari e Maldonado (2008), ilustrado na Figura 3.1, tem-se uma preocupação maior nas atividades Planejamento da Atualização, Filtragem dos Resultados e Combinação dos Resultados. Essas atividades visam viabilizar não somente a realização de um Mapeamento Sistemático original, mas também a atualização e a replicação de estudos existentes. Esse apoio é obtido, por exemplo, com o ajuste dos critérios de seleção, com a criação de filtros para reduzir a sobreposição durante a seleção preliminar e com a mesclagem dos dados extraídos do conjunto de dados original.

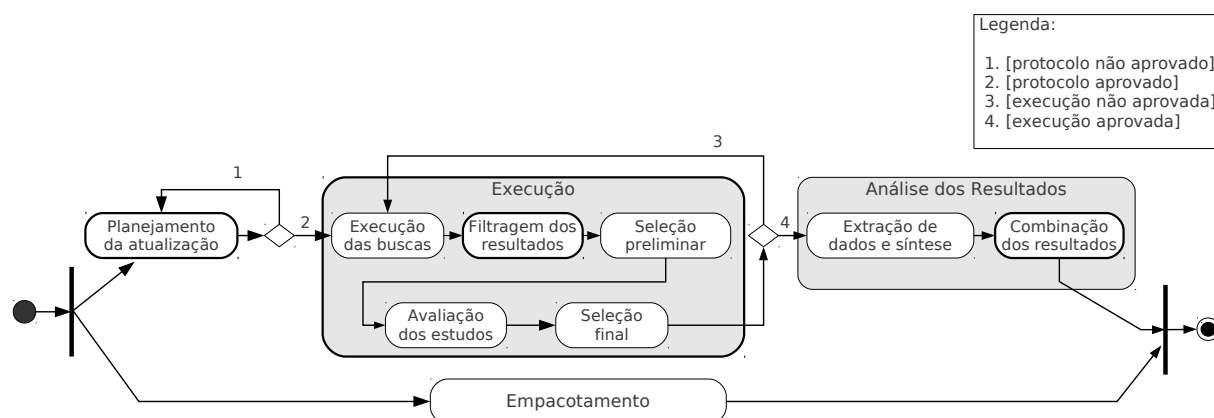


Figura 3.1: Processo para a realização do Mapeamento Sistemático, adaptado do trabalho de Ferrari e Maldonado (2008).

No Mapeamento Sistemático descrito neste capítulo, definiu-se no protocolo apenas um objetivo principal: identificar abordagens (teóricas ou experimentais) para reduzir os custos de

aplicação do teste de mutação. Para atingir esse objetivo, uma questão de pesquisa primária e duas questões secundárias foram investigadas. São elas:

- QP1: Quais abordagens são utilizadas para a redução de custos em teste de mutação?
- QS1: Quais os tipos de objetos são abordados nos estudos que investigam redução de custos em testes de mutação?
- QS2: Quais os operadores de mutação são identificados para compor um conjunto de operadores essenciais?

3.2.1 Critérios de Inclusão e Exclusão

Para apoiar a seleção objetiva de estudos primários, foram definidos e aplicados os seguintes critérios de inclusão (IC) e exclusão (EC) nos estudos:

- IC1: O estudo aborda uma estratégia de redução de custo para teste de mutação.
- EC1: O estudo não aborda uma estratégia de redução de custos para teste de mutação, apesar de o estudo poder ser caracterizado como um estudo relacionado a teste de mutação.
- EC2: O estudo, apesar de abordar a redução de custos para teste de mutação, não está disponível para *download*.

3.2.2 String de Busca

Para obter sucesso em um mapeamento sistemático, é fundamental definir uma boa string de busca que consiga recuperar eficientemente os estudos primários de interesse. Embora cada base de dados contenha a sua própria sintaxe para construção das strings, foi construída uma string genérica utilizando-se como base a sintaxe empregada nas strings da máquina de busca do repositório *ScienceDirect*¹, por ser uma sintaxe mais amigável dentre todas as bases escolhidas para realização das buscas. A string genérica definida foi a seguinte:

“TITLE-ABS-KEY(“mutation testing” OR “mutation analysis” OR “mutant analysis”) AND TITLE-ABS-KEY(“cost reduction” OR “sufficient operator” OR “sufficient mutation” OR “constrained mutation” OR “selective mutation” OR “weak mutation” OR “random selection” OR “random mutation” OR “random mutants”)”

¹<http://www.sciencedirect.com/> acessado em 13/05/2014

3.2.3 Fontes de Estudos Seleccionadas e Resultados

As bases de dados escolhidas para o Mapeamento Sistemático trabalho foram: *ACM Digital Library*,² *IEEE explore*,³ *ScienceDirect*, *Springer Link*,⁴ *Wiley Online Library*.⁵ Essas bases foram escolhidas por se tratarem de repositórios de referência para a divulgação de pesquisas relacionadas ao tópico de pesquisa deste trabalho.

Esse mapeamento trata-se de uma atualização de um mapeamento realizado anteriormente, no período de Novembro de 2012 a Fevereiro de 2013, o qual foi apresentado na dissertação de qualificação do próprio autor deste estudo. A atualização desse mapeamento foi realizada no período de Maio de 2014 a Junho de 2014. Os resultados obtidos com a aplicação da *string* genérica no mapeamento anterior e na atualização deste mapeamento são denominados “Estudos Ant” e “Estudos Novos” respectivamente e são apresentados nos tópicos a seguir.

- **Busca nos Repositórios:** Para a primeira busca nas bases de dados, utilizando a *string* genérica definida, foram retornados os resultados mostrados na Tabela 3.1. Na tabela, estão ordenados o nome das bases de dados e a quantidade de estudos recuperados de cada uma delas.

Tabela 3.1: Quantidade de estudos retornados por cada base de dados

Base de Dados	#Estudos Ant.	#Estudos Novos
ACM	7	1
IEEEExplore	25	4
ScienceDirect	7	0
SpringerLink	4	1
Wiley	198	-
Total	241	6

- **Pré-seleção:** Na Tabela 3.2 são apresentados os resultados referentes à etapa de pré-seleção dos artigos, com o número de estudos aceitos, rejeitados e duplicados por base de dado. Essa etapa consistiu na análise do *título*, *resumo* e *palavras-chaves* de cada estudo. Em alguns casos considerados inconclusivos, foi feita uma análise da introdução e conclusão de cada artigo.
- **Seleção Final:** Na seleção final formou-se um conjunto de 23 estudos para compor o mapeamento sistemático. Observa-se que dois dos estudos da etapa de pré-seleção foram

²<http://dl.acm.org/> acessado em 13/05/2014

³<http://ieeexplore.ieee.org/Xplore/home.jsp> acessado em 13/05/2014

⁴<http://link.springer.com/> acessado em 13/05/2014

⁵<http://onlinelibrary.wiley.com/> acessado em 13/05/2014

Tabela 3.2: Quantidade de estudos aceitos, rejeitados e duplicados na etapa de pré-seleção

Base de Dados	#Ant.	#Novos	#Aceitos	#Rejeitados	#Duplicados
ACM	7	1	5	1	2
IEEEExplore	25	4	11	18	0
ScienceDirect	7	0	5	1	1
SpringerLink	4	1	0	5	0
Wiley	198	-	7	138	53
Total	241	6	28	163	56

excluídos, pois apesar dos estudos estarem relacionados a teste de mutação, ambos não focam em uma estratégia específica para redução de custo. Além dos estudos selecionados na etapa de pré-seleção, um estudo foi inserido manualmente pois a *string* de busca utilizada não conseguiu cobrir esse estudo. Esse estudo foi identificado como de interesse após a análise dos trabalhos relacionados citados nos estudos selecionados para o Mapeamento Sistemático

Os resultados da seleção final são apresentados a seguir na Tabela 3.3

Tabela 3.3: Quantidade de estudos Extraídos para revisão bibliográfica

Base de Dados	Etapa Pré-Seleção	Seleção Final	Rejeitados
ACM	5	5	0
IEEEExplore	11	9	2
ScienceDirect	5	5	0
SpringerLink	0	0	0
Wiley	7	7	0
Especialista	7	7	0
Total	35	33	2

3.3 Sumarização dos Resultados

Os estudos selecionados no Mapeamento Sistemático foram classificados de acordo com um conjunto de características específicas a cada estudo. Cada característica é denominada *faceta*. Nesta seção são apresentadas a análise individual das facetas consideradas, sendo elas *Linguagem-Alvo*, *Tipo de Abordagem* e *Tipo de Objeto*, e uma análise cruzada das duas primeiras facetas baseada em um gráfico de bolhas.

3.3.1 Análise Individual das Facetas

Linguagem-Alvo

A primeira faceta refere-se às linguagens de programação nas quais as abordagens de redução do custo do teste de mutação estão focadas. Foram identificadas as seguintes linguagens de

Tabela 3.4: Descrição dos estudos retornados

Autor	Ano	Abordagem	Tipo de Objeto	Linguagem Alvo
DUNCAN; ROBSON	1990	Outras abordagens	Programas	Sem linguagem Alvo
MARSHALL et al.	1990	Outras abordagens	Programas	Sem linguagem Alvo
WEISS; FLEYSHGAKKER	1993	Algoritmo em serie para analise de mutacao	Programas	Sem linguagem Alvo
OFFUTT; ROTHERMEL; ZAPF	1993	Mutação Seletiva	Programas	Fortran
OFFUTT; LEE	1994	Mutação fraca	Programas	Fortran
OFFUTT et al.	1996	Operadores Essenciais	Programas	Fortran
MRESA; BOTTACI	1999	Operadores Essenciais	Programas	Fortran
HIERONS; HARMAN; DANICIC	1999	Fatiamento de programa	Programas	Imperativa
BARBOSA; MALDONADO; VINCENZI	2001	Operadores Essenciais	Programas	C
MA; OFFUTT; KWON	2005	Combinação de tecnicas	Programas	Java
NAMIN; ANDREWS	2006	Redução Variável	Programas	C
TUYA; SUAREZ-CABAL; RIVA	2007	Mutação Seletiva	Programas	SQL
NAMIN; ANDREWS; MURDOCH	2008	Operadores Essenciais	Programas	C
UNTC	2009	SDL-Mutation	Programas	C
POLO; PIATTINI; Garcia-Rodriguez	2009	Mutação de maior ordem	Programas	Java
PAPADAKIS; MALEVRIS; KALLIA	2010	Outras abordagens	Programas	Java
ZHANG et al.	2010	Combinação de tecnicas	Programas	C
KINTIS; PAPADAKIS; MALEVRIS	2010	Mutação de maior ordem	Programas	C
DOMÍNGUEZ-JIMÉNEZ et al.	2011	Teste de Mutação Evolucionário (EMT, do inglês, <i>Evolutionary Mutation Test</i>)	Processo/ Método	C ou Java
KAMINSKI et al.	2011	Mutação Seletiva	Programas/ queries	SQL SQL
MATEO; USAOLA	2012	Otimização da compilação e/ou execução de mutantes	Programas	Sem linguagem Alvo
DURELLI; OFFUTT; DELAMARO	2012	Otimização da compilação e/ou execução de mutantes	Programas	Java
PAPADAKIS; MALEVRIS	2012	Outras abordagens	Programas	Sem linguagem Alvo
KIM; MA; KWON	2012	Combinação de tecnicas	Programas	Java
GLIGORIC et al.	2012	Combinação de tecnicas	Programas	Java
WEDYAN; GHOSH	2012	Mutação de maior ordem	Processo/ Método	AspectJ
OMAR; GHOSH	2012	Mutação de maior ordem	Programas	AspectJ
GLIGORIC et al.	2013	Mutação Seletiva	Programas	Java
MATEO; USAOLA; ALEMÁN	2013	Mutação de maior ordem	Programas	Java
ZHANG et al.	2013	Mutação Aleatória	Programas	Java
DENG; OFFUTT; LI	2013	Mutação SDL	Programas	Java
DELAMARO et al.	2014	Combinação de Técnicas	Programas	C
AMMANN; DELAMARO; OFFUTT	2014	Combinação de tecnicas	Processo/ Método	C

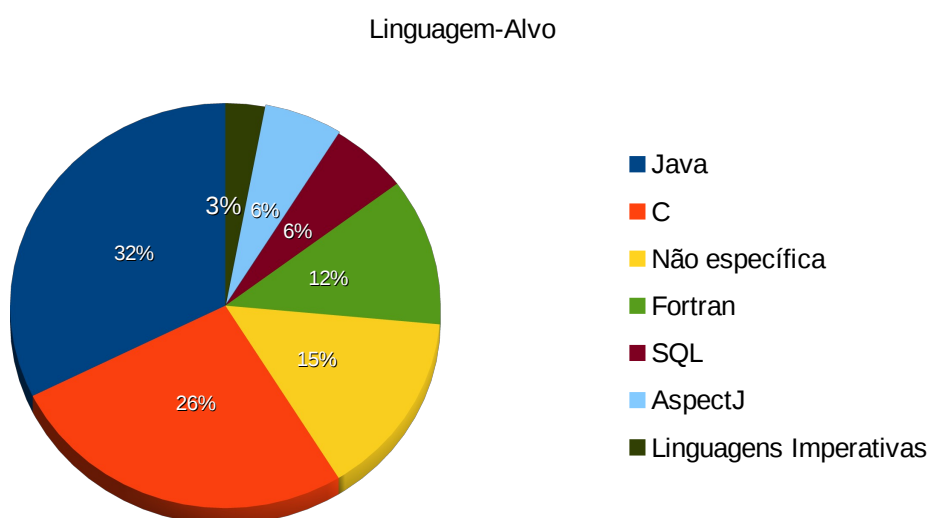
programação: C, Java, Fortran, SQL e AspectJ. Além dessas linguagens, percebeu-se que algumas dessas abordagens focavam em linguagens imperativas⁶ ou não focavam em uma linguagem específica. Os resultados são sumarizados na Tabela 3.5 e ilustrados na Figura 3.2.

De acordo com os resultados quantificados, é possível perceber que duas linguagens de paradigmas distintos se destacaram, são elas: a linguagem C (paradigma procedural) que representa 26% do total e a linguagem Java (paradigma orientado a objetos) que representa 32% do total de artigos. De acordo ainda com a tabela e o gráfico mostrados é possível perceber que cinco estudos (15% do total) que abordam redução de custos não focam em nenhuma linguagem específica e um estudo (3% do total), apesar de não focar em uma linguagem específica, foca

⁶Linguagem imperativa, ou procedural, é o tipo de linguagem que mudam o comportamento ou o estado de um programa através de ações, comandos ou enunciados.

Tabela 3.5: Quantidade de estudos por Linguagem-alvo

Linguagem	#Estudos Ant	#Estudos Novos
Java	7	4
C	6	3
Fortran	4	0
SQL	2	0
AspectJ	1	1
Não especifica	5	0
Linguagens Imperativas	1	0
Total	26	8

**Figura 3.2: Gráfico quantitativo das linguagens-alvo dos estudos primários.**

em linguagens do paradigma imperativo.

Pode-se perceber que na primeira execução desse mapeamento sistemático foi encontrado apenas um estudo que visava redução de custo para programas OA, e durante a atualização desse mapeamento foi possível detectar um estudo mais recente com foco em programas OA.

Tipo de Abordagem

Na seleção final dos artigos foram identificados treze tipos de abordagens nas quais os artigos foram classificados. As abordagens identificadas são as seguintes:

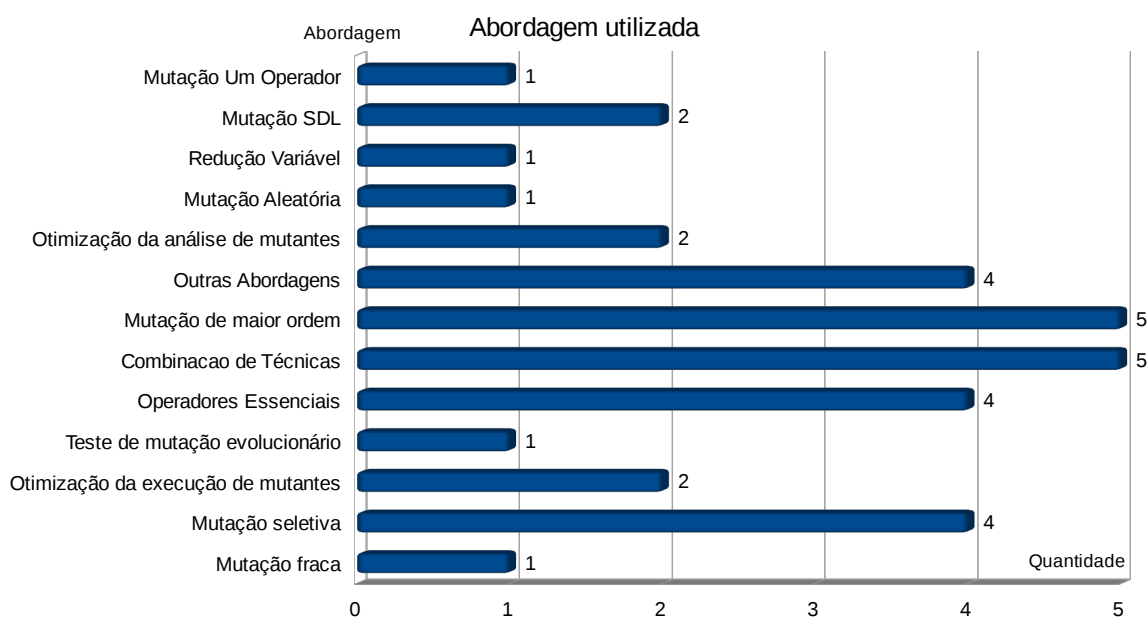
1. *Mutação fraca* – abordagem otimiza a “mutação forte” verificando os mutantes logo após o seu ponto de execução, ao invés de verificar após a execução de todo o programa;

2. *Mutação seletiva* – abordagem que propõe a geração de mutantes utilizando um conjunto reduzido de operadores;
3. *Mutação aleatória* – abordagem que considera apenas uma parte dos mutantes gerados pelos operadores, $x\%$ desses mutantes são escolhidos aleatoriamente para análise de mutantes;
4. *Otimização da execução dos mutantes* – abordagens que propõem a aceleração da execução dos mutantes;
5. *Teste de mutação evolucionário* – abordagem que visa reduzir os mutantes por meio de algoritmos evolutivos;
6. *Operadores essenciais* – abordagem que visa determinar um conjunto essencial de operadores para geração reduzida, porém eficiente de mutantes;
7. *Deleção de sentenças* – também conhecida como Mutação SDL do inglês, *SDL Mutation (statement deletion operator)*, utiliza um operador para gerar mutantes por meio da deleção de instruções no código.
8. *Redução variável* – abordagem formula o problema da mutação seletiva como um problema estatístico, e aplica métodos estatísticos lineares para identificar um subconjunto de operadores de mutação;
9. *Mutação Um Operador* – também conhecida pelo seu termo em inglês *One-op Mutation*. Utiliza apenas um operador mais relevante do conjunto.
10. *Combinação de técnicas* – abordagens que propõem o uso conjunto de duas ou mais técnicas já existentes;
11. *Mutação de maior ordem* – do inglês, *higher order mutation*, é uma abordagem que aplica os operadores de mutação mais de uma vez para geração de mutantes;
12. *Otimização da análise de mutantes* – abordagens que facilitam ou aceleram a análise dos mutantes; e
13. *Outras abordagens* - abordagens que propõem ou que se utilizam de framework, modelos, fluxos de dados, fluxo de controle para reduzir custos.

Com os resultados sumarizados na Tabela 3.6 e mostrados com mais clareza pelo gráfico da Figura 3.3, é possível perceber que as abordagens que mais se destacaram pela quantidade de estudos em que foram utilizadas são *Combinação de Técnicas* (MA; OFFUTT; KWON, 2005; ZHANG

Tabela 3.6: Quantidade de estudos por abordagem

Abordagem	#Estudos Ant	#Estudos Novos
Otimização da análise de mutantes	2	0
Outras Abordagens	4	0
Mutação de maior ordem	3	2
Combinação de Técnicas	4	1
Operadores Essenciais	4	0
Teste de mutação evolucionário	1	0
Otimização da execução de mutantes	2	0
Mutação seletiva	3	1
Mutação fraca	1	0
Mutação Aleatória	0	1
SDL Mutation	1	1
Redução Variável	0	1
Mutação Um Operador	0	1
Total	25	8

**Figura 3.3: Gráfico quantitativo das abordagens utilizadas nos estudos**

et al., 2010; KIM; MA; KWON, 2012; GLIGORIC et al., 2012; AMMANN; DELAMARO; OFFUTT, 2014; DELAMARO et al., 2014) e *Mutação de Maior Ordem* (POLO; PIATTINI; Garcia-Rodriguez, 2009; KINTIS; PAPADAKIS; MALEVRIS, 2010; WEDYAN; GHOSH, 2012; OMAR; GHOSH, 2012; MATEO; USAOLA; ALEMÁN, 2013), seguidas por *Mutação Seletiva* (OFFUTT; ROTHERMEL; ZAPF, 1993; TUYA; SUAREZ-CABAL; RIVA, 2007; KAMINSKI et al., 2011; GLIGORIC et al., 2013) e *Operadores Essenciais* (OFFUTT et al., 1996; MRESA; BOTTACI, 1999; BARBOSA; MALDONADO; VINCENZI, 2001; NAMIN; ANDREWS; MURDOCH, 2008).

A *Combinação de técnicas* é um tipo de abordagem em que há a combinação de pelo menos duas técnicas de redução. Em dois estudos, por exemplo, observou-se que foram combinadas as técnicas *mutação fraca* e *mutação forte*, e em outro estudo foi utilizado a combinação de cinco técnicas diferentes, sendo que quatro dessas técnicas eram de otimização e uma era técnica de heurística.

Observou-se que a abordagem *Operadores Essenciais* é a terceira abordagem mais difundida dentre os estudos sobre redução de custos em testes de mutação, juntamente com *Mutação Seletiva* e *Outras Abordagens*. A determinação de um conjunto essencial de operadores de mutação tem se mostrado uma abordagem bastante eficiente, e por esse motivo ela foi o tipo de abordagem aplicada nesse trabalho.

Tipos de Objetos

Na sumarização dos estudos selecionados, percebeu-se que os tipos de objetos das abordagens de redução de custo, em sua grande maioria – 30 estudos – consistiam em programas, equivalente a 88.2% do total. Apenas quatro estudos (11.7% do total) abordavam outros tipos de artefatos. Destes, um estudo focava em expressões SQL e os outros 3 estudos – denominado por Métodos – não focavam em programas ou expressões em uma linguagem específica, mas sim em um *framework* conceitual, ou um modelo de referência para obter uma redução de custo. Esses resultados aparecem ilustrados na Figura 3.4.

3.3.2 Análise Cruzada das Facetas

Após a análise dos artigos selecionados e a definição de facetas para caracterização do tópico investigado, foi criado um gráfico de bolhas para auxiliar na interpretação dos dados coletados. No gráfico de bolhas são confrontadas duas facetas: *Linguagem-Alvo* e *Tipo de Abordagem*. O gráfico de bolhas dessas duas facetas é apresentado na Figura 3.5, tendo como objetivo elencar as linguagens de programação investigadas em cada abordagem identificada nos estudos primários. No gráfico, a faceta *Tipo de Abordagem* é representada no eixo horizontal e a faceta *Linguagem-Alvo* aparece no eixo vertical.

O cruzamento das facetas permite identificar o tipo de linguagem de programação utilizada em cada uma das abordagens. O valor presente em cada ponto de cruzamento dos eixos determina a quantidade de estudos primários relacionados a um determinado tipo de abordagem para uma determinada linguagem, ou seja, cada ponto de cruzamento entre os dois eixos do gráfico representa a quantidade de estudos retornados no Mapeamento Sistemático que possuem características em comum às duas facetas.

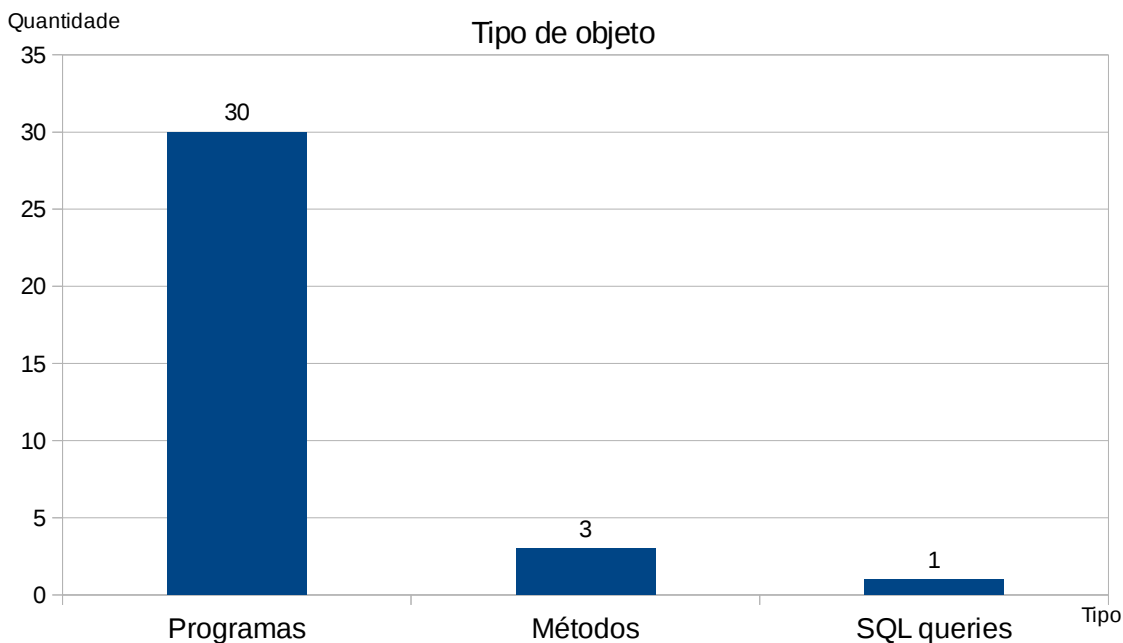


Figura 3.4: Gráfico quantitativo dos tipos de objetos abordados

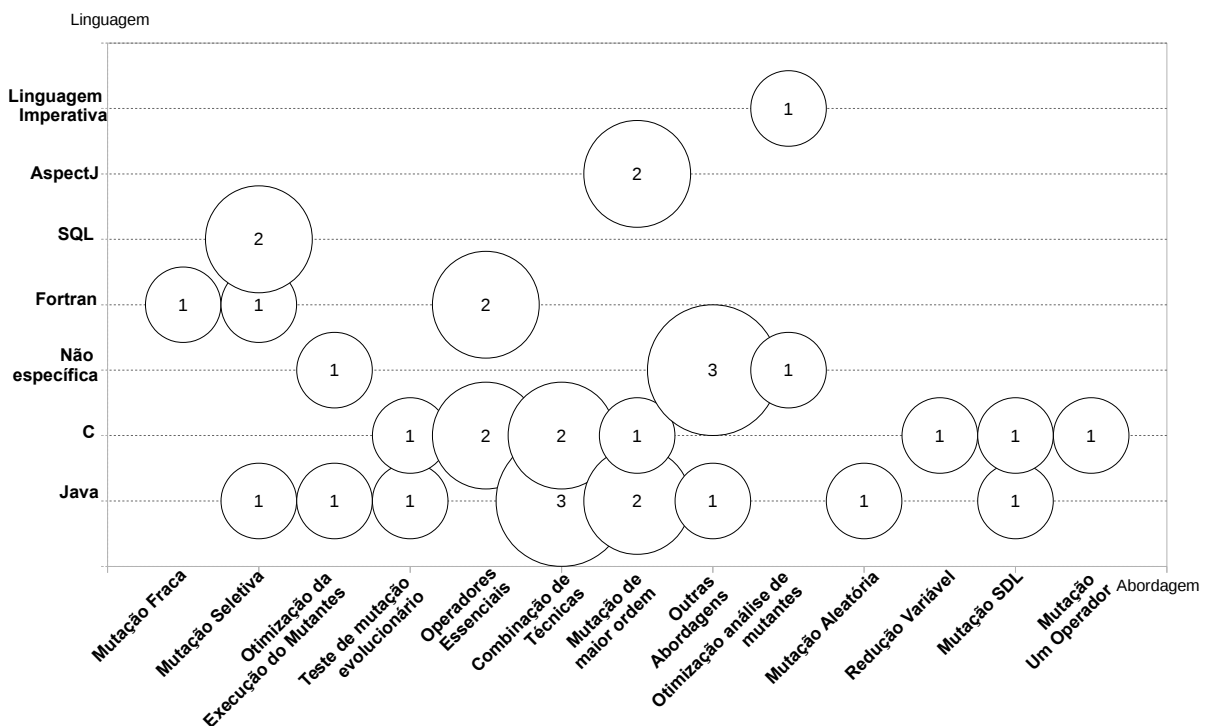


Figura 3.5: Gráfico de bolhas para as diferentes linguagens usadas em cada abordagem identificadas na revisão sistemática

De acordo com o cruzamento apresentado das facetas foi possível observar que:

(i) O uso apenas da abordagem de *mutação fraca* para reduzir custos tem se mostrado uma técnica pouco difundida na literatura, pois foi identificado apenas um estudo que relata o uso dessa técnica e a sua aplicação foi realizada em programas escritos na linguagem Fortran (OFFUTT; ROTHERMEL; ZAPF, 1993; OFFUTT; LEE, 1994; OFFUTT et al., 1996; MRESA; BOTTACI, 1999).

(ii) É possível perceber que em estudos mais recentes há também uma tendência a trabalhar com linguagens mais “recentes” como Java (MA; OFFUTT; KWON, 2005; POLO; PIATTINI; Garcia-Rodriguez, 2009; PAPADAKIS; MALEVRIS; KALLIA, 2010; DOMÍNGUEZ-JIMÉNEZ et al., 2011; DURELLI; OFFUTT; DELAMARO, 2012; GLIGORIC et al., 2012; KIM; MA; KWON, 2012; DENG; OFFUTT; LI, 2013; GLIGORIC et al., 2013; MATEO; USAOLA; ALEMÁN, 2013; ZHANG et al., 2013) e AspectJ (OMAR; GHOSH, 2012; WEDYAN; GHOSH, 2012). Além disso, consideram-se linguagens mais antigas, porém bem difundidas como a linguagem C (BARBOSA; MALDONADO; VINCENZI, 2001; NAMIN; ANDREWS, 2006; NAMIN; ANDREWS; MURDOCH, 2008; UNTCH, 2009; KINTIS; PAPADAKIS; MALEVRIS, 2010; ZHANG et al., 2010; DOMÍNGUEZ-JIMÉNEZ et al., 2011; AMMANN; DELAMARO; OFFUTT, 2014; DELAMARO et al., 2014).

(iii) Os estudos classificados como “Outras Abordagens”, por se tratarem de estudos que utilizam fluxo de dados, fluxo de controle, ou seleção de caminhos para criação de testes, não focam em nenhuma linguagem específica, mas sim no processo proposto pelos estudos. Apenas um estudo que utiliza *framework* como abordagem para obter uma redução de custo, possui foco na linguagem Java.

(iv) Pode-se observar ainda que para a linguagem AspectJ, linguagem-alvo deste estudo, há poucas abordagens de redução de custo.

Dentre as abordagens identificadas, é dado destaque às abordagens que têm como propósito a determinação um conjunto essencial de operadores, pois esse é o tipo de abordagem alvo deste trabalho. Os estudos selecionados que abordam a identificação de um conjunto essencial de operadores são descritos com mais detalhes na próxima seção.

3.4 Estratégias Baseadas em Operadores Essenciais

A seguir serão explanados os estudos do mapeamento sistemático que utilizaram operadores essenciais para reduzir consideravelmente os custos em teste de mutação. Do mapeamento sistemático realizado nesse trabalho foram identificados quatro estudos que se encontram nesse contexto, e eles estão dispostos nos tópicos a seguir, obedecendo a ordem cronológica de publicação dos mesmos.

3.4.1 A Investigação de Offutt et al. (1996)

Offutt et al. (1996) realizaram um estudo comparativo entre as abordagens de técnicas de redução de custo em teste de mutação. A investigação consiste em uma comparação entre a Mutação Seletiva (do inglês, *Selective Mutation*) (OFFUTT; ROTHERMEL; ZAPF, 1993) e o teste de mutação convencional. Segundo os resultados apresentados no trabalho, a mutação seletiva é quase tão forte quanto a mutação convencional. Para avaliar a hipótese proposta no trabalho, os autores criaram conjuntos de testes adequados em relação aos mutantes gerados com a abordagem de Mutação Seletiva e depois calcularam a adequação desses testes levando o conjunto completo de mutantes gerados.

Nesse trabalho, foi utilizado como objeto de estudo um conjunto de 10 programas escritos na linguagem Fortran, sendo que o tamanho médio de cada programa desse conjunto escolhido para o experimento é de cerca de 10 a 48 linhas executáveis. Como resultado da aplicação dos operadores de mutação, cada programa resultou em um faixa de 183 a 3010 mutantes.

Os seguintes passos foram realizados:

1. são criados para cada programa mutantes gerados de acordo com os operadores seletivos como ES-Selective (composto por dois grupos de operadores: *Expression modification* que contém os operadores ABS, AOR, LCR, ROR, UOI; e o grupo de operadores denominado *Statement modification* que contém os operadores DER, DSA, GLR, RSR, SAN, SDL. Os mutantes são gerados também baseando-se em mais outros três conjuntos seletivos: RS-Selective (composto pelos grupos de operadores: *Replacement of operand* que contém os operadores AAR, ACR, ASR, CAR, CNR, CRP, CSR, SAR, SCR, SRC e SVR e pelo grupo *Statement modification*), o RE-Selective (composto também por dois grupos: *Replacement of operand* e o *Expression modification*) e o E-Selective (que significa *expression-selective mutation* e é composto por apenas cinco operadores do grupo *Expression modification*);
2. são eliminados todos os mutantes equivalentes, determinados previamente de forma manual;
3. utilizando a ferramenta Godzilla (DEMILLO; OFFUTT, 1991) são gerados casos de teste para matar todos os mutantes não-equivalentes possíveis, foram gerados cinco conjunto separados de casos de teste seletivos, e gerados quatro variações diferentes para cada conjunto de teste seletivo para um total de dez programas, totalizando 200 casos de teste distintos;

4. para cada programa são criados todos os mutantes *não-seletivos*, ou seja, é gerado todos os mutantes utilizando todos os operadores (conjunto completo);
5. após, são eliminados todos os mutantes equivalentes do grupo dos não-seletivos;
6. cada conjunto de casos de teste seletivo é aplicado sobre os mutantes não seletivos criados;
7. depois é calculado o escore de mutação para cada conjunto de casos de teste, e o escore final de mutação mede a eficácia do conjunto de teste seletivo adequado de mutantes em conjuntos mutantes não seletivos e fornece uma estimativa do quão próximo a mutação seletiva está para mutação não-seletiva.

Os resultados obtidos no estudo mostraram que os conjuntos de testes que obtiveram escore com 100% de adequação para os mutantes seletivos, alcançaram também, aproximadamente, cerca de 100% de adequação para os não-seletivos. Os escores, em geral, alcançaram acima de 98% para os casos de teste RS-Selective, e acima de 99% para os programas em ES-Selective e RE-Selective. No estudo é mostrada uma percentagem de economia obtida pelos operadores, o cálculo dessa percentagem consistiu na subtração do número de mutantes seletivos pelo número de mutantes não seletivos divididos pela quantidade de mutantes não seletivos. A percentagem representa o número de mutantes que não precisam gerados e mortos com mutação seletiva, e os resultados alcançados variam de 6% a 72% dos mutantes. Ressalta-se também que a variação de números de casos de teste usada em cada programa afeta muito pouco no resultado, pois a variação no tamanho e número de mutantes mortos é muito pequena. Comparando o maior conjunto de casos de teste obteve um escore de mutação apenas 0,11% maior que o menor conjunto, ou seja, mesmo o menor dos conjuntos pode matar quase tanto quanto os maiores conjuntos.

Os resultados obtidos por Offutt et al. (1996) apontaram que a implementação da mutação “completa” (isto é, com um conjunto completo e extenso de operadores de mutação) é muito mais cara do que o necessário. Os autores reafirmaram a hipótese de que a mutação seletiva é eficaz e eficiente. Especificamente, os resultados apresentados no trabalho mostram que os operadores de mutação que substituem todos os operandos com todos os operandos sintaticamente corretos, como por exemplo, a modificação de um nome de uma variável, acrescentando uma letra ou um caractere especial, adiciona muito pouco na eficácia do teste de mutação. Um exemplo disso é demonstrado no trabalho, no qual dos 22 operadores de mutação implementados na ferramenta *Mothra*, apenas cinco podem ser considerados essenciais para implementar o teste de mutação eficientemente. Esse conjunto de *operadores essenciais* é formado pelos

operadores *ABS* (inserção valor absoluto), *AOR* (substituição de um operador aritmético), *LCR* (substituição do conector lógico), *ROR* (substituição do operador relacional), *UOI* (inserção de um operador unário).

3.4.2 A Investigação de Barbosa (1998)

Baseando-se em um estudo prévio de Offutt et al. (1996) que relata a determinação de um conjunto essencial de operadores de mutação para redução de custo em teste de mutação, Barbosa investiga abordagens para determinar um conjunto essencial de operadores de mutação para linguagem C, e para isso foi definida uma diretriz para seleção de um conjunto essencial de operadores de mutação. Barbosa ainda aplicou essa diretriz nos operadores de mutação da ferramenta *Proteum*, e obteve um conjunto de 10 operadores de mutação que alcançavam um escore de mutação de aproximadamente 99.5% e com 65% de média na redução dos custos da aplicação.

Em seu trabalho, Barbosa fez uma comparação entre os conjuntos de operadores essenciais determinados por Offutt et al. e dos operadores determinados por Wong et al. Apesar de os resultados alcançados por Wong serem um pouco melhores do que o conjunto determinado pela abordagem de Offutt para os 27 programas estavam entre os mais relevantes conflitando com N-seletiva mutação. Além de os operadores essenciais determinados eram completamente diferentes para cada conjuntos. Esses resultados motivaram a criação de uma diretriz para determinação de um conjunto de operadores de essenciais, esse processo é denominado Procedimento Essencial (*do inglês Sufficient Procedure*).

A *relação de inclusão* definida por (RAPPS; J.WEYUKER, 1985) é um dos principais mecanismos para fazer a comparação entre os critérios de teste. Considerando C1 e C2 como critérios de teste, diz-se então que C1 inclui C2, se para todo conjunto teste T1 que é adequado a C1, é também adequado a C2, e se para alguns T2 é adequado a C2 mas não é adequado a C1, e se para qualquer conjunto teste T for a adequado a C1 e também for adequado para C2 e vice-versa, então diz-se que o C1 e C2 são equivalentes.

Considerando esse conceito definido por Rapps, Barbosa definiu empiricamente um conceito adequado denominado relação *EmpSubsumes*, o qual assume que escores de mutação que possuem valores próximo a 1.0 podem ser considerados como satisfatórios, pois segundo Offutt não há evidências claras de que 100% de cobertura consiga oferecer um teste melhor do que um com uma cobertura com menor percentagem. Utilizando então como referência a relação de inclusão definida por Rapps, é determinado empiricamente pelo testador um novo escore de mutação ms^* , e então são definidas as propriedades que para um critério C e um teste T, diz-se

que T é empiricamente adequado a C se T obtêm um escore de mutação igual ou superior ao que fora definido pelo testador (ms^*). Se para cada conjunto de teste T1 é empiricamente adequado a C1 e T1 é também empiricamente adequado a C2, e algum teste T2 é empiricamente adequado a C2, mas não a C1, então diz-se que C1 EmpSubsumes C2, e o T1 é empiricamente adequado a C1 e a C2 e vice-versa, então diz-se que C1 e C2 são empiricamente equivalentes.

A determinação de um conjunto essencial de operadores de mutação consiste na seleção de um subconjunto de operadores, os quais produzem mutantes e que quando aplicados em um conjunto de teste T, obtêm um escore de mutação igual ou próximo ao escore de mutação alcançados por todos os mutantes gerados quando aplicados ao mesmo conjunto de teste T.

Buscando definir um subconjunto de operadores o estudo estabelece um conjunto de diretrizes, denominado *Procedimento Essencial* (do inglês, *Sufficient procedure*), que visa ajudar a selecionar os melhores operadores de mutação para compor esse subconjunto essencial. Os passos para realizar a seleção de operadores essenciais são descritos a seguir:

- **considerar operadores de mutação que determinam um alto escore de mutação:** selecionar os operadores que podem ser determinantes no aumento do escore de mutação.
- **considerar um operador de cada classe de mutação:** cada classe de mutação refere-se a um tipo de erro específico de uma determinada parte do programa, é preferível a escolha do operador mais representativo de cada classe.
- **avaliar a inclusão empírica entre os operadores:** os operadores de mutação que são empiricamente incluídos por outros operadores do conjunto essencial que elevam o custo da aplicação devem ser removidos.
- **estabelecer uma estratégia incremental:** estabelecer uma estratégia incremental entre os operadores dos conjuntos essenciais, baseada no custo aplicação e os requisitos de teste que cada classe determina. Aplicar primeiro os operadores que são mais relevantes para os requisitos mínimos de teste.
- **considerar os operadores que oferecem um incremento no escore de mutação:** o incremento de 1% no escore de mutação representa cerca de 5% dos mutantes que são realmente significantes. Dessa forma um operador que não foi selecionado e que pode aumentar o escore de mutação, deve ser analisado.
- **considerar os mutantes com alta resistência:** operadores que geram mutantes que são

mais difíceis de serem satisfeitos pelos casos de testes adequados a outros operadores, podem contribuir pra aumentar o escore de mutação do conjunto essencial final.

Após a determinação desse conjunto de diretrizes para seleção de operadores essenciais, Barbosa aplicou o Procedimento Essencial em dois experimentos para determinar um conjunto essencial de operadores para linguagem C. Para esses experimentos foi utilizado a ferramenta de teste *Proteum* para apoiar o teste de mutação nessa linguagem.

Experimento 1: Foi utilizado um conjunto de 27 programas de um editor simples de texto, esses programas possuem de 11 a 71 linhas executáveis, gerando cada cerca de 119 a 1631 mutantes. Para esse conjunto de 27 programas, foram aplicados 39 dos 71 operadores de mutação, sendo 10 operadores de declaração, 21 de operadores, 5 de variáveis e 3 de constantes.

Experimento 2: Já no experimento 2, foi utilizado um conjunto de cinco programas utilitários Unix, cada programa possuía cerca de 76 a 119 linhas executáveis, cada um gerando de 1619 a 4332 mutantes. Após a aplicação das diretrizes foram escolhidos 56 operadores, sendo 14 de declaração, 33 de operadores, 6 de variáveis e 3 de constantes.

Após a aplicação do processo, os resultados relatados no estudo mostram que foi possível atingir uma significativa redução do número de operadores de mutação da ferramenta *Proteum*, os conjuntos de operadores essenciais determinados no estudo alcançaram alto grau de adequação, os quais atingiram escores de mutação acima de 99,5%. Considerando o custo da aplicação em termos de número de mutantes gerados, a redução atingiu uma média de cerca de 65%. Em ambos experimentos, levando em consideração o escore de mutação, redução de custo, resistência e distribuição dos escores, foi possível observar que o procedimento proposto por Barbosa, representa uma boa escolha em relação a outras abordagens de redução de custo em teste de mutação.

3.4.3 A Investigação de Mresa e Bottaci (1999)

O estudo proposto por Mresa e Bottaci (1999) diferentemente de outros trabalhos não faz suposições sobre quais perdas na adequação de conjuntos de testes são aceitáveis. Ao invés disso, é levado em consideração o cálculo de eficiência (custo x escore) das estratégias de mutação seletiva. São definidos os escores e os custos individuais de cada operador e os operadores são comparados em relação a essas duas variáveis. Através dessa comparação é possível

identificar quais os operadores são mais eficientes individualmente, pois operadores que são eficientes individualmente, produzem um conjunto de operadores eficiente.

Um experimento foi realizado aplicando os 21 operadores de mutação da ferramenta Mothra (excluindo-se apenas o operador *dsa* - *data statement alterations operator*) sobre um conjunto de 11 programas. Cada programa desse conjunto possui cerca de 19 a 71 linhas de códigos executáveis, o que dá uma média 43,7 linhas de código no total, gerando no total cerca de 780 a 5728 mutantes. Do conjunto total de programas, apenas quatro programas não geraram mutantes quando aplicados a um ou dois operadores, para esses casos os operadores foram excluídos. O conjunto de teste necessário para matar todos os mutantes é influenciado não só pelo número de mutantes, mas também pela natureza do programa.

O experimento realizado para cada um dos 11 programas seguiu os seguintes passos:

1. todos os operadores de mutação são aplicados nos programas, excluindo apenas o operador *dsa*;
2. uma sequência de testes adequados e efetivos são gerados para os conjunto de mutantes, e a partir dessa sequência são criados conjuntos de testes adequados e não redundantes.
3. Então para cada conjunto não vazio de mutantes não equivalentes: (i) 10 sequências o-adequadas e efetivas e os seus respectivos conjuntos de testes não-redundantes são gerados. Uma sequência de teste efetivas e adequadas é gerada a partir da geração de casos de teste, produzidos pela ferramenta Godzilla. Para alguns operadores e programas, a ferramenta Godzilla conseguia gerar um conjunto de teste o-adequado, e para outros não conseguia gerar um conjunto o-adequado, gerando um conjunto significativamente menor do que o adequado, sendo necessário portanto, gerar testes adicionais usando um gerador de números randômicos. Após a geração dessa sequência de testes foram gerados os casos de teste não redundantes, através de repetidas execuções de permutações cíclicas das sequências efetivas contra um conjunto de mutantes, e em cada ciclo são eliminados os testes inefetivos. Esse ciclo é repetido até que todos os testes tenham sido executados até a última sequência.
(ii) é calculado o escore médio dos 10 o-adequado conjunto de testes, tanto para sequência efetiva quanto para o conjunto de testes não redundantes;
(iii) é calculado o custo médio de 10 o-adequado conjunto de testes, tanto para sequência efetiva quanto para conjunto de teste não redundantes

Através do cálculo do escore e custo médio, foram observados no estudo que:

- os operadores *svr*, *asr* e *csr* geravam muitos mutantes.
- os escores de operadores que tem maior pontuação são próximos de um, o que mostra que os resultados obtidos são consistentes com os resultados obtidos em estudos anteriores de mutação seletiva.
- o desvio padrão dos escores de três operados com os maiores pontuações é muito pequeno, e esse índice pode ser usado como medida aproximada de confiabilidade dos valores dos escores.
- o custo do operador *abs* é o mais alto devido ao grande número de mutantes equivalentes gerados.
- existe uma correlação clara entre o escore e o custo. Essa correlação pode ser vista na Figura 3.6.

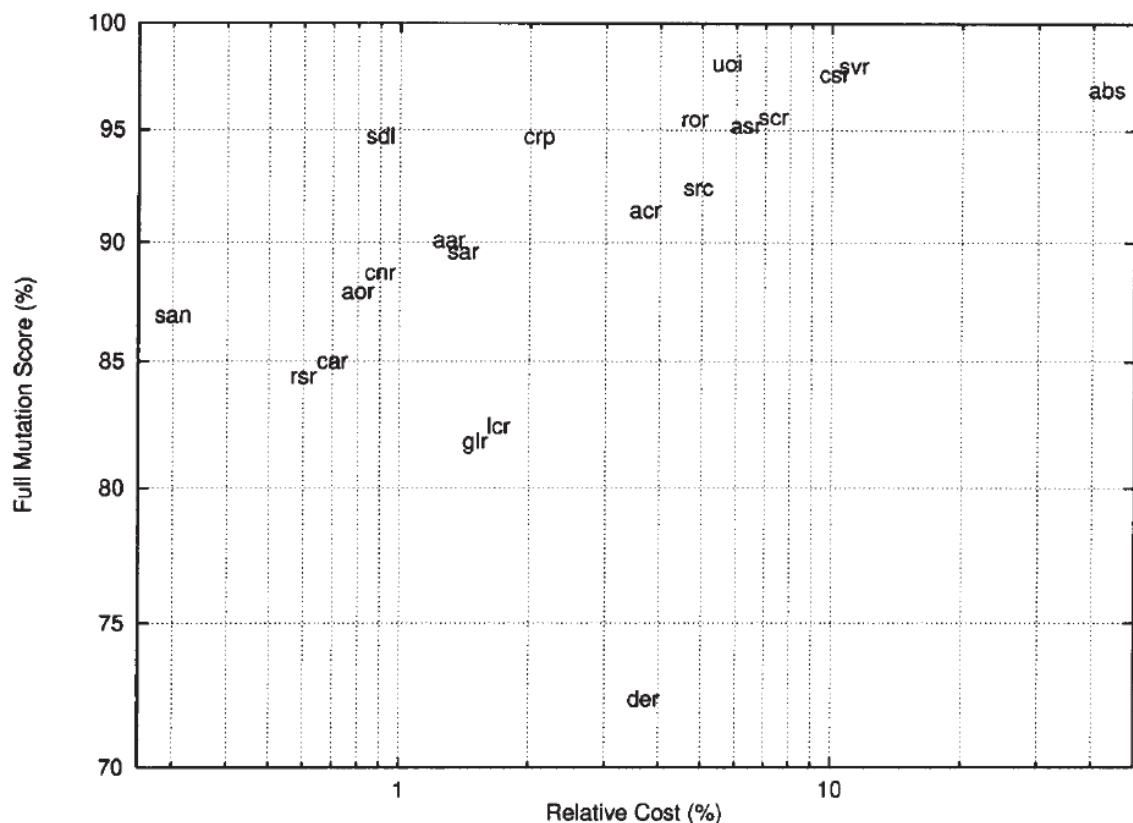


Figura 3.6: Gráfico *escore x cost*(em escala logarítmica) para cada operador de mutação. (Figura adaptada de (MRESA; BOTTACI, 1999))

O estudo apresentado por Mresa e Bottaci (1999) visa utilizar informações sobre custo e escore individual de cada operador para encontrar um subconjunto de operadores que seja

eficiente, mas que tenha um escore de mutação relativamente alto. Para encontrar esse subconjunto foi adotada a premissa de que, se os operadores são eficientes individualmente, logo eles também produzirão um conjunto eficiente de operadores. A partir dessa informação, foi possível determinar um conjunto de operadores mais eficientes, denominado *moreEfficient*, composto pelos operadores *san*, *aor*, *sdl*, *ror*, *uoi*. No estudo esse conjunto de operadores mais eficientes é referenciado por *eff*. Observou-se que em algumas experiências com esse conjunto de operadores, os conjuntos de teste que eram adequados para esse conjunto *eff* de operadores falhavam em matar mutantes *abs* de um conjunto completo do programa. Por esse motivo foi acrescentado o operador *abs* ao conjunto de operadores *eff* gerando um novo conjunto, referenciado no estudo por *efa*.

Utilizando os conjuntos de operadores proposto por Offutt et al. (1996) (*abs*, *uoi*, *lcr*, *aor*, *ror*), referenciado no estudo por *exp*, juntamente com os outros dois conjuntos *eff* e *efa*, foram realizadas comparações de desempenho entre os operadores e também comparações com mutação *x%-seletiva*. O experimento de comparação foi realizado da mesma forma como no processo descrito anteriormente, exceto que, para esse experimento os conjunto de operadores utilizados é menor do que o primeiro experimento, e o mesmo método foi utilizado para criar os 10 conjuntos de testes adequados para os conjuntos de operadores.

Esse experimento resultou em duas tabelas comparativas, uma com escores de mutação das cinco estratégias de mutação seletiva e outra com os custos associados a cada uma delas. As duas tabelas são apresentadas na Tabela 3.8 e na Tabela 3.7 respectivamente, onde as colunas *mean* representa a média obtida pelo operador dentro do conjunto dos 11 programas e a coluna *sd* representa o desvio padrão de cada operador.

Tabela 3.7: Tabela comparativa com custo das cinco estratégias. (Tabela adaptada de Mresa e Bottaci (1999))

Op.	ARSG	BANK	CALV	FCNT	MINV	PAT0	RPC	SEQS	SORT	STRE	TRET	mean	sd
exp	56,5	49,0	41,9	55,7	48,2	62,0	48,9	61,6	56,2	47,1	55,6	53,0	6,4
efa	54,0	51,4	41,4	57,4	48,8	60,9	50,3	63,0	57,5	47,0	56,2	53,4	6,4
xex	30,5	38,5	24,5	34,3	36,7	46,4	29,7	36,7	51,9	47,3	31,5	37,1	8,4
eff	16,4	9,9	9,6	10,0	12,9	11,5	11,5	10,8	18,0	7,5	10,5	11,7	3,1
xef	17,7	20,7	12,7	17,6	21,2	21,8	16,2	20,9	29,8	17,0	13,7	19,0	4,7

Com os resultados apresentados no estudo, os autores concluíram que se o escore de mutação for muito próximo de 100% a mutação *x%-seletiva* pode ser mais eficiente do que a mutação baseada em apenas expressões de conjunto de operadores, como por exemplo o conjunto *eff* e o conjunto *efa*, e também mais eficiente que o conjunto baseado em mutação seletiva como o conjunto *exp* que foi proposto por Offutt et al. (1996). O estudo afirma que se um conjunto de testes que menos rigoroso for aceitável, então, para estes casos, a mutação seletiva baseada

Tabela 3.8: Tabela comparativa com os escores das cinco estratégias. (Tabela adaptada de Mresa e Bottaci (1999))

Op.	ARSG	BANK	CALV	FCNT	MINV	PAT0	RPC	SEQS	SORT	STRE	TRET	mean	sd
exp	99,9	98,4	99,4	99,7	99,6	98,8	99,8	99,8	99,9	99,6	99,5	99,5	0,5
efa	99,9	98,3	99,3	99,8	99,5	98,0	99,8	99,8	99,9	99,6	99,8	99,4	0,6
xex	99,7	99,2	99,4	99,5	99,6	99,8	99,7	99,8	99,5	99,6	99,0	99,5	0,3
eff	99,9	97,0	99,0	99,5	99,0	98,0	98,8	99,4	99,2	99,2	98,5	98,9	0,8
xef	99,7	97,8	99,3	99,1	99,4	99,1	98,7	99,2	99,4	98,9	98,0	99,0	0,6

em um conjunto restrito de operadores de mutação pode ser mais eficiente que a mutação $x\%$ -seletiva. Ressalta-se ainda que as diferenças de custos associadas a cada estratégia é decorrente do custo em identificar os mutantes equivalentes.

3.4.4 A Investigação de Namin, Andrews e Murdoch (2008)

No estudo de Namin, Andrews e Murdoch (2008), é proposto um processo de análise estatística visando encontrar um conjunto essencial de operadores que seja suficiente para calcular a eficácia de um teste, e juntamente com essa análise estatística é definido um modelo linear capaz de prever com bastante precisão a adequação de mutação. É realizada também no estudo uma validação do processo por meio de uma validação cruzada e pela aplicação de outras análises estatísticas alternativas.

Segundo os autores, o estudo proposto difere em três aspectos em relação aos demais estudos realizados anteriormente. (i) Nesse estudo é utilizado um conjunto de programas um pouco maior do que os outros; (ii) os estudos anteriores focam em teste de mutação, enquanto que o estudo foca na análise de mutação, eles estão interessados em determinar um conjunto essencial que permita prever com precisão a adequação de mutação; (iii) trata o problema como um problema de redução padrão de variável, leva em consideração todos os operadores incluindo aqueles não considerados pelos processos de seleção de operadores abordados nas outras pesquisas.

No estudo foi realizado um experimento utilizando um conjunto de sete programas Siemens, que representa um conjunto amplo com uma diversidade de programas escritos na linguagem C. Esse conjunto contém códigos envolvendo desde inteiros e ponto flutuante, até structs, ponteiros, alocação de memória, loops e diversas outras estruturas complexas. O conjunto Siemens já conta um conjunto abrangente de casos de teste, esse conjunto de casos de teste permite a criação de diversos conjuntos de testes para diferentes algoritmos ou critérios de cobertura. Cada programa contém em média 312,57 linhas de códigos executáveis, o que totaliza do conjunto 2188 linhas executáveis. Na Tabela 3.9 é demonstrado com mais clareza os resultados

obtidos através da aplicação dos operadores em cada programa do conjunto Siemens.

Tabela 3.9: Tabela descritiva dos programas objetos

Programas	NOLC	NTC	NMG	NM
printtokens	343	4130	11741	1551
printtokens2	355	4115	10266	1942
replace	513	5542	23847	1969
schedule	296	2650	4130	1760
schedule2	263	2710	6552	1497
tcas	137	1608	4935	4935
totinfo	281	1052	8767	1740
Total	2188	21807	70238	15394

Na Tabela 3.9 são mostrados para cada programa os valores referentes à quantidade total de linhas executáveis, representados pela sigla **NLOC**; a quantidade de casos de teste, representado pela sigla **NTC**; a quantidade de mutantes gerados por todos os operadores de mutação, representado pela sigla **NGM**; e a quantidade de mutantes selecionados que não são equivalentes, representado pela sigla **NM**.

Para cada programa foram gerados todos os mutantes para todos os operadores da *Proteum*. Porém, constatou-se que a utilização de todos os mutantes demandaria muito processamento, então decidiu-se que seriam escolhidos aleatoriamente apenas 2000 mutantes para o experimento. Para distribuir uniformemente todos os mutantes sobre todos os operadores foi utilizada a razão de 2000/NM(P) (número de mutantes). Para cada programa foram gerados 100 conjuntos de teste, composto por 2 casos de teste de tamanhos distintos de cada um dos 50 casos de teste. A partir dessa seleção é calculada a razão de adequação de mutação para cada operador de mutação, e depois essa razão é calculado para todos eles.

Depois dessa etapa é calculado o custo associado a cada operador. É dito que um operador O_i tem menor custo quando ele gera menos mutantes que um operador O_j , na média para todos os programas. A partir de então é definido custo (i,P) que é o custo de um operador O_i para o programa P , calculado pela razão da quantidade do número de mutantes não-equivalentes gerados pelo operador O_i , sobre o número total de mutantes não-equivalentes de P gerados por todos os operadores. Após é calculado o custo global do conjunto pela média de custo de cada operador em todos os programas.

Para auxiliar na seleção de um conjunto é utilizado um método de seleção baseado em custo (CBLARS). O modelo identificou um conjunto de 28 operadores de mutação essenciais que geraram no total 1139 mutantes para todos os programas, Isso representa apenas 7,40% dos mutantes gerados pelo conjunto completo dos 108 operadores da *Proteum*, significando uma economia de aproximadamente 92,6%. A resultante da aplicação do modelo CBLARS é

demonstrada na Tabela 3.10

Tabela 3.10: Tabela Resultante do modelo CBLARS

Name	Descrição	Coeficiente	NumMut
IndVarAriNeg	Inserir negação aritmética em variáveis de interface	0.162469	126
IndVarBitNeg	Inserir negação de Bit em variáveis de interface	0.168583	121
RetStaDel	Excluir declarações de retorno	0.051984	67
ArgDel	Excluir argumento	-0.016532	12
ArgLogNeg	Inserir negação lógica em argumento	0.131566	30
OAAAN	Mutação operador aritmético	0.041376	71
OABN	Aritmética por operador bit a bit	-0.02075	27
OAEA	Atribuição aritmética simples por atribuição	-0.194022	2
OALN	Operador Aritmético por operador lógico	0.022149	40
OBBN	Operador de mutação bit a bit	-0.023452	3
OBNG	Negação bit a bit	-0.00275	6
OBSN	Operador bit a bit por deslocamento de operador	-0.035337	3
OCNG	Negação contexto lógico	0.097971	57
OCOR	Operador Cast por Operador Cast	0.024727	9
Oido	Incremento/Decremento mutação	0.069952	14
OLAN	Operador lógicos por operador aritmético	0.027124	119
OLBN	Operador lógico por operador bit a bit	-0.00362	67
OLLN	Operador de mutação lógico	0.041048	25
OLNG	Negação lógica	0.06966	78
OLSN	Operador lógico por deslocamento de operador	-0.003438	45
ORSN	Operador relacional por deslocamento de operador	0.160376	91
SGLR	Substituição do rótulo goto	0.07605	1
SMTC	Continuação n-trip	0.031519	9
SMVB	Mover para cima e para baixo	-0.062404	2
SSWM	Mutação mudança de declaração	-0.020412	15
STRI	Armadilha da condição if	0.094324	85
SWDD	substituição do while pelo do-while	0.032363	1
VGPR	Mutação referência global do ponteiro	-0.091281	13
(Interceptação)	-	-0.036398	-

Namin et.al. concluíram que usando os métodos estatísticos, é possível determinar um subconjunto de operadores da ferramenta *Proteum* que consiga gerar 8% menos de mutantes gerados pelo conjunto completo. Com esse subconjunto é possível prever com precisão a eficácia do conjunto de testes sobre os mutantes a partir de um conjunto de operadores. Por meio de uma validação cruzada utilizada no estudo foi possível constatar que o processo proposto é razoável e que pode ser estendido a outros programas.

Com a possibilidade de prever com precisão a eficácia de um conjunto de teste o processo proposto por Namin et.al. pode levar a uma forma padronizada de medição de eficácia nas técnicas de testes. No conjunto essencial determinado no estudo foi detectado, por exemplo, a ausência de operadores de mutação constantes e a presença de operadores que tendem a produzir menos mutantes, e isso pode ser útil na determinação de um conjunto adequado de operadores de mutação para ser utilizado nos testes de mutação.

3.5 Considerações Finais

Neste capítulo foram relatados o protocolo para realização de um mapeamento sistemático, e a sumarização dos estudos obtidos com esse mapeamento. O propósito do mapeamento foi a caracterização das estratégias de redução de custo em teste de mutação. Por meio da sumarização dos estudos foi possível obter uma visão geral das estratégias que abordam redução de custo.

Posteriormente, as principais estratégias de redução de custo são descritas com mais detalhes na Seção 3.4. Dentre essas estratégias, destaca-se a abordagem proposta no estudo de Barbosa (1998), pois relata um procedimento denominado Procedimento Essencial, que tem se mostrado bastante promissor e que foi aplicado neste trabalho. A preparação e aplicação desse procedimento é descrita no Capítulo 4.

Capítulo 4

PREPARAÇÃO DO ESTUDO, EXECUÇÃO E ANÁLISE DOS RESULTADOS

4.1 Considerações Iniciais

Muitos estudos têm proposto técnicas para a redução de custo em teste de mutação. No Capítulo 3 foi possível observar que dentre os estudos apresentados, o Procedimento Essencial proposto por Barbosa (1998), em face aos resultados alcançados, foi a abordagem que se mostrou mais promissora. Com o propósito de reduzir custos em programas do paradigma OA, foi realizado um experimento aplicando o Procedimento Essencial em dois conjuntos distintos de programas OA, desenvolvidos na linguagem AspectJ.

O Procedimento Essencial que foi explicado com mais detalhes no Capítulo 3, é descrito novamente de forma mais breve na Subseção 4.2.1. A infraestrutura de apoio utilizada neste trabalho é descrita na Seção 4.2.2. O passo-a-passo da aplicação do procedimento nos dois conjuntos são descritos nas Seções 4.3 e 4.4. Os resultados para os dois conjuntos são apresentados e discutidos nas Subseções 4.3.5 e 4.4.5, e as comparações entre os dois conjuntos de operadores essenciais são apresentadas e discutidas na Seção 4.5.

4.2 Informações Preliminares

Nesta seção são apresentados de forma sucinta os passos do Procedimento Essencial e também a infraestrutura utilizada para apoiar a realização deste trabalho.

4.2.1 Procedimento Essencial

De acordo com o objetivo apresentado no Capítulo 1, para obter uma redução de custo em teste de mutação para programas OA, foi aplicado neste estudo o Procedimento Essencial, proposto por Barbosa (1998). O procedimento é composto por seis passos que são enumerados a seguir:

- **Passo 1 - Selecionar operadores que determinam alto escore de mutação:** Devem ser selecionados operadores que geram mutantes, os quais quando aplicados ao conjunto de teste adequado ao conjunto completo de operadores alcançam altos escores de mutação. Desta forma busca-se garantir que o conjunto essencial de operadores possua um alto escore de mutação.
- **Passo 2 - Procurar selecionar operadores de cada classe de mutação:** pelo fato de cada classe modelar defeitos específicos, é desejável que o conjunto de operadores contenha pelo menos um operador de cada classe.
- **Passo 3 - Avaliar a inclusão empírica entre operadores:** verificar os operadores que já são incluídos empiricamente por outros operadores e removê-los do conjunto essencial. Essa operação deverá ser repetida até que todos os operadores incluídos empiricamente por outros operadores tenham sido removidos.
- **Passo 4 - Estabelecer estratégia incremental de aplicação:** Devido ao alto custo dos operadores, é necessário estabelecer uma estratégia incremental para aplicação dos operadores do conjunto essencial.
- **Passo 5 - Selecionar operadores que proporcionam incremento no escore de mutação:** Buscando melhorar a qualidade do conjunto essencial, busca-se os operadores que não foram selecionados e que, acrescidos ao conjunto almejado, possam incrementar o escore de mutação.
- **Passo 6 - Selecionar operadores de alto *strength*:** Outros operadores devem ser considerados no conjunto essencial. Em particular, são os operadores que possuem um alto *strength* em relação ao conjunto total de operadores. Tais operadores são, em média, pouco incluídos empiricamente pelos demais operadores.

4.2.2 Infraestrutura de Apoio

A escolha da ferramenta é fundamental nesse estudo, pois a ferramenta precisa dar apoio à aplicação do critério análise de mutantes, bem como às atividades inerentes a esse critério como, por exemplo, gerenciamento dos casos de teste, gerenciamento dos operadores de mutação, geração e gerenciamento dos mutantes, análise de resultados e geração de relatórios.

A ferramenta *Proteum/AJ* (FERRARI et al., 2010) foi escolhida, pois atendia a esses requisitos. Além disso a *Proteum/AJ* permite:

- Avaliar individualmente cada mutante, permitindo a visualização do código do mutante gerado em contraste com o código original;
- Possibilita ao usuário o planejamento de uma estratégia de teste de mutação, optando pela opção *clean*, que literalmente limpa todos os mutantes gerados, gerando-os novamente, ou optando pela opção *update*, que adiciona novos mutantes ao conjunto gerado anteriormente.
- Possibilita ao usuário a detecção automática de mutantes equivalentes enquanto os mutantes são compilados. Essa detecção é feita apenas para mutantes cujo alvo da mutação foram *pointcuts* e alguns tipos de mutação em *advices*. A análise de equivalência é feita por meio da comparação de informações produzidas pelo próprio compilador do AspectJ. Essa opção pode ser desabilitada pelo testador antes da compilação dos mutantes pela ferramenta. Mais detalhes sobre os operadores cujos mutantes equivalentes podem ser detectados automaticamente pela ferramenta *Proteum/AJ* pode ser obtidos no trabalho de Ferrari et al. (2010).

Para esse estudo foi utilizada a versão 2.0 da ferramenta *Proteum/AJ* (LEME et al., 2012). Nesta versão, além do suporte a operadores para AspectJ, há a inserção de novos operadores de unidade, possibilitando o suporte ao critério análise de mutante também em programas Java.

Ressalta-se que os experimentos e resultados descritos nesta dissertação compreendem apenas operadores de mutação específicos para programas AspectJ, definidos por Ferrari, Maldonado e Rashid (2008) e implementados na versão 1.0 da ferramenta *Proteum/AJ* (FERRARI et al., 2010).

4.3 Aplicação do Procedimento Essencial no Conjunto 1

Esta seção detalha a aplicação do Procedimento Essencial no primeiro conjunto de programas, doravante denominado **Set1**. A seção é iniciada com uma sucinta descrição dos programas e da origem dos dados de teste, isto é, mutantes e casos de teste.

4.3.1 Programas do conjunto Set1

A seleção de programas configura um importante passo na aplicação de um experimento, pois os resultados gerados pelo experimento estão fortemente associados aos tipos de programas selecionados e às características inerentes a cada um.

Como o objetivo do estudo está relacionado a programas no paradigma OA, para esse primeiro conjunto de programas foram selecionados 12 programas escritos em AspectJ que foram utilizados no estudo de Ferrari (2010). Os programas utilizados são listados na Tabela 4.1 e, logo em seguida, é apresentada uma breve descrição dos programas.

Tabela 4.1: Descrição das aplicações do Experimento

Aplicação	LOC*	Classes	Aspectos
1.BankingSystem	199	9	6
2.Telecom	251	6	3
3.ProdLine	537	8	8
4.FactorialOptimizer	39	1	1
5.MusicOnline	150	7	2
6.VendingMachine	64	1	3
7.PointBoundsChecker	44	1	1
8.StackManager	77	4	3
9.PointShadowManager	66	2	1
10.Math	53	1	1
11.AuthSystem	89	3	2
12.SeqGen	205	8	4
Total	1774	51	35

*É considerado apenas as linhas de código, excluindo comentários e linhas em branco.

BankingSystem é um sistema que gerencia transações de conta de um banco Laddad (2003), nessa aplicação são implementadas os aspectos *logging*, controle mínimo de saldo e operação de cheque especial.

Telecom é um sistema simulador de telefonia, que é desenvolvido originalmente em AspectJ e distribuído, como uma das aplicações exemplo do AspectJ, pela The Eclipse Foundation (2010). O *Telecom* gerencia as ligações telefônicas, e os cálculos de tempo e custo de cada ligação são feitas pelos aspectos.

ProdLine consiste em um sistema de linha de produto para aplicações gráficas que incluem funções em comum de domínio gráfico (Lopez-Herrejon; BATORY, 2002). Os aspectos nessa aplicação são responsáveis por introduzir as características selecionadas em uma determinada instância SPL.

FactorialOptimizer é um sistema matemático que implementa uma otimização no cálculo fatorial de um número (ALEXANDER; BIEMAN; ANDREWS, 2004). O cálculo é gerenciado por um aspecto, ele armazena todos os cálculos em cache para futuramente ser recuperado e reutilizado em outro cálculos, reduzindo a sobrecarga.

MusicOnline é um sistema gerenciador de uma loja online de músicas, ele permite o usuário ouvir músicas e *playlists* inteiras. (LEMONS; FRANCHIN; MASIERO, 2009). O usuário necessita pagar por cada música ou por *playlist*. Os aspectos são responsáveis por gerenciar as contas dos usuários e o sistema de cobrança.

VendingMachine é um sistema que gerencia a interação de vendas em uma máquina de bebidas. O usuário insere moedas para a compra de bebidas e o sistema gerencia a venda, retornando troco ao usuário quando necessário, e liberando a bebida solicitada (LIU; CHANG, 2008). Os aspectos são responsáveis por gerenciar as operações de venda.

PointBoundsChecker é um verificador de ponto de duas dimensões (MORTENSEN; ALEXANDER, 2005). O aspecto verifica se as coordenadas dos pontos pertencem a um determinado intervalo, caso contrário é lançado uma exceção.

StackManager implementa uma pilha simples que melhora as operações de *push* e *pop* (XIE et al., 2005). Os aspectos evitam que números negativos sejam inseridos na pilha, realizam auditoria sobre os números armazenados e contam o número de operações do tipo *push*.

PointShadowManager é uma aplicação que gerencia as coordenadas entre dois pontos (ZHAO, 2003). Um aspecto cria e gerencia a sombra de dois pontos. Quando um ponto é criado, sua sombra deve ter exatamente as mesmas coordenadas, da mesma forma, se as coordenadas desse ponto são atualizadas sua respectiva sombra deve ser alterada.

Math é uma aplicação matemática que calcula a probabilidade de sucesso em uma dada sequência de n (Teorema de Bernoulli) onde a possibilidade de sucesso é sempre a mesma a cada rodada do experimento (LEMONS; FRANCHIN; MASIERO, 2009). O aspecto registra operações de exponenciação, identificando o tipo de expoente.

AuthSystem é uma versão mais simples de um sistema de banco, o sistema exige que o usuário faça autenticação antes de operações simples como débito, crédito e verificar saldo (ZHOU; RICHARDSON; ZIV, 2004). Os aspectos são responsáveis por esse gerenciamento e por

autenticação do usuário.

SeqGen implementa um gerador de uma sequência de números e caracteres (BERNARDI; LUCCA, 2007). Os aspectos modularizam a política de geração e os interesses de *logging*.

4.3.2 Geração dos Conjuntos AM-Adequados

Criação dos Casos de Teste

Para o conjunto Set1, a seleção dos programas e a criação dos conjuntos AM-adequados de casos de teste foram realizadas no trabalho de Ferrari (2010). A geração desse conjunto de casos de teste tomou como base a especificação de requisitos de cada programa e também foi utilizado o critério Teste Funcional Sistemático (do inglês, Systematic Functional Testing - SFT) (LINKMAN; VINCENZI; MALDONADO, 2003). Esse critério de teste utiliza a combinação de dois critérios: Particionamento de Equivalência e Análise de Valor-Limite. Em suma, para cada classe de equivalência é necessário a construção de dois casos de teste. Dessa forma, evita-se que haja uma correta execução para uma entrada de dados que mascara um defeito que poderia ser descoberto por outra entrada de testes de uma mesma partição de domínio.

Aplicação do Critério Análise de Mutantes

Ferrari (2010) aplicou 24 operadores de mutação para os 12 programas selecionados, o resumo da aplicação desses operadores é mostrado na Tabela 4.2.

Tabela 4.2: Mutantes gerados a partir das 12 aplicações do conjunto Set1 (adaptada de Lacerda e Ferrari (2014))

Aplicação	mG1	mG2	mG3	Total	Equiv. Autom.	Anom	Mut. Vivos
1. BankingSystem	108	2	26	136	61	20	55
2. Telecom	82	2	27	111	46	12	53
3. ProdLine	158	0	41	199	125	16	58
4. FactorialOptimizer	14	0	15	29	8	6	15
5. MusicOnline	47	0	10	57	25	5	27
6. VendingMachine	82	2	29	113	58	8	47
7. PointBoundsChecker	46	0	24	70	32	10	28
8. StackManager	34	0	11	45	24	0	21
9. PointShadowManager	38	0	12	50	25	4	21
10. Math	16	0	4	20	13	0	7
11. AuthSystem	45	0	7	52	28	3	21
12. SeqGen	33	0	7	40	19	3	18
Total	703	6	213	922	464	87	371

As colunas “mG1”, “mG2” e “mG3” representam as quantidades de mutantes gerados por cada grupo de operador em cada uma das aplicações, e a coluna “Total” representa a soma dos mutantes gerados por esses três grupos de operadores. A coluna “Equiv. Autom.” repre-

sentam a quantidade de mutantes que foram classificados automaticamente pela ferramenta como equivalentes. Os mutantes não-compiláveis são computados na coluna “Anom” e os mutantes classificados como vivos são representados pela coluna “Mut. Vivos”.

Tomando como exemplo o programa *Telecom* - segunda linha da Tabela 4.2 - foram gerados 111 mutantes, dos quais 82 pertencem ao grupo 1 de operadores (operadores que modelam defeitos de pointcuts), 2 mutantes do grupo 2 (operadores que modelam defeitos para expressões de declarações em AspectJ) e 27 mutantes do grupo 3 (operadores que modelam defeitos relacionados à definição e à implementação de *advices*). Dos 111 mutantes gerados, a ferramenta classificou automaticamente 46 mutantes como equivalentes, 12 como anômalos (mutantes não compiláveis) e 53 mutantes como vivos.

Adequação do conjunto de teste

Após a geração dos mutantes, foram aplicados os conjuntos de casos de teste. O resultado da aplicação e da adequação desses casos de teste ao conjunto de mutantes gerados pode ser visualizado na Tabela 4.3.

Tabela 4.3: Resultado do teste de mutação para as 12 aplicações do conjunto Set1 (adaptada de Lacerda e Ferrari (2014))

Aplicação	Vivos	Mortos T_{SFT}	Reman. Vivos	MS-Ini	Equiv. Man.	MS- Inter	Testes Adds	TM	MS- Final
1. BankingSystem	55	55	0	1,00	-	-	-	58	1,00
2. Telecom	53	31	22	0,58	10	0,72	4	67	1,00
3. ProdLine	58	58	0	1,00	-	-	-	36	1,00
4. FactorialOptimizer	15	14	1	0,93	1	1,00	-	-	1,00
5. MusicOnline	27	22	5	0,81	2	0,88	2	48	1,00
6. VendingMachine	47	23	24	0,49	13	0,68	5	23	1,00
7. PointBoundsChecker	28	28	0	1,00	-	-	-	14	1,00
8. StackManager	21	21	0	1,00	-	-	-	15	1,00
9. PointShadowManager	21	13	8	0,62	5	0,81	2	14	1,00
10. Math	7	4	3	0,57	2	0,80	1	54	1,00
11. AuthSystem	21	17	4	0,81	1	0,85	2	19	1,00
12. SeqGen	18	4	14	0,22	8	0,40	3	42	1,00
Total	366	285	81	0,78	42	0,88	19	409	1,00

Na tabela, a coluna “Vivos” representa a quantidade de mutantes vivos antes da aplicação dos casos de teste no conjunto de programas. Após a aplicação dos casos de teste, alguns mutantes são mortos. A quantidade de mutantes mortos pelos casos de teste é representada pela coluna “Mortos T_{SFT} ”. As quantidades de mutantes que permaneceram vivos após a aplicação dos casos de teste são representadas na coluna “Reman. Vivos”.

O escore de mutação após esse primeiro passo é representado pela coluna “MS-Ini”. Depois de aplicado os casos de teste, são necessários reavaliar e identificar os mutantes equivalentes.

Apesar de a ferramenta fazer uma pré-análise dos mutantes e classificar alguns como equivalentes, é necessário completar essa análise, pois a ferramenta faz a detecção automática apenas de mutantes gerados por alguns operadores que atuam em expressões de *pointcuts* e declaração de *advices*.

Depois de refeita a identificação dos mutantes equivalentes, um novo escore de mutação é calculado. Esse novo escore é representado pela coluna “MS-Inter”. Se após essa análise, o escore de mutação não atingir o valor de 1, são adicionados novos casos de teste ao conjunto já existente. Esses novos casos de teste estão representados na coluna “Testes Adds”, e a totalização é apresentada na coluna “TM”. Por fim, na coluna “MS-Final” encontra-se o escore de mutação final depois de todo o processo de análise dos mutantes e adequação do conjunto de testes.

Para o programa *Telecom* (linha 2), por exemplo, após a compilação dos mutantes e detecção automática dos equivalentes, restaram 53 mutantes vivos (Coluna “Vivos”). A aplicação dos casos de teste matou 31 desses mutantes, restando ainda assim 22 mutantes vivos. Com isso, o escore de mutação inicial é de 0,58. A partir dos mutantes ainda vivos, classificou-se manualmente 10 mutantes como equivalentes, restando ainda 12 mutantes como vivos. Dessa forma o escore de mutação foi recalculado, mudando de 0,58 para 0,72, como pode ser visualizado na Coluna “MS-Inter”. Após essa identificação manual dos mutantes equivalentes, foram acrescentados ao conjunto de testes 4 novos casos de teste, totalizando 67 casos de teste. Por fim, na Coluna “MS-Final” representa o escore de mutação do programa *Telecom* após o processo de aplicação dos casos de teste, análise de mutantes e adequação do conjunto de casos de teste.

4.3.3 Preparação dos Dados

A partir da adequação do conjunto de casos de teste foi possível realizar o cruzamento entre todos os operadores e obter o escore de mutação de cada operador em relação aos demais operadores do conjunto total de operadores. Dessa forma, é possível saber a capacidade de um conjunto de casos de teste adequado a um determinado operador ser também adequado aos outros operadores do conjunto. Uma parte dessa relação pode ser vista na Tabela 4.4.

Tomando como exemplo o operador ABAR (1ª linha), tem-se o escore de mutação de ABAR em relação a todos os operadores. O conjunto de casos de teste adequados a ABAR, por exemplo, possui escore de mutação 0,985 para operadores DAPO e 0,860 para operadores PCCE, ou seja, o conjunto de casos de teste adequado para os mutantes gerados pelo operador ABAR é também considerado adequado para os mutantes gerados pelo operador DAPO. Porém, esse mesmo conjunto não é considerado adequado para os mutantes gerados pelo ope-

Tabela 4.4: Escore de mutação por operador do conjunto Set1

Op/Op	ABAR	ABHA	ABPR	APER	APSR	DAPC	DAPO	PCCE	...	Média
ABAR	1,000	0,935	0,943	1,000	0,896	1,000	0,985	0,860	...	0,925
ABHA	0,996	1,000	0,943	1,000	0,993	1,000	0,985	0,937	...	0,960
ABPR	0,617	0,651	1,000	1,000	0,723	1,000	0,985	0,528	...	0,773
APER	0,142	0,296	0,558	1,000	0,387	0,611	0,532	0,195	...	0,336
APSR	0,523	0,636	0,797	1,000	1,000	1,000	0,985	0,504	...	0,720
DAPC	0,441	0,501	0,715	1,000	0,511	1,000	0,985	0,402	...	0,620
DAPO	0,133	0,236	0,363	0,482	0,285	0,450	1,000	0,160	...	0,274
PCCE	0,871	0,891	0,824	1,000	0,694	1,000	0,985	1,000	...	0,900
Média	0,539	0,552	0,594	0,663	0,518	0,736	0,701	0,476	...	-

rador PCCE (0,860), pois ele está abaixo do valor mínimo para a relação de inclusão.

Neste estudo, foi considerado como valor mínimo para essa relação de inclusão empírica dos operadores, o índice de 0,95. O valor desse índice foi determinado com base no estudo de Barbosa (1998).

Embora Barbosa (1998) tenha usado em seu estudo o índice de 0,99, neste estudo decidiu-se fazer uma redução desse índice para que fosse possível obter um conjunto inicial de operadores de tamanho similar ao identificado pela autora.

A partir desses dados é também possível realizar uma síntese dos dados obtidos como, por exemplo, os operadores com maior escore de mutação, ou seja, operadores que possuem as maiores médias dos escores de mutação quando aplicado a outros operadores, assim como os operadores de maior custo, que são os operadores que mais geraram mutantes para o conjunto de programas e os operadores com maior dificuldade de satisfação (*strength*), que são operadores que possuem menor escore de mutação médio quando executados com testes adequados para mutantes gerados por outros operadores. Uma parte dessa síntese pode ser visualizada na Tabela 4.5, onde são exibidos apenas os 10 primeiros operadores ordenados em ordem decrescente de escore de mutação, *strength* e custo.

4.3.4 Coleta de Dados

Nesta seção, o procedimento essencial é aplicado no conjunto de programas Set1 e os passos do procedimento são descritos e apresentados a seguir.

Passo 1: Selecionar operadores que determinam alto escore de mutação.

Para a aplicação desse passo foi determinado o valor de $0,75 \pm 0,02$ como limiar mínimo do escore de mutação médio para inclusão de operadores no conjunto CE_{pre} , que é um conjunto

Tabela 4.5: Escore de Mutação, strength e custo dos operadores.

MS		Strength		Custo	
Operador	Valor	Operador	Valor	Operador	Valor
ABHA	0,960	PWIW	0,556	PWIW	456
PWIW	0,953	PCCE	0,524	ABAR	75
ABAR	0,925	PCTT	0,508	ABPR	62
PCCE	0,901	POAC	0,505	PCCE	57
POAC	0,873	APSR	0,482	POPL	54
ABPR	0,733	PCGS	0,471	ABHA	52
APSR	0,720	ABAR	0,461	POAC	50
DAPC	0,620	ABHA	0,448	PCCT	31
POPL	0,572	POPL	0,415	PCLO	30
PSWR	0,564	ABPR	0,406	APSR	18
...

prévio de operadores essenciais. Apenas os operadores que possuem escore de mutação médio igual ou superior ao limiar são selecionados para compor o CE_{pre} .

O valor mínimo para inclusão foi determinado com base no estudo de Barbosa (1998). Embora a autora do estudo original tenham optado por um índice maior ($0,90 \pm 0,005$), decidiu-se por aplicar neste estudo um índice um pouco menor com o objetivo de obter um conjunto CE_{pre} com tamanho similar ao conjunto obtido no estudo de Barbosa (1998). A partir desse índice obteve-se 6 operadores para compor o conjunto CE_{pre} definido a seguir:

$$CE_{pre} = \{ABHA, PWIW, ABAR, PCCE, POAC, ABPR\}$$

Passo 2: Procurar selecionar um operador de cada classe de mutação

No passo 2 é determinado que seja incluído ao conjunto CE_{pre} pelo menos um operador de cada classe de mutação. Observa-se que correntemente o CE_{pre} não possui operadores do grupo 2, que modelam defeitos relacionados a expressões de declarações em AspectJ.

Dos cinco operadores do grupo 2 (DAPC, DAPO, DAIC, DEWC, DSSR), apenas dois operadores geraram mutantes para o Set1, sendo eles: DAPC e DAPO. Observa-se que neste estudo o índice determinado para relação de inclusão empírica – relação que mede por meio do escore de mutação se o conjunto de casos de teste adequado para um determinado operador é também adequado a outro operador – é de $MS \geq 0,95$. Por meio da relação de inclusão empírica os dois operadores do grupo 2 aplicados ao conjunto Set1 são incluídos empiricamente pelo conjunto CE_{pre} .

Dessa forma, nesse passo não foi acrescentado nenhum operador ao conjunto. Portanto, o conjunto CE_{pre} ficou composto da seguinte forma:

$$CE_{pre} = \{ABHA, PWIW, ABAR, PCCE, POAC, ABPR\}$$

Passo 3: Avaliar inclusão empírica entre operadores de mutação

A partir do conjunto CE_{pre} estabelecido anteriormente, são analisadas as relações de inclusão empírica entre os operadores do conjunto CE_{pre} , a fim de reduzir o tamanho do conjunto preliminar de operadores. Na Tabela 4.6 pode-se visualizar a relação de inclusão de todos os operadores do conjunto CE_{pre} .

Tabela 4.6: Relação de inclusão empírica entre os operadores do conjunto Set1

Relação de Inclusão	Op.Evidência	Score de Mutação
{'ABHA','PWIW','ABAR','PCCE','POAC'} → {'ABPR'}	ABPR	0,943
{'ABPR','PWIW','ABAR','PCCE','POAC'} → {'ABHA'}	ABHA	0,999
{'ABPR','ABHA','ABAR','PCCE','POAC'} → {'PWIW'}	PWIW	0,743
{'ABPR','ABHA','PWIW','PCCE','POAC'} → {'ABAR'}	ABAR	0,996
{'ABPR','ABHA','PWIW','ABAR','POAC'} → {'PCCE'}	PCCE	0,937
{'ABPR','ABHA','PWIW','ABAR','PCCE'} → {'POAC'}	POAC	0,921

Considerando o escore de mutação da relação de inclusão, os operadores candidatos a exclusão do conjunto CE_{pre} por já serem considerados inclusos empiricamente pelos demais operadores formam o seguinte conjunto: $CadElimin = \{ABHA, ABAR\}$.

Como o processo de eliminação dos operadores do conjunto CE_{pre} é feita de forma incremental, o primeiro operador a ser excluído será aquele com maior escore de mutação do conjunto $CadElimin$. Nesse caso, o operador ABHA possui escore de mutação 0,999, ou seja, é o mais incluído empiricamente em relação aos demais operadores do conjunto.

Após a exclusão do operador ABHA, o conjunto CE_{pre} foi reavaliado e verificou-se que os valores dos escores de mutação da relação de inclusão empírica permaneceram os mesmos. A nova relação de inclusão após a exclusão do operador ABHA, pode ser visualizado na Tabela 4.7.

Tabela 4.7: Relação de inclusão empírica entre os operadores do conjunto Set1 após a exclusão do operador ABHA

Relação de Inclusão	Op.Evidência	Score de Mutação
{'PWIW','ABAR','PCCE','POAC'} → {'ABPR'}	ABPR	0,943
{'ABPR','ABAR','PCCE','POAC'} → {'PWIW'}	PWIW	0,743
{'ABPR','PWIW','PCCE','POAC'} → {'ABAR'}	ABAR	0,996
{'ABPR','PWIW','ABAR','POAC'} → {'PCCE'}	PCCE	0,937
{'ABPR','PWIW','ABAR','PCCE'} → {'POAC'}	POAC	0,921

A partir dessa nova reavaliação, o próximo operador do conjunto $CadElimin$ a ser excluído do conjunto CE_{pre} será o operador ABAR (MS = 0,996).

Após a exclusão do operador ABAR do conjunto CE_{pre} , ocorreram mudanças em alguns escores de mutação, porém nenhuma dessas mudanças incluíram empiricamente nenhum outro operador. Os valores dessa reavaliação podem ser melhor visualizados na Tabela 4.8.

Tabela 4.8: Relação de inclusão empírica entre os operadores do conjunto Set1 após a exclusão do operador ABAR

Relação de Inclusão	Op.Evidência	Score de Mutação
{'PWIW','PCCE','POAC'} → {'ABPR'}	ABPR	0,862
{'ABPR','PCCE','POAC'} → {'PWIW'}	PWIW	0,688
{'ABPR','PWIW','POAC'} → {'PCCE'}	PCCE	0,937
{'ABPR','PWIW','PCCE'} → {'POAC'}	POAC	0,921

Dessa forma, como mais nenhum outro operador foi considerado incluído empiricamente, o passo 3 do procedimento essencial é finalizado e é formado o seguinte conjunto CE_{pre} :

$$CE_{pre} = \{ABPR, PWIW, PCCE, POAC\}$$

Passo 4: Estabelecer uma estratégia incremental de aplicação

Com base nos os operadores estabelecidos no passo 3, é estabelecida uma ordem a qual os operadores devem ser aplicados, tomando como base o custo dos operadores. Os operadores devem ser aplicados seguindo a ordem crescente do operador de menor custo, para o operador de maior custo. Dessa forma, é estabelecida a seguinte ordem de aplicação: POAC (50 mutantes), PCCE (57 mutantes), ABPR (62 mutantes), PWIW (456 mutantes). O conjunto CE_{pre} formado ao final do passo 4 será o seguinte:

$$CE_{pre} = \{POAC, PCCE, ABPR, PWIW\}$$

Passo 5: Selecionar operadores que proporcionam incremento no escore de mutação

Para este passo, Barbosa (1998) determinou em seu estudo um índice de incremento mínimo (IIM) no valor de 0,001, que é o índice mínimo que um operador incrementa o escore de mutação do conjunto CE_{pre} . O valor do IIM utilizado para este trabalho é de 0,002.

A partir desse índice, observou-se que de todos os operadores de mutação – excetuando os operadores que já fazem parte do conjunto CE_{pre} – apenas os operadores PCLO e POPL não foram incluídos empiricamente pelo conjunto CE_{pre} . Dessa forma, o conjunto *CandIns* (conjunto de operadores candidatos a serem inseridos no CE_{pre}) é formado apenas pelos operadores PCLO e POPL.

Pelo fato de POPL e PCLO pertencerem à mesma classe de operadores, apenas um dos operadores – o que possuir maior *strength* – deverá ser inserido no conjunto CE_{pre} . Porém, neste conjunto, ambos os operadores não ofereceram nenhum incremento ao conjunto, permanecendo inalterado a composição do conjunto CE_{pre} .

Dessa forma, ao final do passo 5 nenhum operador é acrescentado ao conjunto CE_{pre} , permanecendo com a seguinte composição:

$$CE_{pre} = \{POAC, PCCE, ABPR, PWIW\}$$

Passo 6: Selecionar operadores de alto *strength*

Para esse passo, Barbosa (1998) definiu um índice mínimo de *strength* denominado IMS (Índice Mínimo de Strength), que é o índice mínimo de *strength* que os operadores que não fazem parte do conjunto CE_{pre} , e que ainda não foram incluídos empiricamente pelo próprio conjunto CE_{pre} devem ter para compor o Conjunto Essencial final. O IMS definido para este trabalho é de $0,400 \pm 0,02$, um valor pouco maior ao determinado no estudo de BARBOSA, de $0,300 \pm 0,005$.

A partir desse índice têm-se os seguintes operadores: “PWIW”, “PCCE”, “PCTT”, “POAC”, “APSR”, “PCGS”, “ABAR”, “ABHA”, “POPL”, “ABPR” e “PCLO”. Destes, os operadores “PWIW”, “PCCE”, “POAC” e “ABPR” são desconsiderados por já fazerem parte do conjunto CE_{pre} . Dos demais operadores, apenas os operadores “POPL” e “PCLO” não são incluídos empiricamente pelo conjunto CE_{pre} , portanto somente dois operadores serão considerados como candidatos a inserção no conjunto CE_{pre} .

Pelo fato de POPL e PCLO pertencerem a mesma classe de operadores, apenas um dos dois operadores será incluído no conjunto CE_{pre} . O operador POPL (0,415) será então o único a ser incluído no conjunto CE_{pre} , por possuir *strength* maior que o do operador PCLO (0,385).

Ao final do passo 6, tem-se o Conjunto Essencial final CE , que é o conjunto resultante da aplicação do Procedimento Essencial:

$$CE = \{POAC, POPL, PCCE, ABPR, PWIW\}$$

Seguindo a diretriz estabelecida no passo 4 do procedimento, na qual os operadores devem ser aplicados seguindo a ordem de custo de cada operador, o operador POPL (54 mutantes) – último operador adicionado ao conjunto – foi alocado entre os operadores POAC (50 mutantes) e PCCE (57 mutantes).

4.3.5 Análise dos Dados - Conjunto Set1

Nesta seção analisam-se os resultados alcançados com a aplicação do Procedimento Essencial. Para isso, compara-se o custo do conjunto *CE* obtido com a aplicação do procedimento com o custo do conjunto completo de operadores. A Tabela 4.9 sumariza o custo de cada operador do conjunto *CE*. Mostra-se na tabela, respectivamente, o nome do operador, o número de mutantes, o número de mutantes compiláveis, o número de mutantes classificados automaticamente como equivalentes pela ferramenta *Proteum/AJ*, e o número de mutantes significativos (isto é, mutantes mortos pelos casos de teste, ou classificados manualmente como equivalentes.)

Tabela 4.9: Custo de cada operador essencial do conjunto Set1

Operador	Custo	Mut.Compil	Mut.Equiv.	Total Mut.
POAC	50	44	0	44
POPL	54	50	40	10
PCCE	57	55	0	55
ABPR	62	20	0	20
PWIW	456	446	402	44
Total	679	615	442	173

Em uma primeira análise, com o conjunto *CE* formado foi possível obter uma redução de 26,4%, considerando os 922 mutantes gerados pelo conjunto completo de operadores e os 679 mutantes gerados pelos operadores essenciais.

Porém, em uma análise mais minuciosa, descartou-se os mutantes anômalos e os mutantes que foram classificados automaticamente como equivalentes pela ferramenta. Em ambos os casos não foi necessário nenhum esforço manual para identificar esses mutantes.

Dessa forma, a partir dos 922 mutantes gerados, 464 mutantes foram classificados automaticamente como equivalentes e 87 como anômalos, restando apenas 371 mutantes significativos. Comparando esse número com o valor de 173 (número de mutantes significativos gerados pelos operadores do conjunto *CE*), obteve-se uma redução de custo de 53%. Esta redução é um pouco menor que os 65% alcançados por Barbosa (1998) em seu estudo. Contudo, é uma redução de custo expressiva.

4.4 Aplicação do Procedimento Essencial no Conjunto 2

Esta seção detalha a aplicação do Procedimento Essencial no segundo conjunto de programas, denominado *Set2*. A seção é iniciada com uma breve descrição das aplicações, e do planejamento e desenvolvimento dos dados de testes, isto é, mutantes e casos de teste.

4.4.1 Seleção dos Programas

O conjunto de programas selecionados para compor esse segundo grupo é composto por seis pequenas aplicações escritas em AspectJ. As aplicações selecionadas fazem parte de um conjunto de aplicações que possuem versões equivalentes tanto no paradigma orientado a objetos quanto no paradigma OA. A escolha por aplicações com essas características deveu-se ao fato de que os artefatos produzidos neste trabalho também seriam utilizados em outros estudos conduzidos no mesmo grupo de pesquisa do autor desta dissertação. Em particular, os conjuntos de testes seriam reutilizados em um trabalho que envolve a comparação de testes inter-paradigmas (orientado a objetos e orientado a aspectos).

Os programas selecionados nesse segundo grupo podem ser visualizados na Tabela 4.10

Tabela 4.10: Descrição das aplicações do Experimento

Aplicação	LOC*	Classes	Aspectos
1.TelecomTimingBilling	206	6	2
2.ShopSystem	405	9	8
3.ChainOfResponsability	150	6	2
4.AtmLog	528	12	1
5.VendingMachine	259	10	1
6.Chess	1004	16	1
Total	2552	59	15

*É considerado apenas as linhas de código, excluindo comentários e linhas em branco.

TelecomTimingBilling é um sistema simulador de telefonia que realiza cálculos de tempo e custo de cada ligação realizada. O sistema é desenvolvido e distribuído pela The Eclipse Foundation (2010) junto com o AspectJ. Os cálculos de tempo e custo são feitos pelos aspectos, os quais identificam o tipo de chamada (local ou longa distância), cronometram o tempo da ligação e, a partir da taxa atribuída a cada ligação, realizam os cálculos de custo de cada ligação.

ShopSystem é um sistema que simula um loja *online*, desenvolvido por Bartsch e Harrison (2008). Nesse sistema, cada usuário (administrador ou cliente) desempenha um papel diferente na loja, e cada um possui acesso a diferentes funções do sistema. Os responsáveis por gerenciar as funcionalidades à qual cada usuário tem acesso são desempenhadas pelos aspectos.

ChainOfResponsability é um programa simples que promove a interação do usuário com uma *interface* gráfica, e para cada interação é exibida uma mensagem com a ação que o usuário acabou de executar. O programa foi desenvolvido por Hannemann e Kiczales (2002) para aplicar em seu estudo o padrão de projeto *chain of responsibility* em programas Java e AspectJ. O usuário, ao interagir com a *interface*, opta por clicar em um botão pressionando juntamente a tecla *Shift*, *Ctrl* ou *Alt*. Para cada uma dessas ações, os aspectos são responsáveis por exibir uma mensagem específica na tela.

AtmLog é um sistema que simula as transações de um cliente em um terminal bancário. Esse sistema além de realizar as funções básicas de um terminal bancário, como depósitos, saques e verificar saldo da conta, ele registra em um arquivo de *log* todas as operações realizadas até o momento no terminal. Essa funcionalidade de registro em *log* é desempenhada por um aspecto.

VendingMachine é um sistema que simula uma máquina de vendas automática. Nele, o usuário insere moedas, escolhe o produto e, quando necessário, o troco é retornado ao usuário. O gerenciamento do itens selecionados e do montante envolvido na compra é feito por um aspecto.

Chess é um típico jogo de xadrez. É um jogo com uma *interface* simples, e permite apenas jogar com outro jogador real. A cada movimento, são exibidas no próprio jogo algumas mensagens como, por exemplo, qual jogador deve ser o próximo a mover uma peça, aviso de movimentos incorretos, mensagem de sucesso quando há um vencedor no jogo. Esse conjunto de mensagens é gerido por um aspecto.

4.4.2 Geração dos Conjuntos AM-Adequados

Para esse conjunto de programas foi necessária a criação dos planos de teste e dos casos de teste para os programas. Conforme mencionado na Seção 4.4.1, essas duas atividades foram realizadas em colaboração com outro estudante de mestrado, pois o mesmo utilizou as mesmas aplicações em seu estudo. Inicialmente o conjunto era composto por 12 aplicações, ou seja, foram construídos 12 planos de teste e casos de teste para os 12 programas. Porém, após a criação dos planos e casos de teste, apenas 6 dos 12 programas foram considerados relevantes para aplicação do critério análise de mutantes. Esses programas foram descritos na Seção 4.4.1.

Criação do Plano de Teste

Antes da criação dos casos de teste, é necessário a construção do plano de testes. No plano de testes foram definidos os requisitos de cada programa e, para cada requisito, foram definidas as condições de entrada, as classes válidas, as classes inválidas e os valores limites. Um exemplo referente à funcionalidade “Saque da Conta”, do programa *AtmLog*, pode ser visto na Tabela 4.11.

Criação dos Casos de Teste

Na criação dos casos de teste, assim como no conjunto *Set1*, foi aplicado o critério Teste Funcional Sistemático (LINKMAN; VINCENZI; MALDONADO, 2003). Como explicado na Subseção

Tabela 4.11: Exemplo de Tabela de Plano de Testes com classes de equivalência e valores limites

UC01: Withdraw value			
Condição de entrada	Classe Válida	Classe Inválida	Valores Limite
Saque de um valor da conta:			
Valor saque	(C1) valor saque <= saldo conta	(I1) valor saque > saldo conta	(B1) valor saque = 0 (B2) valor saque = saldo (B3) valor saque = saldo+1
Condição de saída	Classe Válida	Classe Inválida	
Mensagem de sucesso	(O1) exibe mensagem: "saque realizado com sucesso"		
Registro no log	(O2) operação é registrada no arquivo de log		

4.3.2, com a utilização do critério SFT, cada classe de equivalência deve ser coberta por pelo menos dois casos de teste. Dessa forma, determinou-se o seguinte padrão de nomenclatura para a construção dos casos de teste:

"test[Classe]_[metodo]_[IDClasseEquivalencia]" (caso de teste)
 "test[Classe]_[metodo]_[IDClasseEquivalencia]_SFT" (caso de teste adicional para critério SFT)

A seguir, como exemplo, é exibido o trecho de código de um caso de teste referente ao método `authenticateUser` da classe `Log` do programa `AtmLog`.

Aplicação do Critério Análise de Mutantes

Nos programas do conjunto `Set2`, foram aplicados os mesmos 24 operadores de mutação aplicados nos programas do conjunto `Set1`. O resumo da aplicação desses operadores nos programas pode ser visualizado na Tabela 4.12.

Tabela 4.12: Mutantes gerados a partir das seis aplicações do conjunto Set2)

Aplicação	mG1	mG2	mG3	Total	Equiv. Autom.	Anom	Mut. Vivos
1. TelecomTimingBilling	47	2	12	61	27	6	28
2. ShopSystem	311	24	19	354	178	38	138
3. ChainOfResponsability	33	0	2	35	24	4	7
4. AtmLog	67	0	32	99	47	22	30
5. VendingMachine	34	1	6	41	27	6	8
6. Chess	25	0	6	31	12	6	13
Total	517	27	77	621	315	82	224

As Colunas "mG1", "mG2" e "mG3" representam as quantidades de mutantes gerados por cada grupo de operadores de mutação e a Coluna "Total" representa a soma dos mutantes

```
1  public void testLog_authenticateUser_C4
    () {
2      bankDatabase.authenticateUser
        (12345, 54321);
3      Log.printLog();
4      assertEquals("User:12345",allOutput
        .toString().trim());
5  }
6  public void
    testLog_authenticateUser_C4_SFT(){
7      bankDatabase.authenticateUser
        (12345, 54321);
8      bankDatabase.authenticateUser
        (98765, 56789);
9      Log.printLog();
10     assertEquals("User:12345\n"+"User:
        98765", allOutput.toString().
        trim());
11 }
```

Figura 4.1: Exemplo de caso de teste - método *authenticateUser* da Classe *Log*

gerados pelos operadores.

Tomando como exemplo o programa *VendingMachine* – quinta linha da Tabela 4.12 –, foram gerados 41 mutantes, dos quais 34 mutantes pertencem ao grupo 1 de operadores (operadores que modelam defeitos em pointcuts), um mutante do grupo 2 (operadores para declarações gerais) e seis mutantes do grupo 3 (operadores para definição e implementação de *advices*). Dos 41 mutantes gerados, a ferramenta classificou automaticamente 27 mutantes como equivalentes, 6 mutantes como anômalos (mutantes não compiláveis) e 8 mutantes como vivos.

Adequação do Conjunto de Programas

Após a aplicação do critério análise de mutantes, foram aplicados os casos de teste para o conjunto de programas. O resultado da aplicação e adequação desses casos de teste ao conjunto de mutantes pode ser visualizado na Tabela 4.13.

Após a compilação dos mutantes e a detecção automática dos mutantes equivalentes e anômalos pela ferramenta Proteum/AJ, no programa *Chess* (linha 6), por exemplo, restaram apenas 13 mutantes vivos. A partir desses mutantes vivos, foram aplicados os casos de teste. Com a aplicação dos casos de teste foi possível matar 6 dos 13 mutantes vivos, restando ainda sete mutantes. O escore de mutação após a aplicação dos casos de teste foi calculado e é representado pela coluna “MS-Ini” com o valor 0,462. Depois foram classificados manualmente três mutantes como equivalentes, chegando a um escore de mutação de 0.600, representado pela Co-

Tabela 4.13: Resultado do teste de mutação para as seis aplicações do conjunto Set2

Aplicação	Vivos	Mortos T_{SFT}	Reman. Vivos	MS-Ini	Equiv. Man.	MS- Inter	Testes Adds	TM	MS- Final
1. TelecomTimingBilling	28	11	17	0,39	8	0,55	1	23	1,00
2. ShopSystem	138	99	39	0,72	16	0,81	1	37	1,00
3. ChainOfResponsability	7	3	4	0,43	4	1,00	-	6	1,00
4. AtmLog	30	13	17	0,43	14	0,81	1	16	1,00
5. VendingMachine	8	7	1	0,88	1	1,00	0	11	1,00
6. Chess	13	6	7	0,46	3	0,60	1	43	1,00
Total	224	139	85	0,62	46	0,78	4	136	1,00

luna “MS-Inter”. Para a adequação do conjunto de casos de teste foi adicionado mais um caso de teste ao conjunto, totalizando 43 casos de teste (Coluna “TM”). Por fim, após essa adequação foi recalculado o escore de mutação, obtendo o escore de mutação final (Coluna “MS-Final”) de um, ou seja, 100% dos mutantes mortos.

Ressalta-se que o baixo número de casos de teste adicionados aos conjuntos SFT-adequados já era esperado. No conjunto Set1, por exemplo, houve um acréscimo de apenas 5% para atingir 100% de cobertura de mutantes. No trabalho original do critério SFT, os testes adequados para esse critério foram capazes matar todos os mutantes gerados (LINKMAN; VINCENZI; MALDONADO, 2003), evidenciando que esse critério resulta em conjuntos de testes bastante significativos.

4.4.3 Preparação dos Dados

Similarmente à preparação dos dados realizada para o conjunto Set1, a partir da adequação do conjunto de casos de teste, é possível realizar o cruzamento entre os operadores e obter o escore de mutação de um operador em relação aos demais. Pode-se então identificar a capacidade de um conjunto de casos de teste adequados para um operador ser adequado também a outros operadores do conjunto. Uma parte dessa relação pode ser visualizada na Tabela 4.14

Tabela 4.14: Escore de mutação por operador do conjunto Set2

Op/Op	ABAR	ABHA	ABPR	APSR	DAPC	DAPO	DSSR	PCCE	...	Média
ABAR	1,000	0,947	0,765	0,953	1,000	1,000	0,000	0,916	...	0,863
ABHA	0,983	1,000	0,765	1,000	1,000	1,000	1,000	1,000	...	0,954
ABPR	0,124	0,295	1,000	0,000	0,032	0,489	0,000	0,065	...	0,292
APSR	0,404	0,516	0,000	1,000	0,968	0,511	1,000	0,569	...	0,520
DAPC	0,534	0,644	0,451	0,953	1,000	1,000	0,000	0,675	...	0,644
DAPO	0,344	0,477	0,451	0,618	0,798	1,000	0,000	0,380	...	0,500
DSSR	0,000	0,040	0,000	0,047	0,000	0,000	1,000	0,019	...	0,081
PCCE	0,534	0,684	0,451	1,000	1,000	1,000	1,000	1,000	...	0,745
Média	0,489	0,574	0,472	0,629	0,655	0,699	0,333	0,528	...	-

A partir dos dados mostrados na Tabela 4.14 é realizada uma síntese do cruzamento entre os operadores, que pode ser melhor visualizada na Tabela 4.15, as quais contêm o escore de mutação, o *strength* e custo dos operadores, respectivamente, em ordem decrescente.

Tabela 4.15: Escore de Mutação, strength e custo dos operadores.

MS		Strength		Custo	
Operador	Valor	Operador	Valor	Operador	Valor
ABHA	0,954	DSSR	0,667	PWIW	335
PWIW	0,885	PWIW	0,596	PCLO	51
POAC	0,871	PCCR	0,573	POPL	44
ABAR	0,863	ABPR	0,528	PCCE	40
POPL	0,748	ABAR	0,511	POAC	35
PCCE	0,745	PCCE	0,472	ABPR	28
DAPC	0,644	POAC	0,468	DAPC	24
APSR	0,520	PCTT	0,451	ABAR	21
DAPO	0,500	ABHA	0,426	ABHA	21
PCLO	0,439	POPL	0,379	APSR	7
...

4.4.4 Aplicação Passo-a-Passo do Procedimento Essencial

Nesta seção são descritos todos os passos do Procedimento Essencial aplicados no conjunto Set2. O resultado parcial é exibido ao final de cada etapa.

Passo 1: Selecionar operadores que determinam alto escore de mutação

Para a aplicação desse passo foi utilizado o mesmo índice definido para o conjunto Set1, isto é, $MS = 0,75 \pm 0,02$, como limiar de escore de mutação médio para inclusão de operadores com os maiores escores no conjunto CE_{pre} . A partir desse índice obteve-se os seguintes operadores para compor o conjunto CE_{pre} :

$$CE_{pre} = \{ABHA, PWIW, POAC, ABAR, POPL, PCCE\}$$

Passo 2: Procurar selecionar um operador de cada classe de mutação

No passo 2 é determinado que seja incluído ao conjunto CE_{pre} pelo menos um operador de cada classe de mutação. Observa-se que correntemente o CE_{pre} não possui operadores do grupo 2, que modelam defeitos relacionados a expressões de declarações em AspectJ. Dos cinco operadores do grupo 2 (DAPC, DAPO, DAIC, DEWC, DSSR), apenas três geraram mutantes para o conjunto Set2, sendo eles: DAPC, DAPO e DSSR.

Na relação de inclusão empírica – índice também determinado no conjunto *Set1* – os três operadores aplicados no conjunto *Set2* já foram considerados incluídos empiricamente pelo conjunto CE_{pre} . Dessa forma, nesse passo não foi acrescentado nenhum operador ao conjunto CE_{pre} . O conjunto CE_{pre} formado ao final desse passo mantém-se o mesmo:

$$CE_{pre} = \{ABHA, PWIW, POAC, ABAR, POPL, PCCE\}$$

Passo 3: Avaliar inclusão empírica entre operadores de mutação

A partir do conjunto CE_{pre} corrente, são estabelecidas relações de inclusão empírica, que podem ser visualizadas na Tabela 4.16.

Tabela 4.16: Relação de inclusão empírica entre os operadores do conjunto *Set2*

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'ABHA','PWIW','POAC','ABAR','POPL'} → {'PCCE'}	PCCE	1,000
{'PCCE','PWIW','POAC','ABAR','POPL'} → {'ABHA'}	ABHA	1,000
{'PCCE','ABHA','POAC','ABAR','POPL'} → {'PWIW'}	PWIW	0.632
{'PCCE','ABHA','PWIW','ABAR','POPL'} → {'POAC'}	POAC	0.971
{'PCCE','ABHA','PWIW','POAC','POPL'} → {'ABAR'}	ABAR	0.983
{'PCCE','ABHA','PWIW','POAC','ABAR'} → {'POPL'}	POPL	0.983

Considerando o índice de escore de mutação de $MS \geq 0.95$ – o mesmo determinado para o conjunto *Set1* – os operadores de mutação candidatos a exclusão por já serem incluídos empiricamente formam o seguinte conjunto: $CadElimin = \{PCCE, ABHA, POPL, ABAR, POAC\}$.

Como o processo de eliminação dos operadores do conjunto é feita de forma incremental, o primeiro operador a ser excluído será aquele com maior escore de mutação do conjunto *CadElimin*. Nesse conjunto em particular, os operadores PCCE e ABHA possuem o mesmo escore de mutação ($MS = 1$). Entretanto, o Procedimento Essencial não prescreve uma solução em uma situação de “empate”, ou seja, qual operador deve ser escolhido para ser retirado do conjunto CE_{pre} . Sendo assim, tomou-se a decisão de, a partir desse ponto, seguir com a aplicação do Procedimento Essencial por dois caminhos paralelos: (i) removendo o operador PCCE; e (ii) removendo o operador ABHA. Adotou-se a nomenclatura CE_{pre-1} para o conjunto CE_{pre} no caso (i) e CE_{pre-2} para o conjunto CE_{pre} no caso (ii). Ao final da aplicação de todos os passos, fazer-se-á a união dos dois conjuntos de operadores essenciais resultantes.

- **Aplicação do Passo 3 no caso (i) – CE_{pre-1}**

Nesse caminho o primeiro operador candidato a ser excluído do conjunto CE_{pre-1} foi o operador PCCE ($MS = 1$). Após a exclusão do operador PCCE, o conjunto CE_{pre-1} foi

reavaliado e verificou-se que houve mudança no escore de mutação apenas na relação de inclusão do operador ABHA, que passou de 1 para 0,958. A nova relação de inclusão dos operadores após a exclusão do operador PCCE pode ser visualizada na Tabela 4.17.

Tabela 4.17: Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-1} após a exclusão do operador PCCE

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'PWIW','POAC','ABAR','POPL'} → {'ABHA'}	ABHA	0,958
{'ABHA','POAC','ABAR','POPL'} → {'PWIW'}	PWIW	0,632
{'ABHA','PWIW','ABAR','POPL'} → {'POAC'}	POAC	0,971
{'ABHA','PWIW','POAC','POPL'} → {'ABAR'}	ABAR	0,983
{'ABHA','PWIW','POAC','ABAR'} → {'POPL'}	POPL	0,983

Após a exclusão de PCCE, o conjunto *CadElimin* é composto pelos seguintes operadores: POPL, ABAR, POAC e ABHA. Ao analisar o conjunto o próximo operador a ser eliminado do conjunto será o operador POPL (MS = 0,983) por ser o mais incluído empiricamente. Depois da exclusão do POPL, o conjunto CE_{pre} foi reavaliado e verificou-se que todos os valores na relação de inclusão permaneceram os mesmos. A nova relação de inclusão empírica pode ser vista na Tabela 4.18.

Tabela 4.18: Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-1} após a exclusão do operador POPL

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'PWIW','POAC','ABAR'} → {'ABHA'}	ABHA	0,958
{'ABHA','POAC','ABAR'} → {'PWIW'}	PWIW	0,632
{'ABHA','PWIW','ABAR'} → {'POAC'}	POAC	0,971
{'ABHA','PWIW','POAC'} → {'ABAR'}	ABAR	0,983

Na sequência, o próximo operador a ser eliminado será o operador ABAR (MS = 0,9829). Após a eliminação do operador ABAR, os valores da nova relação de inclusão permaneceram os mesmos. A nova relação pode ser visualizada na Tabela 4.19

Tabela 4.19: Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-1} após a exclusão do operador ABAR

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'PWIW','POAC'} → {'ABHA'}	ABHA	0,958
{'ABHA','POAC'} → {'PWIW'}	PWIW	0,632
{'ABHA','PWIW'} → {'POAC'}	POAC	0,971

Seguindo a mesma lógica, o operador POAC é eliminado e o conjunto CE_{pre} é reavaliado. Após a essa reavaliação, foi verificada uma mudança no escore de mutação para o operador ABHA (reduzindo de 0,958 para 0,947), deixando, assim, de ser incluído

empiricamente pelos demais operadores do conjunto CE_{pre} . Essa nova relação pode ser visualizada na Tabela 4.20.

Tabela 4.20: Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-1} após a exclusão do operador POAC

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'PWIW'} → {'ABHA'}	ABHA	0,947
{'ABHA'} → {'PWIW'}	PWIW	0,632

Portanto, após o Passo 3 do Procedimento Essencial para o caminho CE_{pre-1} , restaram apenas 2 operadores: ABHA e PWIW, os quais passam a compor o seguinte conjunto:

$$CE_{pre-1} = \{ABHA, PWIW\}$$

• **Aplicação do Passo 3 no caso (ii) – CE_{pre-2}**

Nesse caminho o primeiro operador excluído do conjunto foi o operador ABHA (MS=1). Após a exclusão do operador ABHA, o conjunto CE_{pre} foi reavaliado e verificou-se que ocorreram mudanças nos escores de mutação dos operadores POAC e PCCE. Essa nova relação de inclusão dos operadores pode ser visualizada na Tabela 4.21.

Tabela 4.21: Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-2} após a exclusão do operador ABHA

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'PWIW','POAC','ABAR','POPL'} → {'PCCE'}	PCCE	0,928
{'PCCE','POAC','ABAR','POPL'} → {'PWIW'}	PWIW	0,632
{'PCCE','PWIW','ABAR','POPL'} → {'POAC'}	POAC	0,944
{'PCCE','PWIW','POAC','POPL'} → {'ABAR'}	ABAR	0,983
{'PCCE','PWIW','POAC','ABAR'} → {'POPL'}	POPL	0,983

Depois da exclusão de ABHA, o conjunto *CadElimin* é formado pelos seguintes operadores POPL e ABAR. O próximo operador candidato a ser eliminado do conjunto será o operador POPL (MS = 0,983234), pois possui escore de mutação maior que o operador ABAR (MS = 0,982905). Apesar dos valores na tabela indicarem que os operadores POPL e ABAR possuem os mesmos escores de mutação, devido ao arredondamento para três casas decimais, foi levado em consideração os valores sem o arredondamento. Dessa forma, o operador POPL é eliminado do conjunto por ser mais incluído empiricamente do que o operador ABAR.

Após a exclusão do operador POPL, o conjunto CE_{pre} foi reavaliado e verificou-se que todos os valores da relação de inclusão permaneceram os mesmos. Essa nova relação de inclusão pode ser visualizado na Tabela 4.22.

Tabela 4.22: Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-2} após a exclusão do operador POPL

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'PWIW','POAC','ABAR'} → {'PCCE'}	PCCE	0,928
{'PCCE','POAC','ABAR'} → {'PWIW'}	PWIW	0,632
{'PCCE','PWIW','ABAR'} → {'POAC'}	POAC	0,944
{'PCCE','PWIW','POAC'} → {'ABAR'}	ABAR	0,983

Como pode ser visto na Tabela 4.22, o último operador do conjunto *CadElimin* a ser eliminado será o operador ABAR (MS = 0,982905). Após a eliminação do operador ABAR, o conjunto CE_{pre} foi reavaliado e os valores da relação de inclusão empírica permaneceram os mesmos. Dessa forma, nenhum outro operador foi considerado incluído empiricamente pelos demais. A relação de inclusão entre os operadores que restaram pode ser visualizada na Tabela 4.23

Tabela 4.23: Relação de inclusão empírica entre os operadores do conjunto Set2 CE_{pre-2} após a exclusão do operador ABAR

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'PWIW','POAC'} → {'PCCE'}	PCCE	0,928
{'PCCE','POAC'} → {'PWIW'}	PWIW	0,632
{'PCCE','PWIW'} → {'POAC'}	POAC	0,944

Após o Passo 3 do Procedimento Essencial para o caminho CE_{pre-2} , restaram apenas 3 operadores: PCCE, PWIW e POAC, os quais passaram a compor o seguinte conjunto:

$$CE_{pre-2} = \{PCCE, PWIW, POAC\}$$

Passo 4: Estabelecer uma estratégia incremental de aplicação

Como citado no passo anterior, o procedimento essencial para esse conjunto necessitou ser dividido em dois subprocessos concorrentes, e consequentemente essa divisão se estenderá aos demais passos do procedimento, inclusive este.

- **Aplicação do Passo 4 no caso (i) (CE_{pre-1})**

Para o conjunto CE_{pre-1} , a aplicação dos operadores deverá seguir a ordem do operador de menor custo para o de maior custo. Dessa forma, é estabelecida a seguinte ordem de aplicação: ABHA (21 mutantes), PWIW (335 mutantes). E o conjunto CE_{pre-1} ficou com a seguinte ordem de aplicação de operadores:

$$CE_{pre-1} = \{ABHA, PWIW\}$$

- **Aplicação do Passo 4 no caso (ii) (CE_{pre-2})**

Seguindo a mesma lógica, estabelece-se a seguinte ordem de aplicação dos operadores do conjunto CE_{pre-2} : POAC (35 mutantes), PCCE (40 mutantes), PWIW (335 mutantes). O conjunto CE_{pre-2} configura-se da seguinte forma:

$$CE_{pre-2} = \{POAC, PCCE, PWIW\}$$

Passo 5: Selecionar operadores que proporcionam incremento no escore de mutação

Nesse passo considerou-se o mesmo índice de incremento mínimo (IIM) aplicado no conjunto $Set1$, com o valor $IIM = 0,02$. Esse índice foi considerado para os dois caminhos: CE_{pre-1} e CE_{pre-2} .

- **Aplicação do Passo 5 no caso (i) (CE_{pre-1})**

Observou-se que de todos os operadores de mutação, com exceção dos operadores que já compõem o conjunto CE_{pre-1} , apenas o operador ABPR não foi incluído empiricamente pelo conjunto CE_{pre-1} . Dessa forma o operador ABPR compõe o conjunto de operadores candidatos a serem inseridos no conjunto CE_{pre-1} , denominado $CandIns$.

O operador ABPR não conseguiu alcançar o índice de incremento superior ao estabelecido, dessa forma ao final do passo 5 deste subprocesso não foi incluído nenhum novo operador ao conjunto CE_{pre-1} . Ao final do passo 5 o conjunto CE_{pre-1} ficou composto da seguinte forma:

$$CE_{pre-1} = \{ABHA, PWIW\}$$

- **Aplicação do Passo 5 no caso (ii) (CE_{pre-2})**

Observou-se que todos os operadores de mutação, com exceção dos operadores que já compõem o conjunto CE_{pre-2} , apenas o operador ABPR não foi incluído empiricamente pelo conjunto CE_{pre-2} . Portanto o operador ABPR é o único operador candidato, conjunto $CandIns$, a ser inserido no conjunto CE_{pre-2} . Porém o operador ABPR não conseguiu alcançar o índice de incremento maior do que o estabelecido.

Dessa forma, ao final do passo 5 nenhum operador foi incluído ao conjunto CE_{pre-2} , ficando com a seguinte composição:

$$CE_{pre-2} = \{POAC, PCCE, PWIW\}$$

Passo 6: Selecionar operadores de alto *strength*

Para esse passo definiu-se um índice mínimo de *strength* (*IMS*) de $0,400 + -0,02$ que é o *strength* mínimo que um operador deve ter para ser incluído no conjunto CE_{pre} . Esse índice é o mesmo definido para o conjunto *Set1* e foi também adotado para os dois caminhos em execução para o conjunto *Set2*.

- **Aplicação do Passo 6 no caso (i) (CE_{pre-1})**

A partir desse índice, no subprocesso CE_{pre-1} obteve-se os seguintes operadores: DSSR, PWIW, PCCR, ABPR, ABAR, PCCE, POAC, PCTT, ABHA. Destes, os operadores PWIW e ABHA são excluídos por já fazerem parte do conjunto CE_{pre-1} . Dos demais operadores, apenas o ABPR não foi incluído empiricamente. Dessa forma, o operador ABPR será acrescentado ao conjunto CE_{pre-1} .

Ao final do passo 6 é obtido o conjunto final de operadores essenciais representado por $CE-1$:

$$CE-1 = \{ABHA, ABPR, PWIW\}$$

Seguindo a diretriz do passo 4 do procedimento, o operador ABPR (28 mutantes) – último operador a ser adicionado ao conjunto – foi alocado logo após o operador ABHA (21 mutantes).

- **Aplicação do Passo 6 no caso (ii) (CE_{pre-2})**

A partir desse índice, com relação ao conjunto CE_{pre-2} obteve-se os seguintes operadores: DSSR, PWIW, PCCR, ABPR, ABAR, PCCE, POAC, PCTT, ABHA. Destes, os operadores PWIW, PCCE E POAC são excluídos por já fazerem parte do conjunto CE_{pre-2} . Dos demais operadores, apenas o ABPR não está incluído empiricamente. Dessa forma, o operador ABPR será acrescentado ao conjunto CE_{pre-2} .

Ao final do passo 6, obteve-se o seguinte conjunto final de operadores essenciais representado por $CE-2$:

$$CE-2 = \{ABPR, POAC, PCCE, PWIW\}$$

Seguindo a diretriz do passo 4 do procedimento, o operador ABPR (28 mutantes) – último operador a ser adicionado ao conjunto – foi alocado à frente do operador POAC (35 mutantes).

Junção dos Conjuntos *CE-1* e *CE-2*:

Ao final do passo 6, tem-se os conjuntos finais dos dois caminhos paralelos *CE-1* e *CE-2*, com a seguinte composição: $CE-1 = \{ABHA, ABPR, PWIW\}$ e $CE-2 = \{ABPR, POAC, PCCE, PWIW\}$

Decidiu-se que para este grupo de programas, devido a subdivisão ocorrida durante o procedimento essencial, o conjunto final de operadores será composto pelo conjunto união dos dois conjuntos gerados pelos subprocessos *CE-1* e *CE-2*, ou seja, $CE \leftarrow CE-1 \cup CE-2$. Assim sendo, o conjunto essencial para o conjunto **Set2** ficou composto da seguinte forma:

$$CE = \{ABHA, ABPR, POAC, PCCE, PWIW\}$$

A união dos dois conjuntos essenciais foi realizada, também seguindo a diretriz estabelecida no passo 4 do procedimento, que determina que os operadores devem ser ordenados segundo o custo de cada operador. Os operadores, portanto, ficaram dispostos da seguinte forma: ABHA (21 mutantes), ABPR (28 mutantes), POAC (35 mutantes), PCCE (40 mutantes) e PWIW (335 mutantes)

4.4.5 Análise dos Dados - Conjunto **Set2**

Nesta subseção, analisam-se os resultados alcançados com a aplicação do Procedimento Essencial no conjunto **Set2**. Para isso, é feita uma comparação do custo do conjunto *CE* obtido pelo Procedimento Essencial com o custo do conjunto completo de operadores.

A Tabela 4.24 sumariza o custo de cada operador do conjunto *CE*. A Tabela mostra, respectivamente, o nome do operador, o número de mutantes, o número de mutantes compiláveis, o número de mutantes classificados automaticamente como equivalentes pela *Proteum/AJ*, e o número de mutantes significativos (isto é, mutantes mortos pelos casos de teste ou classificados manualmente como equivalentes.)

Tabela 4.24: Custo de cada operador essencial do conjunto **Set2**

Operador	Custo	Mut.Compil	Mut.Equiv.	Total Mut.
ABHA	21	21	0	21
PWIW	335	315	296	19
ABPR	28	4	0	4
PCCE	40	40	0	40
POAC	35	23	1	22
Total	459	403	297	106

Uma análise preliminar aponta uma redução de 26,08%, considerando os 621 mutantes gerados pelo conjunto completo de operadores, e os 459 mutantes do conjunto essencial. Porém, em uma análise mais minuciosa, descartou-se os mutantes anômalos e os mutantes que foram classificados automaticamente como equivalentes pela *Proteum/AJ*. Conforme destacado durante a análise do conjunto Set1, em ambos os casos não foi necessário nenhum esforço manual para identificar esses mutantes.

A partir dos 621 mutantes gerados, 315 mutantes foram classificados como equivalentes e 82 como anômalos, restando apenas 224 mutantes significativos. Comparando esse número com o valor de 106 (número de mutantes significativos gerados pelos operadores do conjunto CE) obteve-se uma redução de 52,67% no custo.

4.5 Comparação entre os conjuntos Set1 e Set2

Nesta seção é realizada uma comparação e uma visão geral dos operadores essenciais obtidos para os conjuntos Set1 e Set2. Na Figura 4.2 é possível visualizar essa comparação entre os operadores obtidos nos dois conjuntos.

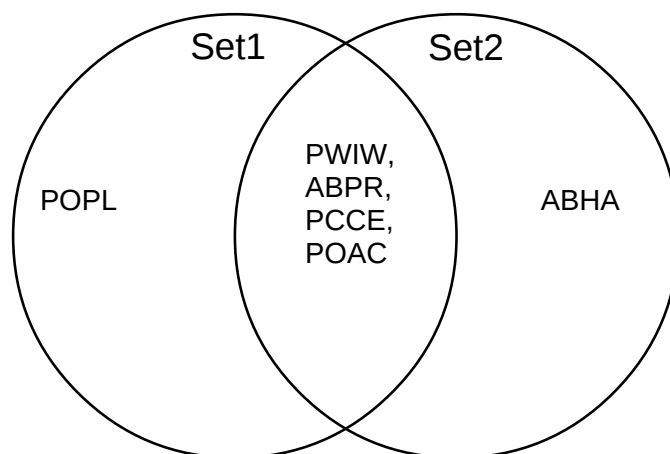


Figura 4.2: Diagrama comparativo dos dois conjuntos essenciais

Para ambos os conjuntos de aplicações obteve-se um conjunto essencial de operadores de mutação composto por 5 elementos, sendo que a intersecção desses conjuntos essenciais contém os operadores PWIW, ABPR, PCCE e POAC. Os únicos operadores que diferem os dois conjuntos são os operadores POPL (Set1) e ABHA (Set2).

Dessa forma, não é possível afirmar que o conjunto de operadores formado pela intersecção

ou pela união dos conjuntos essenciais $CE-1$ e $CE-2$, seria um conjunto essencial de operadores ideal para os programas utilizados em ambos os conjuntos $Set1$ e $Set2$.

4.6 Considerações Finais

Visando a redução de custo em teste de mutação no paradigma OA, foram descritos, neste capítulo, dois experimentos aplicando o Procedimento Essencial em dois conjuntos de programas, denominados $Set1$ e $Set2$. Para cada um desses conjuntos foi obtido um conjunto essencial de operadores, $CE-1$ e $CE-2$, respectivamente. Realizou-se uma análise das reduções de custo obtida por cada conjunto essencial, e posteriormente foi realizada uma comparação entre os operadores essenciais dos dois conjuntos.

A partir dessa relação entre os dois conjuntos, no Capítulo 5 é realizada uma segunda análise, na qual o Procedimento Essencial é aplicado a um conjunto maior de programas, denominado $SetAll$, que é na verdade o conjunto união dos programas utilizados nos conjuntos $Set1$ e $Set2$, ou seja, $SetAll \leftarrow Set1 \cup Set2$. O objetivo dessa reaplicação do procedimento é fazer uma reaplicação do Procedimento Essencial nos dois conjuntos de programas, agora unidos, e investigar a relação entre os operadores obtidos nos conjuntos essenciais $CE-all$, $CE-1$ e $CE-2$.

Capítulo 5

UM ESTUDO COMPLEMENTAR - O PROCEDIMENTO ESSENCIAL APLICADO AO CONJUNTO SETALL

5.1 Considerações Iniciais

Com o objetivo de realizar uma análise mais detalhada dos conjuntos essenciais obtidos nos Conjuntos Set1 e Set2, e na tentativa de entender as variações de operadores obtidos nos dois conjuntos, é realizada neste capítulo a aplicação do Procedimento Essencial em um terceiro conjunto de programas, este, por sua vez, com uma quantidade maior de programas. Os resultados da aplicação do procedimento neste terceiro conjunto são comparados com os resultados obtidos nas Seções 4.3.5 e 4.4.5.

Posteriormente, uma análise das reduções de custo e escores de mutação alcançados nos três conjuntos é comparada com outros estudos que realizaram experimentos semelhantes ao relatados neste trabalho.

5.2 Aplicação do Procedimento Essencial no conjunto SetAll

Nesta seção será apresentada a aplicação do Procedimento Essencial no conjunto de programas denominado SetAll, que é um conjunto composto pela união dos programas presentes nos conjuntos Set1 e Set2. A seção se inicia com uma breve descrição dos programas utilizados e da origem dos dados de testes, isto é, mutantes e casos de testes. Nas demais seções são

apresentadas a aplicação do Procedimento Essencial nesse conjunto de programas, uma análise do conjunto essencial obtido e ao final é realizada uma comparação com o conjunto essencial obtido nos conjuntos Set1 e Set2, cujos resultados foram apresentados no Capítulo 4.

5.2.1 Seleção dos Programas

O conjunto de programas utilizados nesse experimento resultou da união dos programas utilizados nos conjuntos Set1 e Set2. Esse conjunto união é denominado SetAll e é dado pela relação $\text{SetAll} \leftarrow \text{Set1} \cup \text{Set2}$. Os programas utilizadas nesse conjunto união podem ser melhor visualizados na Tabela 5.1. Observa-se que se optou por exibir novamente as características gerais desses programas em uma única tabela, pois tais informações encontram-se espalhadas por diversas tabelas ao longo do Capítulo 4. A mesma observação é válida para as Tabelas 5.2 a 5.3.

Tabela 5.1: Descrição das aplicações do conjunto União dos Experimentos Set1 e Set2

Application	LOC*	Classes	Aspectos
1.BankingSystem	199	9	6
2.Telecom	251	6	3
3.ProdLine	537	8	8
4.FactorialOptimizer	39	1	1
5.MusicOnline	150	7	2
6.VendingMachine	64	1	3
7.PointBoundsChecker	44	1	1
8.StackManager	77	4	3
9.PointShadowManager	66	2	1
10.Math	53	1	1
11.AuthSystem	89	3	2
12.SeqGen	205	8	4
13.TelecomTimingBilling	206	6	2
14.ShopSystem	405	9	8
15.ChainOfResponsability	150	6	2
16.AtmLog	528	12	1
17.VendingMachine	259	10	1
18.Chess	1004	16	1
Total	4326	110	50

*São considerados apenas as linhas de código, excluindo comentários e linhas em branco.

5.2.2 Geração dos Conjuntos AM-adequados

Aplicação do Critério Análise de Mutantes

Por se tratar dos mesmos programas aplicados nos Conjuntos Set1 e Set2, os dados referentes à aplicação do critério análise de mutantes são os mesmos dos outros dois conjuntos.

Dessa forma, na Tabela 5.2 são reapresentados, agora de forma unificada, os resultados da aplicação desse critério.

Tabela 5.2: Mutantes gerados a partir das 18 aplicações do conjunto união Set1 U Set2

Aplicação	mG1	mG2	mG3	Total	Equiv. Autom.	Anom	Mut. Vivos
1. BankingSystem	108	2	26	136	61	20	55
2. Telecom	82	2	27	111	46	12	53
3. ProdLine	158	0	41	199	125	16	58
4. FactorialOptimizer	14	0	15	29	8	6	15
5. MusicOnline	47	0	10	57	25	5	27
6. VendingMachine	82	2	29	113	58	8	47
7. PointBoundsChecker	46	0	24	70	32	10	28
8. StackManager	34	0	11	45	24	0	21
9. PointShadowManager	38	0	12	50	25	4	21
10. Math	16	0	4	20	13	0	7
11. AuthSystem	45	0	7	52	28	3	21
12. SeqGen	33	0	7	40	19	3	18
13. TelecomTimingBilling	47	2	12	61	27	6	28
14. ShopSystem	311	24	19	354	178	38	138
15. ChainOfResponsability	33	0	2	35	24	4	7
16. AtmLog	67	0	32	99	47	22	30
17. VendingMachine	34	1	6	41	27	6	8
18. Chess	25	0	6	31	12	6	13
Total	1220	33	290	1543	779	169	595

Adequação do conjunto de teste

Ressalta-se que os conjuntos AM-adequados para o conjunto SetAll também correspondem à união dos conjuntos AM-adequados dos conjuntos Set1 e Set2. O conjunto união AM-adequado dos conjuntos Set1 e Set2 é apresentado na Tabela 5.3.

5.2.3 Preparação dos Dados

De forma similar ao procedimento realizado para os estudos com os conjuntos Set1 e Set2, realizou-se o cruzamento entre os operadores de mutação e analisou-se o escore de mutação de um operador em relação aos demais. Assim, é possível observar a capacidade de um conjunto de testes adequado para um determinado operador ser também adequado para outros operadores. Uma parte dessa relação pode ser visualizada na Tabela 5.4

A partir desses dados é feita uma síntese do cruzamento entre os operadores, que pode ser melhor visualizada na Tabela 5.5. Nessa tabela, é possível observar, respectivamente, o escore de mutação, o *strength* e o custo de cada operador em ordem decrescente.

Tabela 5.3: Resultado do teste de mutação para as 18 aplicações do conjunto SetAll

Aplicação	Vivos	Mortos T_{SFT}	Reman. Vivos	MS-Ini	Equiv. Man.	MS- Inter	Testes Addts	TM	MS- Final
1. BankingSystem	55	55	0	1,00	-	-	-	58	1,00
2. Telecom	53	31	22	0,58	10	0,72	4	67	1,00
3. ProdLine	58	58	0	1,00	-	-	-	36	1,00
4. FactorialOptimizer	15	14	1	0,93	1	1,00	-	-	1,00
5. MusicOnline	27	22	5	0,81	2	0,88	2	48	1,00
6. VendingMachine	47	23	24	0,49	13	0,68	5	23	1,00
7. PointBoundsChecker	28	28	0	1,00	-	-	-	14	1,00
8. StackManager	21	21	0	1,00	-	-	-	15	1,00
9. PointShadowManager	21	13	8	0,62	5	0,81	2	14	1,00
10. Math	7	4	3	0,57	2	0,80	1	54	1,00
11. AuthSystem	21	17	4	0,81	1	0,85	2	19	1,00
12. SeqGen	18	4	14	0,22	8	0,40	3	42	1,00
13. TelecomTimingBilling	28	11	17	0,39	8	0,55	1	23	1,00
14. ShopSystem	138	99	39	0,72	16	0,81	1	37	1,00
15. ChainOfResponsability	7	3	4	0,43	4	1,00	-	6	1,00
16. AtmLog	30	13	17	0,43	14	0,81	1	16	1,00
17. VendingMachine	8	7	1	0,88	1	1,00	0	11	1,00
18. Chess	13	6	7	0,46	3	0,60	1	43	1,00
Total	590	424	166	0,72	88	0,85	23	526	1,00

Tabela 5.4: Escore de mutação por operador do conjunto Set-All

Op/Op	ABAR	ABHA	ABPR	APER	APSR	DAPC	DAPO	DSSR	PCCE	...	Média
ABAR	1,000	0,938	0,921	1,000	0,916	1,000	0,991	0,000	0,879	...	0,880
ABHA	0,995	1,000	0,921	1,000	0,996	1,000	0,991	1,000	0,957	...	0,966
ABPR	0,562	0,575	1,000	1,000	0,484	0,197	0,783	0,000	0,331	...	0,571
APER	0,121	0,233	0,490	1,000	0,259	0,104	0,387	0,000	0,112	...	0,229
APSR	0,514	0,610	0,700	1,000	1,000	0,972	0,791	1,000	0,533	...	0,704
DAPC	0,459	0,532	0,683	1,000	0,657	1,000	0,991	0,000	0,491	...	0,620
DAPO	0,165	0,288	0,374	0,482	0,395	0,728	1,000	0,000	0,240	...	0,337
DSSR	0,000	0,009	0,000	0,000	0,016	0,000	0,000	1,000	0,007	...	0,058
PCCE	0,835	0,847	0,779	1,000	0,801	1,000	0,991	1,000	1,000	...	0,891
Média	0,498	0,512	0,540	0,627	0,501	0,571	0,634	0,278	0,438	...	-

5.2.4 Aplicação Passo-a-Passo do Procedimento Essencial

Passo 1: Selecionar operadores que determinam alto escore de mutação

Para a aplicação desse passo foi utilizado o mesmo índice definido para os conjuntos Set1 e Set2. O valor $0,75 + -0,02$ foi utilizado como limiar do escore de mutação para inclusão de operadores com maiores escores no conjunto CE_{pre} .

A partir desse índice obteve-se os seguintes operadores para compor o conjunto CE_{pre} :

$$CE_{pre} = \{ABHA, PWIW, PCCE, ABAR, POAC\}$$

Tabela 5.5: Escore de Mutação, strength e custo dos operadores.

MS		Strength		Custo	
Operador	Valor	Operador	Valor	Operador	Valor
ABHA	0,966	DSSR	0,722	PWIW	791
PWIW	0,907	PWIW	0,610	POPL	98
PCCE	0,891	PCCE	0,562	PCCE	97
ABAR	0,880	POAC	0,545	ABAR	96
POAC	0,858	PCTT	0,537	ABPR	90
APSR	0,704	ABAR	0,502	POAC	85
DAPC	0,621	PCGS	0,500	PCLO	81
POPL	0,592	APSR	0,499	ABHA	73
ABPR	0,571	ABHA	0,488	PCTT	37
PCLO	0,444	POPL	0,471	DAPC	27
...

Passo 2: Procurar selecionar um operador de cada classe de mutação

Neste passo determina-se que o conjunto contenha pelo menos um operador de cada classe de mutação. Para o conjunto SetAll, apenas os operadores do grupo 2 (operadores que modelam defeitos relacionados a expressões de declarações em AspectJ) não estão presentes no conjunto CE_{pre} .

Dos operadores do grupo 2 (DAPC, DAPO, DAIC, DEWC, DSSR), apenas 3 desses operadores foram aplicados nesse conjunto, são eles: DAPC, DAPO, DSSR. Porém, por meio da relação de inclusão empírica, tais operadores já foram considerados incluídos empiricamente pelos demais operadores do conjunto CE_{pre} .

Dessa forma, neste passo não foi acrescentado nenhum operador ao conjunto. O conjunto CE_{pre} permaneceu o seguinte:

$$CE_{pre} = \{ABHA, PWIW, PCCE, ABAR, POAC\}$$

Passo 3: Avaliar inclusão empírica entre operadores de mutação

Considerando os operadores obtidos nos passos 1 e 2, são estabelecidas as relações de inclusão empírica mostradas na Tabela 5.6.

Tabela 5.6: Relação de inclusão empírica entre os operadores do conjunto Set-All

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'ABHA','PWIW','PCCE','ABAR'} → {'POAC'}	POAC	0,935
{'POAC','PWIW','PCCE','ABAR'} → {'ABHA'}	ABHA	1,000
{'POAC','ABHA','PCCE','ABAR'} → {'PWIW'}	PWIW	0,705
{'POAC','ABHA','PWIW','ABAR'} → {'PCCE'}	PCCE	0,957
{'POAC','ABHA','PWIW','PCCE'} → {'ABAR'}	ABAR	0,995

Considerando o índice de escore de mutação de 0,95 - o mesmo para os conjuntos *Set1* e *Set2* - os operadores candidatos a exclusão por serem considerados incluídos empiricamente pelos demais operadores são determinados pelo conjunto $CadElimin = \{ABHA, ABAR, PCCE\}$.

Como o processo de eliminação dos operadores é realizado de forma incremental, o primeiro operador a ser excluído do conjunto será o que for mais incluído empiricamente pelos demais operadores do conjunto CE_{pre} , ou seja, o operador com maior escore de mutação. Nesse caso, trata-se do operador ABHA ($MS = 1$).

Após a exclusão do operador ABHA, o conjunto CE_{pre} foi reavaliado e verificou-se que apenas o escore de mutação para o operador PWIW permaneceu inalterado. Essa nova reavaliação das relações de inclusão pode ser vista na Tabela 5.7.

Tabela 5.7: Relação de inclusão empírica entre os operadores do conjunto *Set-All*

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'PWIW','PCCE','ABAR'} → {'POAC'}	POAC	0,927
{'POAC','PCCE','ABAR'} → {'PWIW'}	PWIW	0,705
{'POAC','PWIW','ABAR'} → {'PCCE'}	PCCE	0,936
{'POAC','PWIW','PCCE'} → {'ABAR'}	ABAR	0,974

Após a essa nova relação é definido um novo conjunto *CadElimin* composto apenas por ABAR ($MS = 0,974$). Dessa forma o próximo candidato a ser excluído de CE_{pre} será o operador ABAR.

Após a exclusão de ABAR, o conjunto CE_{pre} é reavaliado e houve uma mudança no escore de mutação apenas o operador PWIW. O resultado dessa nova relação de inclusão pode ser visualizado na Tabela 5.8.

Tabela 5.8: Relação de inclusão empírica entre os operadores do conjunto *Set-All*

Relação de Inclusão	Op.Evidência	Escore de Mutação
{'PWIW','PCCE','ABAR'} → {'POAC'}	POAC	0,927
{'POAC','PCCE','ABAR'} → {'PWIW'}	PWIW	0,651
{'POAC','PWIW','ABAR'} → {'PCCE'}	PCCE	0,936

Após essa nova reavaliação do conjunto CE_{pre} , nenhum outro operador foi considerado incluído empiricamente pelo demais operadores. Ao final deste passo chegou-se ao seguinte conjunto CE_{pre} :

$$CE_{pre} = \{PWIW, PCCE, POAC\}$$

Passo 4: Estabelecer uma estratégia incremental de aplicação

Neste passo é estabelecida uma ordem de aplicação dos operadores, seguindo a ordem do operador de menor custo para o operador de maior custo. Dessa forma, os operadores deverão ser aplicados na seguinte ordem: POAC (85 mutantes), PCCE (97 mutantes), PWIW (791 mutantes).

Ao final do passo 4 têm-se o seguinte conjunto $CE_{pre} = \{POAC, PCCE, PWIW\}$.

$$CE_{pre} = \{POAC, PCCE, PWIW\}$$

Passo 5: Selecionar operadores que proporcionam incremento no escore de mutação

Neste passo foi determinado o mesmo índice de incremento mínimo estabelecido nos conjuntos *Set1* e *Set2*, isto é, definiu-se o valor de 0,02.

Observou-se que dentre os operadores de mutação – excetuando os operadores que já fazem parte do conjunto CE_{pre} – apenas os operadores ABPR e APSR não foram incluídos empiricamente pelos operadores de CE_{pre} . Como ABPR e APSR pertencem à mesma classe de operador de mutação, será incluído no conjunto CE_{pre} aquele que possuir o maior *strength*.

Porém, neste caso especificamente, independentemente do valor de *strength* de cada um dos operadores, ambos operadores não conseguiram alcançar o índice de incremento mínimo estabelecido.

Dessa forma, ao final do passo 5, o conjunto CE_{pre} permaneceu inalterado.

Passo 6: Selecionar operadores de alto strength.

Neste passo, o índice mínimo de *strength* é o mesmo definido para os conjuntos *Set1* e *Set2* ($0,400 + -0,02$). A partir desse índice têm-se os seguintes operadores: DSSR, PWIW, PCCE, POAC, PCTT, ABAR, PCGS, APSR, ABHA, POPL. ABPR, PCLO, PCCR, DAPC, PSWR. Destes, os operadores PWIW, PCCE, POAC são descartados pois já fazem parte do conjunto CE_{pre} .

Observando os operadores restantes, todos foram considerados incluídos empiricamente pelo conjunto CE_{pre} . Portanto, neste passo não foi acrescentado nenhum outro novo operador ao conjunto CE_{pre} . Dessa forma, ao final desse passo os operadores que formam o conjunto CE_{pre} será o conjunto essencial final determinado por *CE-All*, conforme apresentado a seguir:

$$CE-All = \{POAC, PCCE, PWIW\}$$

5.2.5 Análise dos Dados - Conjunto SetAll

Nesta subseção são analisados os resultados obtidos com a aplicação do Procedimento Essencial no conjunto SetAll. Para isso, comparou-se o custo do conjunto CE-All obtido pelo procedimento com o custo do conjunto completo de operadores. Inicialmente, a Tabela 5.9 sumariza o custo de cada operador do conjunto essencial CE-All. A tabela exhibe, respectivamente, o nome do operador, o total de mutantes gerados, o número de mutantes compiláveis, a quantidade de mutantes classificados automaticamente pela *Proteum/AJ* como equivalentes e a quantidade de mutantes significativos (isto é, mutantes mortos pelos casos de testes, ou classificados manualmente como equivalentes).

Tabela 5.9: Custo de cada operador essencial do conjunto Set-All

Operador	Custo	Mut.Compil	Mut.Equiv.	Total Mut.
POAC	85	67	1	66
PCCE	97	95	0	95
PWIW	791	761	698	63
Total	973	923	699	224

Similarmente à análise feita para os conjuntos Set1 e Set2, à primeira vista foi possível obter uma redução de 36,9%, considerando os 1543 mutantes gerados pelo conjunto completo de operadores e os 973 mutantes do conjunto essencial.

Desconsiderou-se então os mutantes anômalos e os mutantes que foram classificados automaticamente como equivalentes pela *Proteum/AJ*. Dos 1543 mutantes gerados, 779 mutantes foram classificados como equivalentes e 169 como anômalos, restando apenas 595 mutantes significativos. Comparando esse número com o valor de 224 (número de mutantes significativos gerados pelos operadores do conjunto CE-All) obteve-se uma redução de 62% no custo. Comparando esse resultado com os resultados obtidos para os conjuntos Set1 e Set2, observa-se um aumento na redução de custo quando um conjunto maior de aplicações foi analisado. Mais detalhes sobre as diferenças de resultados nos três estudos realizados são apresentados na próxima seção.

5.3 Comparação com os Conjuntos Set1 e Set2

Nesta seção é realizada uma análise comparativa entre os conjuntos de operadores obtidos com a aplicação do Procedimento Essencial nos conjuntos Set1, Set2 e SetAll. Na figura 5.1 é possível perceber que o conjunto SetAll é composto basicamente por operadores formados pela interseção dos conjuntos Set1 e Set2, com exceção do operador ABPR, que é comum apenas aos conjuntos Set1 e Set2.

Pelo fato de os conjuntos Set1 e Set2 serem compostos por programas diferentes, os operadores POPL e ABHA tem relevâncias diferentes em cada um dos conjuntos. Dessa forma, quando é feita a união dos programas no conjunto SetAll e aplicado o Procedimento Essencial, ambos os operadores não aparecem no conjunto essencial CE-All. Em contrapartida, apesar de o operador ABPR fazer parte de ambos os conjuntos, ele não está presente no conjunto CE-All pois, como mostrado na subseção 5.2.4, foi excluído antes mesmo da execução do procedimento por possuir score de mutação abaixo do limiar mínimo determinado no passo 1.

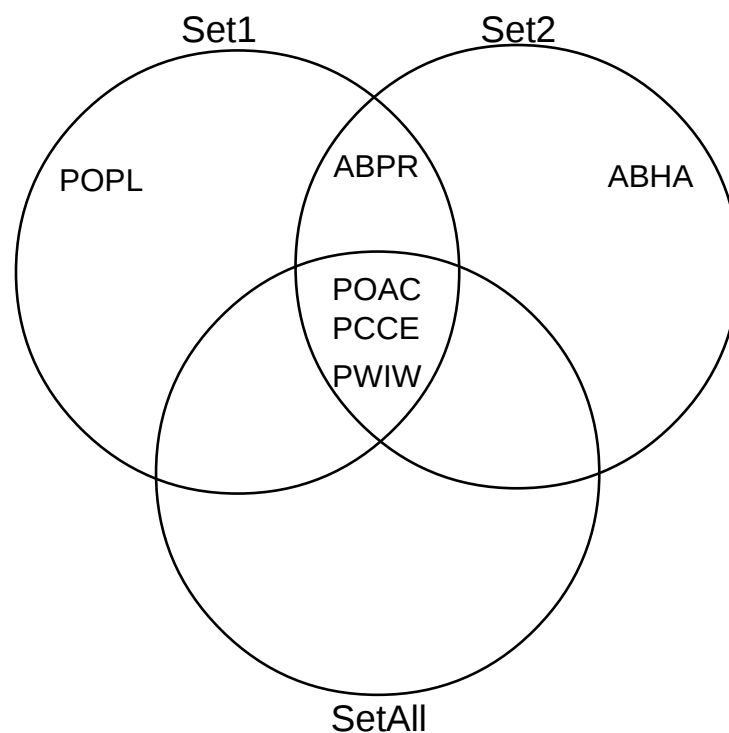


Figura 5.1: Diagrama comparativo dos operadores essenciais identificados para os conjuntos Set1, Set2 e SetAll de aplicações

Observou-se ainda que para os conjuntos Set1 e Set2 foram obtidos conjuntos essenciais compostos por cinco operadores cada. Para o conjunto SetAll, por outro lado, obteve-se um conjunto essencial um pouco mais reduzido, com apenas três operadores, ou seja, 85% menos operadores que o conjunto completo (20 operadores conseguiram ser aplicados ao conjunto de programas SetAll).

No conjunto SetAll percebeu-se que o Procedimento Essencial, diferentemente do comportamento mostrado nos conjuntos Set1 e Set2, mostrou uma tendência a apresentar um conjunto de operadores essenciais mais reduzido. Ressalta-se que esse comportamento foi observado durante o desenvolvimento deste estudo, sendo então necessária a realização de outros experimentos com as mesmas características e sistemática deste experimento, para obter uma conclusão

mais precisa acerca dessa relação entre o conjunto essencial e o tamanho do conjunto de programas.

Além dessa análise, foi realizada uma outra, mais detalhada, dos operadores obtidos com o procedimento nos 3 conjuntos de programas. A partir dos dados obtidos e descritos nos estudos envolvendo os conjuntos Set1, Set2 e SetAll, é apresentada na Tabela 5.10 uma aplicação dos conjuntos de testes adequados com respeito aos operadores essenciais no conjunto completo de mutantes gerados. Essa aplicação é feita de forma gradual, obedecendo a ordem de custos dos operadores, com o intuito de observar a evolução do escore de mutação à medida que os operadores essenciais são inseridos ao conjunto.

Tomando como exemplo o conjunto essencial de Set1, os testes para o operador POAC serão os primeiros a serem aplicados no conjunto completo de mutantes, pois o operador POAC é o mais barato dentre os essenciais identificados para o conjunto Set1 (50 mutantes). Quando os testes adequados aos mutantes de POAC são aplicados ao conjunto completo de mutantes, observa-se que escore de mutação já atinge o valor de 0,811. Logo após, ao acrescentar o operador POPL (54 mutantes) ao conjunto, obtêm-se um pequeno incremento no escore de mutação, obtendo-se o escore de 0,860. Esse processo vai ocorrendo até que sejam inseridos todos os operadores do conjunto essencial. Ao final das inserções, obtêm-se o escore de mutação do conjunto essencial de 0,940.

Tabela 5.10: Tabela com a inserção gradual dos operadores dos Conjuntos Essenciais

Combinações Set1	Total Execuções	Mortos	Equival.	Escore de Mutação
'POAC' → OP	13639	9917	1410	0.811
'POAC', 'POPL' → OP	13639	10456	1483	0.860
'POAC', 'POPL', 'PCCE' → OP	13639	10456	1483	0.860
'POAC', 'POPL', 'PCCE', 'ABPR' → OP	13639	10932	1679	0.914
'POAC', 'POPL', 'PCCE', 'ABPR', 'PWIW' → OP	13639	11172	1754	0.940
Combinações Set2	Total Execuções	Mortos	Equival.	Escore de Mutação
'ABHA' → OP	7137	5576	1164	0.934
'ABHA', 'ABPR' → OP	7137	5576	1164	0.934
'ABHA', 'ABPR', 'POAC' → OP	7137	5576	1164	0.934
'ABHA', 'ABPR', 'POAC', 'PCCE' → OP	7137	5576	1164	0.934
'ABHA', 'ABPR', 'POAC', 'PCCE', 'PWIW' → OP	7137	5576	1164	0.934
Combinações SetAll	Total Execuções	Mortos	Equival.	Escore de Mutação
'POAC' → OP	20779	15419	2563	0.846
'POAC', 'PCCE' → OP	20779	15819	2593	0.870
'POAC', 'PCCE', 'PWIW' → OP	20779	16471	2918	0.922

Na Tabela 5.10 foi possível observar a aplicação gradual dos operadores do conjunto essencial e o incremento no escore de mutação que cada operador proporciona. Com o intuito de observar o comportamento dos operadores do três conjuntos – Set1, Set2 e SetAll – durante

a formação do conjunto essencial, foram elaborados diagramas de Venn para cada passo do procedimento com os três conjuntos de programas. Esses diagramas são exibidos na Figura 5.2.

Levando em consideração as particularidades do conjunto **Set2** e a subdivisão ocorrida no mesmo durante a aplicação do procedimento, é possível observar que o operador PWIW foi o único operador que persistiu em todos os conjuntos – inclusive nos dois subprocessos de **Set2** – desde a execução do primeiro passo do procedimento, até a formação final do conjunto essencial. Mostra-se, dessa forma, a importância do operador PWIW na composição dos conjuntos essenciais. Outros dois operadores que se destacaram pelas suas constantes presenças em quase em todos os conjuntos essenciais – excetuando apenas um dos subprocessos de **Set2** – são os operadores PCCE e POAC. Destaca-se que os três operadores citados anteriormente são exatamente os mesmos operadores que compõem o conjunto essencial do conjunto **SetAll**.

Em face à variedade de aplicações utilizadas nos três conjuntos, esses operadores que se destacaram na execução dos procedimentos podem representar uma possível composição de um conjunto essencial ideal para outras aplicações deste paradigma. Porém, seria necessária a execução de outros experimentos para que fosse possível chegar a uma conclusão mais concreta.

5.4 Comparação com Outros Conjuntos Essenciais

No Capítulo 3 foram sumarizados alguns estudos que abordam a determinação de conjuntos essenciais de operadores de mutação. Entretanto, esses estudos foram realizados para linguagens de programação específicas como Fortran (OFFUTT et al., 1996) e C (BARBOSA, 1998).

Esta seção apresenta uma comparação entre as reduções de custo que podem ser obtidas com os operadores essenciais identificados neste trabalho, com as reduções de custo obtidas pelos conjuntos reduzidos de operadores de mutação relatados por outros autores em outras linguagens de programação. Em particular, os resultados deste trabalho são comparados com os resultados relatados por Offutt et al. (1996) e Barbosa (1998), pois se tratam de trabalhos que também buscam determinar um conjunto essencial de operadores, porém para programas desenvolvidos em diferentes linguagens de programação.

Na Tabela 5.11, são sumarizados os valores obtidos por OFFUTT et al. com a aplicação de cada um dos conjuntos de programas usando a mutação seletiva (*ES-S*, *RS-S* e *RE-S*) e o conjunto de operadores essenciais, dado por *E-S*, em um determinado conjunto de programas. Também são mostrados os valores das reduções de custo obtidas com os conjuntos essenciais – formados por meio da aplicação do Procedimento Essencial – nos experimentos realizados por BARBOSA, dados por *CE-27* e *CE-5*.

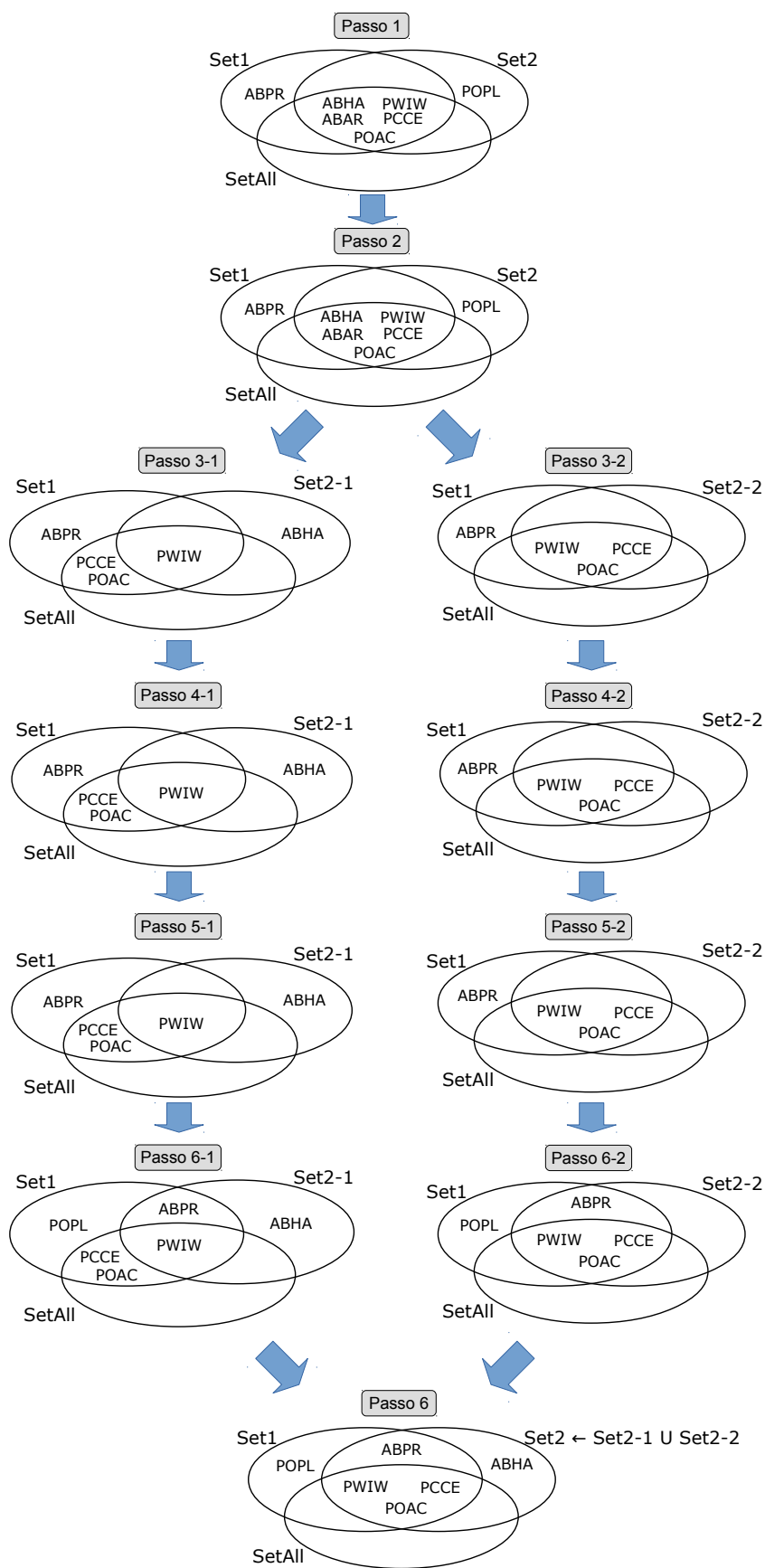


Figura 5.2: Evolução dos conjuntos CE_{pre} para cada conjunto de aplicações (Set1, Set2 e SetAll)

Juntamente com os estudos apresentados anteriormente, na Tabela 5.11 são listados os resultados alcançados pelos conjuntos essenciais desenvolvidos ao longo deste trabalho. Os resultados mostrados na tabela estão associados aos nomes dos autores. Os mesmos resultados são apresentados de forma mais clara no gráfico presente na Figura 5.3.

Tabela 5.11: Tabelas comparativa com as reduções de custos relatadas por outros estudos

Autor	Conjunto de Programas	% Redução	Linguagem
Offutt et al. (1996)	ES-S	71,52	Fortran
Offutt et al. (1996)	RS-S	22,44	Fortran
Offutt et al. (1996)	RE-S	6,04	Fortran
Offutt et al. (1996)	E-S	77,56	Fortran
Barbosa (1998)	CE-27	65,89	C
Barbosa (1998)	CE-5	82,04	C
Lacerda	Set1	53,37	AspectJ
Lacerda	Set2	52,68	AspectJ
Lacerda	SetAll	62,40	AspectJ

Para os estudos considerados, é possível observar que, independentemente da linguagem alvo, a determinação de um conjunto essencial de operadores tem se mostrado uma abordagem bastante efetiva para a redução de custo em teste de mutação. Apesar das diferenças evidentes nos percentuais de redução de custo, principalmente em relação aos estudos de Barbosa (1998), cujos percentuais variam de 65% a 82%, as reduções obtidas neste estudo são significativas, variando entre 52% e 62%. Ressalta-se que os operadores de mutação aplicados nos trabalhos de Offutt et al. (1996) e Barbosa (1998) são de granularidade mais refinada, atuando em detalhes de implementação internos às menores unidades de implementação, no caso funções e métodos, conforme caracterizadas por Vincenzi (2004) e Lemos et al. (2007). Os operadores para programas AspectJ, por sua vez, concentram suas mudanças nas interfaces entre as unidades. Exemplos são as mutações que ocorrem em pointcuts, que podem aumentar ou reduzir o número de interações entre os aspectos e o código base da aplicação em teste.

A partir dos dados relatados nos estudos, foi possível também realizar a comparação com os escores de mutação obtidos com cada conjunto essencial. Na Tabela 5.12 são descritos os escores obtidos com a aplicação dos operadores essenciais no conjunto completo de operadores.

Em geral, os valores dos escores de mutação alcançados nos estudos foram superiores a 92%. Apesar dos escores apresentados para este estudo serem inferiores aos escores apresentados nos estudos de Barbosa (1998) e Offutt et al. (1996), ainda assim representam um escore de mutação significativo. Essa constatação vem do fato de que os operadores para a linguagem AspectJ atuam em uma granularidade menos refinada do que os operadores de unidade para

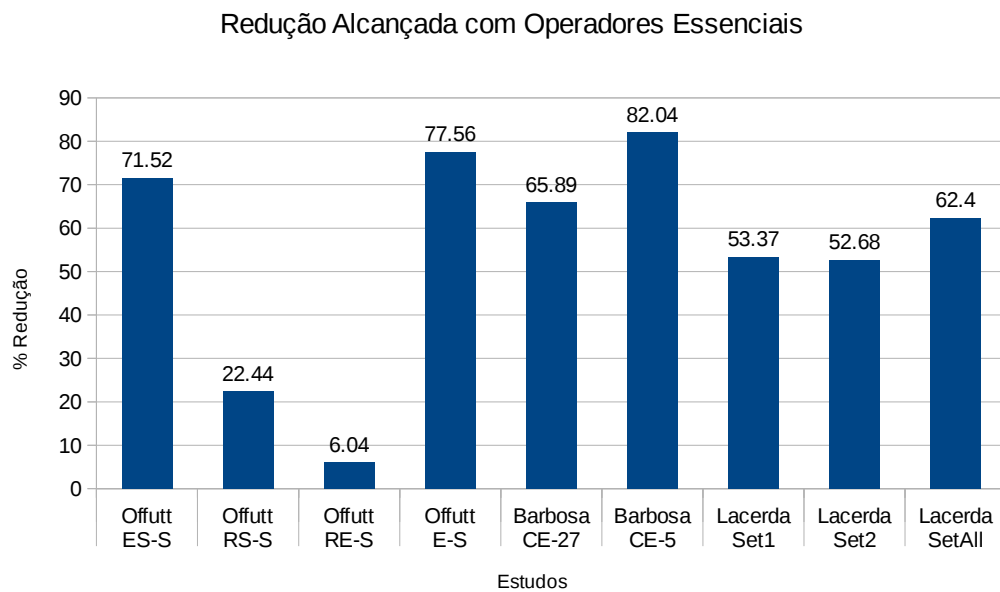


Figura 5.3: Gráfico comparativo com a redução de custo obtida nesse estudo e nos estudos de Offutt et al. (1996), Barbosa (1998)

programas Fortran e C. Uma estratégia combinada de aplicação de operadores de unidade e específicos para AspectJ poderiam levar a escores de mutação próximos a 100%, entretanto tal estratégia foge ao escopo deste trabalho.

Tabela 5.12: Tabelas comparativa com os escores de mutação alcançados por outros estudos

Autor	Estudo	MS	Linguagem
Offutt	ES-S	99,54	Fortran
Offutt	RS-S	97,31	Fortran
Offutt	RE-S	99,97	Fortran
Offutt	E-S	99,51	Fortran
Barbosa	CE-27	99,6	C
Barbosa	CE-5	99,7	C
Lacerda	Set1	94,00	AspectJ
Lacerda	Set2	93,35	AspectJ
Lacerda	SetAll	92,22	AspectJ

Os resultados mostram que a determinação de um conjunto essencial de operadores, independentemente da linguagem de programação, além de diminuir o custo do teste de mutação, se mostrou efetiva na manutenção da eficácia do teste em relação ao conjunto original. A comparação com os escores de mutação obtidos neste estudo com os escores de mutação relatados por outros estudos pode ser melhor visualizada no gráfico da Figura 5.4.

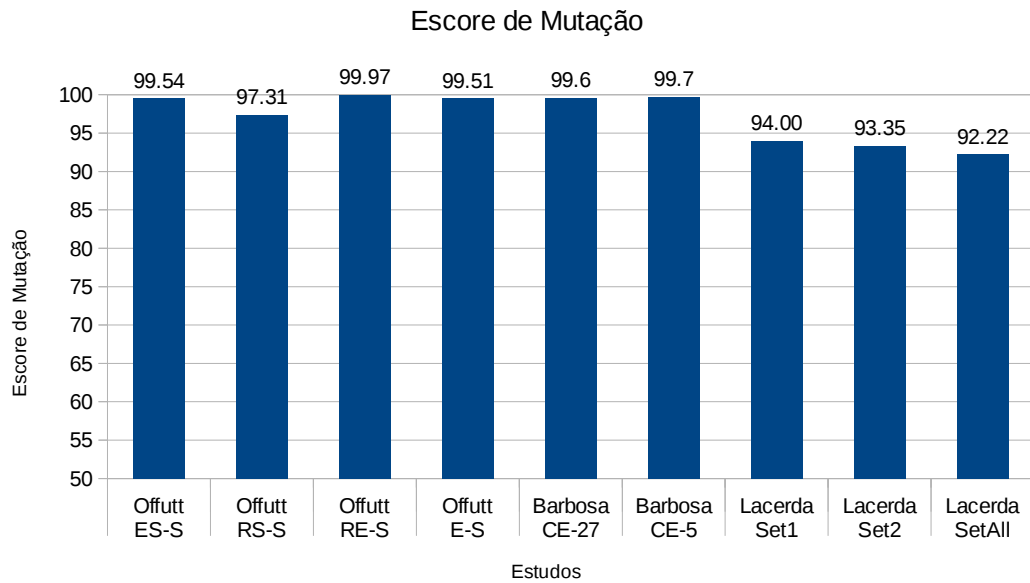


Figura 5.4: Gráfico comparativo com o escore de mutação obtido com o conjunto essencial formado nesse estudo e nos estudos de Offutt et al. (1996), Barbosa (1998)

5.5 Considerações Finais

Visando a investigação das variações nos conjuntos essenciais de operadores, foi descrito neste capítulo um terceiro experimento similar aos realizados no Capítulo 4. Porém, o conjunto de programas utilizados neste é composto pela união dos programas utilizados nos Conjuntos Set1 e Set2. Aplicou-se o Procedimento Essencial nesse terceiro conjunto de programas e obteve-se um conjunto essencial *CE-All*, composto por três operadores (POAC, PCCE e PWIW).

Observou-se que o conjunto essencial *CE-All* era composto, quase que na totalidade, por operadores resultantes da intersecção dos conjuntos essenciais *CE-1* e *CE-2*. Essa composição de operadores sugere que esse seja um conjunto ideal de operadores para outras aplicações semelhantes às utilizadas nos experimentos.

Após a aplicação do procedimento e análise dos operadores, foi realizada também uma análise das reduções de custo alcançadas com o procedimento. Posteriormente, os resultados dessas reduções foram comparados com os resultados das reduções obtidas por outros estudos que relataram determinação de um conjunto de operadores essenciais.

Capítulo 6

CONCLUSÃO E TRABALHOS FUTUROS

As atividades de Verificação e Validação (V&V) têm o objetivo de ajudar a aumentar a qualidade do software em desenvolvimento. Uma dessas atividades é o teste de software, cujo objetivo é revelar defeitos presentes nos artefatos produzidos (MYERS et al., 2004), que mesmo com o emprego de técnicas avançadas de engenharia e ferramentas de apoio, ainda persistem no software. Dentre os critérios seleção de casos de teste mais investigados pela comunidade científica, o teste de mutação (DEMILLO; LIPTON; SAYWARD, 1978) tem se mostrado uma importante alternativa na avaliação tanto do software em si, quanto do conjunto de testes utilizado para testá-lo.

Ainda em relação à qualidade de um software, outra maneira de melhorá-la está relacionada à aplicação de técnicas contemporâneas de desenvolvimento. Essas abordagens visam lidar com a crescente complexidade dos produtos de software. Pode-se citar como exemplo dessas abordagens a Programação Orientada a Objetos (POO), a qual se beneficia da decomposição de problemas baseada em objetos (BOOCH, 1994), e a Programação Orientada a Aspectos (KICZALES et al., 1997), a qual propõe-se a melhorar a modularização de interesses transversais, que podem representar tanto requisitos funcionais como requisitos não-funcionais do software.

Embora a POA ofereça benefícios no desenvolvimento do software, a aplicação dessa técnica (ou paradigma) pode introduzir novos tipos de defeitos de software. Dessa forma, para garantir a qualidade do software, analisou-se neste estudo a aplicação do critério Análise de Mutantes no contexto do paradigma OA. Porém, apesar de ter se mostrado bastante efetivo, a aplicação desse critério apresenta um custo muito elevado, devido à geração da grande quantidade de versões modificadas para um mesmo programa.

Muitos estudos têm proposto diferentes abordagens visando a redução de custo nesse critério.

Dentre esses, o estudo de Barbosa (1998) demonstrou ser muito promissor, pois além de buscar a redução de custo em teste de mutação, visa também manter a eficácia do conjunto de teste. Eles propõe um procedimento, denominado **Procedimento Essencial**, composto por seis passos que objetiva a determinação de um conjunto essencial de operadores de mutação, por meio da seleção dos melhores operadores.

Com o propósito de reduzir custo em teste de mutação em programas OA, o Procedimento Essencial foi analisado e adaptado para ser aplicado no paradigma OA. Ainda não há na literatura nenhum estudo relatando a obtenção de um conjunto reduzido de operadores em programas OA. Dessa forma, nesse estudo foram conduzidos três experimentos aplicando o Procedimento Essencial em programas do paradigma OA, como forma de avaliar a eficácia do procedimento.

Dois desses experimentos, foram conduzidos utilizando dois conjuntos distintos de programas desenvolvidos em AspectJ. Os conjuntos denominados, **Set1** e **Set2**, possuíam respectivamente 12 e 6 programas cada. Com a aplicação do Procedimento Essencial nos conjuntos foi possível obter uma redução de 53,37% no conjunto **Set1** e de 52,68% no Conjunto **Set2**, obtendo dessa forma uma significativa redução de custo. Além disso, os conjuntos essenciais obtidos nos dois experimentos, conseguiram obter um escore de mutação bastante satisfatório, 94% e 93% respectivamente.

Em um terceiro experimento, foi feita uma reaplicação do procedimento aplicado nos dois experimentos anteriores. Porém, neste, o conjunto de programas utilizados, é composto pela união dos programas utilizados nos conjuntos **Set1** e **Set2**, totalizando 18 programas. Na aplicação do procedimento para esse conjunto, obteve-se uma redução de custo de 62,4%. Além da grande redução alcançada, o conjunto essencial conseguiu atingir um escore de mutação de 92,2%. Apesar do escore obtido ser um pouco menor do que o escore alcançado nos outros dois experimentos, ainda assim, é um valor bastante expressivo para um conjunto reduzido de operadores.

Com a aplicação do procedimento obteve-se os seguintes conjuntos essenciais: $CE-1 = \{POAC, POPL, PCCE, ABPR, PWIW\}$, $CE-2 = \{ABHA, ABPR, POAC, PCCE, PWIW\}$ E $CE-All = \{POAC, PCCE, PWIW\}$. Nos três conjuntos essenciais foi possível observar a frequência dos operadores relacionados a pointcuts (POAC, PCCE, PWIW).

Considerando os resultados alcançados pelos operadores (reduções de custo e efetividade do teste) e as características desses operadores, sugere-se que o conjunto ideal de operadores essenciais para ser aplicado em qualquer conjunto de aplicação seja composto por operadores que modelam defeitos relacionados a pointcuts, pois eles geram menos mutantes que o conjunto de operadores total, e esses mutantes gerados são suficientes para manter a efetividade

do teste de mutação. Porém, ressalta-se que essa afirmação é apenas uma tendência levando-se em consideração as características dos programas utilizados neste trabalho. A variação dos operadores que formam o conjunto essencial dependerá das características de cada aplicação. Dessa forma, um conjunto de programas que possua, por exemplo, mais declarações intertipos e declarações gerais do que pointcuts poderá ter um conjunto de operadores essenciais diferente do que foi estabelecido nos experimentos realizados neste trabalho.

6.1 Contribuições

Pode-se destacar como principais contribuições deste trabalho:

- Aplicação experimental do Procedimento Essencial (BARBOSA, 1998) em três conjuntos de programas OA, desenvolvidos na linguagem AspectJ, e obtenção de um conjunto reduzido de operadores de mutação.
- Redução satisfatória do custo com a aplicação do procedimento em programas OA e obtenção de um alto score de mutação com o conjunto essencial. Obteve-se para os conjuntos *Set1*, *Set2* e *SetAll* scores de mutação com valores superiores a 92, os valores para cada conjunto são respectivamente 94, 93,35 e 92,22.
- Comparação entre os conjuntos essenciais de operadores obtidos nos três experimentos e análise das similaridades entre os conjuntos de operadores obtidos nos mesmos. Nos três conjuntos essenciais obtidos nos experimentos foi possível observar a frequência dos operadores PWIW, POAC e PCCE. Ressalta-se a importância do operador PWIW, devido à presença do mesmo na formação dos conjuntos essenciais nos três experimentos.
- Comparação dos resultados alcançados neste estudo com os resultados relatados em outros dois estudos – Offutt et al. (1996) e Barbosa (1998) – que também determinaram um conjunto essencial de operadores, porém para outras linguagens de programação. Por meio da obtenção de um conjunto essencial de operadores, Offutt et al. (1996) conseguiram em seu estudo obter reduções de custos que variam de 6% a 77%, já Barbosa (1998) conseguiu em seu estudo obter reduções que variam de 65% a 82%, e neste trabalho obteve-se reduções de custo de 52% a 62%.
- Realização de um mapeamento sistemático para caracterizar as principais estratégias de redução de custo em teste de mutação.

- Adaptação do Procedimento Essencial para aplicação em programas do paradigma OA. Foi realizada uma adaptação no Procedimento Essencial, pois devido a uma particularidade dos programas utilizados no Conjunto Set2, ocorreu uma situação de empate entre os operadores em um dos passos e decidiu-se subdividir o Procedimento Essencial em dois subprocessos paralelos.

6.2 Limitações

Algumas limitações do trabalho são:

- Os limiares (do inglês, *thresholds*) utilizados nos experimentos foram baseados exclusivamente nos resultados apresentados nos experimentos de Barbosa (1998). Uma análise mais detalhada poderia revelar *thresholds* mais precisos que pudessem alcançar resultados ainda melhores.
- As aplicações utilizadas nos experimentos são aplicações de pequeno porte (44 a 1004 LOC¹), em sua maioria exemplos de uso acadêmico. Esse fato restringe a generalização dos resultados, pois as aplicações consideradas podem não refletir o estado da prática da Programação Orientada a Aspectos no âmbito industrial.
- Não foram definidas hipóteses formais e, conseqüentemente, não foram realizadas análises estatísticas dos resultados obtidos. Essas análises não foram realizadas devido ao tamanho do conjunto de aplicações não ser suficientemente grande.
- Os casos de teste utilizados neste trabalho foram casos de teste mais simples, que são baseados em classes de equivalência individuais. A investigação de uma estratégia para criação dos conjuntos de teste com casos de teste mais complexos pode impactar no escore final de mutação, pois casos de teste mais complexos podem matar mais mutantes que os casos de teste mais simples.

6.3 Trabalhos Futuros

Dentre as atividades que podem ser realizadas para dar continuidade ao trabalho e contribuir para a melhoria do mesmo, destacam-se:

¹São considerados apenas as linhas de código, excluindo comentários e linhas em branco.

- Aplicação do Procedimento Essencial em conjuntos com programas mais complexos e comparação com os resultados alcançados neste trabalho.
- Investigação da relação entre o tamanho dos conjuntos de programas utilizados com o tamanho dos conjuntos essenciais de cada um dos programas.
- Possível composição do Procedimento Essencial com outras abordagens de redução de custo, com o propósito de potencializar as reduções obtidas nos experimentos realizados neste estudo. Há vários estudos relatados no mapeamento sistemático que apresentam grande potencial na redução de custo, e que poderiam ser utilizados concomitantemente com o Procedimento Essencial.
- Automatização da abordagem de identificação de operadores essenciais com o apoio da ferramenta Proteum/AJ (FERRARI et al., 2010). Novas funcionalidades podem ser incorporadas à ferramenta para automatizar a preparação dos dados e a aplicação dos seis passos do Procedimento Essencial.

REFERÊNCIAS

- abc Development Team. *abc: The AspectBench Compiler for AspectJ*. 2009. Online. <http://www.sable.mcgill.ca/abc> - last accessed on 23/02/2013.
- ALEXANDER, R. T.; BIEMAN, J. M.; ANDREWS, A. A. *Towards the Systematic Testing of Aspect-Oriented Programs*. Fort Collins/Colorado - USA, 2004.
- AMMANN, P.; DELAMARO, M.; OFFUTT, J. Establishing theoretical minimal sets of mutants. In: *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*. Cleveland/OH - USA: IEEE Computer Society, 2014. p. 21–30.
- ANBALAGAN, P.; XIE, T. Automated generation of pointcut mutants for testing pointcuts in AspectJ programs. In: *Proceedings of the 19th International Symposium on Software Reliability Engineering (ISSRE)*. Seattle/WA - USA: IEEE Computer Society, 2008. p. 239–248. ISSN 1071-9458.
- BARBOSA, E. F. *Uma Contribuição para Determinação de um Conjunto Essencial de Operadores de Mutação no Teste de Programas C*. Dissertação (Mestrado) — ICMC/USP, São Carlos/SP - Brasil, 1998.
- BARBOSA, E. F.; MALDONADO, J. C.; VINCENZI, A. M. R. Toward the determination of sufficient mutant operators for C. *The Journal of Software Testing, Verification and Reliability*, John Wiley & Sons, v. 11, n. 2, p. 113–136, 2001.
- BARTSCH, M.; HARRISON, R. An exploratory study of the effect of aspect-oriented programming on maintainability. *Software Quality Journal*, Springer US, v. 16, n. 1, p. 23–44, 2008. ISSN 0963-9314. Disponível em: <<http://dx.doi.org/10.1007/s11219-007-9022-7>>.
- BEIZER, B. *Black Box Testing: Techniques For Functional Testing Of Software And Systems*. New York/NY - USA: John Wiley & Sons, 1995. 294 p.
- BERNARDI, M. L.; LUCCA, G. A. D. Testing aspect oriented programs: an approach based on the coverage of the interactions among advices and methods. In: *Proceedings of the 6th International Conference on Quality of Information and Communications Technology (QUATIC)*. Lisbon - Portugal: IEEE Computer Society, 2007. p. 65–76. ISBN 0-7695-2948-8.
- BIOLCHINI, J. et al. *Systematic Review in Software Engineering*. Rio de Janeiro/RJ - Brazil, 2005.
- BOOCH, G. *Object-Oriented Analysis and Design with Applications*. 2nd.. ed. Redwood City/CA - USA: Addison Wesley, 1994.

- BORGES, K. N. et al. Poke-Tool versão Clipper - Uma ferramenta para suporte ao teste estrutural de programas baseado em análise de fluxo de dados. In: *Sessão de Ferramentas do 9º Simpósio Brasileiro de Engenharia de Software (SBES)*. Recife/PE - Brasil: Brazilian Computer Society, 1995. p. 483–486.
- CHAIM, M. L. *Poke-Tool - Uma Ferramenta Para suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados*. Dissertação (Mestrado) — FEEC/UNICAMP, Campinas/SP - Brasil, 1991.
- CHAIM, M. L.; MALDONADO, J. C.; JIHO, M. Ferramenta para o teste estrutural de software baseado em análise de fluxo de dados: O caso Poke-Tool. In: *Workshop do Projeto de Validação e Teste de Sistemas de Operação*. Águas de Lindóia/SP - Brasil: [s.n.], 1997. p. 29–39.
- CHEVALLEY, P.; Thévenod-Fosse, P. A mutation analysis tool for Java programs. *International Journal on Software Tools for Technology Transfer (STTT)*, Springer-Verlag GmbH, v. 5, n. 1, p. 90–103, 2003.
- DELAMARE, R.; BAUDRY, B.; Le Traon, Y. AjMutator: A tool for the mutation analysis of aspectj pointcut descriptors. In: *Proceedings of the 4th International Workshop on Mutation Analysis (Mutation)*. Denver/CO - USA: IEEE, 2009. p. 200–204.
- DELAMARO, M. E. *Proteum: Um Ambiente de Teste Baseado na Análise de Mutantes*. Dissertação (Mestrado) — ICMC/USP, São Carlos/SP - Brasil, 1993.
- DELAMARO, M. E. *Mutação de Interface: Um critério de adequação interprocedimental para o teste de integração*. Tese (Doutorado) — IFSC/USP, 1997.
- DELAMARO, M. E. et al. Experimental evaluation of sdl and one-op mutation for c. In: *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation*. Washington, DC, USA: IEEE Computer Society, 2014. (ICST '14), p. 203–212. ISBN 978-1-4799-2255-0. Disponível em: <<http://dx.doi.org/10.1109/ICST.2014.33>>.
- DELAMARO, M. E.; MALDONADO, J. C. Proteum: A tool for the assessment of test adequacy for C programs. In: *Conference on Performability in Computing Systems (PCS)*. New Brunswick/NJ - USA: [s.n.], 1996. p. 79–95.
- DELAMARO, M. E.; MALDONADO, J. C.; MATHUR, A. P. Interface Mutation: An approach for integration testing. *IEEE Transactions on Software Engineering*, IEEE Press, v. 27, n. 3, p. 228–247, 2001.
- DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, v. 11, n. 4, p. 34–43, 1978.
- DEMILLO, R. A.; OFFUTT, A. J. Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering*, IEEE Press, v. 17, n. 9, p. 900–910, September 1991.
- DENG, L.; OFFUTT, J.; LI, N. Empirical evaluation of the statement deletion mutation operator. In: *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. Washington, DC, USA: IEEE Computer Society, 2013. (ICST '13), p. 84–93. ISBN 978-0-7695-4968-2. Disponível em: <<http://dx.doi.org/10.1109/ICST.2013.20>>.

- DIJKSTRA, E. W. *A Discipline of Programming*. Englewood Cliffs/NJ - USA: Prentice-Hall, 1976. ISBN 0-13-215871-X.
- DOMÍNGUEZ-JIMÉNEZ, J. et al. Evolutionary mutation testing. *Information and Software Technology*, v. 53, n. 10, p. 1108 – 1123, 2011. ISSN 0950-5849. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S095058491100084X>>.
- DUNCAN, I. M. M.; ROBSON, D. J. Ordered mutation testing. *SIGSOFT Softw. Eng. Notes*, v. 15, n. 2, p. 29–30, abr. 1990. ISSN 0163-5948. Disponível em: <<http://doi.acm.org/10.1145/382296.382699>>.
- DURELLI, V.; OFFUTT, J.; DELAMARO, M. Toward harnessing high-level language virtual machines for further speeding up weak mutation testing. In: *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. [S.l.: s.n.], 2012. p. 681 –690.
- FERRARI, F. C. *A contribution to the fault-based testing of aspect-oriented software*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo (ICMC/USP), São Carlos/SP - Brasil, 2010.
- FERRARI, F. C.; MALDONADO, J. C. Experimenting with a multi-iteration systematic review in software engineering. In: *Proceedings of the 5th Experimental Software Engineering Latin American Workshop (ESELAW)*. Salvador/BA - Brazil: [s.n.], 2008. ISBN 978-85-87837-15-8.
- FERRARI, F. C.; MALDONADO, J. C.; RASHID, A. Mutation testing for aspect-oriented programs. In: *Proceedings of the 1st International Conference on Software Testing, Verification and Validation (ICST)*. Lillehammer - Norway: IEEE, 2008. p. 52–61. ISBN 978-0-7695-3127-4.
- FERRARI, F. C. et al. Automating the mutation testing of aspect-oriented Java programs. In: *Proceedings of the 5th ICSE International Workshop on Automation of Software Test (AST)*. Cape Town - South Africa: ACM Press, 2010. p. 51–58. ISBN 978-1-60558-970-1.
- FILMAN, R. E.; FRIEDMAN, D. Aspect-oriented programming is quantification and obliviousness. In: FILMAN, R. E. et al. (Ed.). *Aspect-Oriented Software Development*. Boston: Addison-Wesley, 2004. cap. 2, p. 21–35.
- FONSECA, R. P. *Suporte ao Teste Estrutural de Programas Fortran no Ambiente Poke-Tool*. Dissertação (Mestrado) — DCA/FEEC/UNICAMP, Campinas/SP - Brasil, 1993.
- FRANKL, P. G.; WEYUKER, E. J. Testing software to detect and reduce risk. *Journal of Systems and Software*, Elsevier Science Inc., v. 53, n. 3, p. 275–286, 2000. ISSN 0164-1212.
- GLIGORIC, M. et al. Efficient mutation testing of multithreaded code. *Software Testing, Verification and Reliability*, v. 23, n. 5, p. 375–403, 2012. ISSN 1099-1689. Disponível em: <<http://dx.doi.org/10.1002/stvr.1469>>.
- GLIGORIC, M. et al. Selective mutation testing for concurrent code. In: *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2013. (ISSTA 2013), p. 224–234. ISBN 978-1-4503-2159-4. Disponível em: <<http://doi.acm.org/10.1145/2483760.2483773>>.

- HANNEMANN, J.; KICZALES, G. Design pattern implementation in java and aspectj. In: *Proceedings of the 17th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. New York, NY, USA: ACM, 2002. (OOPSLA '02), p. 161–173. ISBN 1-58113-471-1. Disponível em: <<http://doi.acm.org/10.1145/582419.582436>>.
- HARROLD, M. J. Testing: A roadmap. In: *Proceedings of the Conference on the Future of Software Engineering - held in conjunction with ICSE*. Limerick - Ireland: ACM Press, 2000. p. 61–72.
- HIERONS, R.; HARMAN, M.; DANICIC, S. Using program slicing to assist in the detection of equivalent mutants. *Software Testing, Verification and Reliability*, v. 9, n. 4, p. 233–262, 1999. ISSN 1099-1689.
- JIA, Y.; HARMAN, M. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, IEEE Computer Society, 2010. ISSN 0098-5589. (in press).
- KAMINSKI, G. et al. A logic mutation approach to selective mutation for programs and queries. *Information and Software Technology*, v. 53, n. 10, p. 1137 – 1152, 2011. ISSN 0950-5849.
- KICZALES, G. et al. An overview of AspectJ. In: *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP)*. Budapest - Hungary: Springer-Verlag, 2001. p. 327–353 (LNCS v.2072). ISBN 3-540-42206-4.
- KICZALES, G. et al. Aspect-oriented programming. In: *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP)*. Jyvaskyla - Finland: Springer-Verlag, 1997. p. 220–242 (LNCS v.1241).
- KIM, S.; MA, Y.; KWON, Y. Combining weak and strong mutation for a noninterpretive java mutation system. *Software Testing, Verification and Reliability*, 2012. ISSN 1099-1689. Disponível em: <<http://onlinelibrary.wiley.com/doi/10.1002/stvr.1480/abstract>>.
- KINTIS, M.; PAPADAKIS, M.; MALEVRIS, N. Evaluating mutation testing alternatives: A collateral experiment. In: *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*. [S.l.: s.n.], 2010. p. 300 –309. ISSN 1530-1362.
- LACERDA, J. T. S.; FERRARI, F. C. Towards the establishment of a sufficient set of mutation operators for AspectJ programs. In: *Proceedings of the 8th Brazilian Workshop on Systematic and Automated Software Testing (SAST) (to appear)*. Maceio/AL - Brazil: Brazilian Computer Society, 2014. p. 21–30.
- LADDAD, R. Aspect-oriented programming will improve quality. *IEEE Software*, v. 20, n. 6, p. 90–91, 2003.
- Leitão Jr., P. S. *Suporte ao Teste Estrutural de Programas Cobol no Ambiente Poke-Tool*. Dissertação (Mestrado) — DCA/FEEC/UNICAMP, Campina/SP - Brasil, 1992.
- LEME, F. G. et al. ProteumAJv2: A mutation-based testing tool for Java and AspectJ programs. In: *Proceedings of the 6th Latin American Workshop on Aspect-Oriented Software Development (LA-WASP) – Poster Session*. São Paulo/SP - Brazil: Brazilian Computer Society, 2012. p. 47–48. ISSN 2178-6097.

- LEMOS, O. A. L.; FRANCHIN, I. G.; MASIERO, P. C. Integration testing of object-oriented and aspect-oriented programs: A structural pairwise approach for Java. *Science of Computer Programming*, Elsevier North-Holland, Inc., Amsterdam - The Netherlands, v. 74, n. 10, p. 861–878, 2009. ISSN 0167-6423.
- 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) - held in conjunction with AOSD*. Chicago/IL - USA: [s.n.], 2005.
- LEMOS, O. A. L. et al. Control and data flow structural testing criteria for aspect-oriented programs. *The Journal of Systems and Software*, Elsevier Science Inc., v. 80, n. 6, p. 862–882, 2007. ISSN 0164-1212.
- LINKMAN, S.; VINCENZI, A. M. R.; MALDONADO, J. C. An evaluation of systematic functional testing using mutation testing. In: *Proceedings of the 7th International Conference on Empirical Assessment in Software Engineering (EASE)*. Keele - UK: [s.n.], 2003. p. 1–15.
- LIU, C.-H.; CHANG, C.-W. A state-based testing approach for aspect-oriented programming. *Journal of Information Science and Engineering*, Institute of Information Science, Academia Sinica, Taiwan, v. 24, n. 1, p. 11–31, 2008.
- Lopez-Herrejon, R. E.; BATORY, D. *Using AspectJ to Implement Product-Lines: A Case Study*. Austin, Texas- USA, 2002.
- MA, Y.; OFFUTT, J.; KWON, Y. R. MuJava: an automated class mutation system. *Software Testing, Verification and Reliability*, v. 15, n. 2, p. 97–133, 2005. ISSN 1099-1689. Disponível em: <<http://onlinelibrary.wiley.com/doi/10.1002/stvr.308/abstract>>.
- MALDONADO, J. C. *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*. Tese (Doutorado) — DCA/FEE, Universidade Estadual de Campinas (UNICAMP), Campinas, SP - Brasil, 1991.
- MALDONADO, J. C. et al. *Introdução ao Teste de Software*. 2000. Minicurso apresentado no 14^o Simpósio Brasileiro de Engenharia de Software (SBES 2000). João Pessoa, PB/Brasil.
- MARSHALL, A. et al. Static dataflow-aided weak mutation analysis (sdawm). *Information and Software Technology*, v. 32, n. 1, p. 99 – 104, 1990. ISSN 0950-5849. Special Issue on Software Quality Assurance. Disponível em: <<http://www.sciencedirect.com/science/article/pii/095058499090053T>>.
- MATEO, P.; USAOLA, M. Mutant execution cost reduction: Through music (mutant schema improved with extra code). In: *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. [S.l.: s.n.], 2012. p. 664 –672.
- MATEO, P. R.; USAOLA, M. P.; ALEMÁN, J. L. F. Validating second-order mutation at system level. *IEEE Transactions on Software Engineering*, v. 39, n. 4, p. 570–587, April 2013. ISSN 0098-5589.
- MATHUR, A. P. *Foundations of Software Testing*. [S.l.]: Addison-Wesley Professional, 2007.
- MORELL, L. J. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, IEEE Computer Society, v. 16, n. 8, p. 844–857, 1990. ISSN 0098-5589.

- 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) - held in conjunction with AOSD*. Chicago/IL - USA: [s.n.], 2005.
- MRESA, E. S.; BOTTACI, L. Efficiency of mutation operators and selective mutation strategies: an empirical study. *Software Testing, Verification and Reliability*, John Wiley & Sons, Ltd., v. 9, n. 4, p. 205–232, 1999. ISSN 1099-1689.
- MYERS, G. J. et al. *The Art of Software Testing*. 2nd. ed. Hoboken/NJ - USA: John Wiley & Sons, 2004.
- NAKAGAWA, E. Y. et al. Towards a reference architecture for software testing tools. In: *Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE)*. Boston/MA - USA: [s.n.], 2007. p. 157–162. ISBN 1-891706-20-9.
- NAMIN, A.; ANDREWS, J. Finding sufficient mutation operators via variable reduction. In: *Second Workshop on Mutation Analysis*. [S.l.: s.n.], 2006. p. 5–5.
- NAMIN, A.; ANDREWS, J.; MURDOCH, D. Sufficient mutation operators for measuring test effectiveness. In: *ACM/IEEE 30th International Conference on Software Engineering, 2008. ICSE '08*. Address: publisher, 2008. p. 351–360. ISSN 0270-5257.
- OFFUTT, A. J. et al. An experimental determination of sufficient mutant operators. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 5, n. 2, p. 99–118, apr 1996. ISSN 1049-331X. Disponível em: <<http://doi.acm.org/10.1145/227607.227610>>.
- OFFUTT, A. J.; LEE, S. D. An empirical evaluation of weak mutation. *IEEE Transactions on Software Engineering*, IEEE Press, v. 20, n. 5, p. 337–344, may 1994. ISSN 0098-5589.
- OFFUTT, A. J.; ROTHERMEL, G.; ZAPF, C. An experimental evaluation of selective mutation. In: *Proceedings of the 15th International Conference on Software Engineering (ICSE)*. Baltimore/MD - USA: IEEE Computer Society, 1993. p. 100–107.
- OMAR, E.; GHOSH, S. An exploratory study of higher order mutation testing in aspect-oriented programming. In: *IEEE 23rd International Symposium on Software Reliability Engineering (ISSRE)*. [S.l.: s.n.], 2012. p. 1–10. ISSN 1071-9458.
- PAPADAKIS, M.; MALEVRIS, N. Mutation based test case generation via a path selection strategy. *Information and Software Technology*, v. 54, n. 9, p. 915 – 932, 2012. ISSN 0950-5849. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S095058491200047X>>.
- PAPADAKIS, M.; MALEVRIS, N.; KALLIA, M. Towards automating the generation of mutation tests. In: *Proceedings of the 5th Workshop on Automation of Software Test*. New York, NY, USA: ACM, 2010. (AST '10), p. 111–118. ISBN 978-1-60558-970-1. Disponível em: <<http://doi.acm.org/10.1145/1808266.1808283>>.
- PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. Bari - Italy: The British Computer Society, 2008. p. 1–10. ISBN 0-7695-2945-3.

- POLO, M.; PIATTINI, M.; Garcia-Rodriguez, I. Decreasing the cost of mutation testing with second-order mutants. *Software Testing, Verification and Reliability*, v. 19, n. 2, p. 111–131, 2009. ISSN 1099-1689. Disponível em: <<http://onlinelibrary.wiley.com/doi/10.1002/stvr.392/abstract>>.
- PRESSMAN, R. S. *Software engineering: A Practitioner's Approach*. 7th.. ed. New York/NY - USA: McGraw-Hill, 2010. ISBN 978-0-07-337597-7.
- RAPPS, S.; J.WEYUKER, E. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, IEEE Press, v. 11, n. 4, p. 367–375, 1985.
- 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)*. Tokio - Japan: IEEE Computer Society, 1982. p. 272–278.
- SHAHRIAR, H.; ZULKERNINE, M. Music: Mutation-based sql injection vulnerability checking. In: IEEE. *The Eighth International Conference on Quality Software. QSIC'08*. [S.l.], 2008. p. 77–86.
- SMITH, B. *Muclipse: An open source mutation testing plug-in for Eclipse*. 2012. Online. <http://muclipse.sourceforge.net/> - last accessed on 23/02/2013.
- SMITH, B.; WILLIAMS, L. An empirical evaluation of the mujava mutation operators. In: IEEE. *Testing: Academic and Industrial Conference Practice and Research Techniques- MUTATION, 2007. TAICPART-MUTATION 2007*. [S.l.], 2007. p. 193–202.
- The Eclipse Foundation. *AJDT Eclipse Plugin*. 2010. Online. <http://www.eclipse.org/ajdt/> - last accessed on 27/04/2010.
- TUYA, J.; SUAREZ-CABAL, M. J.; RIVA, C. de la. Mutating database queries. *Information and Software Technology*, v. 49, n. 4, p. 398 – 417, 2007. ISSN 0950-5849. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0950584906000814>>.
- UNTCH, R. H. On reduced neighborhood mutation analysis using a single mutagenic operator. In: *Proceedings of the 47th Annual Southeast Regional Conference*. New York, NY, USA: ACM, 2009. (ACM-SE 47), p. 71:1–71:4. ISBN 978-1-60558-421-8. Disponível em: <<http://doi.acm.org/10.1145/1566445.1566540>>.
- VINCENZI, A. M. R. *Orientação a Objeto: Definição, Implementação e Análise de Recursos de Teste e Validação*. Tese (Doutorado) — ICMC/USP, São Carlos, SP - Brasil, 2004.
- VINCENZI, A. M. R. et al. Jabuti: A coverage analysis tool for java programs. In: *Anais do 17º Simpósio Brasileiro de Engenharia de Software (SBES), Sessão de Ferramentas*. Manaus/AM - Brasil: Sociedade Brasileira de Computação, 2003. p. 79–84.
- WEDYAN, F.; GHOSH, S. On generating mutants for aspectj programs. *Information and Software Technology*, Elsevier, (in press), 2011.
- WEDYAN, F.; GHOSH, S. On generating mutants for aspectj programs. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 54, n. 8, p. 900–914, ago. 2012. ISSN 0950-5849. Disponível em: <<http://dx.doi.org/10.1016/j.infsof.2011.12.001>>.

- WEISS, S. N.; FLEYSHGAKKER, V. N. Improved serial algorithms for mutation analysis. In: *Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis*. New York, NY, USA: ACM, 1993. (ISSTA '93), p. 149–158. ISBN 0-89791-608-5. Disponível em: <<http://doi.acm.org/10.1145/154183.154266>>.
- WEYUKER, E. J. Using failure cost information for testing and reliability assessment. *ACM TOSEM*, ACM Press, v. 5, n. 2, p. 87–98, 1996.
- WONG, W. E. *On Mutation and Data Flow*. Tese (Doutorado) — Department of Computer Science, Purdue University, West Lafayette/IN - USA, 1993.
- XIE, T. et al. Automated test generation for AspectJ programs. In: *Proceedings of the 1st Workshop on Testing Aspect Oriented Programs (WTAOP) - held in conjunction with AOSD*. Chicago/IL - USA: [s.n.], 2005.
- YAMAZAKI, Y. et al. A unit testing framework for aspects without weaving. In: *Proceedings of the 1st Workshop on Testing Aspect Oriented Programs (WTAOP) - held in conjunction with AOSD*. Chicago/IL - USA: [s.n.], 2005.
- YANO, T. *Estudo do Teste de Mutação em Programas Funcionais SML*. Dissertação (Mestrado) — ICMC/USP, São Carlos/SP - Brasil, 2004.
- ZHANG, L. et al. Operator-based and random mutant selection: Better together. In: *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. [S.l.: s.n.], 2013. p. 92–102.
- ZHANG, L. et al. Is operator-based mutant selection superior to random mutant selection? In: *ACM/IEEE 32nd International Conference on Software Engineering*. [S.l.: s.n.], 2010. v. 1, p. 435–444. ISSN 0270-5257.
- 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)*. 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)*. Germany: [s.n.], 2004. p. 1–16.

GLOSSÁRIO

GFC – *Grafo de Fluxo de Controle*

OA – *Orientado(s) a Aspectos*

POA – *Programação Orientada a Aspectos*

POO – *Programação Orientado a Objetos*

V&V – *Verificação e Validação*