UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

ANDRÉ ASSIS LÔBO DE OLIVEIRA

FTMES@r: Um Método de Localização de Defeitos Baseado em Estratégias de Execução de Mutantes

Goiânia
2018

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[ ] Dissertação** **[X] Tese**

2. Identificação da Tese ou Dissertação:

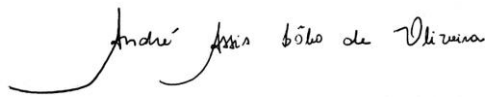
Nome completo do autor: André Assis Lôbo de Oliveira

Título do trabalho:
FTMES@r: Um Método de Localização de Defeitos Baseado em Estratégias de Execução de Mutantes

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [**X**] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.



Assinatura do(a) autor(a)[2]

Ciente e de acordo:

Assinatura do(a) orientador(a)[2]

Data: 19 /11/ 2018

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

Casos de embargo:
- Solicitação de registro de patente
- Submissão de artigo em revista científica
- Publicação como capítulo de livro
- Publicação da dissertação/tese em livro

[2] A assinatura deve ser escaneada.

ANDRÉ ASSIS LÔBO DE OLIVEIRA

FTMES@r: Um Método de Localização de Defeitos Baseado em Estratégias de Execução de Mutantes

Trabalho apresentado ao Programa de Pós–Graduação em Ciência da Computação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação.

Orientador: Prof. Dr. Celso Gonçalves Camilo Júnior

Goiânia
2018

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

de Oliveira, André Assis Lôbo
FTMES@r: Um Método de Localização de Defeitos Baseado em Estratégias de Execução de Mutantes [manuscrito] / André Assis Lôbo de Oliveira. - 2018.
XCVII, 97 f.: il.

Orientador: Prof. Dr. Celso Gonçalves Camilo-Junior.
Tese (Doutorado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2018.
Bibliografia.
Inclui algoritmos, lista de figuras, lista de tabelas.

1. Teste de Software. 2. Depuração de Software. 3. Localização de Defeitos Baseada no Espectro do Programa. 4. Localização de Defeitos Baseada em Mutação. 5. Estratégias de Execução de Mutantes. I. Gonçalves Camilo-Junior, Celso, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

UFMS

Ata de Defesa de Tese de Doutorado

Aos dezoito dias do mês de dezembro de dois mil e dezoito, no horário das dezessete horas e trinta minutos, foi realizada, nas dependências do Instituto de Informática da UFG, a defesa pública da Tese de Doutorado do aluno André Assis Lôbo de Oliveira, matrícula no. 2014 0101, intitulada **"FTMES@r: Um Método de Localização de Defeitos Baseado em Estratégias de Execução de Mutantes"**.

A Banca Examinadora, constituída pelos professores:

Prof. Dr. Celso Gonçalves Camilo Júnior – INF/UFG - orientador

Prof. Dr. Auri Marcelo Rizzo Vincenzi – DC/UFSCar

Prof. Dr. Cássio Leonardo Rodrigues – INF/UFG

Prof. Dr. Eduardo Noronha de Andrade Freitas – IFG

Prof. Dr. Plínio de Sá Leitão Júnior – INF/UFG

emitiu o resultado:

(X) Aprovado

( ) Aprovado com revisão

(A Banca Examinadora deve definir as exigências a serem cumpridas pelo aluno na revisão, ficando o orientador responsável pela verificação do cumprimento das mesmas.)

( ) Reprovado

com o seguinte parecer: _____

_____

_____

_____

Prof. Dr. Celso Gonçalves Camilo Júnior

Prof. Dr. Auri Marcelo Rizzo Vincenzi

Prof. Dr. Cássio Leonardo Rodrigues

Prof. Dr. Eduardo Noronha de Andrade Freitas

Prof. Dr. Plínio de Sá Leitão Júnior

André Assis Lôbo de Oliveira

Licenciado em Informática pela Universidade Federal de Mato Grosso (UFMT) em 2008. Especialista em Gerência de Projetos pelas Faculdades Alves Faria (ALFA) em 2011. Mestre em Ciência da Computação pela Universidade Federal de Goiás em 2013. Professor do IFMT desde 2015.

Dedico,

A **Deus**, porque visualizei em alta definição a Sua poderosa mão abrir as portas para o cumprimento dos Seus bons propósitos em minha vida. Pai amado, sem a Tua intervenção eu não teria conseguido! Eu Te agradeço porque tens me dado muito mais do que mereço! És digno de toda honra, glória e louvor! Obrigado pelo infinito amor, cuidado, direção e longanimidade entregues a mim durante toda essa trajetória!

À minha amada esposa, **Célia Márcia**, pelo amor, carinho, companheirismo, apoio, dedicação e sabedoria que me sustentaram durante toda a minha trajetória acadêmica. Obrigado por ser a minha vida! Você é o melhor dos meus dias!

Aos meus filhos, **Alice** e **Josué**, por ornarem como pedras preciosas a coroa que o próprio Deus colocou sob a minha cabeça a fim de que brilhem!

Aos meus pais, **Francisco Assis de Oliveira** e **Rozelha Lôbo de Oliveira**, pelo amor e torcida durante todo esse tempo. Por terem se esforçado, dia após dia, para que eu estudasse e encontrasse o tesouro do conhecimento. Quero honrá-los enquanto eu existir!

Às minhas irmãs, **Paula Kamilla Ferreira Lôbo** e **Lidianne Lôbo de Oliveira**, pelo amor, carinho e torcida que me acompanharam durante toda a minha trajetória.

Ao meu sogros, **José Donizete Nunes Machado** e **Creusa Senhorinha Gonçalves Nunes**, pelo apoio, cuidado e torcida durante toda a minha trajetória acadêmica.

Agradecimentos

Ao meu Orientador Professor **Celso Gonçalves Camilo Júnior**, pelos ensinamentos que perpassam a minha trajetória acadêmica. Sou muito grato pelo seu caráter, serenidade e a confiança a mim conferida. O seu apoio e auxílio foram fundamentais para tornarem reais sonhos que antes pareciam tão distantes.

Ao Professor **Auri Vincenzi**, pelo apoio dado em vários momentos.

Ao amigo **Eduardo Noronha** que Deus designou para me acompanhar na etapa final da minha pesquisa para que eu cruzasse a linha de chegada!

Aos Professores **Auri, Cássio, Eduardo** e **Plínio**, membros da banca de defesa do doutorado, por terem aceitado o convite e pelas contribuições geradas.

Ao amigo **Kenyo Faria**, pelos seus conselhos e sua importante participação na formação da minha carreira como professor!

À **"família amendoim" (Adaílton e Leandra, Leonardo, Aline e Daniel)**, pela amizade, alegria e torcida durante a minha trajetória na pós-graduação. Vocês são show!

Aos meus amigos, **Sandino Jardim** e **Aline Barros**, pela amizade de longa data, participação em muitos momentos importantes da minha vida, afeto e pelas orações.

Às famílias **ICEC** e **IBAS**, pelo acolhimento, afeto e pelas orações.

Aos meus **colegas da pós** e do grupo de pesquisa *I4Soft*, pela interação durante essa trajetória. Em especial, aos colegas **Diogo Machado** e **Eduardo Souza**.

Aos colegas **Bruno Machado, Acabias Marques, Gleibson Borges, Beatriz Proto e Edward Neto** pela interação e parceria em pesquisa.

Aos cunhados, **Wilhan**, pelo auxílio nas traduções, e **Paulo**, pela torcida.

À minha colega de trabalho **Mônica da Silva** que não mediu esforços para disponibilizar a infraestrutura necessária à realização da minha pesquisa. Sou grato pela companhia e torcida durante esse tempo. Agradeço também aos "colegas de TI" **João Victor** e **Cláudia Regina** pelo apoio e companhia.

Ao **servidores da pós-graduação** do Instituto de Informática (INF) da UFG e da Faculdade de Computação (FACOM) da UFMS, pelo auxílio e excelência no trabalho.

À **CAPES**, à **Faculdade de Computação (FACOM)** e ao **Instituto Federal de Mato Grosso (IFMT)**, pelo apoio financeiro.

"... Seja forte e corajoso! Não se apavore, nem se desanime, pois o Senhor, o seu Deus, estará com você por onde você andar".

Josué,
1:9.

Resumo

de Oliveira, André Assis Lôbo. **FTMES@r: Um Método de Localização de Defeitos Baseado em Estratégias de Execução de Mutantes**. Goiânia, 2018. 97p. Tese de Doutorado . Instituto de Informática, Universidade Federal de Goiás.

A localização de defeitos é considerada uma atividade manual e mais custosa dentre as de depuração. As técnicas de Localização de Defeitos Baseadas no Espectro (SBFL - *Spectrum-based Fault Localization*) são uma das abordagens mais estudadas e avaliadas. A Localização de Defeitos Baseada em Mutação (MBFL - *Mutation Based Fault Localization*) é outra abordagem que traz resultados promissores em eficácia de localização, mas apresenta um alto custo computacional na execução entre casos de teste e programas mutantes. Nesse contexto, esta Tese propõe FTMES@r: um método de localização de defeitos que visa reduzir o custo computacional da abordagem MBFL sem perda da eficácia de localização. Diferindo-se de todas as técnicas de redução, FTMES@r otimiza duas etapas: i) a seleção dos elementos de programa (SFilter@r) e ii) a execução dos mutantes (FTMES). O componente SFilter@r usa a acurácia da abordagem SBFL na formação de um ranking menor pela seleção dos elementos de programa até uma determinada posição @r do ranking de todos os elementos. Assim, SFilter@r emprega o primeiro nível de redução de custo da MBFL porque a geração dos mutantes basea-se somente nos elementos de programa desse ranking reduzido. Na etapa de execução de mutantes, o componente FTMES (*Failed-Test-Oriented Mutant Execution Strategy*) aplica o segundo nível de redução de custo executando mutantes somente com o conjunto dos casos de testes que falham (T_f) e usando a cobertura dos mutantes com o conjunto dos casos de teste que passam (T_p). A experimentação compreende uma comparaçao de 10 técnicas de localização, 221 defeitos reais e 6 métricas de avaliação. Os resultados revelam que FTMES@r apresenta a melhor relação custo-benefício dentre as técnicas estudadas.

Palavras–chave

Teste de Software, Depuração, Localização de Defeitos Baseada no Espectro do Programa, Localização de Defeitos Baseada em Mutação, Estratégias de Execução de Mutantes.

Abstract

de Oliveira, André Assis Lôbo. **FTMES@r: A Fault Localization Method Based Mutation Execution Strategies**. Goiânia, 2018. 97p. PhD. Thesis Instituto de Informática, Universidade Federal de Goiás.

Fault localization has been one of the most manual and costly software debugging activities. The spectrum-based fault localization is the most studied and evaluated fault localization approach. Mutation-based fault localization is a promising approach to the efficacy of localization but with a high computational cost due to the executions between test cases and programs mutants. In this context, this thesis purposes FTMES@r: a fault localization method to reduce the computational MBFL cost while maintaining the efficacy of localization. Differing from all reduction techniques, FTMES@r optimizes two stages: i) the selection of program elements (SFilter@r) and ii) the execution of the mutants (FTMES). The SFilter@r component uses the accuracy of the SBFL approach in forming a smaller ranking by selecting the program elements up to a given position @r of the ranking of all elements. Thus, SFilter@r employs the first level of cost reduction of MBFL because the generation of mutants considers only the program elements of this reduced rank. In the mutants execution stage, the Failed-Test-Oriented Mutant Execution Strategy (FTMES) component applies the second level of cost reduction by running mutants only with the set of failed test cases (T_f) and using the mutants with the set of test cases that pass (T_p). The experimentation comprises a comparison of 10 localization techniques, 221 real defects, and 6 evaluation metrics. The results show that FTMES@r presents the best cost-benefit relationship among the studied techniques.

Keywords

Software Testing, Spectrum-based Fault Localization, Mutation-based Fault Localization, Mutation Execution Strategies.

Sumário

Lista de Figuras **13**

Lista de Tabelas **14**

1 Introdução **15**
- 1.1 Motivação 15
- 1.2 Objetivos 19
- 1.3 Metodologia de Pesquisa 19
- 1.4 Contribuições 20
- 1.5 Publicações 21
- 1.6 Organização da Tese 22

2 Conceitos **23**
- 2.1 Teste de Software 23
  - 2.1.1 Fases 24
  - 2.1.2 Técnicas 25
  - 2.1.3 Análise de Mutantes 26
    - Passos de Aplicação 27
  - 2.1.4 Ferramentas 29
  - 2.1.5 Aplicabilidade 30
- 2.2 Depuração de Software 31
- 2.3 Técnicas de Localização de Defeitos 31
  - 2.3.1 Localização de Defeitos Baseada no Espectro do Programa 33
  - 2.3.2 Localização de Defeitos Baseada em Mutação 34
    - Abordagem Genérica 34
    - Condições para um caso de teste matar um mutante 36
    - Fórmulas de cálculo do valor de suspeição 36

3 Trabalhos Correlatos **39**
- 3.1 O Desafio das Estratégias de Redução de Custo 39
- 3.2 Mutação Seletiva 40
- 3.3 Mutação Aleatória 41
- 3.4 Estratégias de Execução de Mutantes 42

4 O Método Localizador de Defeitos Baseado em Estratégias de Execução de Mutantes **47**
- 4.1 O Filtro dos Elementos de Programa 48
- 4.2 Priorização dos Elementos de Programa 49

4.2.1 A Estratégia de Execução de Mutantes Orientada a Casos de Teste que Falham (FTMES) 50
4.3 Considerações Finais 53

5 Experimentos e Resultados **54**
5.1 Materiais 54
5.1.1 Ferramentas 54
5.1.2 Programas 54
5.2 Métricas 56
5.2.1 Eficiência 56
5.2.2 Eficácia 56
5.3 Técnicas Investigadas 58
5.4 Perguntas de Pesquisa (PP) 60
5.5 Análise da PP1 60
5.6 Análise da PP2 64
5.7 Análise da PP3 70
5.8 Ameaças à Validade 85

6 Conclusão **88**

Referências Bibliográficas **91**

Lista de Figuras

2.1 Exemplo de programa mutante(LAENEN, 2012). 28

3.1 Estratégia de Execução de Mutantes DMES: A) Mutation Execution Optimization Strategy – MEO e; B) Test Case Execution Optimization - TEO. 43

4.1 O Processo de Localização de FTMES@r. 48
4.2 Etapas da Estratégia de Execução de Mutantes: FTMES 51

5.1 Médias das quantidades de execuções de mutantes (*Mutant-Test Pair - MTP)* alcançadas pelas estratégias de execução de mutantes. 61
5.2 Percentual de redução da quantidade de MTP empregado por FTMES@r e MCBFL@r por meio de SFilter@r em relação à MBFL tradicional. 72
5.3 Comparativo do percentual de aproximação do tempo de execução em relação à técnica SBFL. 74
5.4 Percentual de melhoria de acurácia de SFilter@r em relação à SBFL. 77
5.5 Performance geral das técnicas em acurácia (*acc@n*) e esforço desperdiçado (*wef*). 78
5.6 Fronteira de Pareto: Tempo e Acurácia (*acc@20*). 80
5.7 Fronteira de Pareto: Tempo e Wef. 82
5.8 Distância geral de performance de FTMES@30 em relação às técnicas MCBFL, MCBFL@50 e SBFL. A comparação da performance é descrita em termos de tempo, acc@20 e wef. 83
5.9 Fronteira de Pareto com Pesos para Priorizar Custo e Benefício. 84

Lista de Tabelas

2.1 Lista com algumas das ferramentas disponíveis para Análise de Mutantes. Tabela adaptada de Papadakis et al. (2018). 30
2.2 Exemplo de técnicas de Localização de Defeitos Baseadas no Espectro do Programa. 33
2.3 Fórmulas das técnicas de Localização de Defeitos Baseada em Mutação (**MBFL**). 36

5.1 Informações dos Programas. 55
5.2 Técnicas de localização de defeitos. 58
5.3 Escore de mutação médio do conjunto T_f e T_f sobre os mutantes dos programas estudados. 62
5.4 Tempo de execução em minutos das técnicas por programa. 63
5.5 Taxa Média de Redução de Valores MTP das Estratégias de Execução de Mutantes. 64
5.6 Comparação da eficácia de localização das técnicas em termos de *EXAM Score*. 65
5.7 Comparação estatística entre as técnicas MBFL - Test t pareado e teste t de Cohen (d). 66
5.8 Comparação de performance em acurácia (acc@n) e esforço desperdiçado (wef). Células com **fundo em cor preta** destacam a técnica que obteve **melhor resultado** dentre as técnicas comparadas. Já as células com **fundo em cor cinza** destacam a técnica com o **segundo melhor resultado**. 67
5.9 Tempo de execução das técnicas MBFL (em minutos) com diferentes tamanhos de $@r$ na aplicação de SFilter@r. 71
5.10 Aplicação da SFilter@r e a eficácia de localização. 76
5.11 Valor MAP das técnicas SBFL e MCBFL-hybrid-avg (MCBFL). 81

CAPÍTULO 1

Introdução

A localização de defeitos consiste em uma atividade de grande importância no processo de depuração de um software. Apesar das tecnologias existentes que subsidiam o desenvolvimento de software, muitas empresas e organizações de software ainda dependem da experiência e do esforço manual de especialistas para cumprir essa atividade com êxito.

Este Capítulo apresenta alguns fatores que motivaram a realização desta pesquisa, destacando a importância da atividade de depuração e as limitações das técnicas existentes que utilizam a Análise de Mutantes para localizar defeitos, em termos de custo computacional e eficácia de localização. Em seguida, os objetivos desta pesquisa são apresentados para delimitação do escopo da investigação realizada. Em seguida, a metodologia é apresentada descrevendo os passos e os instrumentos adotados. Por fim, as contribuições e os artigos publicados são enumerados.

1.1 Motivação

Uma pesquisa do Instituto Nacional de Padrões e Tecnologia (NIST) afirmou que usuários de software norte-americanos sofrem perdas anuais maiores que $59 bilhões de dólares (TASSEY, 2002). Passados 16 anos dessa publicação, metade da população mundial é impactada por defeitos em software, somando perdas na grandeza de $1,7 trilhões em ativos de 314 companhias, conforme relatório recente de uma empresa do segmento de Teste de Software (TRICENTS, 2018).

Nesse contexto, diversas abordagens de depuração têm sido propostas para auxiliar na localização e no reparo de defeitos em software. Grande parte dessas abordagens fazem uso do conjunto de teste para localizar os elementos defeituosos do programa de maneira automática (PAPADAKIS; TRAON, 2014). Apesar dos avanços dessa área, a literatura ainda referencia a localização de defeitos como uma das atividades mais custosa, tediosa e que consome grande parte do tempo dentre todas as atividades de depuração do software (WONG et al., 2016). Esses fatores, motivam a uma grande comunidade mundial

de pesquisadores que visam o aprimoramento e automatização da atividade de localização de defeitos.

As técnicas de Localização Baseadas no Espectro do Programa são as mais estudadas e avaliadas da atualidade [1] (***S**pectrum-**b**ased **F**ault **L**ocalization Techniques* - **SBFL**) (JONES; HARROLD, 2005; WONG et al., 2016). Elas geram um valor de suspeição $S(s)$ para cada elemento de programa s. Uma lista dos elementos de programa são ordenados em ordem descrescente a partir dos valores de suspeição formando o *ranking* (WONG et al., 2012). O mecanismo da técnica de localização de defeitos visa posicionar os elementos de programa defeituosos no topo desse *ranking* (ABREU; ZOETEWEIJ; GEMUND, 2007). Quanto mais próximos da primeira posição os elementos defeituosos estiverem ranqueados, maior será a eficácia de localização da técnica.

Pesquisadores utilizam-se da combinação de diferentes técnicas visando aumentar a eficácia de localização. Um exemplo disso, são as técnicas de Localização de Defeitos Baseadas em Mutação (***M**utation-**b**ased **F**ault **L**ocalization* - **MBFL**) que combinam técnicas SBFL com informações advindas da Análise de Mutantes (PAPADAKIS; TRAON, 2012; PAPADAKIS; TRAON, 2014).

A Análise de Mutantes tem um alto custo computacional devido à sua grande quantidade de execuções entre casos de teste e programas mutantes (JIA; HARMAN, 2011b). Consequentemente, o custo de uma técnica MBFL também é alto. Assim, uma abordagem MBFL enfrenta dois objetivos conflitantes: i) aprimorar a eficácia de localização (eficácia) e ii) reduzir o custo computacional para ser aplicável (eficiência).

Um estudo recente (PEARSON et al., 2017) demonstra que a Análise de Mutantes demandou de 32 até 168 horas de CPU para a obtenção do *ranking*. Apesar do alto custo computacional, uma técnica MBFL híbrida demonstrou maior eficácia de localização. A abordagem SBFL obteve o *ranking* em poucos minutos, mas com uma menor eficácia de localização quando comparada a MBFL. Esse desafio motiva pesquisadores da área para investigar oportunidades de otimização da MBFL.

Existe uma grande variedade de técnicas de redução de custo computacional da Análise de Mutantes (FERRARI; PIZZOLETO; OFFUTT, 2018). Entretanto, no contexto de localização de defeitos, são poucas as pesquisas publicadas cujo o objetivo consiste em otimizar a MBFL. Até o momento, foram encontradas somente três abordagens com o propósito de reduzir seu custo: i) mutação seletiva (PAPADAKIS; TRAON, 2014); ii) mutação aleatória (LIU et al., 2017) e; iii) estratégia de execução de mutantes (GONG; ZHAO; LI, 2015).

[1] O termo **SBFL** é usado nesta Tese para referir-se a uma técnica ou à classe de técnicas de Localização de Defeitos Baseadas no Espectro. Similarmente, o termo **MBFL** é utilizado para referir-se à abordagem de Localização de Defeitos Baseada em Mutantes.

A mutação seletiva e a mutação aleatória são duas abordagens advindas da Análise de Mutantes, com uso recente no contexto das técnicas MBFL. Gong, Zhao e Li (2015) propõem a Estratégia de Execução de Mutantes Dinâmica (***D**ynamic **M**utation **E**xecution **S**trategy* - **DMES**) para otimizar a MBFL. DMES difere-se das abordagens de Papadakis e Traon (2014) e de Liu et al. (2017) em dois aspectos principais: i) no estágio de sua aplicação e ii) por ser uma técnica específica para a localização de defeitos.

Uma técnica de redução de custo alcança seu objetivo se a redução não comprometer sua eficácia de localização. Além de aplicar técnicas de redução de custo advindas da Análise de Mutantes, o desenvolvimento de uma técnica de redução para MBFL pode explorar particularidades inerentes à localização de defeitos para obtenção de resultados ainda mais satisfatórios. Dentre as técnicas de redução de custo, somente Gong, Zhao e Li (2015) propõem uma abordagem de redução específica para MBFL, uma vez que Papadakis e Traon (2014) e Liu et al. (2017) simplesmente aplicaram técnicas advindas da Análise de Mutantes à MBFL.

Este trabalho propõe o componente FTMES que otimiza a execução de mutantes, atuando, portanto, na mesma etapa que DMES. Conforme será apresentado no Capítulo 3, DMES prioriza a seleção de casos de teste e mutantes dinamicamente pelo cálculo dos valores de suspeição. FTMES, por sua vez, não executa casos de teste que passam, inferindo os resultados das execuções pela cobertura da linha de mutação obtida da execução do programa defeituoso.

Nesse mesmo contexto, Zhang et al. (2016) propuseram o Teste de Mutação Preditivo (*Predictive Mutation Testing* - PMT) que infere o resultado do teste sem realizar a execução. Eles utilizam-se de aprendizado de máquina com diversas métricas coletadas em versões anteriores, como, por exemplo, das informações de execução, injeção e propagação do defeito pelo programa em teste. O objetivo principal consiste em inferir o resultado da execução entre um caso de teste e um mutante (sem realizar a execução do par). Em seus resultados, descrevem uma alta redução de custo da Análise de Mutantes, alcançando uma redução do custo computacional até 151.4 vezes menor do que a abordagem tradicional com efetividade aproximada. Uma das métricas utilizadas por Zhang et al. (2016) é a cobertura da linha de mutação.

A proposição de uma técnica MBFL deve ser validada em programas com defeitos reais. A razão para isso consiste no fato de que resultados obtidos considerando defeitos artificialmente inseridos podem gerar conclusões equivocadas sobre o real desempenho da nova técnica MBFL em relação a técnicas de localização existentes na literatura. Pearson et al. (2017) provaram que técnicas MBFL publicadas como melhores em cenários artificiais demonstraram resultados inferiores aos obtidos pelas técnicas SBFL em cenários reais. Um fator negativo da avaliação da abordagem DMES consiste no fato de ter sido validada somente com defeitos artificiais, o que pode acarretar na extração de

conclusões não representativas para os cenários reais. Por esse motivo, todas as conclusões desta Tese foram pautadas em resultados obtidos por avaliações de defeitos reais, uma característica imprescindível para avaliação de qualquer nova abordagem MBFL.

Além do problema de custo da MBFL, a usabilidade do *ranking* define um outro problema compartilhado tanto pela MBFL quanto pela SBFL. Na prática, programadores iniciam o processo de identificação de defeitos guiados pela ordem do ranking, mas realizam "saltos de posições" realizando "idas e vindas" fazendo com que o *ranking* não seja utilizado de maneira sequencial, conforme sugerido pelas técnicas que o fornecem como saída. Esse fenômeno é conhecido como "efeito zigzag" (PARNIN; ORSO, 2011), acontece devido ao tamanho do *ranking* e também pela dependência entre os elementos de programa.

O "efeito zigzag" representa uma tentativa do programador compreender o relacionamento entre as regiões de código investigadas na identificação do defeito, sendo, portanto, uma parte do processo de depuração. Talvez, na prática, essa ação seja natural e possa ajudar na identificação de defeitos. Por outro lado, existe a possibilidade das posições defeituosas serem "saltadas". Se isso acontecer, o programador poderá ter um esforço extra para identificar a posição defeituosa. Além disso, ele terá desperdiçado o custo computacional de obtenção do *ranking*.

O tamanho do *ranking* é um fator que não contribui para sua usabilidade, seja em uma técnica SBFL ou MBFL. Pearson et al. (2017) demonstrou que as técnicas SBFL são menos eficazes do que a técnica híbrida MCBFL-hybrid-avg, uma técnica que combina informações de cobertura e de mutação para localização de defeitos. Contudo, o mesmo estudo reporta que a classe das técnicas SBFL alcança bons resultados de eficácia em uma quantidade significativa de versões com defeitos reais a um custo muito menor do que a técnica MBFL.

Nesse contexto, outras oportunidades de otimização ainda não exploradas surgem. Por exemplo, o uso de apenas uma fatia de *ranking* de uma técnica SBFL que garanta a seleção do elemento defeituoso para ser aprimorada posteriormente por uma técnica MBFL.

Diante disso, esta Tese propõe o método FTMES@r para fornecer um *ranking* menor e mais acurado, obtido a partir da aplicação da técnica SBFL e aprimorado pela abordagem MBFL, conforme detalhado no Capítulo 4. A abordagem incorpora o componente FTMES, uma estratégia de execução de mutantes que aplica a MBFL eficientemente nos elementos de programa selecionadas.

Não foi encontrada na literatura nenhuma pesquisa que reduz o tamanho do *ranking* (seleção de elementos defeituosos) com base na acurácia da abordagem SBFL para reduzir o custo posterior da Análise de Mutantes. Assim, a proposta deste trabalho reduz o custo da MBFL em duas etapas: i) na seleção de elementos defeituosos (SFilter)

e ii) na execução de mutantes (FTMES). Todos os trabalhos encontrados na literatura reduzem o custo da MBFL ou na etapa de geração ou na etapa de execução de mutantes, de maneira isolada. A presente Tese promove a redução de custo da MBFL na seleção de elementos de programa e na execução de mutantes.

Assim, este estudo foi desenvolvido diante do desafio de alcançar uma localização de defeitos efetiva em fornecer um ranking menor, aprimorado eficientemente pela MBFL e, ao mesmo tempo, útil ao programador.

1.2 Objetivos

Diante das motivações descritas, o principal objetivo desta pesquisa consiste em propor um método localizador de defeitos que combina informações de mutação e do espectro de cobertura do programa (híbrido) que integre diferentes estratégias de redução de custo para obter uma boa relação de custo-benefício no *ranking* obtido. Para alcançá-lo, são pontuados os seguintes objetivos específicos:

1. Conduzir estudos experimentais para avaliação das técnicas estudadas em programas reais.
2. Verificar se FTMES é a estratégia de execução de mutantes mais eficiente entre as abordagens existentes na literatura.
3. Analisar se FTMES prejudica a eficácia de localização das abordagens MBFL.
4. Avaliar se o método FTMES@r, composto pelos componentes SFilter@r e FTMES, melhora a localização de defeitos alcançando menor custo computacional para o emprego da MBFL, uma maior eficácia de localização na priorização dos elementos defeituosos da SBFL e uma melhor relação de custo-benefício entre as técnicas estudadas.

1.3 Metodologia de Pesquisa

A MBFL pode ser uma técnica de localização de defeitos subutilizada por dois motivos: i) ausência de pesquisas que a torne mais eficiente e ii) resultados reais que forneçam o custo-benefício de sua aplicação.

Nesta Tese, é adotado um método de pesquisa quantitativo na investigação do problema de localização de defeitos para uma análise sistemática e experimentais da abordagem proposta. A hipótese de que a localização de defeitos pode ser efetiva com a interação entre uma abordagem SBFL e uma estratégia de execução de mutantes foi testada:

1. Identificando as características mais importantes das abordagens SBFL e MBFL;

2. Comparando as técnicas de redução de custo da MBFL e com as possibilidades de otimização percebidas no nível de seleção de elementos defeituosos e na etapa de execução de mutantes e casos de teste;
3. Investigando a eficácia das diferentes técnicas obtidas por meio das interações com as estratégias de execução de mutantes;
4. Analisando o custo-benefício das técnicas com diferentes métricas de qualidade.

Diante disso, são formuladas as seguintes Perguntas de Pesquisa (PPs):

(PP1) - FTMES supera a eficiência das estratégias de execução de mutantes existentes?

(PP2) - FTMES mantém a eficácia de localização das técnicas MBFL apresentando maior eficiência?

(PP3) - FTMES@r melhora a performance da localização de defeitos produzindo:

- **(PP3.1)** - menor custo computacional em relação às técnicas MBFL?
- **(PP3.2)** - maior eficácia?
- **(PP3.3)** - melhor relação custo-benefício (efetividade) entre as técnicas de localização de defeitos?

As perguntas de pesquisa foram respondidas por estudos empíricos baseados na análise quantitativa dos resultados.

O *benchmark* Defects4J (JUST; JALALI; ERNST, 2014) foi escolhido para comparar a abordagem proposta com as técnicas estudadas. Tal escolha foi subsidiada pelo fato do Defects4J ser considerado atualmente a maior e mais bem organizada base de dados com defeitos reais em Java (CAMPOS; MAIA, 2017), utilizado também em pesquisas de Teste e Reparo Automático de Software (JUST; SCHWEIGGERT, 2015; MOTWANI et al., 2018).

1.4 Contribuições

Os resultados mostram que FTMES@r é um método efetivo na localização de defeitos em software. As principais contribuições resultantes desta Tese são:

1. Uma nova estratégia de execução de mutantes orientada à execução de casos de teste que falham para tornar a aplicação da MBFL mais eficiente.
2. O aprimoramento de *ranking* dos elementos de programa baseado em novo método de interação entre as abordagens SBFL e MBFL.
3. Uma proposta de localização de defeitos com redução de custo da MBFL nos níveis de seleção dos elementos de programa e na execução de programas mutantes.
4. Uma avaliação de eficiência e eficácia de diferentes classes de técnicas de localização e estratégias de redução de custo considerando defeitos reais.

1.5 Publicações

(1) O artigo intitulado "*FTMES@r: A Mutation Execution Strategy Method to Improve the Top@r Positions of Spectrum-based Fault Localization Techniques*" submetido ao Software Quality Journal Special Issue on "Testing and Repair for Software Engineering Technologies and Applications", (SQJ 2018) (OLIVEIRA et al., 2018b).

(2) O artigo intitulado "*FTMES: A Failed-Test-Oriented Mutant Execution Strategy for Mutation-based Fault Localization*" publicado em The 29th IEEE International Symposium on Software Reliability Engineering (ISSRE 2018), October 15-18, 2018 (OLIVEIRA et al., 2018a).

(3) O artigo intitulado "*FTScMES: A New Mutation Execution Strategy Based on Failed Tests' Mutation Score for Fault Localization*" publicado como artigo de pesquisa do *International Symposium on Computer and Information Sciences* (ISCIS 2018) e como parte de capítulo de livro pela *Springer International Publishing - Computer and Information Sciences* (OLIVEIRA et al., 2018c).

(4) O artigo intitulado "*SBSTFrame: uma proposta de framework para o teste de software baseado em busca*" publicado no Workshop de Engenharia de Software baseada em Busca (WESB), co-alocado ao Congresso Brasileiro de Software Teoria e Prática (CBSoft) 2014 (MACHADO et al., 2014).

(5) O artigo intitulado "*Um Algoritmo Genético no Modelo de Ilhas para Seleção de Casos de Teste na Análise de Mutantes*" publicado no Workshop de Engenharia de Software baseada em Busca (WESB), co-alocado ao Congresso Brasileiro de Software Teoria e Prática (CBSoft) 2014 (BORGES et al., 2014).

(6) O artigo intitulado "*Uso de Algoritmo Genético Distribuído na Seleção de Casos de Teste para o Teste de Mutação*" publicado no Workshop de Engenharia de Software baseada em Busca (WESB), co-alocado ao Congresso Brasileiro de Software Teoria e Prática (CBSoft) 2014 (LUIZ et al., 2014).

(7) O artigo intitulado "*O Algoritmo Genético Coevolucionário para Redução de Subconjuntos de Casos de Teste da Análise de Mutantes*" publicado no V Encontro Anual de Tecnologia da Informação (EATI) 2014 (OLIVEIRA et al., 2014).

(8) O artigo intitulado "*Um Operador de Mutação para Algoritmos Evolucionários na Seleção de Casos de Teste da Análise de Mutantes*" publicado no V Encontro Anual de Tecnologia da Informação (EATI) 2014 (MARTINS et al., 2014).

(9) O artigo intitulado "*Avaliação de Técnicas para Redução de Base de Dados de Produção*" publicado no V Encontro Anual de Tecnologia da Informação (EATI) 2014 (NETO et al., 2014).

1.6 Organização da Tese

Este primeiro Capítulo expôs a motivação, o objetivos e as principais contribuições desta Tese. Os Capítulos seguintes estão organizados conforme descrito nos próximos parágrafos.

O **Capítulo 2** apresenta os conceitos e as terminologias de Teste de Software, Análise de Mutantes, Localização de Defeitos Baseada no Espectro do Programa (SBFL) e em Mutação (MBFL).

O **Capítulo 3** descreve um resumo dos trabalhos correlatos encontrados na literatura, apresentando uma discussão das lacunas e das oportunidades percebidas nesse campo de estudo.

O **Capítulo 4**, disserta detalhadamente a formulação do método de localização de defeitos proposto nesta Tese.

O **Capítulo 5**, apresenta a configuração dos experimentos, as técnicas dos estudos, o *benchmark* e detalhes da análise das perguntas de pesquisa suscitadas. Além disso, uma discussão sobre as ameaças à validade foi inserida.

Finalmente, no **Capítulo 6**, as conclusões e as perspectivas de trabalhos futuros sumarizam resultados importantes que foram alcançados, bem como evidenciam possibilidades de pesquisas futuras.

CAPÍTULO 2

Conceitos

Este Capítulo apresenta conceitos básicos sobre teste de software, depuração de software e localização de defeitos. Essas áreas de pesquisa que integram o contexto desta Tese. Inicialmente, os fundamentos sobre o teste de software são apresentados e posteriormente os conceitos sobre depuração e localização de defeitos, considerando que a atividade de teste subsidia a atividade de localização (WONG; DEBROY; CHOI, 2010). As técnicas de localização de defeitos baseadas em mutação (Seção 2.3.2) merecem destaque por constituírem o principal objeto de estudo desta pesquisa.

2.1 Teste de Software

O teste de software possui terminologias básicas que facilitam o seu estudo e entendimento. A presente Tese adota o padrão *IEEE 24765-2010* (IEEE, 2010) para os seguintes termos [1]:

1. Produto/Artefato sob Teste ou Sistema sob Teste (*Product Under Test, System Under Test ou SUT*): sistema, subsistema ou componente de software que está sendo testado;
2. Dado de Teste (*Test Data*): Dado de entrada fornecido à interface pública do artefato sob teste durante a execução do caso de teste;
3. Caso de Teste (*Test Case*): Conjunto de dados de teste, condições de execução e resultados esperados desenvolvidos para um objetivo particular, tal como verificar a conformidade de requisito específico;
4. Conjunto de Teste (*Test Suite*): Coleção de um ou mais casos de teste;
5. Critério de Teste (*Test Criteria*): Define os requisitos que o conjunto de teste deve atender para ser considerado adequado ao teste.
6. Engano (*Mistake*): Ação humana que ocasiona um resultado incorreto;

[1] Na mesma perspectiva de (PRADO, 2018), os termos podem conter diferentes interpretações no contexto de trabalhos de autores referenciados na Tese.

7. Defeito (*Fault*): Passo, processo ou definição de dados incorretos (instrução ou comando incorreto);
8. Erro (*Error*): um estado equivocado do sistema; e
9. Falha (*Failure*): Consiste em uma saída incorreta em relação a uma especificação gerada por um evento de um sistema ou componente de sistema.

O teste de software pode ser definido como um processo de execução de um programa cuja a intenção consiste em encontrar erros (MYERS, 1979). Assim, a qualidade de um conjunto de teste está associada a sua capacidade de revelação de falhas.

O ideal é que as falhas sejam minimizadas por meio de um processo adequado à garantia da qualidade de software (Ré, 2009). Nesse contexto, estão inseridas as atividades de Validação, Verificação e Teste de Software (VV&T) que devem ser praticadas durante todo processo de desenvolvimento de software (DELAMARO; JINO; MALDONADO, 2007). Tais atividades são agrupadas em diferentes fases ou níveis de teste.

2.1.1 Fases

O teste de software é uma atividade complexa. As fases ou níveis visam minimizar tal complexidade. Tradicionalmente, as atividades são divididas em quatro fases em sequência (FERRARI, 2010):

1. **Teste de Unidade**: essa fase objetiva testar cada unidade de forma isolada com a intenção de revelar falhas na implementação lógica do código;
2. **Teste de Integração**: essa fase auxilia na identificação de problemas que competem à interface entre as unidades e partes do software;
3. **Teste de Aceitação**: é desempenhada após o teste de integração e auxilia na revelação de falhas grosseiras que não funcionam conforme a especificação;
4. **Teste de Sistema**: nessa fase, os testadores tentam contabilizar todos os elementos envolvidos na execução do software. Tem como principal objetivo testar as funções e o desempenho do sistema em termos gerais.

Além dessas fases, há uma fase que não ocorre durante o processo normal de desenvolvimento do software. É o chamado **Teste de Regressão** em que, após a liberação do sistema, modificações podem acontecer e novos defeitos podem surgir como consequência dessas modificações. O teste de regressão consiste na reexecução de algum subconjunto de funcionalidades que já foi testado para garantir se as modificações não inseriram efeitos colaterais indesejados (PRESSMAN, 2006).

2.1.2 Técnicas

Em geral, é impossível provar a corretude de um programa (MYERS, 1979) por meio do teste, considerando, por exemplo, que seu domínio de entrada pode conter uma quantidade infinita de elementos. Os critérios de teste auxiliam na redução desse domínio, definindo os elementos do programa em teste (ou alguma outra parte do produto) que deverão ser exercitados durante a execução do software, orientando o engenheiro em todo o processo de teste (FERRARI, 2010).

Os critérios de teste podem ser classificados em três técnicas: *funcional*, *estrutural* e *baseado em defeitos*. Tais técnicas se diferenciam no tipo de informação utilizada para a formação de subconjuntos de dados de teste (subdomínios) (DELAMARO; JINO; MALDONADO, 2007). Nenhuma dessas técnicas é completa, mas se complementam para melhorar a garantia da qualidade do software (VINCENZI, 1998).

Teste Funcional

O **Teste Funcional** é conhecido como **Caixa Preta** (MYERS; SANDLER, 2004). O foco desse critério está em verificar como a caixa (o programa) se comporta sob diferentes circunstâncias, sem considerar o conteúdo da caixa (detalhes de implementação). Possui o objetivo de observar se, para uma entrada dada, o software produz saída correta (ARAKI, 2009). Dessa forma, o testador deriva os casos de teste utilizando-se das especificações funcionais do programa (VINCENZI, 1998). Exemplos desse critério são: particionamento de equivalência, análise do valor limite, teste funcional sistemático, grafo de causa e efeito (DELAMARO; JINO; MALDONADO, 2007).

Teste Estrutural

O **Teste Estrutural** é conhecido como **Caixa Branca** (MYERS; SANDLER, 2004) porque a estrutura interna do programa é considerada na escolha dos casos de teste, ao contrário do **Caixa Preta**. A maioria dos critérios dessa técnica, em geral, utiliza uma representação de programa conhecida como *grafo de fluxo de controle* ou *grafo de programa*.

Seguindo a descrição formal de Junior (2005), a estrutura de fluxo de controle de um módulo M (ou unidade M) é representada por um grafo de fluxo de controle, denotado por $G(M) = (N, E, n_{in}, n_{out})$, que é um grafo dirigido, onde: N é o conjunto de nós; $E \subset (NxN)$ é o conjunto de arcos; n_{in} e n_{out} são os nós de entrada e de saída, respectivamente.

Cada nó do grafo de programa está associado a um bloco de comandos do módulo. A execução do primeiro conjunto de comandos traz como consequência a execução de todos os outros comandos do mesmo bloco, seguindo a sequência de

codificação do programa. Com exceção do primeiro e do último comando, cada bloco tem apenas um único predecessor e um único sucessor. Um caminho completo no *grafo de fluxo de controle* consiste em um caminho que vai do nó de entrada a um nó de saída. Como exemplos de critérios estruturais podem-se destacar: critérios baseados em fluxo de controle e critérios baseados em fluxo de dados.

Teste Baseado em Defeitos

O teste baseado em defeitos utiliza informações sobre os tipos específicos de defeitos que se deseja revelar e os enganos mais frequentes cometidos no processo de desenvolvimento de software. Dentre os critérios baseados em defeitos, valem ser destacados: semeadura de erros e análise de mutantes.

O critério de *semeadura de erros* (BUDD, 1981) semeia artificialmente no programa uma quantidade conhecida de defeitos. Após a realização do teste, dentre o total de defeitos encontrados, verificam-se quais são defeitos naturais e quais são artificiais. Usando estimativas de probabilidade, a quantidade de defeitos naturais ainda existentes no programa pode ser calculado.

A *análise de mutantes* utiliza um conjunto de **programas mutantes P'** (modificados a partir de um programa original **P**) para avaliar se um conjunto de teste **T** é adequado para testar o programa **P** (DEMILLO; LIPTON; SAYWARD, 1978). O objetivo consiste em encontrar um conjunto de teste capaz de revelar as diferenças entre os *programas mutantes* **P'** e o programa original **P**. Uma vez que a *análise de mutantes* é o critério utilizado nesta pesquisa, a Seção 2.1.3 descreverá essa técnica com maiores detalhes.

2.1.3 Análise de Mutantes

A Análise de Mutantes ou Teste de Mutação (*Mutation Analysis* ou *Mutation Testing*) surgiu na década de 1970, na Yale University e Georgia Institute of Technology. Um dos primeiros artigos publicados sobre o Teste de Mutação foi o de DeMillo (DEMILLO; LIPTON; SAYWARD, 1978). Nesse artigo, é apresentada a ideia central da técnica que está baseada na *hipótese do programador competente* e no *efeito de acoplamento*.

A *hipótese do programador competente* assume que programadores experientes escrevem programas muitos próximos do correto de modo que, assumindo-se a validade dessa hipótese, pode-se dizer que os defeitos são introduzidos no programa por meio de pequenos desvios sintáticos que, embora não causem erros sintáticos, alteram a semântica do programa. Por sua vez, o *efeito de acoplamento* assume que defeitos complexos estão ligados a defeitos simples e, por isso, a detecção de um defeito simples pode levar a descoberta de defeitos complexos.

Assumindo a validade dessas hipóteses, a Análise de Mutantes utiliza um conjunto de programas mutantes P' para verificar o quanto um conjunto de teste T é adequado para realizar o teste (DEMILLO; LIPTON; SAYWARD, 1978). Portanto, é possível avaliar o desempenho desse conjunto de casos de teste sobre os programas mutantes baseando-se no programa original P. A geração dos programas mutantes é feita por meio dos operadores de mutação, responsáveis por modelar os desvios sintáticos mais comuns em determinado contexto. Quando um operador de mutação é aplicado no programa original P, e este possui a estrutura sintática de que o operador de mutação necessita, um conjunto de mutantes P' é gerado.

Se o programa original e um programa mutante geram saídas diferentes sobre determinado caso de teste, este programa mutante é considerado como um mutante *morto*. O conceito de matar mutantes é a identificação do programa mutante realizada pelo caso de teste na comparação da saída de P com as saídas de P'.

DeMilli e Offutt (1991) definem três condições para que um caso de teste possa matar um mutante:

1. Alcançabilidade: o código mutado deve ser coberto pelo caso de teste.
2. Necessidade: A execução da mutação deve produzir um estado infectado em que a expressão mutada computa valores diferentes comparada a expressão não mutada.
3. Suficiência: a diferença nos estados deve ser suficiente para propagar diferentes saídas entre o programa mutado e o original.

O Escore de Mutação é um número real que varia entre 0.0 e 1.0 (Equação 2-1). Por meio do cálculo do Escore de Mutação é possível avaliar a qualidade do conjunto de teste T ou de um caso de teste particular. Assim, quanto maior o valor do Escore de Mutação, maior é a qualidade do conjunto de teste.

$$MS(P,T) = \frac{DM(P,T)}{M(P) - EM(P)} \tag{2-1}$$

Onde:

- $DM(P,T)$: número de mutantes mortos pelo conjunto de teste em T;
- $M(P)$: número total de mutantes gerados;
- $EM(P)$: número de mutantes gerados equivalentes a P.

Passos de Aplicação

Dado um programa P e um conjunto de teste T cuja a qualidade se deseja avaliar, a Análise de Mutantes pode ser aplicada pelos seguintes passos (DEMILLO; LIPTON; SAYWARD, 1978):

1. Geração do conjunto de programas mutantes P'
2. Execução de P com T;
3. Execução do conjunto P' com T;
4. Análise dos Mutantes Vivos.

Na ***geração do conjunto de mutantes*** P' são utilizados os operadores de mutação. Os operadores de mutação são os elementos que contêm regras sintáticas responsáveis pelas alterações do programa original P, formando o conjunto de programas mutantes P'. A aplicação de um operador de mutação pode gerar mais que um mutante, pois o programa original P pode conter mais de uma estrutura sintática que está no domínio do operador. Dessa forma, o operador, quando aplicado, desempenha ações de mutação sobre cada uma dessas estruturas. A escolha dos operadores de mutação depende da linguagem e do defeito que se deseja modelar. A Figura 2.1 apresenta um exemplo de mutante. A função *def_sum(n)* é o programa original e a função *def_sum_mutante(n)* é o programa mutante. O sinal $<$ em destaque, no programa mutante, representa a mutação realizada pelo operador de mutação que gerou o mutante.

```
def sum(n) {
  s ← 0;
  for (i ← 1; i ≤ n; i++) {
       s ← s + n;
  }
  return s;
}

def sum_mutant(n) {
  s ← 0;
  for (i ← 1; i < n; i++) {
       s ← s + n;
  }
  return s;
}
```

Figura 2.1: *Exemplo de programa mutante(LAENEN, 2012).*

A etapa de ***execução do programa em teste*** P consiste na execução do programa original P utilizando os casos de teste selecionados. Neste passo, verifica-se o comportamento de P. Se o comportamento do programa for conforme o esperado, executa-se o passo seguinte, caso contrário, o processo termina, uma vez que o programa original falhou para algum caso de teste.

Geralmente, da mesma forma que nas outras técnicas, o testador possui a tarefa de avaliar se o programa em teste está correto (DELAMARO; JINO; MALDONADO, 2007).

A ***execução dos programas mutantes*** P' é um passo que pode ser totalmente automatizado (VINCENZI, 1998). Cada um dos mutantes contidos em P' é executado

contra os casos de teste de T. Os resultados das execuções dos mutantes são comparados com o resultado da execução do programa original P (saída esperada). Considerando cada mutante individualmente, se o resultado do mutante for diferente de P, este mutante é dito morto. Caso contrário, o mutante continua vivo. Um mutante pode continuar vivo por um dentre dois motivos:

1. Ser equivalente ao programa original P;
2. O conjunto de teste selecionado não é adequado o suficiente para distinguir o comportamento entre o mutante e o programa original P.

Após a execução dos mutantes, o escore de mutação dos casos de teste pode ser calculado, conforme Equação 2-1.

A ***Análise dos Mutantes Vivos*** requer maior intervenção humana (DELAMARO; JINO; MALDONADO, 2007), pois deve-se decidir quais dos mutantes vivos são equivalentes ao programa original. Pode-se descrever esta verificação, brevemente, da seguinte forma:

1. Se o programa mutante for equivalente, ele deve ser descartado;
2. Caso não seja, os casos de teste selecionados não foram capazes de diferenciá-lo do programa original P. Logo, existe a necessidade de incluir novos casos de teste ao conjunto T para torná-lo mais adequado à realização do teste. Após isso, deve-se voltar ao passo de execução do programa original.

Dependendo do número de mutantes gerados, bem como das características do programa em teste, para que a aplicação deste critério seja eficiente, é de fundamental importância o uso de ferramentas computacionais, principalmente na execução e comparação de resultados. A próxima seção, realiza uma breve descrição de algumas ferramentas que auxiliam na aplicação deste critério.

2.1.4 Ferramentas

Uma ferramenta de teste pode contribuir na interação e comparação de várias técnicas (GUIDETTI, 2005) para uma maior eficácia na revelação da presença de defeitos. Além disso, no contexto do teste de regressão, uma ferramenta de teste pode facilitar o reuso de casos de teste durante a vida do software contribuindo para a avaliação do impacto das mudanças de novas funcionalidades do software (VINCENZI, 1998).

Muitas ferramentas são desenvolvidas para dar suporte à Análise de Mutantes. A Tabela 2.1 apresenta algumas ferramentas que dão apoio à realização da Análise de Mutantes.

Dentre as ferramentas apresentadas na Tabela 2.1, a ferramenta Major (JUST, 2014) foi utilizada na presente Tese para geração dos programas mutantes pelo fato

Tabela 2.1: *Lista com algumas das ferramentas disponíveis para Análise de Mutantes. Tabela adaptada de Papadakis et al. (2018).*

Nome	Ano	Aplicação	Nome	Ano	Aplicação
mutate	-	C	*ILMutator*	2011	C#
Jester	2001	Java	*Smutant*	2011	Smalltalk
Proteum/IM	2001	C	*jMuHLPSL*	2011	HLPSL
mutgen	2003	C	*SMT-C*	2012	C
muJava	2004	Java	*mutant (muRuby)*	2012	Ruby Policies
ByteME	2006	Java	*CCMUTATOR*	2013	C/C++
SQLMutation	2006	SQL	*Comutation*	2013	Java
Jumble	2007	Java	*SchemaAnalyst*	2013	SQL
ESTP	2008	C	*XACMUT*	2013	XACML
Milu	2008	C	*Mutandis*	2013	JavaScript
Javalanche	2009	Java	*MutPy*	2014	Python
Jdama	2009	SQL/JDBC	*MuCheck*	2014	Haskell
AjMutator	2009	AspectJ	*HOMAJ*	2014	AspectJ/Java
Gamera	2009	WS-BPEL	*MutaLog*	2014	Logic Mutation
PASTE	2009	TFSM	*MoMut*	2015	UML models
PIT	2010	Java	*MuVM*	2016	C
MutMut	2010	Java	*Vibes*	2016	Transition Systems, Statechart Models
GenMutants	2010	.Net	*uDroid*	2017	Android apps
Judy	2010	Java	*Mdroid+*	2017	Android apps
webMuJava	2010	HTML/JSP	*LittleDarwin*	2017	Java
Bacterio	2010	Java	*MuCPP*	2017	C++
Major	2011	Java	*MutRex*	2017	Regular Expressions
Parau	2011	Java	BacterioWeb	2017	Android apps

de estar integrada ao benchmark Defects4J (JUST; JALALI; ERNST, 2014), conforme descrito no Capítulo 5.

2.1.5 Aplicabilidade

Apesar do Teste de Mutação possibilitar uma boa avaliação de um conjunto de casos de teste, ainda é um critério de teste que apresenta problemas acerca de sua aplicabilidade. Um deles consiste no alto custo computacional na etapa de execução dos programas mutantes pelos casos de teste devido a grande quantidade de mutantes que são gerados. Além disso, muitas vezes é necessário aumentar a quantidade de casos de teste para se obter melhores resultados, o que eleva ainda mais o custo. Outro problema enfrentado diz respeito ao grande esforço humano exigido na verificação da equivalência entre programas. Como a verificação automática da equivalência entre programas é uma questão indecidível, o olhar humano ainda consiste no principal guia desse processo (JIA; HARMAN, 2011a).

Jia e Harman (2011a) agrupam as técnicas de redução de custo computacional da análise de mutantes em duas grandes áreas: a) redução dos mutantes gerados e b) redução do custo de execução dos mutantes. Na redução dos mutantes gerados, destacam-se as pesquisas no campo da mutação por amostra (ou aleatória - *mutant sampling*),

clusterização (HUSSAIN, 2008), mutação seletiva (MATHUR, 1991), mutação de ordem superior (JIA; HARMAN, 2008) e estabelecimento do conjunto essencial de operadores (NOBRE, 2011). Na redução da execução dos mutantes, destacam-se as pesquisas no campo da mutação fraca (DEMILLO; LIPTON; SAYWARD, 1978), as técnicas de otimização do tempo de execução (FICICI; BUCCI, 2007; MA; OFFUTT; KWON, 2005), e plataformas de suporte avançado ao teste de mutação (KRAUSER et al., 1991).

Nesse mesmo contexto, em um estudo recente, Ferrari, Pizzoleto e Offutt (2018) elencam seis linhas de pesquisa com maior número de estudos para redução do custo da análise de mutantes: otimização na geração, execução e análise dos mutantes; análise de controle de fluxo; algoritmos evolucionários; mutação seletiva; análise de fluxo de dados e mutação de alta ordem.

2.2 Depuração de Software

Depuração de software é o processo de detectar erros ou defeitos de um software ou sistema, e resolvê-los para que funcione corretamente (SRIVASTVA; DHIR, 2017) envolvendo ações de análise e modificação de programas. É uma atividade essencial e compreende todo o ciclo de vida do processo de desenvolvimento de software (SINGH; GAUTAM, 2016).

Existem dois tipos de abordagens de depuração de software: análise estática e dinâmica. A análise estática é realizada sem a execução do programa, diferentemente da análise dinâmica que envolve execução.

A análise dinâmica pode ser dividida em quatro atividades principais (ADRAGNA, 2010):

1. Localização de defeitos;
2. Classificação de defeitos;
3. Entendimento do defeito
4. Reparo do defeito.

Dentre as atividades de depuração de software, a localização de defeitos é considerada uma das mais difíceis e que consome maior parte do tempo e dos recursos utilizados durante a depuração (WONG et al., 2016; LIU et al., 2017; PEARSON et al., 2017).

2.3 Técnicas de Localização de Defeitos

Um relatório do Instituto Nacional de Padrões e Tecnologias (NIST) aponta que usuários norte-americanos sofrem perdas anuais maiores que $59 bilhões devido

à presença de defeitos em software. Passados 16 anos dessa publicação, metade da população mundial é impactada por defeitos em software, somando perdas na grandeza de $1,7 trilhões em ativos de 314 companhias, conforme relatório recente de uma empresa do segmento de Teste de Software (TRICENTS, 2018). Diante disso, pesquisas vêm sendo desenvolvidas a fim de aprimorar a efetividade das atividades de teste, localização e reparo de defeitos. O Teste de Software auxilia na revelação de defeitos para aperfeiçoar a qualidade dos artefatos de software. As técnicas de Localização de Defeitos visam identificar o local do defeito no software quando as atividades de teste revelam defeitos (SOHN; YOO, 2017).

Em geral, as técnicas de Localização de Defeitos calculam um valor de suspeição para os elementos de programa (*statements* - linhas, funções ou métodos). Após a execução dos casos de teste e cálculo dessas probabilidades, as técnicas geram um *ranking* decrescente dos elementos do programa baseados nesses valores. Os elementos com valores mais altos de suspeição ocupam as primeiras posições desse ranking. A ideia é que o *ranking* guie o programador na verificação das reais localizações dos defeitos. Portanto, uma técnica de Localização de Defeitos possui alta acurácia ou eficácia de localização quando posiciona os elementos defeituosos nas primeiras posições, diminuindo o esforço do programador.

Wong et al. (2016) classifica as técnicas como tradicionais e avançadas. As técnicas tradicionais incluem: registro de programa (*program logging*), asserções (*assertions*), pontos de interrupção (*breakpoints*) e perfilamento (*profiling*). Dentre as técnicas avançadas estão aquelas baseadas: em fatias (*slice-based*), no espectro do programa (*spectrum-based*), em estatísticas (*statistic-based*), em aprendizado de máquina (*machine learning-based*), em mineração de dados (*data mining-based*) e em modelos (*model-based*).

As técnicas de Localização de Defeitos Baseadas no Espectro do Programa (*Spectrum-based Fault Localization - SBFL*[2]) são as mais estudadas (PEARSON et al., 2017). A Seção 2.3.1 descreve maiores detalhes sobre essa classe de técnicas. Outra classe mais recente, não incorporada na classificação de Wong et al. (2016), corresponde às técnicas de Localização de Defeitos Baseadas em Mutação (*Mutation-based Fault Localization - MBFL*[3]). A Seção 2.3.2 descreve essa classe com maiores detalhes por ser o objeto de estudo desta pesquisa.

[2]O acrônimo SBFL é adotado para referir-se à classe de técnicas de Localização de Defeitos Baseadas no Espectro do Programa.

[3]O acrônimo MBFL é adotado para referir-se à classe de técnicas de Localização de Defeitos Baseadas em Mutação.

2.3.1 Localização de Defeitos Baseada no Espectro do Programa

As técnicas de Localização de Defeitos Baseadas no Espectro do Programa (ou *Spectrum-based Fault Localization - SBFL*) utilizam-se da frequência de execuções dos elementos de programa (*statements*) para gerar um valor de suspeição *S(s)* (*suspiciousness*) para em elemento *s* associado.

Em geral, o valor de suspeição *S(s)* é alto para um elemento de programa quando existe uma alta frequência de execução pelos casos de teste que falharam e uma baixa frequência de execução pelos casos de teste que passaram. Em outras palavras, as entidades que falharam tendem a ser responsáveis pela falha do programa ao invés daquelas que passaram pelos casos de teste. O valor de suspeição representa a probabilidade da falha.

A Tabela 2.2 apresenta as cinco técnicas *SBFL* mais estudadas em diversas pesquisas na literatura (PEARSON et al., 2017). O elemento *totalpassed* representa o número de casos de teste que passam. O elemento *passed(s)* corresponde ao número de casos de teste que passam e que executam *s*. Similarmente, os elementos *totalfailed* e *failed(s)* são definidos para o conjunto dos casos de teste que falham.

Tabela 2.2: *Exemplo de técnicas de Localização de Defeitos Baseadas no Espectro do Programa.*

Técnica	Fórmula
Tarantula (JONES; HARROLD, 2005):	$S(s) = \frac{failed(s)/totalfailed}{failed(s)/totalfailed + passed(s)/totalpassed)}$
Ochiai (ABREU; ZOETEWEIJ; GEMUND, 2006):	$S(s) = \frac{failed(s)}{\sqrt{totalfailed.(failed(s)+passed(s))}}$
Dstar (WONG et al., 2014):	$S(s) = \frac{failed(s)^*}{\sqrt{passed(s)+(totalfailed-failed(s))}}$
Op2 (NAISH; LEE; RAMAMOHANARAO, 2011):	$S(s) = failed(s) - \frac{passed(s)}{totalpassed+1}$
Barinel (ABREU; ZOETEWEIJ; GEMUND, 2009):	$S(s) = 1 - \frac{passed(s)}{failed(s)}$

Todo o conjunto de teste T precisa ser executado contra o programa em teste P a fim de que os elementos das fórmulas da Tabela 2.2 calculem os valores de suspeição de cada elemento s do programa P. Dependendo do tamanho do software que esteja sendo testado, esse custo de execução dos testes pode ser alto. Todavia, o custo de uma técnica *SBFL* é menor do que o de uma técnica de Localização de Defeitos Baseada em Mutação, descrita na próxima seção.

2.3.2 Localização de Defeitos Baseada em Mutação

As técnicas de Localização de Defeitos Baseadas em Mutação (*MBFL*) aplicam a *Análise de Mutantes* no processo de localização. Um valor de suspeição é atribuído aos mutantes na hipótese de que os casos de teste com alto escore de mutação possuem um poder de localização que aprimora a acurácia de localização do ranking gerado (PEARSON, 2016). Um caso de teste mata um mutante quando detecta a diferença entre o programa original e o mutante, após suas execuções.

Metallaxis (PAPADAKIS; TRAON, 2012) e **MUSE** (MOON et al., 2014) são as primeiras técnicas que utilizam a Análise de Mutantes no processo de localização de defeitos. Nesta Tese, essas duas técnicas são consideradas como *canônicas* ou *tradicionais* porque utilizam somente as informações dos mutantes na atribuição dos valores suspeitos aos elementos de programa. Tais técnicas possibilitam calcular valores de suspeição para os elementos de programa que possuem mutantes correspondentes, ou seja, elementos em que os operadores de mutação são exercitados. Entretanto, uma limitação da *MBFL* tradicional acontece quando o defeito encontra-se nos elementos imutáveis, isto é, que não permitem a atuação de operadores de mutação para gerar mutantes.

Recentemente a *MBFL* tradicional foi proposta de maneira combinada com as técnicas *SBFL*, configurando uma nova classe de técnicas *MBFL*: *as técnicas híbridas*. A ideia dessas técnicas consiste capacitar a *MBFL* para atribuir valores de suspeição aos elementos de programa imutáveis. Para essa classe de técnicas, os valores de suspeição de cada elemento de programa consideram: i) informações de escore de mutação da Análise dos Mutantes (*MBFL* tradicional) e ii) informações de cobertura dos casos de teste sobre o programa defeituoso (*SBFL*). Pearson et al. (2017) apresentam a técnica *MCBFL-hybrid-avg* com melhor eficácia de localização. Além disso, expõem a técnica híbrida *MRSBFL-hybrid-max* como uma alternativa com maior escalabilidade.

Abordagem Genérica

Uma abordagem genérica para as técnicas *MBFL* pode ser abstraída dentre as existentes e implementada por meio de cinco etapas principais: i) classificação dos casos de teste; ii) geração dos mutantes; iii) execução dos mutantes pelos casos de teste; iv) cálculo dos valores de suspeição e; v) criação do *ranking* dos elementos de programa.

1. ***Classificação dos Casos de Teste***: inicialmente, T executa P, sendo P o programa em teste (*PUT*) e T o conjunto de casos de teste de P. Em seguida, as informações de cobertura e dos resultados dos testes são coletadas (se falhou ou se passou). Assim, T é divido em dois subconjuntos: T_p e T_f, sendo que T_p são os casos de teste que passam e T_f aqueles que falham. Além disso, $cov(T_f)$ é o conjunto dos elementos de programa cobertas por T_f.

2. ***Geração dos Mutantes***: as técnicas *MBFL* usam o conjunto $cov(T_f)$ para gerar os mutantes. Em geral, um elemento de programa s tem um conjunto de mutantes denotado por $M(s)$ gerado a partir do elemento de programa s.
3. ***Execução dos Mutantes pelos Casos de teste***: Os casos de teste executam cada mutante e as informações de cobertura e morte são armazenadas. Então, para cada mutante m, T é dividido em outros dois subconjuntos: T_n e T_k, sendo que T_n é o subconjunto que *não mata o mutante m*, e T_k é o subconjunto que *mata m*.
4. ***Cálculo dos Valores de suspeição***: a suspeição de um mutante m pode ser calculada considerando diferentes fórmulas das técnicas de Localização de Defeitos Baseada no Espectro do Programa. Todavia, as informações de cobertura precisam ser substituídas por informações correspondentes aos mutantes. Essa transformação pode ser realizada considerando quatro métricas: $a_{np} = |T_n \cap T_p|$, $a_{kp} = |T_k \cap T_p|$, $a_{nf} = |T_n \cap T_f|$ e $a_{kf} = |T_k \cap T_f|$. a_{kp} é a quantidade de casos de teste dentre os que passam (T_p) que **matam** o mutante m. (a_{np} é o quantidade de casos de teste que passam que **não matam** o mutante m. Similarmente, a_{kf} e a_{kp}, são as quantidades de casos de teste, dentre os que falham (T_f), que **matam** e **não matam** o mutante Essas métricas também satisfazem: $a_{np} + a_{kp} = totalpassed = |T_p|$ e $a_{nf} + a_{kf} = totalfailed = |T_f|$. A Equação 2-2 apresenta a aplicação dessas quatro métricas na fórmula Ochiai. Essa aplicação também pode ser realizada em outras fórmulas das técnicas *SBFL* (por exemplo, as listadas na Tabela 2.2) após a transformação das métricas de cobertura em métricas de mutantes. Assim, o próximo passo consiste em calcular o valor de suspeição $S(m)$ para cada mutante $m \in M(s)$ associado ao elemento de programa s.

$$S(m) = \frac{a_{kf}}{\sqrt{(a_{kf} + a_{nf}).(a_{kf} + a_{kp})}} \tag{2-2}$$

   O método de agregação do valor de suspeição dos mutantes ao elemento de programa s pode variar, por exemplo, pelo valor máximo ou pelo valor médio. Se adotado o valor máximo, a agregação será da seguinte forma: $S(s) = Max(S(m_1), ..., S(m_n))$, sendo que $m_1, \ldots, m_n$ são todos os mutantes em $M(s)$ e o valor de $S(s)$ é a probabilidade do elemento de programa s ser defeituoso.
5. ***Criação do Ranking dos Elementos de Programa***: o *ranking* dos elementos de programa é gerado pela ordenação decrescente dos valores de suspeição $S(s)$ de P. Essa ordem tem grande importância no processo de localização porque guiará a verificação manual do local do defeito e o posterior reparo.

Em termos gerais, existem dois aspectos que diferenciam as técnicas *MBFL*: i) condição para um caso de teste matar um mutante e; ii) a fórmula para o cálculo do valor de suspeição dos elementos de programa.

Tabela 2.3: *Fórmulas das técnicas de Localização de Defeitos Baseada em Mutação (**MBFL**).*

Técnica	Fórmula de suspeição do Mutante S(m(s))	Método de agregação de suspeição S(s)
Metallaxis	$S(m) = \frac{a_{kf}}{\sqrt{(a_{kf}+a_{nf}).(a_{kf}+a_{kp})}}$	$S(s) = Max(S(m))$
MUSE	$S(m) = failed(m) - \frac{f2p}{p2f}$	$S(s) = Med(S(m))$
MCBFL-hybrid-avg	$S(m) = \frac{\frac{{a_{kf}}^2}{\sqrt{(a_{kp}+a_{np})+(a_{kf}+a_{np})-a_{kf}}} + \frac{{a_{covf}}^2}{\sqrt{(a_{covp}+a_{ncovp})+(a_{covf}+a_{ncovf^-})}} + \frac{failed(s)}{\sqrt{failed(s).(failed(s)+passed(s))}}}{2}$	$S(s) = Max(S(m))$
MRSBFL-hybrid-max	$S(m) = \frac{\frac{a_{covf}}{\sqrt{(a_{covf}+a_{ncovf}).(a_{covf}+a_{covp})}} + \frac{failed(s)}{\sqrt{failed(s).(failed(s)+passed(s))}}}{2}$	$S(s) = Max(S(m))$

Condições para um caso de teste matar um mutante

1. ***Metallaxis*** e ***MCBFL-hybrid-avg***: matam um mutante *m* quando existe diferença entre as saídas do mutante e do programa defeituoso. Tal diferença pode ser detectada, por exemplo, pelas mensagens de falha e exceções lançadas. Assim, um caso de teste *t* mata um mutante *m* se satisfaz a condição de suficiência, definida Seção 2.1.2.
2. ***MUSE***: mata um mutante quando não responde conforme o especificado pelo caso de teste. Assim, um mutante *m* pode ser considerado como *morto* para as técnicas *Metallaxis* e *MCBFL-hybrid-avg* e, ao mesmo tempo, pode ser considerado como *vivo* para a técnica MUSE. Assim, um caso de teste *t* mata um mutante *m* se satisfaz a condição de suficiência e além disso, pela informação do caso de teste se falhou (FAIL) ou passou (PASS).
3. ***MRSBFL-hybrid-max***: não há execução de mutantes, somente as informações de cobertura dos mutantes são consideradas. Assim, um caso de teste mata um mutante *m* quando cobre sua linha de mutação no programa original defeituoso pela condição de alcançabilidade, definida na Seção 2.1.2.

Fórmulas de cálculo do valor de suspeição

A Tabela 2.3 apresenta os diferentes cálculos de valor de suspeição das técnicas *MBFL*. A primeira coluna mostra o nome das técnicas *MBFL*. A segunda coluna contém as fórmulas das técnicas *MBFL* para o cálculo do valor de suspeição de cada mutante *m* associado a um elemento de programa *s*. Os operadores de mutação geram um conjunto de mutantes $M(s)$ para cada elemento de programa *s*, em geral, $|M(s)| > 1$. Por esse motivo, as técnicas *MBFL* precisam de um método de agregação para indicar como os mutantes definem o valor de suspeição dos elementos de programa. A terceira coluna apresenta o método de agregação do valor de suspeição de cada técnica. A seguir, o cálculo do valor de suspeição de cada técnica *MBFL* é detalhado:

1. ***Metallaxis***: a fórmula de cálculo de suspeição do mutante ***S(m(s))*** utiliza as *métricas de transformação* do valor de cobertura em informações de escore de mutação explicadas na Seção 2.3.2. Originalmente, Metallaxis aplica tais métricas na fórmula Ochiai (ABREU; ZOETEWEIJ; GEMUND, 2006) e possui o método de agregação pelo mutante com maior valor de suspeição. Informações detalhadas sobre o mecanismo Metallaxis são detalhadas por Papadakis e Traon (2012).
2. ***MUSE***: considera como $failed(m(s))$ o número de casos de testes que tiveram suas saídas modificadas de FAIL para PASS pela execução de m. O elemento $f2p$ é o somatório total de mudanças de FAIL para PASS realizando a execução de todos os casos de teste que falham sobre todos os mutantes. Similarmente, $passed(m)$ e $p2f$ são definidos. Originalmente, o método de agregação da MUSE utiliza a média dos valores de suspeição dos mutantes. Informações detalhadas sobre o mecanismo MUSE é detalhada por Moon et al. (2014).

3. ***MRSBFL-hybrid-max***: calcula o valor de suspeição de um elemento s em quatro etapas. A **primeira etapa** calcula o valor de suspeição $S(m)$ de cada $m \in M(s)$ semelhantemente à técnica *Metallaxis*, porém, substituindo as informações de execução por informações de cobertura do mutante. Nesse caso, T_n é o conjunto de casos de testes que *não cobrem* o mutante m, e T_k é o conjunto de casos de testes que *cobrem* m. A **segunda etapa** calcula o valor de suspeição $S(s)$ para o mesmo elemento de programa s aplicando uma técnica *SBFL*, por exemplo, a fórmula *Ochiai* (ABREU; ZOETEWEIJ; GEMUND, 2006), conforme Tabela 2.2. A **terceira etapa** soma o valor de suspeição do mutante m (**etapa 1 - *MFBL***) com o valor de suspeição calculado pela técnica *SBFL* (**etapa 2 - *SBFL***). Tal soma é o *processo de hibridização* que utiliza as informações de duas técnicas para definir a suspeita de cada mutante m em $M(s)$. A **etapa 4** finaliza o processo aplicando o método de agregação definindo o valor de suspeição do elemento s como o maior valor de suspeição dos mutantes em $M(s)$. Informações detalhadas sobre o mecanismo MRSBFL-hybrid-max podem ser obtidas em Pearson (2016).
4. ***MCBFL-hybrid-avg***: calcula o valor de suspeição em seis etapas. **A primeira etapa** calcula o valor de suspeição $S(m)$ do mutante m da mesma forma que a técnica *Metallaxis* (Seção 2.3.2 - passo 3). A **segunda etapa** é idêntica à primeira etapa da técnica MRSBFL-hybrid-avg descrita anteriormente (técnica *mirror*, conforme Pearson (2016)). A **terceira etapa** calcula o valor de suspeição de cada mutante somando os valores das duas etapas anteriores. A **quarta etapa** calcula o valor de suspeição $S(s)$ para o mesmo elemento de programa s aplicando uma técnica *SBFL*, por exemplo, *Ochiai*. Na **quinta etapa** ocorre o processo de hibridização somando-se os valores obtidos nas etapas 3 e 4. A **sexta etapa** finaliza o processo aplicando

o método de agregação definindo o valor de suspeição do elemento s pela média dos valores de suspeição dos mutantes em $M(s)$. Informações detalhadas sobre o mecanismo MCBFL-hybrid-avg podem ser obtidas em Pearson (2016).

Com exceção da técnica MUSE, todas as técnicas descritas anteriormente são consideradas nesta Tese. MUSE não é avaliada porque pesquisas anteriores reportam que a técnica Metallaxis a supera em eficácia de localização com menor custo computacional (DEBROY; WONG, 2014; PEARSON et al., 2017).

CAPÍTULO 3

Trabalhos Correlatos

Este Capítulo apresenta uma visão geral sobre o estado da arte das Estratégias de Redução de Custo aplicadas às técnicas de Localização de Defeitos Baseadas em Mutação (*MBFL*).

A Seção 3.1 enfatiza o grande desafio das estratégias de redução de custo para as abordagens MBFL. As Seções 3.2 e 3.3 descrevem duas técnicas de redução do custo para MBFL. Tais técnicas reduzem o custo na etapa de geração/seleção de mutantes otimizando apenas a segunda etapa da abordagem genérica da MBFL (Seção 2.3.2). O método proposto nesta Tese aplica a redução de custo em etapas distintas, conforme detalhado no Capítulo 4. Já a Seção 3.4 apresenta a classe das Estratégias de Execução de Mutantes.

3.1 O Desafio das Estratégias de Redução de Custo

A técnica de Localização de Defeitos Baseada em Mutantes (*Mutation-based Fault Localization - MBFL*) foi concebida visando o aumento da performance de localização das técnicas de Localização de Defeitos Baseadas no Espectro do Programa (*Spectrum-based Fault Localization - SBFL*). A ideia é que as informações de mutação advindas da Análise de Mutantes aprimorem a eficácia de localização.

Pesquisas vêm demonstrando que as técnicas MBFL têm melhor eficácia de localização do que as técnicas SBFL (PAPADAKIS; TRAON, 2012; PAPADAKIS; TRAON, 2014; MOON et al., 2014). Mais recentemente, Pearson et al. (2017) comprovam que ambas as técnicas combinadas (técnicas híbridas) possuem maior eficácia de localização do que quando usadas isoladamente. Isso torna-se mais evidente no contexto de defeitos reais que podem acontecer em locais não tocados pelos operadores de mutação, isto é, os elementos de programa imutáveis. O fato é que a MBFL tradicional perde eficácia de localização quando comparada a uma técnica SBFL que pode atribuir valor de suspeição em locais imutáveis. No conjunto de programas do *benchmark* Defects4J, por exemplo, 10% dos defeitos não possuem mutantes correspondentes, o que significa que a MBFL tradicional tem menor eficácia em 10% dos defeitos do benchmark Defects4J, conforme

Pearson et al. (2017) demonstrou. Dessa forma, o uso isolado da MBFL tradicional não justifica-se devido a natureza de certos tipos de defeitos reais.

O custo é o principal problema de aplicabilidade da Análise de Mutantes (FERRARI; PIZZOLETO; OFFUTT, 2018). O uso de uma técnica MBFL agrega um alto custo computacional advindo da Análise de Mutantes, especificamente, da etapa de *execução dos mutantes com a suíte de teste*. Em geral, as estratégias de redução de custo da MBFL têm aplicado técnicas advindas da Análise de Mutantes: **Mutação Seletiva** (PAPADAKIS; TRAON, 2014) e **Mutação Aleatória (*Mutant Sampling*)** (LIU et al., 2017; PAPADAKIS; TRAON, 2015). Outra classe de estratégias de redução de custo da MBFL são as **Estratégias de Execução de Mutantes** (GONG; ZHAO; LI, 2015).

Um dos principais objetivos do método proposto nessa Tese consiste em realizar a MBFL eficientemente. Apesar das diferenças entre a proposta da Tese e as abordagens existentes na literatura, o mesmo desafio é compartilhado: reduzir o custo computacional sem perda de eficácia de localização.

Após publicar a abordagem MBFL, Papadakis e Traon (2012) realizaram estudos que envolvem técnica de redução a fim de melhorar a sua aplicabilidade (PAPADAKIS; TRAON, 2014; PAPADAKIS; TRAON, 2015). Na mesma perspectiva, outras pesquisas mais recentes foram realizadas, tais como Gong, Zhao e Li (2015), Liu et al. (2017) e Liu et al. (2018). Pearson et al. (2017) descrevem um custo computacional entre 32 a 168 horas de processamento para realizar a Análise de Mutantes no contexto das técnicas MBFL. Diante disso, as estratégias para a redução do custo da MBFL têm grande importância.

Apesar da importância das estratégias de redução de custo da MBFL, um problema perceptível em todas essas estratégias consiste no cenário das suas proposições que se baseiam em problemas com defeitos inseridos artificialmente (DO; ELBAUM; ROTHERMEL, 2005). Muitas técnicas MBFL que previamente afirmaram ser mais eficazes do que as técnicas SBFL em cenários artificiais se demonstraram menos eficazes em programas com defeitos reais (PEARSON et al., 2017). Portanto, a validação em defeitos reais se faz necessária tanto para uma técnica MBFL quanto para uma estratégia de redução de custo que aprimora a MBFL.

As próximas seções abordam sobre cada uma das estratégias de redução de custo que têm sido aplicadas no contexto MBFL, destacando as diferenças em relação a abordagem da Tese.

3.2 Mutação Seletiva

A Mutação Seletiva realiza a seleção dos operadores de mutação visando reduzir custo da Análise de Mutantes. Foi originalmente proposta por Mathur (1991) com

diversos estudos posteriores para estabelecer um subconjunto essencial de operadores com a mesma efetividade que todo o conjunto de operadores (VINCENZI, 1998; NAMIN; ANDREWS, 2006; NAMIN; ANDREWS; MURDOCH, 2008; NOBRE, 2011; BANZI et al., 2012).

Papadakis e Traon (2012) aplicaram a Mutação Seletiva para reduzir o custo da MBFL (PAPADAKIS; TRAON, 2014). Eles estabelecem subconjuntos de operadores por meio de um procedimento incremental dos operadores que geram menores quantidades de mutantes. Os mutantes são gerados a partir dos elementos de programa cobertos pelos casos de teste que falham. Nos experimentos, os autores apresentam distintos subconjuntos de operadores com diferentes quantidades de mutantes selecionados. Eles compararam a eficácia de localização com técnicas de localização diversas (SANTELICES et al., 2009; YU et al., 2011), incluindo uma técnica SBFL (ABREU; ZOETEWEIJ; GEMUND, 2006). Como resultados, reportam maior eficácia de localização e uma redução do custo de 37% a 80% da Análise de Mutantes. Os experimentos foram realizados contemplando defeitos inseridos artificialmente.

Considerando a abordagem genérica da MBFL, Debroy e Wong (2014) propõem a redução de custo na etapa de geração dos programas mutantes por meio da seleção de operadores, especificamente, na segunda etapa da abordagem genérica da MBFL (Seção 2.3.2). Diferindo-se de Papadakis e Traon (2014), a Tese propõe a redução de custo da MBFL em duas etapas distintas: i) na seleção dos elementos de programa por meio do componente SFilter@r (veja Seção 4.1 e ii) na execução entre casos de teste e mutantes por meio do componente FTMES (veja Seção 4.2). A seleção dos elementos de programa consiste em uma etapa anterior à geração dos programas mutantes. A execução dos mutantes representa uma etapa posterior à geração dos programas mutantes.

Diante disso, a proposta desta Tese pode ser utilizada de maneira combinada com a abordagem de Papadakis e Traon (2014). Portanto, a Mutação Seletiva não foi considerada como uma técnica de comparação na avaliação dos resultados.

3.3 Mutação Aleatória

Mutação Aleatória constitui uma técnica que seleciona um pequeno subconjunto de mutantes de todo o conjunto (ACREE, 1980). Foi primeiramente proposta por (ACREE, 1980) e (BUDD, 1980). Nessa abordagem, uma porcentagem desses mutantes são selecionados aleatoriamente e os remanescentes são descartados (JIA; HARMAN, 2011b). Ao contrário da Mutação Seletiva, que ignora a diversidade de operadores existentes, a Mutação Aleatória não remove intencionalmente operadores de mutação, preservando tipos de mutantes associados às entidades de código.

Recentemente, Liu et al. (2017) aplicaram a Mutação Aleatória para redução de custo da MBFL. Eles propuseram a técnica SOME (*Statement-Oriented Mutant Reduction Strategy for Mutation Based Fault Localization*). Os mutantes são gerados a partir dos elementos de programa cobertos pelos casos de teste que falham. Nos experimentos, eles realizaram testes com três percentuais de amostras de mutantes: 10%, 20% e 30%, comparando com a técnica de Mutação Seletiva e com a MBFL tradicional. Como resultados, eles apresentam maiores reduções de custo computacional do que a Mutação Seletiva. A mutação seletiva demonstrou-se mais eficaz apesar dos resultados similares, considerando que a redução do custo computacional pode acarretar na perca de eficácia. Além disso, quando Liu et al. (2017) comparam a abordagem com a MBFL tradicional, que utiliza todos os mutantes, descrevem que não existem diferenças significativas na eficácia de localização.

Da mesma forma que a Mutação Seletiva, a estratégia de Liu et al. (2017) é empregada na segunda etapa da abordagem genérica da MBFL (Seção 2.3.2). Assim, a presente pesquisa difere-se da abordagem de Liu et al. (2017) porque o custo da MBFL é reduzido em etapas distintas. Similarmente à Mutação Seletiva, a Mutação Aleatória não é considerada como uma técnica de comparação e pode ser utilizada de maneira combinada à proposta desta Tese.

3.4 Estratégias de Execução de Mutantes

Gong, Zhao e Li (2015) propuseram a Estratégia de Execução Dinâmica de Mutantes (*Dynamic Mutation Execution Strategy - DMES*), configurando outra classe de técnicas para redução de custo da MBFL. Diferente das técnicas de mutação seletiva e aleatória, a redução de custo acontece por meio de uma priorização dinâmica das execuções entre casos de teste e mutantes.

A abordagem DMES parte da seguinte ideia: se os valores limites de suspeição dos mutantes pudessem ser estimados antes das suas respectivas execuções, todos os mutantes com baixo valor de suspeição não precisariam ser executados. Deste modo, bastaria apenas selecionar e executar o mutante com o máximo valor de suspeição, evitando, assim, as "não contribuições" de execuções.

A Figura 3.1[1] descreve os passos da estratégia DMES (GONG; ZHAO; LI, 2015). Em termos gerais (*DMES*), baseia-se em duas etapas: A) execução dos mutantes (*Mutation Execution Optimization Strategy (MEO)* e B) execução dos casos de testes (*Test Case Execution Optimization Strategy (TEO)*.

[1] A Figura 3.1 foi extraída de Gong, Zhao e Li (2015) visando uma compreensão mais detalhada da estratégia DMES.

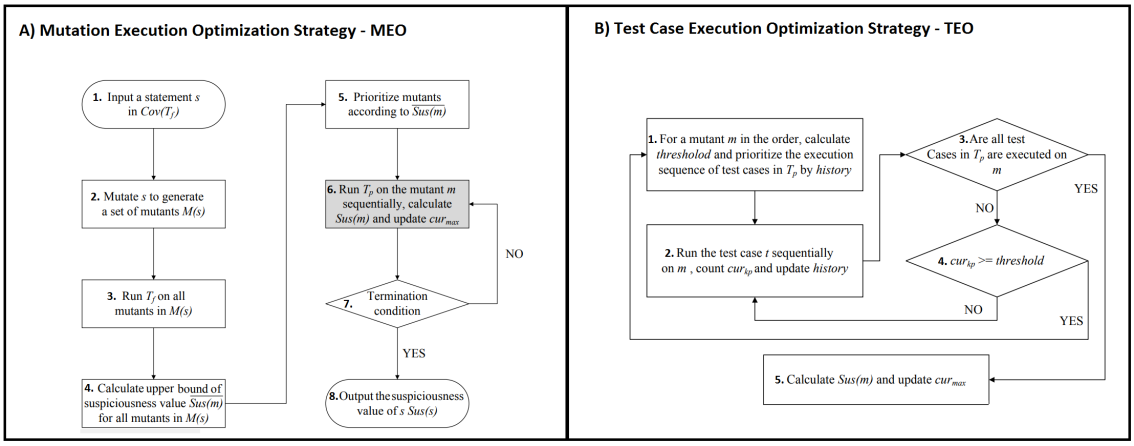


Figura 3.1: *Estratégia de Execução de Mutantes DMES: A) Mutation Execution Optimization Strategy – MEO e; B) Test Case Execution Optimization - TEO.*

A parte A) da Figura 3.1 apresenta a etapa MEO que objetiva evitar execuções de mutantes. Ela pode ser compreendida por meio de 8 passos principais:

1. Inicialmente, um elemento de programa s coberto pelos casos de teste que falham $Cov(T_f)$ é fornecido.
2. O conjunto $M(s)$ de mutantes é gerado pela ferramenta de mutação.
3. Todo o conjunto dos casos de teste que falham T_f executam sobre todo o conjunto de mutantes $M(s)$.
4. As informações de execução de $M(s)$ com T_f são utilizadas no cálculo do valor de suspeição de cada mutante baseado no seu limite superior $\overline{Sus(m)}$. Conforme Gong, Zhao e Li (2015) detalham, $Sus(m)$ alcança seu limite superior $\overline{Sus(m)}$ se e somente se $a_{kp} = 0$ e $a_{np} = |T_p|$. Com essa percepção, o limite superior $\overline{Sus(m)}$ é calculado pela atribuição de $a_{kp} = 0$ e $a_{np} = |T_p|$ na mesma fórmula de cálculo do valor de suspeição do mutante (Tabela 2.3, fórmula Metallaxis).
5. O valor de suspeição define a ordem de execução dos mutantes com os casos de teste de modo que os mutantes mais suspeitos são executados primeiro.
6. O conjunto de mutantes $M(s)$ são executados sequencialmente pelo conjunto de casos de teste que passam T_p. A ideia desse passo consiste em pular execuções dos mutantes por T_p quando um valor limite é alcançado. À medida que cada mutante m é executado, o valor de suspeição $\overline{Sus(m)}$ de cada mutante é atualizado assim como o valor cur_{max} (o maior valor de suspeição dentre os mutantes). Vale observar que a otimização TEO acontece também nessa etapa.
7. Se para um mutante m a condição $\overline{Sus(NEXT(m))} < cur_{max}$ for satisfeita, o limite superior foi alcançado, indicando que o próximo mutante não ultrapassará o valor de suspeição máximo já alcançado (cur_{max}). Essa predição considera que o valor

máximo de suspeição define a suspeição do elemento de programa s. Por isso, os mutantes seguintes não são executados por T_p.

8. Por fim, o algoritmo finaliza quando o limite superior é alcançado ou quando todos os mutantes são executados por T_p.

Desse modo, a etapa *MEO* constitui-se como um processo que evita execuções de mutantes quando predizem que as execuções seguintes não contribuirão para o aumento dos valores de suspeição de s. Em outras palavras, é um processo que "pula" execuções de mutantes.

A abordagem DMES também visa "pular" casos de teste na etapa de execução de $M(s)$ com T_p. Retomando o processo MEO, se $Sus(m) < cur_{max}$, a execução de m pode ser pulada. Todavia, se $Sus(m) \geq cur_{max}$, pode ser que $Sus(m)$ ainda seja menor que cur_{max}. Para cada mutante, se a relação $Sus(m) \leq cur_{max}$ puder ser determinada sem executar todos os testes, o custo computacional pode ser reduzido. Gong, Zhao e Li (2015) observaram que a_{kp} (total de mutantes mortos por T_p) tem uma correlação negativa para o estabelecimento de $Sus(m)$. Então, existe um valor máximo (*threshold*) que faz $Sus(m) \leq cur_{max}$. Com base nessa observação, eles formularam a Equação 3-1.

$$threshold = \begin{cases} totalpassed, & \textbf{se } cur_{max} = 0 \\ \frac{a_{kf}^2}{cur_{max}^2 * totalfailed} - a_{kf}, & \textbf{se } cur_{max} \neq 0. \end{cases} \tag{3-1}$$

Outro ponto importante consiste no armazenamento do histórico do valor do total de mutantes mortos pelo caso de teste. A variável cur_{kp} armazena para cada mutante m a quantidade de casos de teste que o mata. À medida que cur_{kp} é calculada, o histórico da quantidade de mutantes mortos por cada caso de teste também é atualizado alterando a ordem de execução dos casos de teste. A ideia consiste em alcançar o *threshold* mais rapidamente executando primeiro os casos de teste que matam mais mutantes. Quando algum caso de teste satisfaz a condição $cur_{max} \geq threshold$, o valor limite foi alcançado e os casos de teste seguintes são evitados e não precisam ser executados. Assim, *TEO* é um processo que seleciona casos de teste reduzindo o custo das execuções dinamicamente.

A parte B) da Figura 3.1 apresenta a etapa TEO que acontece no passo 6 da otimização MEO (fundo cinza). O processo TEO pode ser descrito através de 5 passos principais:

1. O *threshold* é calculado para cada mutante e os casos de teste são priorizados pelo seu histórico de execução de mutantes.
2. Os casos de teste são executados sequencialmente sobre o mutante m. Após isso, $curr_{kp}$ e o histórico de matança de cada caso de teste é atualizado.
3. Se todos os casos de teste em T_p tiverem executado o mutante m, os valores $Sus(m)$ e cur_{max} são atualizados.

4. Se ainda existirem casos de teste para executar m e o *threshold* não tiver sido alcançado ($cur_{kp} \leq threshold$), a execução de m continua com o próximo caso de teste em T_p. Porém, se o *threshold* tiver sido alcançado ($cur_{kp} \geq threshold$), os próximos casos de teste são "pulados" no processo de execução e inicia-se novamente a execução do mutante $m+1$ com T_p a partir do caso de teste que mata a maior quantidade de mutantes, conforme as informações do histórico.

A abordagem de *DMES* faz parte do escopo da Tese porque reduz o custo da MBFL otimizando a terceira etapa da abordagem genérica (Seção 2.3.2): *a execução de mutantes*. Por esse motivo, o componente DMES é utilizado como abordagem comparativa nos experimentos realizados.

Pearson et al. (2017) ao avaliar diferentes técnicas MBFL realizou uma simples otimização para evitar a execução do conjunto de mutantes $M(s)$ com todo o conjunto de casos de teste que passam T_p: *pular a execução de mutantes que permanecem com o status de vivo após sua execução com o conjunto dos casos te teste que falham (T_f)*. Pearson et al. (2017) não trata tal otimização como uma estratégia de execução de mutantes. Para fins de comparação, essa otimização de Pearson et al. (2017) foi nomeada como SAM (*Skipping Alive Mutants*), conforme Seção 5.3.

Da mesma forma que Gong, Zhao e Li (2015) na proposição de *DMES*, foi percebido que as execuções do conjunto de casos de teste que falham (T_f) são necessárias porque contribuem significativamente na localização de defeitos. Além disso, um dos fatores que mais onera as técnicas MBFL são as execuções do conjunto de mutantes M pelos casos de teste que passam (T_p). Quanto mais alta a cardinalidade dos conjuntos T_p e M, maior é o custo da MBFL. Portanto, a otimização das execuções de T_p sobre M é um aspecto relevante a ser considerado pelas Estratégias de Execução de Mutantes no contexto de localização de defeitos.

Apesar do método proposto nesta Tese também otimizar a execução de mutantes por meio do componente FTMES, mesma etapa que *DMES*, dois pontos centrais que diferenciam ambas estratégias podem ser destacados: i) o mecanismo de redução de custo e ii) o cálculo do valor de suspeição do mutante.

O mecanismo de redução de custo de *DMES* aplica priorização e predição dinâmica baseando-se nas informações de mutação para executar e evitar execuções. Conforme descrito, a priorização acontece no conjunto de mutantes M e no conjunto de casos de teste que passam T_p. Essa priorização visa acelerar o alcance de valores de suspeição mais altos, enquanto a predição estabelece um valor limite que evita execuções futuras.

DMES calcula o valor de suspeição dos mutantes usando as informações das execuções de T_p e M até o alcance de um valor de *threshold*, seja no processo *MEO* (pular execuções de mutantes) ou no processo *TEO* (pular execuções dos casos de teste que

passam). Em relação ao conjunto T_p, o componente FTMES calcula o valor de suspeição dos mutantes utilizando somente as informações de cobertura dos mutantes.

Assim, FTMES otimiza a etapa de Execução de Mutantes diferentemente da abordagem *DMES* de Gong, Zhao e Li (2015), conforme detalhado no Capítulo 4, que apresenta FTMES como parte do método proposto.

CAPÍTULO 4

O Método Localizador de Defeitos Baseado em Estratégias de Execução de Mutantes

O uso de uma técnica de Localização de Defeitos Baseada no Espectro (SBFL) requer: i) o programa defeituoso, ii) o conjunto de casos de teste e iii) a técnica SBFL como mecanismo de cálculo dos valores de suspeição. Todos os casos de teste são executados no programa defeituoso. As informações de cobertura são coletadas e os valores suspeitos dos elementos de programas são calculados conforme a fórmula de cálculo da técnica SBFL. Por fim, o programador usa a ordem do *ranking* gerado como guia para encontrar e reparar o defeito.

A acurácia do *ranking* tem grande importância porque, em geral, os programas reais contêm milhares de linhas. Além disso, existem dependências nos elementos de programa do *ranking*. Esses dois fatores contribuem para o "efeito zigzag" (PARNIN; ORSO, 2011): os programadores percorrem o *ranking* de forma não sequencial. Na prática, são as primeiras posições do *ranking* que guiam o processo de identificação para o reparo do defeito. As outras posições do *ranking* são mais utilizadas no processo de identificação do defeito juntamente com os elementos de programa relacionadas àqueles ranqueados nas primeiras posições.

Na área de localização de defeitos existem muitas técnicas que afirmam ser melhores do que outras (WONG et al., 2016) para uma variedade de cenários que também consideram muitas combinações de técnicas SBFL. As técnicas de Localização de Defeitos Baseadas em Mutantes (MBFL) propõem maior acurácia do que as técnicas SBFL. Entretanto, as técnicas SBFL demonstraram maior eficácia de localização do que as técnicas MBFL tradicionais em alguns tipos de defeitos a um custo muito mais baixo (PEARSON et al., 2017).

Diante disso, o método FTMES@r é proposto para fornecer um *ranking* menor, mais eficaz e obtido com maior eficiência contendo dois passos principais: i) o filtro dos elementos de programa e ii) a priorização de elementos do programa. Inicialmente uma técnica SBFL seleciona elementos de maior suspeição de defeitos formando um *ranking*

de tamanho *r*. Em seguida a abordagem MBFL refina/reordena o *ranking* visando priorizar os elementos defeituosos para alcançar uma maior eficácia de localização.

4.1 O Filtro dos Elementos de Programa

A Figura 4.1 apresenta uma visão geral da abordagem FTMES@r. Primeiramente, a pessoa responsável pela atividade de depuração fornece as seguintes entradas: i) o programa defeituoso P; ii) o conjunto T que testa P; iii) o valor @r que define o tamanho de Ranking@r e; as técnicas de localização: iv) SBFL e v) MBFL.

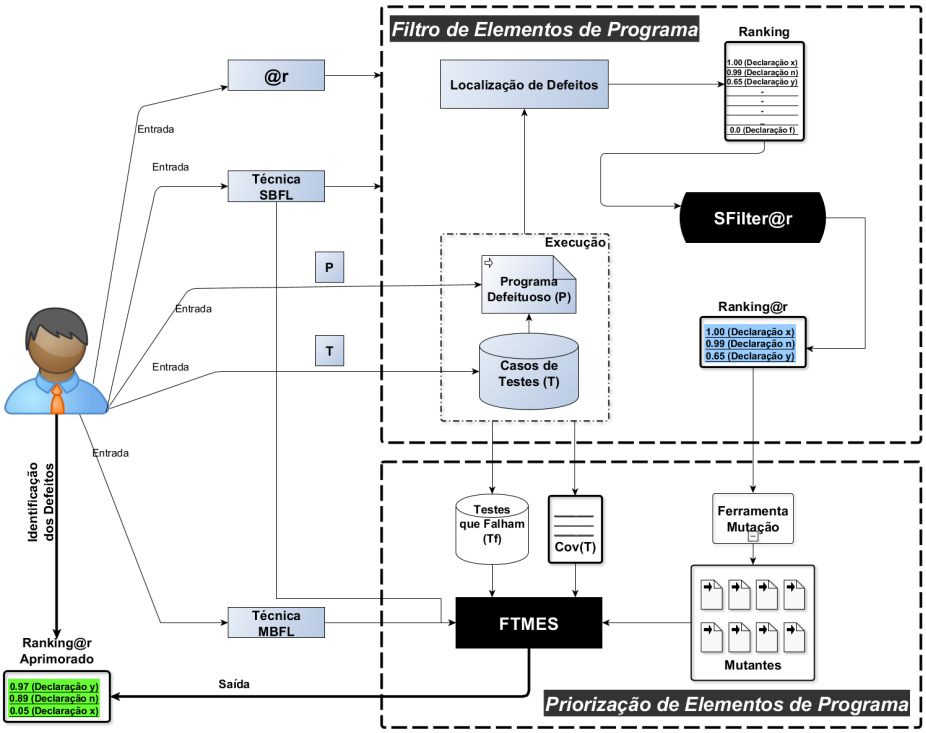


Figura 4.1: *O Processo de Localização de FTMES@r.*

O conjunto de teste *T* executa sobre *P* e as informações de cobertura são coletadas. Na sequência, a técnica SBFL realiza a localização de defeitos usando as informações de cobertura. A saída consiste em um Ranking dos valores de suspeição contendo todos os elementos de programa de *P* em ordem decrescente.

Após isso, o Ranking@r é obtido considerando as primeiras @r posições do *ranking* original. Essa etapa foi nomeada como *SFilter@r* porque realiza o *Filtro dos Elementos de Programa* pela acurácia de localização da técnica SBFL (SFilter), sendo @r o parâmetro fornecido para definir o tamanho de Ranking@r.

O valor @r ajuda o programador a concentrar-se no tamanho do ranking que ele determinou em conformidade com o domínio do problema e sua realidade para

desempenhar a atividade de localização de defeitos. Vale notar que o valor @r filtra os elementos de programa com maiores valores de suspeição, o que torna esse passo dependente da acurácia de localização da SBFL.

Por outro lado, como efeito dessa redução, a técnica MBFL será realizada mais eficientemente. É importante destacar que existe a possibilidade de reduzir a eficácia de localização devido à redução dos elementos de programa que, consequentemente, reduzirá a quantidade de mutantes a serem gerados. A próxima Seção descreve a priorização dos elementos de programa contidos em Ranking@r.

4.2 Priorização dos Elementos de Programa

Conforme Figura 4.1, o passo de priorização de elementos de programa considera: i) o conjunto de mutantes M gerado a partir dos elementos de programa selecionados por *SFilter@r*; ii) o conjunto de casos de teste que falham T_f classificado previamente na execução de T sobre P; iii) as informações de cobertura $cov(T)$ dos casos de teste que passam e que falham; as técnicas iv) SBFL e; v) MBFL.

O componente FTMES é a estratégia de execução de mutantes responsável por executar, calcular os valores de suspeição dos elementos de programa e fornecer um *Ranking@r* aprimorado ao programador. FTMES e *SFilter@r* são dois componentes de FTMES@r que, além de trabalharem juntos, podem ser usados independentemente. *SFilter@r* também possibilita gerar mutantes para outras estratégias de execução. Semelhantemente, FTMES pode executar os casos de teste sobre todos os mutantes gerados a partir de todos os elementos de P que são cobertos pelo conjunto de casos de teste que falham (T_f).

Tais componentes formam a abordagem proposta porque configuram as hipóteses da pesquisa: i) as técnicas SBFL conseguem posicionar os elementos de programa defeituosos até uma posição @r do *ranking* gerado eficientemente; ii) a técnica FTMES tem o potencial de executar mutantes a um custo muito inferior ao empregado pela abordagem tradicional da MBFL mantendo a eficácia de localização e; iii) que a união dessas duas técnicas pode tornar a localização de defeitos mais útil e com maior poder de localização de defeitos.

A Seção 4.2.1 descreve mais detalhes do componente FTMES como uma estratégia de execução de mutantes.

4.2.1 A Estratégia de Execução de Mutantes Orientada a Casos de Teste que Falham (FTMES)

A Estratégia de Execução de Mutantes Orientada a Casos de Teste que Falham (**FTMES**: ***F**ailed-**T**est-Oriented **M**utant **E**xecution **S**trategy*) visa obter eficácia de localização no *ranking* dos elementos de programa de uma forma mais eficiente. FTMES usa somente o conjunto de casos de teste que falham para gerar e executar mutantes. Assim, as execuções dos casos de teste que passam são evitadas pela substituição das informações de matança pelas informações de cobertura dos mutantes.

No processo de localização de defeitos, T executa o programa em teste P, sendo T seu conjunto de teste. Assim, os resultados dos testes (FAIL ou PASS) e as informações de cobertura são coletadas. Em seguida, T é dividido em conjuntos: T_p e T_f, sendo T_p os casos de teste que passam e T_f os casos de teste que falham. Então, $cov(T_f)$ consiste no conjunto dos elementos de programa cobertos por T_f. O conjunto $cov(T_f)$ é usado na geração dos mutantes [1] das técnicas MBFL.

A percepção por traz de FTMES está relacionada à execução do conjunto de teste. A execução de todo o conjunto dos testes que falham T_f sobre os mutantes parece ser um custo necessário porque esses apontam o comportamento não esperado do programa defeituoso. Da mesma forma que as outras técnicas MBFL, os mutantes foram gerados a partir dos elementos de programa cobertos por T_f.

A Figura 4.2 apresenta o processo de FTMES destacando as suas principais etapas. Todas as etapas são realizadas para cada elemento de programa presente em $cov(T_f)$. Ao final, o mecanismo de localização de defeitos gera o *ranking* dos valores de suspeição dos elementos de programa. Duas etapas destacam-se nesse processo de cálculo de suspeição de FTMES: i) cálculo de ak_f e an_f e; ii) cálculo de $aCov_p$ e $anCov_p$.

O **cálculo das métricas** ak_f **e** an_f acontece a partir da execução de T_f com o conjunto $M(s)$ de mutantes do elemento de programa s, conforme Seção 2.3.2.

O **cálculo das métricas** $aCov_p$ **e** $anCov_p$ não envolve execução de mutantes. Essa etapa somente usa informações de cobertura baseada nas informações de cobertura da execução prévia de T contra P. Assim, ***informações de cobertura substituem as informações do status do mutante (vivo ou morto)*** a fim de evitar execuções dos mutantes pelos casos de teste.

Desse modo, $ak_p = aCov_p$, sendo $aCov_p$ o número de casos de teste que passam em T_p que cobrem o mutante $m \in M(s)$. Similarmente, $an_p = anCov_p$, sendo $anCov_p$ o número de casos de teste em T_p que não cobrem o mutante $m \in M(s)$. FTMES retorna

[1] Quando FTMES é o componente de execução de mutantes da abordagem FTMES@r, a geração de mutantes é realizada a partir dos elementos de programas do Ranking@r filtrado por SFilter@r, conforme Seção 4.1.

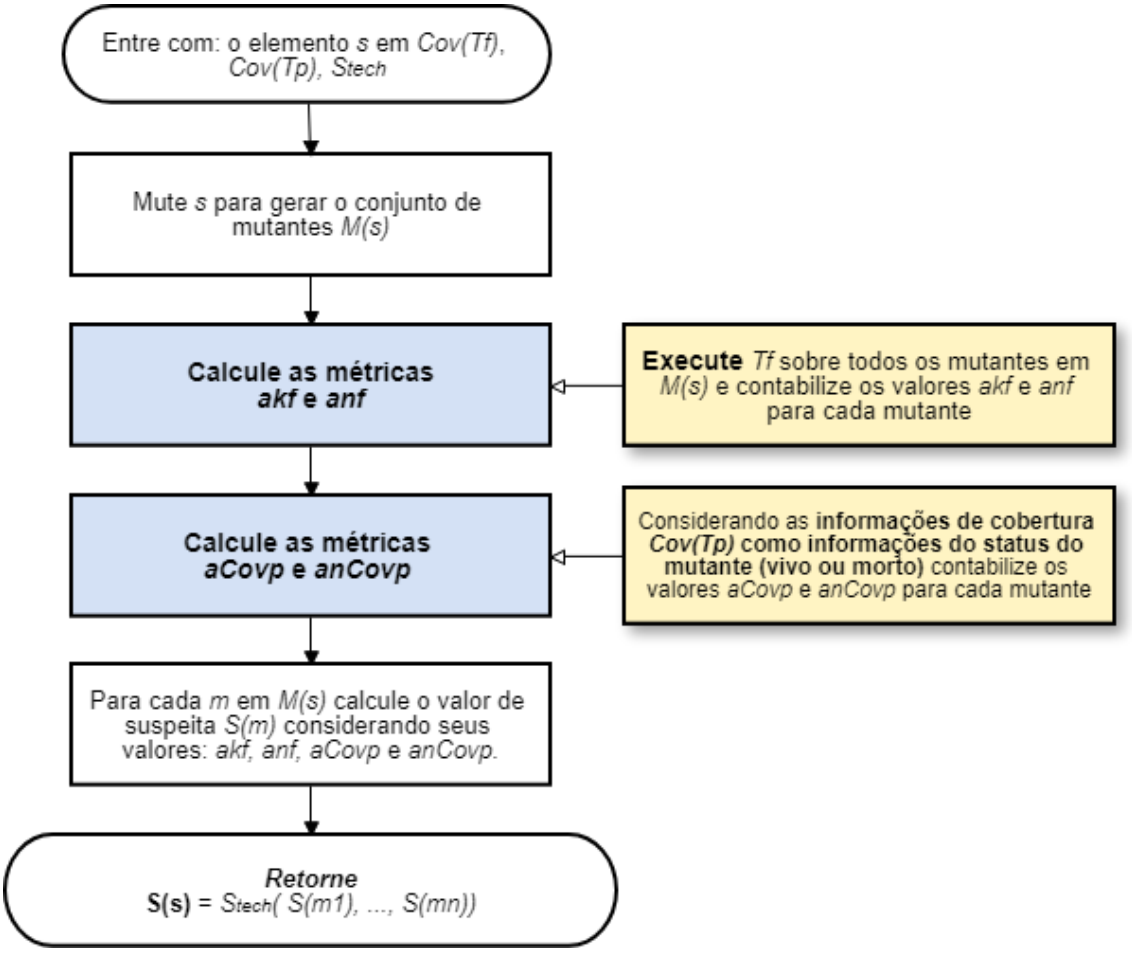


Figura 4.2: *Etapas da Estratégia de Execução de Mutantes: FTMES*

uma lista *SL* de valores de suspeição relacionados aos elementos de programa *s*, de acordo com o último passo expresso na Figura 4.2. Essa lista é usada pelas técnicas MBFL para atribuir o valor de suspeição final $S(s)$ para um elemento de programa *s*.

O Algoritmo 1 descreve o processo de cálculo de valores de suspeição que FTMES realiza para um dado elemento de programa *s*. As entradas do Algoritmo 1 incluem: i) o programa *P*; ii) o conjunto de teste *T*; iii) o elemento de programa *s* coberto pelos casos de teste que falham; iv) as informações de cobertura dos casos de teste *C* de *s*; v) o conjunto *R* de resultados de T; vi) a fórmula de cálculo de suspeição S_{form} e vii) a técnica MBFL S_{Tech} que consiste no mecanismo que atribui o valor de suspeição $S(s)$ considerando a lista de suspeição *SL* de cada mutante em $M(s)$ relacionada ao elemento de programa *s*.

C é um vetor em que cada elemento $C(t)$ tem dois possíveis valores: coberto ou não coberto, que indica se o teste *t* cobre ou não o elemento de programa *s*. *R* é também um vetor com valores que passaram ou falharam. $R(t) = Passou$ representa o caso de teste *t* passando em *P*, caso contrário $R(t) = Falhou$ representa *t* falhando em *P*.

Algorithm 1: Algoritmo da Abordagem FTMES

1 **Entrada**: FTMES(P, T, R, S, C, S_{form}, S_{form})
2 **Saída**: *Susp(s)*

1: $M(s) \leftarrow Muta(s)$
2: $Tf, Tp \leftarrow$ Particione T usando R
3: **for** $m \in M(s)$ **do**
4: **for** $(t_f \in T_f)\&\&(C(t_f) == Cobertura)$ **do**
5: $Execute < m, t_f >$
6: Contabilize ak_f e an_f
7: **end for**
8: **for** $(t_p \in T_p)$ **do**
9: Contabilize $aCov_p$ e $anCov_p$
10: **end for**
11: $SL_m = S_{form}(ak_f, an_f, aCov_p, anCov_p)$
12: **end for**
13: **return** $Susp(s) = S_{Tech}(SL_M(s))$

A seguir, o algoritmo é explicado em detalhes. A linha 1 muta o elemento de programa s para gerar os mutantes $M(s)$ pela aplicação dos operadores de mutação. A linha 2 divide T em dois conjuntos: T_f é o conjunto de casos de teste que falham e T_p é o conjunto de casos de teste que passam. As linhas de 4-7 iteram sobre o conjunto $M(s)$ para capturar os valores ak_f e an_f de cada mutante em $M(s)$ (passo 3 do processo de FTMES, conforme Figura 4.1).

Semelhantemente, as linhas de 8-10 iteram sobre M(s) para capturar os valores $aCov_p$ e $anCov_p$ (passo 4 do processo FTMES, conforme Figura 4.1). A linha 11 calcula os valores de suspeição para cada mutante m conforme a fórmula S_{form} e armazena na lista SL. Finalmente, a suspeita $Susp(s)$ é retornada para ser atribuída ao elemento de programa s, como a técnica S_{Tech} guia.

Nesta Tese, S_{Tech} pode ser a MBFL tradicional ou a técnica híbrida MCBFL-hybrid-avg, conforme Seção 2.3.2. Para a MBFL, FTMES atribui o valor de suspeição de s com o valor máximo em SL, ou seja, $S_{MBFL}(s) = max(SL)$. Para a técnica híbrida MCBFL-hybrid-avg, FTMES atribui da seguinte forma: $S_{MCBFL-hybrid-avg}(s) = media(max(SL), S_{SBFL}(s))$. A técnica MCBFL-hybrid-avg é detalhada por Pearson et al. (2017).

FTMES não adiciona custo computacional à implementação da MBFL, diferentemente da técnica de comparação DMES (GONG; ZHAO; LI, 2015). O efeito da utilização de FTMES é avaliado considerando 8 técnicas de comparação incluindo a abordagem DMES (GONG; ZHAO; LI, 2015).

4.3 Considerações Finais

Este Capítulo apresentou o método FTMES@r que propõe a localização de defeitos por meio de dois componentes: SFilter@r e FTMES. O componente SFilter@r é responsável por filtrar os elementos de programa com base na acurácia da abordagem SBFL. O componente FTMES é a estratégia de execução de mutantes responsável por priorizar os elementos de programa filtrados pela abordagem SBFL eficientemente.

FTMES@r, por meio desses dois componentes, emprega dois níveis de otimização para redução do custo computacional da abordagem MBFL. Até o presente momento, não existe nenhuma proposta de redução de custo da MBFL que emprega dois níveis de redução. As propostas existentes (PAPADAKIS; TRAON, 2014; GONG; ZHAO; LI, 2015; LIU et al., 2017), propõe redução de custo em apenas um dos níveis isoladamente. Além disso, o nível de filtrar elementos de programa com base na eficácia da técnica SBFL para reduzir o custo da MBFL é inovador.

O componente FTMES foi avaliado diante da atual estratégia de execução de mutantes DMES (GONG; ZHAO; LI, 2015) e provado ser superior tanto em eficiência de obtenção do ranking e quanto em eficácia de localização com uma avaliação em defeitos reais (OLIVEIRA et al., 2018a; OLIVEIRA et al., 2018c).

No Capítulo seguinte, são apresentadas as configurações dos experimentos e a análise dos resultados.

CAPÍTULO 5

Experimentos e Resultados

Este Capítulo descreve o desenvolvimento experimental da abordagem proposta. Após a descrição dos materiais, métodos e técnicas investigadas, a capacidade de redução da estratégia de execução de mutantes FTMES foi avaliada a fim de apresentar sua performance diante das abordagens existentes, comprovando que FTMES é a atual melhor técnica de execução de mutantes da literatura (OLIVEIRA et al., 2018a). Em seguida, o componente SFilter@r é avaliado por meio das seguintes interações: i) entre SFilter@r e FTMES, e ii) a interação entre SFilter@r e MCBFL-hybrid-avg (sem FTMES). O objetivo consiste em demonstrar a capacidade de redução e eficácia de localização de FTMES@r que sugere a interação entre SFilter@r e FTMES.

Visando consolidar a representatividade dos resultados, o estudo compreende a avaliação de eficácia e eficiência de 10 técnicas de localização de defeitos comparadas em um conjunto de programas que somam 221 defeitos reais.

5.1 Materiais

5.1.1 Ferramentas

A ferramenta Major (JUST, 2014) foi usada na geração dos mutantes e no fornecimento dos dados da Análise de Mutantes para as técnicas de Localização de Defeitos Baseada em Mutantes (MBFL). GZoltar é muito utilizada na área de localização de defeitos (CAMPOS et al., 2012). Tal ferramenta foi empregada para o fornecimento dos dados de cobertura para as técnicas de localização de defeitos.

5.1.2 Programas

Defects4J é uma base de dados (*benchmark*) e um arcabouço que fornece defeitos reais para habilitar a reprodução de estudos em Teste de Software e Localização de Defeitos (JUST; JALALI; ERNST, 2014). O Defects4J foi utilizado porque é um conjunto de programas escritos em linguagem Java com defeitos reais, além de ser de grande porte,

revisado por pares e considerado atualmente a maior e mais bem organizada base de dados com defeitos reais em Java (CAMPOS; MAIA, 2017). Foram considerados os seguintes programas do Defects4J: JFreeChart (26 defeitos), Apache Commons Lang (65 defeitos), Apache Commons Math (106 defeitos), Mockito (38 defeitos), e Joda-Time(27 defeitos). A Tabela 5.1 apresenta algumas informações do estudo realizado.

A coluna 2 da Tabela 5.1 corresponde ao número de versões defeituosas. Tal coluna apresenta dois números: o total de versões fornecido pelo Defects4J e o total de experimentos considerado. No uso com o *benchmark* para algumas versões de programa não foi possível obter algumas informações: (i) identificação das linhas defeituosas; ii) identificação das linhas candidatas para a correção dos defeitos que existem pela omissão de código e; iii) versões com somente casos de teste que falham. Versões que apresentaram tais características foram excluídas. A coluna KLOC consiste na quantidade de linhas de código dos programas. As colunas Média $|T_f|$ e Média $|T_p|$ é o tamanho médio dos conjuntos de casos de teste que falham e que passam. A coluna Média Mutantes é a quantidade média de mutantes considerados nos experimentos para programa. A coluna Média ED é a quantidade média de elementos defeituosos (ED) na caracterização de um *bug* entre as versões de programa. Por fim, as colunas Média EM T_f e T_p consiste no valor médio do escore de mutação (EM) dos conjuntos de casos de teste T_f e T_p.

Tabela 5.1: *Informações dos Programas.*

Programas	**Versões**	**KLOC**	**Média** $\|T_f\|$	**Média** $\|T_p\|$	**Média Mutantes**	**Média ED**	**Média EM** T_f	T_p
Chart	26 (24)	50	3.8	235.9	5284	2.8	0.09	0.05
Lang	65 (49)	6	1.8	169.8	2153	2.53	0.09	0.07
Math	106 (93)	19	3.5	186.0	9664	3.8	0.08	0.07
Mockito	38 (29)	22	3.9	661.5	2353	3.06	0.15	0.15
Time	27 (26)	53	3.5	2541.4	11031	4.04	0.06	0.06

As informações da Tabela 5.1 são úteis porque configuram características do benchmark que permite a comparação entre as técnicas. O componente FTMES do método proposto parte da heurística de que a execução de mutantes somente com o conjunto de casos de teste que falham (T_f) reduz o custo computacional da MBFL. Da mesma forma que a maioria dos programas em teste da realidade e os programas que permitem a avaliação de técnicas de localização de defeitos, o bechmark Defects4J possui uma quantidade muito menor no tamanho do conjunto dos casos de teste que falham (T_f) em relação aos que passam (T_p). Assim, as colunas 3 e 4 que da Tabela 5.1 possuem o objetivo de apresentar esses tamanhos para os programas do Defects4J. Similarmente, o escore de mutação desses conjuntos podem influenciar nos resultados das diferentes técnicas de localização. Por esse motivo, a Tabela 5.1 também detalha as informações

de escore de mutação para os diferentes programas devido a sua influência nas técnicas estudadas.

5.2 Métricas

A **eficiência** é considerada como a capacidade da técnica de reduzir a quantidade de execuções de mutantes e casos de teste. Já a **eficácia** equivale à capacidade de ranquear os elementos de programa nas posições iniciais da lista de suspeição do programa. As técnicas de localização são avaliadas sob essas duas dimensões.

5.2.1 Eficiência

O Par Teste-Mutante (*Mutant-Test Pair*), ou simplesmente as execuções MTP, contabilizam o número de execuções de mutantes pelos casos de teste. Essa métrica é usada para avaliar a capacidade das técnicas de redução de custo da MBFL(LIU et al., 2018; GONG; ZHAO; LI, 2015; LIU et al., 2017). Tal métrica relaciona o custo computacional ao número de execuções de mutantes para obtenção do ranking dos elementos de programa. Portanto, quanto menor for o valor MTP de uma técnica MBFL, maior é a sua eficiência. Essa métrica foi utilizada para avaliar as técnicas que executam mutantes.

Além disso, o tempo de execução dos casos de teste sobre os mutantes foram coletados. O valor de tempo de execução compreende o tempo total para executar os casos de teste sobre o programa original e sobre os mutantes. O valor do tempo de execução não compreende o tempo de compilação ou uso adicional para o uso de uma técnica específica.

5.2.2 Eficácia

A eficácia de localização das técnicas de localização de defeito diminui o esforço manual empregado na atividade de depuração de software. Todavia, essa atividade ainda precisa ser aprimorada porque, na prática, a experiência do programador tem sido mais efetiva do que o ranking gerado pelas técnicas (PARNIN; ORSO, 2011).

Considerando a importância da qualidade desse ranking, as técnicas estudadas foram avaliadas com 4 métricas de eficácia: i) *EXAM Score* (WONG et al., 2008); ii) *acc@n* (Accuracy) e; iii) *wef* (Wasted Effort) (SOHN; YOO, 2017) e iv) MAP. As métricas *acc@n* e *wef* são baseadas na proposta de contar elementos de programa ao invés de apresentar valores em percentuais, seguindo algumas diretrizes de literaturas mais recentes (PARNIN; ORSO, 2011).

1. **EXAM Score**: também chamado somente de *score*, consiste na métrica mais popular para avaliar a eficácia de localização da lista de elementos de programa ranqueadas pelas técnicas de localização de defeitos (PEARSON et al., 2017; WONG et al., 2016). O *EXAM Score* é obtido por uma simples divisão: n/N; em que n denota o número de elementos que necessitam ser inspecionadas antes de encontrar a linha defeituosa do programa em teste, e N diz respeito ao número de elementos de programa. O *score* é um número real que pertence ao intervalo [0,1]. *Quanto menor for o valor do score, melhor foi o desempenho da técnica*. O presente estudo avalia as técnicas no cenário do melhor caso, isto é, qualquer elemento de programa defeituoso que seja localizado é o suficiente para se entender e reparar o defeito. Se múltiplos elementos de programa estiverem com o mesmo valor de suspeição, todas elas serão tratadas como o *n-ésimo* elemento na saída, em que n é a média das posições (PEARSON et al., 2017; STEIMANN; FRENKEL; ABREU, 2013; WONG et al., 2016).
2. **acc@n**: contabiliza o número de defeitos que foram localizados nas *n posições do topo* do ranking dos elementos de programa (SOHN; YOO, 2017). Os valores 1,3,5,10,20,30,50,100 e 200 foram utilizados para definição de n. Quanto maior forem os valores para acc@n, melhor é a acurácia de localização da técnica. Os programas estudados do Defects4J, contém *bugs* com múltiplos elementos defeituosos (linhas). Na expetimentação realizada, foi considerado o primeiro elemento defeituoso para contabilizar o $acc@n$. Ou seja, assume-se que o programador ao encontrar o primeiro elemento defeituoso será capaz de identificar o defeito para posterior reparo.
3. **wef**: mede a quantia de esforço perdido na procura dos elementos de programa defeituosos pela contabilização dos elementos não defeituosos. *Quanto menor forem os valores para wef, melhor é a técnica sob essa perspectiva* (SOHN; YOO, 2017).
4. **MAP**: do termo em inglês *Mean Average Precision* (SOHN; YOO, 2017), consiste na Média da Precisão Média para avaliação do *ranking* considerando todos os elementos defeituosos.

   Primeiramente, define-se a previsão de localização $P(i)$ da técnica, conforme Equação 5-1, sendo NED_i o número de elementos defeituosos até a posição i do topo do *ranking*. Conforme Equação 5-2 corresponde na razão entre NED_i e i.

$$P(i) = \frac{NED_i}{i} \tag{5-1}$$

   Em seguida, define-se a precisão média (*Average Precision*). O elemento $Defeituoso(i) = 0$ quando o elemento de posição i não é defeituoso e $Defeituoso(i) = 1$ quando o elemento i é defeituoso. Dessa forma, o valor quanto

mais a técnica posicionar os elementos defeituosos mais próximos ao topo do *ranking* maior será o valor de *P(i)* e de AP, por consequência.

$$AP = \sum_{i=1}^{|NED|} \frac{P(i) * Defeituoso(i)}{NED} \tag{5-2}$$

Por fim, MAP é a média dos valores AP considerando todas as versões de programa sendo uma métrica calculada ao longo das versões de um mesmo projeto.

5.3 Técnicas Investigadas

A abordagem proposta nesta Tese relaciona-se com 3 classes de técnicas de localização de defeitos: i) SBFL; ii) MBFL e iii) híbridas, conforme Seção 2.3.2.

A Tabela 5.2 apresenta a configuração das técnicas quando somada as estratégias de execução de mutantes. A primeira coluna expõe um pequeno nome atribuído à junção de uma técnica de localização e uma estratégia de execução de mutantes. As estratégias de execução de mutantes usadas são descritas na segunda coluna. Por fim, a terceira coluna evidencia o nome das técnicas.

Tabela 5.2: *Técnicas de localização de defeitos.*

Abreviação	Estratégia de Execução de Mutantes	Técnica
B1	-	**SBFL** (Dstar utilizando variável* = 2)
B2	-	***MRSBFL-hybrid-max***
B3	SAM	**MBFL** (mata mutantes igual a Metallaxis)
B4	SAM	***MCBFL-hybrid-avg***
B5	DMES	**MBFL**
B6	DMES	***MCBFL-hybrid-avg***
B7	DMES_SAM	**MBFL**
B8	DMES_SAM	***MCBFL-hybrid-avg***
B9	FTMES	**MBFL**
B10	FTMES	***MCBFL-hybrid-avg***

As linhas 1 e 2 da Tabela 5.2 correspondem às técnicas SBFL (B1) e MRSBFL-hybrid-max (B2). Essas técnicas não executam mutantes. Portanto, as estratégias de execução de mutantes não se aplicam a elas. As linhas seguintes da Tabela são correspondentes a técnicas que executam mutantes.

Dstar (WONG et al., 2014) é a técnica SBFL (B1) utilizada na experimentação, uma vez que ela foi reportada como a melhor técnica SBFL validada com defeitos reais (PEARSON et al., 2017; PEARSON, 2016). Por essa razão, ela está presente em todos

os experimentos, incluindo técnicas MBFL tradicionais e híbridas. A técnica MRSBFL-hybrid-max (B2) é apresentada na próxima linha.

A técnica MBFL (B3) é considerada como a MBFL tradicional nos experimentos por representar a Metallaxis (PAPADAKIS; TRAON, 2012), como detalhada na Seção 2.3.2. Todavia, Dstar é considerada como a fórmula de cálculo dos valores de suspeição para Metallaxis, que foi proposta originalmente com a fórmula Ochiai (PAPADAKIS; TRAON, 2014). A razão de considerarmos Dstar ao invés de Ochiai como fórmula da MBFL tradicional foi motivada pelos resultados de Pearson et al. (2017) os quais reportam uma melhor interação entre Dstar e a MBFL.

A técnica MCBFL-hybrid-avg (B4) é descrita na próxima linha da Tabela 5.2. As técnicas (B3 e B4) consideram a otimização SAM *Skipping Alive Mutants (SAM)*, conforme Seção 3. Assim, elas contêm a mesma quantidade de execuções, ou seja, o mesmo custo computacional em termos de execuções MTP. Além disso, observa-se o mesmo tempo de execução para as técnicas, uma vez que elas executam os mesmos casos de teste sobre os mesmos mutantes.

A experimentação, aplica DMES (*Dynamic Mutation Execution Strategy)* (GONG; ZHAO; LI, 2015) em duas classes de técnicas MBFL: i) incorporada à MBFL tradicional (MBFL - B3) e ii) incorporada à classe das técnicas MBFL híbridas (MCBFL-hybrid-avg - B4). Quando DMES é aplicada canonicamente com as técnicas MBFL e MCBFL-hybrid-avg, é referenciada como B5 e B6, respectivamente. A estratégia DMES_SAM consiste em uma pequena variação produzida na DMES canônica pela inserção da otimização SAM em seu mecanismo. A proposta dessa combinação é comparar a qualidade das soluções produzidas pela interação entre DMES, SAM e DMES_SAM. Na Tabela 5.2, B7 e B8 correspondem à estratégia DMES_SAM aplicada sobre as técnicas MBFL (B7) e a MCBFL-hybrid-avg (B8).

Finalmente, FTMES também é aplicada às técnicas MBFL e MCBFL-hybrid-avg, sendo referenciadas como B9 e B10 na Tabela 5.2, respectivamente. As quantias de execuções de casos de teste e de mutantes são equivalentes em ambas interações MBFL (B9) e MCBFL-hybrid-avg (B10), uma vez que o mesmo subconjunto de casos de teste (T_f) é executado sobre os mutantes gerados a partir dos elementos de programa cobertos igualmente por T_f.

As estratégias de execução de mutantes visam reduzir o custo computacional das técnicas MBFL. Portanto, as técnicas MBFL (B3) e MCBFL-hybrid-avg (B4) são as principais técnicas a serem otimizadas pela aplicação das estratégias de redução: i) DMES; ii) DMES_SAM; e iii) FTMES. Logo, espera-se que todas essas aplicações tornem a MBFL mais eficiente.

5.4 Perguntas de Pesquisa (PP)

(PP1) - FTMES supera a eficiência das estratégias de execução de mutantes existentes?

(PP2) - FTMES mantém a eficácia de localização das técnicas MBFL apresentando maior eficiência?

(PP3) - FTMES@r melhora a performance da localização de defeitos produzindo:

- **(PP3.1)** - menor custo computacional em relação às técnicas MBFL?
- **(PP3.2)** - maior eficácia?
- **(PP3.3)** - melhor relação custo-benefício (efetividade) entre as técnicas de localização de defeitos?

5.5 Análise da PP1

PP1: FTMES supera a eficiência das estratégias de execução de mutantes existentes? Para responder à PP1 foram utilizadas as seguintes métricas: i) o tempo de execução dos casos de teste e ii) a quantidade de execuções entre mutantes e casos de teste (ou execuções MTP).

A Figura 5.1 apresenta a eficiência de cada técnica em termos de execuções MTP. Vale observar que a quantidade de execuções MTP alcançadas por cada técnica não está ligada somente com a quantidade de mutantes gerados para cada programa. Existem outros fatores que influenciam nos resultados alcançados [1], vale destacar: a cardinalidade dos conjuntos de casos de teste, as variações na cardinalidade dos conjuntos (mutantes e casos de teste) ao longo das diferentes versões consideradas, informações de cobertura e o mecanismo de cada técnica que determina as seleções dos pares de execução. Assim, considerando os diferentes programas estudados, é possível que uma versão de programa tenha uma cardinalidade alta de casos de teste e mutantes gerados, mas apresente valores menores de MTP para uma outra versão de programa que tenha uma menor cardinalidade para esses mesmos conjuntos, seja, por exemplo, pelas informações de cobertura, seja pelo mecanismo de seleção da estratégia de execução de mutantes.

Dentre as técnicas MBFL da Figura 5.1, FTMES é a estratégia de execução de mutantes mais eficiente em todos os programas estudados. Esse resultado acontece porque FTMES realiza execuções entre mutantes e casos de teste somente com o conjunto dos

[1] Até mesmo a quantidade de MTPs da técnica FTMES não pode ser obtida somente com a quantidade de mutantes gerados e casos de teste que falham para uma determinada versão de programa. Na verdade, a quantidade de pares de execução MTPs serão obtidos considerando o tamanho do conjunto dos casos de teste que falham e as respectivas quantidade de mutantes cobertos por cada um desses casos de teste que falham. Ou seja, os valores MTP também devem consideram as informações de cobertura inerentes ao conjunto dos casos de teste.

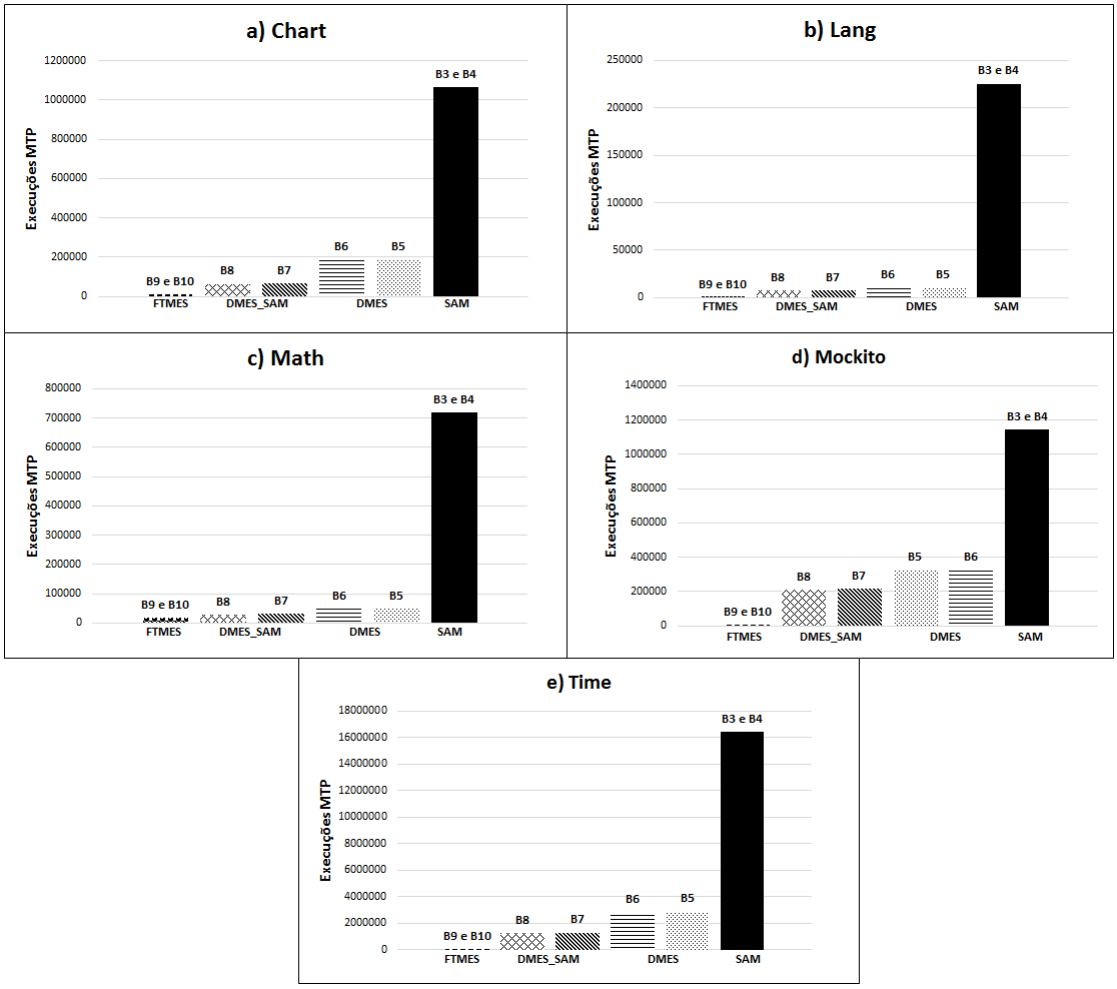


Figura 5.1: *Médias das quantidades de execuções de mutantes (Mutant-Test Pair - MTP) alcançadas pelas estratégias de execução de mutantes.*

casos de teste que falham, um conjunto muito menor do que o conjunto dos casos de teste que passam. Assim, FTMES apresenta uma maior eficiência do que a obtida pelas outras técnicas (DMES, DMES_SAM e SAM) que empregam execuções entre os mutantes, casos de teste que falham e casos de teste que passam.

Outro resultado notório é o observado entre DMES e suas variações. No geral, DMES_SAM demonstrou-se mais eficiente do que a versão original DMES. Esse resultado acontece porque DMES_SAM possui um recurso adicional ao mecanismo original de DMES no processo de execuções de mutantes. Esse recurso consiste em não executar mutantes que não morrem com nenhum caso de teste que falha (recurso *Skipping Alive Mutants* - SAM). A versão original DMES não possui esse recurso. Assim, a interação entre DMES e o recurso SAM, expresso pela estratégia DMES_SAM, potencializou a capacidade da DMES canônica de evitar execuções entre mutantes. Em outras palavras, a DMES canônica não pulou algumas execuções evitadas pelo recurso SAM.

A otimização SAM demonstrou-se menos eficiente do que todas as outras. O pior desempenho de SAM é explicado pelo fato da quantidade de execuções evitadas com o mecanismo de pular mutantes que não morrem com os negativos de SAM ser menor do que a quantidade de execuções evitadas pelo mecanismo das outras técnicas. A maior eficiência de FTMES em relação a abordagem SAM era esperada porque SAM executa mutantes com casos de teste que passam e FTMES não executa. Por outro lado, a maior eficiência de DMES em relação à otimização SAM destaca que o mecanismo de DMES possui maior potencial de evitar execuções de mutantes. Isso acontece porque DMES possui dois mecanismos de otimização: 1) descarte de execuções de mutantes que atingem um valor limite de suspeição e 2) também o descarte execuções de casos de teste que passam com baixo escore de mutação.

Ao contrário de FTMES e SAM que produzem a mesma eficiência quando aplicadas a MBFL tradicional e híbrida, DMES_SAM e DMES produziram diferentes resultados de eficiência entre a MBFL tradicional e a MBFL híbrida ao longo dos diferentes programas. Para DMES_SAM, em todos os programas, a interação mais eficiente foi obtida com MCBFL-hybrid-avg (B8). Para DMES, a interação mais eficiente, percebida em 4 dos 5 dos programas estudados, foi obtida também com MCBFL-hybrid-avg (B6). No caso de DMES, a única exceção ocorre no programa *Mockito* que apresenta uma interação mais eficiente entre DMES com a MBFL (B5). Tal fenômeno acontece no programa *Mockito* porque o escore de mutação do conjunto de teste (T_f e T_p) é maior do que o escore de mutação percebido nos outros programas, conforme a Tabela 5.3 que apresenta os escores de mutação dos conjuntos T_f e T_p. Isso faz com o valor *threshold* seja alcançado mais rapidamente no programa *Mockito* porque os casos de teste de T_p são priorizados pela sua capacidade de matar mutantes (escore de mutação).

Tabela 5.3: *Escore de mutação médio do conjunto T_f e T_f sobre os mutantes dos programas estudados.*

	Escore de Mutação (T_f)	**Escore de Mutação (T_p)**
Chart	0,09	0,05
Lang	0,09	0,07
Math	0,08	0,07
Mockito	**0,15**	**0,15**
Time	0,06	0,06

A Tabela 5.4 fornece o tempo de execução médio de todas as técnicas estudadas em minutos. Diferentemente da Figura 5.1, que fornece o custo em termos de valores MTP (independente de tempo), a Tabela 5.4 fornece uma dimensão de tempo na descrição da performance das técnicas. O objetivo é ter uma dimensão de tempo na performance das técnicas estudadas, o que também permite uma comparação direta de uma técnica SBFL com uma técnica MBFL. Vale observar que diversos fatores podem influenciar

em diferentes tempos de execução. A quantidade de execuções (valores MTP) obtidas é apenas um desses fatores [2] e não determina sozinha o tempo alcançado por uma técnica.

Um dos resultados com maior destaque na Tabela 5.4 é o alto custo computacional de uma técnica MBFL em relação à técnica SBFL (B1). Dentre as técnicas que executam mutantes, a estratégia FTMES é a que torna o custo da MBFL mais aproximado ao custo da SBFL (B1). Após FTMES, as estratégias com melhor performance, em sequência, são: DMES_SAM, DMES e a otimização SAM. Apesar de não ser o fator determinante, em termos gerais, a análise MTP da Figura 5.1 forneceu essa mesma sequência de melhores estratégias de execução de mutantes, em termos de eficiência.

FTMES e DMES são duas estratégias de execução de mutantes. SAM é uma otimização da MBFL realizada em trabalhos prévios (PAPADAKIS; TRAON, 2012; PEARSON et al., 2017). Nesse sentido, a otimização SAM foi considerada como parâmetro para as estratégias de execução de mutantes. Sendo assim, as técnicas MBFL (B3) e MCBFL-hybrid-avg (B4) são as principais linhas de base para avaliação do percentual de redução que as outras estratégias de execução de mutantes podem desempenhar. A Tabela 5.5 apresenta o percentual de redução de execuções MTP que as técnicas FTMES, DMES e DMES_SAM desempenharam em relação à abordagem SAM.

Tabela 5.4: *Tempo de execução em minutos das técnicas por programa.*

			Estratégias de Execução de Mutantes							
			SAM		***DMES***		***DMES_SAM***		***FTMES***	
	SBFL	**MRSBFL***	**MBFL**	**MCBFL***	**MBFL**	**MCBFL***	**MBFL**	**MCBFL***	**MBFL**	**MCBFL***
Programas	(B1)	(B2)	(B3)	(B4)	(B5)	(B6)	(B7)	(B8)	(B9)	(B10)
Chart	0.79		801.43		141.79	140.91	51.50	48.93	8.91	
Lang	0.34		82.04		4.18	4.15	3.09	3.00	0.52	
Math	2.52		1034.47		71.94	70.77	46.67	42.40	27.23	
Mockito	4.04		2958.89		839.19	839.25	567.78	551.30	13.00	
Time	9.79		11303.18		1995.08	1950.85	907.57	858.34	16.98	
Média	**3.50**		**3233.00**		**610.44**	**601.19**	**315.32**	**300.79**	**13.33**	

A Tabela 5.5 descreve a redução produzida pela aplicação de DMES, DMES_SAM e FTMES nas técnicas MBFL (B3) e MCBFL-hybrid-avg (B4), em termos de execuções MTP. Por exemplo, a estratégia FTMES (coluna 4) reduziu 98,99% da quantidade de execuções MTP comparada às técnicas MBFL convencionais (B3 e B4) para o programa *Chart* (linha 3).

[2]Questões de *hardware* podem influenciar, para um mesmo experimento, na obtenção de diferentes tempos de execução. Além disso, vale destacar que o tempo de execução de um mutante, pode variar para: i) diferentes versões; ii) diferentes casos de teste e iii) diferentes programas. Assim, é possível que um programa com uma quantidade menor de valores MTP possa demorar mais tempo que outro cujos valores de MTP obtidos sejam menores.

Tabela 5.5: *Taxa Média de Redução de Valores MTP das Estratégias de Execução de Mutantes.*

	DMES		*DMES_SAM*		*FTMES*
Programas	**(B5)**	**(B6)**	**(B7)**	**(B8)**	**(B10)**
Chart	82,39%	82,50%	93,67%	93,99%	98,99%
Lang	95,30%	95,33%	96,64%	96,74%	99,77%
Math	93,27%	93,39%	95,72%	96,14%	97,61%
Mockito	71,74%	71,73%	80,92%	81,48%	99,70%
Time	82,42%	82,81%	92,05%	92,49%	99,94%
Média de Redução	**85,02%**	**85,15%**	**91,80%**	**92,17%**	**99,20%**

De acordo com a Tabela 5.5, a DMES canônica foi menos eficiente que sua variação DMES_SAM. Na média geral, a estratégia DMES aplicada à MBFL (B5) produziu uma redução média de 85,02%, enquanto a aplicação em MCBFL-hybrid-avg (B6) produziu 85,15%. Similarmente, a estratégia DMES_SAM aplicada à MCBFL (B7) produziu uma redução média de 91,80%, enquanto a aplicação em MCBFL-hybrid-avg (B8) produziu 92,17%. Assim, tanto a DMES canônica quanto a DMES_SAM demonstraram maior eficiência quando combinadas à técnica MCBFL-hybrid-avg. Apesar das boas reduções alcançadas por DMES_SAM e DMES canônica, FTMES as superou alcançando maiores percentuais de redução em todos os programas estudados.

Resposta para PP1: FTMES obteve uma redução de custo entre 7% e 14% maior do que as estratégias de execução DMES e DMES_SAM, respectivamente. Além disso, alcançou um custo 99,20% menor do que o empregado pela otimização SAM. Portanto, FTMES demonstrou maior eficiência do que todas estratégias de execução de mutantes existentes.

5.6 Análise da PP2

PP2: FTMES mantém a eficácia de localização das técnicas MBFL apresentando maior eficiência? Para responder a PP2, as técnicas foram comparadas considerando três métricas que avaliam a eficácia de localização de defeitos. A Tabela 5.6 apresenta a performance das técnicas de EXAM Score em uma visão independente de projeto. A primeira coluna lista valores de *EXAM Score (Score)* e as outras colunas representam a performance das técnicas cujos percentuais de *score* foram menores ou iguais aos valores correspondentes da primeira coluna. Por exemplo, a técnica SBFL (B1) reportou valores menores ou iguais a 0.01 em 48% (0.489) das diferentes versões de programas.

A técnica MCBFL-hybrid-avg (B4) realiza uma melhor localização de defeitos em termos de EXAM Score, conforme Tabela 5.6. FTMES (B10) aparece com a segunda melhor performance. A terceira e a quarta melhor performance são realizadas pelas

técnicas SBFL (B1) e MRSBFL-hybrid-max (B2), respectivamente. Vale destacar que a técnica DMES está no grupo das técnicas com piores valores de *EXAM score*. Além disso, a variação produzida com DMES_SAM é melhor do que a DMES canônica.

Tabela 5.6: *Comparação da eficácia de localização das técnicas em termos de EXAM Score.*

			Estratégias de Execução de Mutantes							
			SAM		***DMES***		***DMES_SAM***		***FTMES***	
	SBFL	**MRSBFL***	**MBFL**	**MCBFL***	**MBFL**	**MCBFL***	**MBFL**	**MCBFL***	**MBFL**	**MCBFL***
Score <=	(B1)	(B2)	(B3)	(B4)	(B5)	(B6)	(B7)	(B8)	(B9)	(B10)
0,01	0,489	0,471	0,385	0,548	0,403	0,376	0,412	0,430	0,131	0,534
0,05	0,778	0,756	0,688	0,851	0,624	0,656	0,661	0,701	0,570	0,796
0,10	0,905	0,896	0,738	0,928	0,670	0,742	0,715	0,792	0,742	0,910
0,15	0,950	0,941	0,756	0,968	0,697	0,783	0,742	0,828	0,819	0,946
0,20	0,968	0,968	0,769	0,977	0,701	0,796	0,751	0,842	0,837	0,973
0,30	0,982	0,986	0,774	0,982	0,701	0,796	0,751	0,851	0,864	0,982
0,40	0,982	0,986	0,774	0,991	0,701	0,801	0,751	0,855	0,864	0,986
0,50	0,982	0,986	0,774	0,991	0,701	0,801	0,751	0,855	0,864	0,986
0,60	0,991	0,995	0,977	1	0,991	0,986	0,986	0,986	0,995	0,995
0,70	1	1	0,991	1	1	0,995	1	0,995	0,995	1
0,80	1	1	0,995	1	1	0,995	1	0,995	0,995	1
0,90	1	1	0,995	1	1	0,995	1	0,995	0,995	1
1	1	1	1	1	1	1	1	1	1	1

A Tabela 5.7 apresenta uma comparação estatística entre as técnicas MBFL. O layout de apresentação dos resultados dessa Tabela é idêntico à avaliação realizada por Pearson et al. (2017), relacionando o Teste t com o tamanho do efeito (*Cohen's d*) [3]. Na primeira coluna está a pergunta "ganhador > perdedor" para fazer a comparação pareada entre FTMES e todas as outras técnicas que executam mutantes. A hipótese é que FTMES ganha das outras técnicas sendo expressada da seguinte forma: "FTMES (B10) > MBFL (B3)", ou seja, uma afirmação de que FTMES (B10) tem melhor performance do que MBFL(B3). Os outros valores seguintes dessa linha vão comprovar ou não tal afirmação.

A segunda coluna da Tabela 5.7 denominada "verdadeiro?", refere-se ao Teste t e responde "sim" ou "não" à pergunta de comparação da primeira coluna sendo que a formatação das respostas indica o nível de significância sobre o valor p para a resposta, podendo ser: **p<0.01**, p<0.05, *(p>0.05)*. Por exemplo, na terceira linha existe a pergunta "FTMES (B10) > MBFL (B3)", a resposta na segunda coluna é **sim**, sua formatação indica que a assertiva é verdadeira e que $p < 0.01$ (significância de 99%). A terceira coluna da Tabela 5.7 indica o tamanho do efeito do valor *d* no teste de *Cohen*. Da mesma forma, as diferentes formatações (negrito, itálico, entre parêntese) destacam o tamanho do efeito.

[3] A Tabela 5.7 desta Tese evidencia a mesma análise estatística apresentada na Tabela IV do estudo realizado por Pearson et al. (2017)

Por exemplo: **efeito grande**, efeito médio, (efeito pequeno) e *(efeito negligenciável)*. Perceba que o teste "FTMES (B10) > MBFL (B3)", na terceira linha da Tabela, é de "0.624" e sua formatação indica efeito médio. A coluna "95% IC" fornece o intervalo de confiança do teste para diferenciar a média. A coluna "(m – ig – p)" são contadores por defeito que foram "ganhador melhor" (m), "igual a"(ig) e "perdedor pior" (p), ignorando a magnitude da diferença. Por exemplo, Na terceira linha do teste "FTMES (B10) > MBFL (B3)", tem-se (218-1-3) significando que FTMES teve um EXAM Score 218 vezes melhor do que MBFL, 1 vez igual e 3 vezes pior.

Dessa forma, FTMES foi comparada estatisticamente com todas as técnicas MBFL estudadas que executam mutantes, a fim de verificar se FTMES é estatisticamente melhor do que as outras. O EXAM Score foi utilizado porque apresentou uma distribuição normal provada mediante a testes de normalidade [4]. Assim, o *teste t* pareado e o teste de *Cohen* foram utilizados para avaliar a significância e a magnitude da performance de FTMES comparada às outras técnicas.

Com exceção da técnica MCBFL-hybrid-avg (B4), FTMES (B10) é melhor do que todas as demais técnicas com um nível de confiança de 99% ($p \leq 0.001$) e com um *efeito de tamanho médio*. MCBFL-hybrid-avg é melhor do que FTMES em termos de EXAM Score, mas **esse resultado tem uma baixa significância estatística [5] e um efeito de tamanho negligenciável, segundo o teste *Cohen* (d)**. Em síntese, tal resultado mostra que FTMES aprimora a eficiência da MCBFL-hybrid-avg (B4) sem perda significativa de eficácia de localização.

Tabela 5.7: *Comparação estatística entre as técnicas MBFL - Test t pareado e teste t de Cohen (d).*

Comparações	**EXAM Score**		**Defeitos**	
ganhador >perdedor	verdadeiro?	d (tam. do efeito)	95% IC	(m - ig - p)
FTMES (B10) >MBFL (B3)	**sim**	0.624	[-0.136, -0.073]	(218-1-2)
FTMES (B10) >MCBFL* (B4)	*(não)*	*(0.065)*	[-0.009, 0.0189]	(85-37-99)
FTMES (B10) >DMES (B5)	**sim**	0.767	[-0.166, -0.101]	(112-13-96)
FTMES (B10) >DMES (B6)	**sim**	0.589	[-0.123, -0.064]	(133-1-87)
FTMES (B10) >DMES_SAM (B7)	**sim**	0.669	[-0.143, -0.080]	(110-12-99)
FTMES (B10) >DMES_SAM (B8)	**sim**	(0.470)	[-0.095, -0.041]	(122-2-97)

A Tabela 5.8 apresenta a performance das técnicas em termos de acurácia ($acc@n$) e esforço desperdiçado (wef) para cada programa, fornecendo uma dimensão em cada projeto.

[4] A normalidade dos dados das amostras de EXAM Score obtidas foi verificada considerando Anderson Darling e Shapiro-Wilk.

[5] O valor de p referente ao Teste de *Student* da comparação "FTMES (B10) > MCBFL* (B4)" foi de $p = 0.4902$, o que significa uma baixa significância estatística.

Tabela 5.8: *Comparação de performance em acurácia (acc@n) e esforço desperdiçado (wef). Células com* ***fundo em cor preta*** *destacam a técnica que obteve* ***melhor resultado*** *dentre as técnicas comparadas. Já as células com* ***fundo em cor cinza*** *destacam a técnica com o* ***segundo melhor resultado****.*

Técnica	Prog.	Def.	acc@1	acc@3	acc@5	acc@10	acc@20	wef_med	wef_desv
Nome:	Chart	24	0	2	3	7	12	**155,00**	551,75
SBFL	Lang	49	1	5	13	24	30	32,02	43,82
Estratégia:	Math	93	6	22	27	34	43	**60,47**	**125,06**
sem	Mockito	29	**4**	**9**	**13**	**13**	**15**	324,83	912,52
(B1)	Time	26	0	4	8	11	13	97,77	151,87
	Total	**221**	**11**	**42**	**64**	**89**	**113**	**670,09**	**1785,02**
Nome:	Chart	24	0	2	3	7	12	155,00	551,75
*MRSBFL**	Lang	49	1	5	13	24	30	32,02	43,82
Estratégia:	Math	93	6	21	26	33	42	107,54	264,91
sem	Mockito	29	**4**	**9**	**12**	12	14	234,83	776,37
(B2)	Time	26	0	4	8	11	13	97,77	151,87
	Total	**221**	**11**	**41**	**62**	**87**	**111**	**627,15**	**1788,71**
Nome:	Chart	24	**1**	4	7	9	12	976,04	2441,83
MBFL	Lang	49	5	12	19	23	30	218,53	504,91
Estratégia:	Math	93	8	18	22	34	44	540,61	1286,40
sem	Mockito	29	1	**5**	6	9	10	247,48	**610,64**
(B3)	Time	26	**6**	**7**	**10**	**13**	**16**	395,65	1124,89
	Total	**221**	**21**	**46**	**64**	**88**	**112**	**2378,32**	**5968,68**
Nome:	Chart	24	**2**	**7**	**9**	**11**	**14**	**60,54**	**133,22**
*MCBFL**	Lang	49	7	**17**	**22**	**31**	**37**	**19,39**	**36,04**
Estratégia:	Math	93	**10**	**27**	**35**	**44**	**53**	**53,53**	**134,84**
sem	Mockito	29	**5**	**9**	9	**15**	**18**	**186,83**	771,41
(B4)	Time	26	**5**	**10**	**10**	**14**	**16**	**70,73**	**115,01**
	Total	**221**	**29**	**70**	**85**	**115**	**138**	**391,01**	**1190,52**
Nome:	Chart	24	**1**	3	5	8	9	1166,46	2780,58
MBFL	Lang	49	**9**	12	15	20	27	346,86	626,07
Estratégia:	Math	93	8	20	25	37	43	673,46	1472,27
DMES	Mockito	29	1	2	2	4	6	492,93	938,39
(B5)	Time	26	**5**	6	8	11	**17**	442,58	1296,42
	Total	**221**	**24**	**43**	**55**	**80**	**102**	**3122,29**	**7113,72**
Nome:	Chart	24	**1**	3	6	8	10	1219,21	2729,04
*MCBFL**	Lang	49	**10**	**14**	15	23	32	233,02	556,31
Estratégia:	Math	93	8	15	22	37	43	467,10	1317,12
DMES	Mockito	29	2	2	2	4	4	432,59	886,10
(B6)	Time	26	**5**	6	6	8	11	483,88	1232,15
	Total	**221**	26	**40**	**51**	**80**	**100**	**2835,80**	**6720,71**
Nome:	Chart	24	**1**	**5**	7	9	11	1117,21	2792,23
MBFL	Lang	49	7	13	18	22	27	304,43	612,71
Estratégia:	Math	93	7	19	21	34	43	592,61	1427,29
DMES_SAM	Mockito	29	0	3	3	6	8	462,66	938,26
(B7)	Time	26	**5**	**5**	8	10	17	448,69	1294,96
	Total	**221**	**20**	**45**	**57**	**81**	**106**	**2925,60**	**7065,45**
Nome:	Chart	24	**1**	**5**	**8**	**10**	12	1064,38	2744,75
*MCBFL**	Lang	49	**9**	**17**	**20**	**25**	31	119,02	434,45
Estratégia:	Math	93	7	15	18	36	45	440,62	1304,56
DMES_SAM	Mockito	29	0	3	3	7	8	435,31	900,07
(B8)	Time	26	**5**	5	8	10	15	435,12	1236,62
	Total	**221**	**22**	**45**	**57**	**88**	**111**	**2494,44**	**6620,46**
Nome:	Chart	24	0	0	0	0	1	3625,75	5765,03
MBFL	Lang	49	0	0	0	0	0	756,55	624,14
Estratégia:	Math	93	0	0	0	0	0	2051,02	1748,26
FTMES	Mockito	29	0	0	0	0	0	1401,69	**613,13**
(B9)	Time	26	0	0	0	0	0	4497,19	1528,61
	Total	**221**	**0**	**0**	**0**	**0**	**1**	**12332,20**	**10279,18**
Nome:	Chart	24	0	**5**	7	**10**	**13**	**53,46**	**89,84**
*MCBFL**	Lang	49	1	5	16	**25**	**36**	**25,47**	**38,74**
Estratégia:	Math	93	**12**	**27**	**30**	**40**	**50**	75,35	242,29
FTMES	Mockito	29	3	**9**	**12**	12	**15**	**210,79**	774,94
(B10)	Time	26	0	4	**9**	11	14	**74,92**	**107,29**
	Total	**221**	**16**	**50**	**74**	**98**	**128**	**440,00**	**1253,09**

A primeira coluna da Tabela 5.8 lista as siglas de cada técnica estudada. A segunda coluna mostra o número de defeitos considerados. A partir da quarta até a oitava coluna são apresentados diferentes valores de *n* para o cálculo de acurácia *acc@n*. Por fim, as duas últimas colunas fornecem a média e o desvio padrão para as métricas de esforço desperdiçado *wef_med* e *wef_std)*, respectivamente. Além disso, o **fundo em cor preta**, presente em algumas células, sinaliza que a referida técnica apresentou o **primeiro melhor resultado** para o respectivo parâmetro. Similarmente, o **fundo em cor cinza** representa o **segundo melhor resultado** para o parâmetro.

É possível visualizar que MCBFL-hybrid-avg (B4) possui melhor performance em termos esforço desperdiçado (*wef*) e também em acurácia (*acc@n*), em uma visão geral da Tabela 5.8. As células com fundo em cor preta destaca nitidamente a superioridade de MCBFL-hybrid-avg (B4) ao longo dos diferentes programas e parâmetros. Uma das razões para esse resultado está no fato de que a MCBFL-hybrid-avg (B4) consegue uma boa combinação entre as informações de cobertura com as informações de mutação, conforme demonstrado em trabalhos prévios (PEARSON et al., 2017).

A técnica FTMES (B10) apresenta o segundo melhor resultado, tanto em acurácia (*acc@n*), quanto em esforço desperdiçado (*wef*). As células com fundo em cor cinza fornecem um bom destaque para esse resultado, principalmente, quando olha-se para os totais das métricas. A única métrica que FTMES (B10) não obteve o segundo melhor resultado foi para o experimento *acc@1*. Para todos os outros valores de *n*, FTMES (B10) demonstrou o segundo melhor resultado geral. Isso fortalece que FTMES (B10) consegue reduzir consideravelmente o custo da MBFL híbrida mantendo uma acurácia de localização aproximada.

É importante destacar que FTMES (B10) alcançou um melhor resultado do que MCBFL-hybrid-avg (B4) no programa Math para o experimento de acc@1. Embora seja um caso isolado, esse resultado é interessante porque FTMES, além de reduzir o custo computacional, conseguiu aprimorar a acurácia. Para todos os outros experimentos, ou FTMES (B10) empata, ou se aproxima de MCBFL-hybrid-avg (B4), conforme Tabela 5.8.

Pearson et al. (2017) já demonstraram que a MBFL tradicional (B3) possui menor eficácia de localização do que a MBFL híbrida (MCBFL-hybrid-avg - B4). Uma das razões está nas limitações dos operadores de mutação. Existem linhas que não possuem mutantes correspondentes, isso faz com que a MBFL tradicional não consiga atribuir valores de suspeição para determinados tipos de defeitos. Nesse contexto, a MBFL híbrida consegue alcançar melhores resultados porque possui informações de cobertura até mesmo nos elementos de programas imutáveis, conseguindo atribuir valores de suspeição para esse tipo de defeito.

A técnica B9, composta pela aplicação de FTMES à MBFL tradicional, obteve o pior resultado entre as técnicas. Acredita-se que a principal razão de FTMES ter piorado o desempenho da MBFL tradicional esta na redução das informações de mutação provocada por FTMES, uma vez que ela considera apenas as execuções dos casos de teste que falham. Isso demonstra que a abordagem FTMES é mais indicada para ser aplicada em conjunto com a MBFL híbrida para não haver perda em eficácia de localização.

É possível concluir que DMES_SAM obteve melhores resultados do que sua proposta original DMES. Esse resultado indica que ainda existem possibilidades de melhoria da abordagem original DMES.

As melhores aplicações das estratégias de execução de mutantes de DMES e FTMES acontecem na abordagem MBFL híbrida, conforme Tabela 5.8. Comparando as duas estratégias diretamente, observe que FTMES (B10) superou DMES_SAM (B8) em termos gerais. Todavia, considerando os programas Lang e Chart, DMES_SAM (B8) obteve melhores resultados que FTMES (B10) nos parâmetros acc@1, acc@3 e acc@5. Por outro lado, FTMES (B10) apresentou melhor resultado do que DMES_SAM (B8) para o acc@20. Nos programas Math, Mockito e Time, FTMES (B10) exibiu melhores resultados do que DMES_SAM na maioria dos valores de n para os $acc@n$ estudados.

Do ponto de vista de comparação das diferentes estratégias de execução de mutantes, a otimização SAM alcança os melhores resultados em eficácia de localização, entretanto possui o maior custo computacional. A estratégia DMES consegue maiores reduções de custo do que a estratégia SAM, porém perde em eficácia de localização com uma distância considerável. Por outro lado, a estratégia FTMES alcança as maiores reduções de custo mantendo uma eficácia de localização mais aproximada da otimização SAM.

Resposta para PP2: FTMES aplicada a técnica MCBFL-hybrid-avg obteve uma eficácia de localização aproximada em termos de EXAM Score, acurácia ($acc@n$) e esforço desperdiçado (wef) quando comparada a versão original de MCBFL-hybrid-avg. Todavia, a aplicação de FTMES na MBFL tradicional gerou perdas consideráveis em eficácia de localização indicando o uso de FTMES é mais apropriado à MBFL híbrida para que a redução de custo alcançada não prejudique a eficácia de localização da técnica quando executa mutantes originalmente.

5.7 Análise da PP3

PP3: A pergunta de pesquisa PP3 visa avaliar a interação entre: i) SFilter@r com FTMES (FTMES@r) e ii) SFilter@r e MCBFL-hybrid-avg (MCBFL@r) [6]. O objetivo consiste em verificar essa interação na perspectiva de custo e benefício. Especificamente, espera-se aprimorar: i) o custo em relação à MBFL quando não utiliza SFilter@r e ii) o benefício em relação a Ranking@r obtido a partir da abordagem SBFL. Assim, a PP3 foi respondida sob três perspectivas: PP3.1) redução do custo computacional das técnicas MBFL; PP3.2) aumento da eficácia de localização de defeitos das técnicas SBFL e; PP3.3) análise de custo-benefício (efetividade). As seguintes técnicas foram utilizadas:

1. **SBFL**: representada pela técnica Dstar (WONG et al., 2012), conforme Seção 5.3. Foi selecionada por ser a técnica SBFL com melhor interação com a abordagem MBFL, conforme trabalhos prévios (PEARSON et al., 2017).
2. **MCBFL**: representada pela técnica híbrida MCBFL-hybrid-avg (B4 da Tabela 5.2). Foi escolhida por ser a técnica MBFL com maior eficácia de localização, conforme Seção 5.6.
3. **MCBFL@r**: representa a interação entre SFilter@r com a técnica híbrida MCBFL-hybrid-avg (B4 da Tabela 5.2).
4. **FTMES**: representada pela técnica B10 da Tabela 5.2. Foi adotada por ser a técnica MBFL que incorpora a estratégia de execução de mutantes com maior percentual de redução de custo computacional, conforme Seções 5.5 e 5.6.
5. **FTMES@r**: representa a interação entre SFilter@r com a técnica FTMES (B10 da Tabela 5.2).

Redução do custo computacional das técnicas MBFL

PP3.1: FTMES@r aprimora a localização de defeitos produzindo menor custo computacional em relação as técnicas MBFL? O objetivo dessa pergunta de pesquisa consiste em avaliar se a interação de SFilter@r com FTMES (FTMES@r) produz maior redução do que a interação de SFilter@r com MCBFL. A Tabela 5.9 expressa o custo computacional das técnicas a partir dos diferentes tamanhos de *r* na aplicação de SFilter@r. O tempo de execução é apresentado em minutos.

[6] Observe que, no texto, as interações entre SFilter@r com FTMES e MCBFL em gráficos, tabelas e legendas podem ser representadas por FTMES@r e MCBFL@r. Além dessas representações, algum valor de @r pode ser referenciado, por exemplo com @r=30, fazendo com que as referências sejam da seguinte forma: FTMES@30 e MCBFL@30; para FTMES e MCBFL, respectivamente.

Tabela 5.9: *Tempo de execução das técnicas MBFL (em minutos) com diferentes tamanhos de @r na aplicação de SFilter@r.*

Técnicas	Prog.	Com SFilter@r						Sem SFilter@r
		@r=10	*@r=20*	*@r=30*	*@r=50*	*@r=100*	*@r=200*	
					MCBFL@r			**MCBFL**
MCBFL	Chart	7,94	15,23	21,46	39,42	71,33	136,7	801,43
	Lang	1,97	3,54	4,67	6,51	12,66	26,27	82,04
	Math	16,65	30,18	43,88	74,3	115,26	219,41	1034,47
	Mockito	210,83	442,6	640,8	992,26	1595,01	2146,56	2958,89
	Time	123,91	241,49	361,64	598,85	1187,21	2408	11303,18
	Média	**72,26**	**146,61**	**214,49**	**342,27**	**596,29**	**987,39**	**3236,00**
					FTMES@r			**FTMES**
FTMES	Chart	0,89	0,98	1,05	1,29	1,67	2,49	8,91
	Lang	0,35	0,37	0,38	0,39	0,44	0,49	0,52
	Math	2,82	3,18	3,47	4,12	5	7,75	27,23
	Mockito	4,76	5,6	6,27	7,52	9,47	11,21	13
	Time	9,89	9,99	10,09	10,3	10,79	11,73	16,98
	Média	**3,74**	**4,02**	**4,25**	**4,72**	**5,48**	**6,73**	**13,33**

Considerando os diferentes valores de $@r$ e todos os programas da Tabela 5.9, FTMES@r obteve maiores reduções de custo computacional do que MCBFL@r. Além disso, menores valores de $@r$ geram menores custos, uma vez que a aplicação de SFilter@r reduz consideravelmente a quantidade dos elementos de programa e, consequentemente, as possibilidades de gerações de programas mutantes. Assim, os respectivos tempos de execução das técnicas são reduzidos. Tal redução pode ser vista pela comparação dos diferentes resultados dos valores de $@r$ com a coluna ***Sem SFilter***, que representa os resultados de FTMES e MCBFL quando não usam o componente SFilter@r.

A Figura 5.2 exibe os percentuais de redução das execuções MTP de FTMES@r e MCBFL@r em relação à MBFL tradicional.

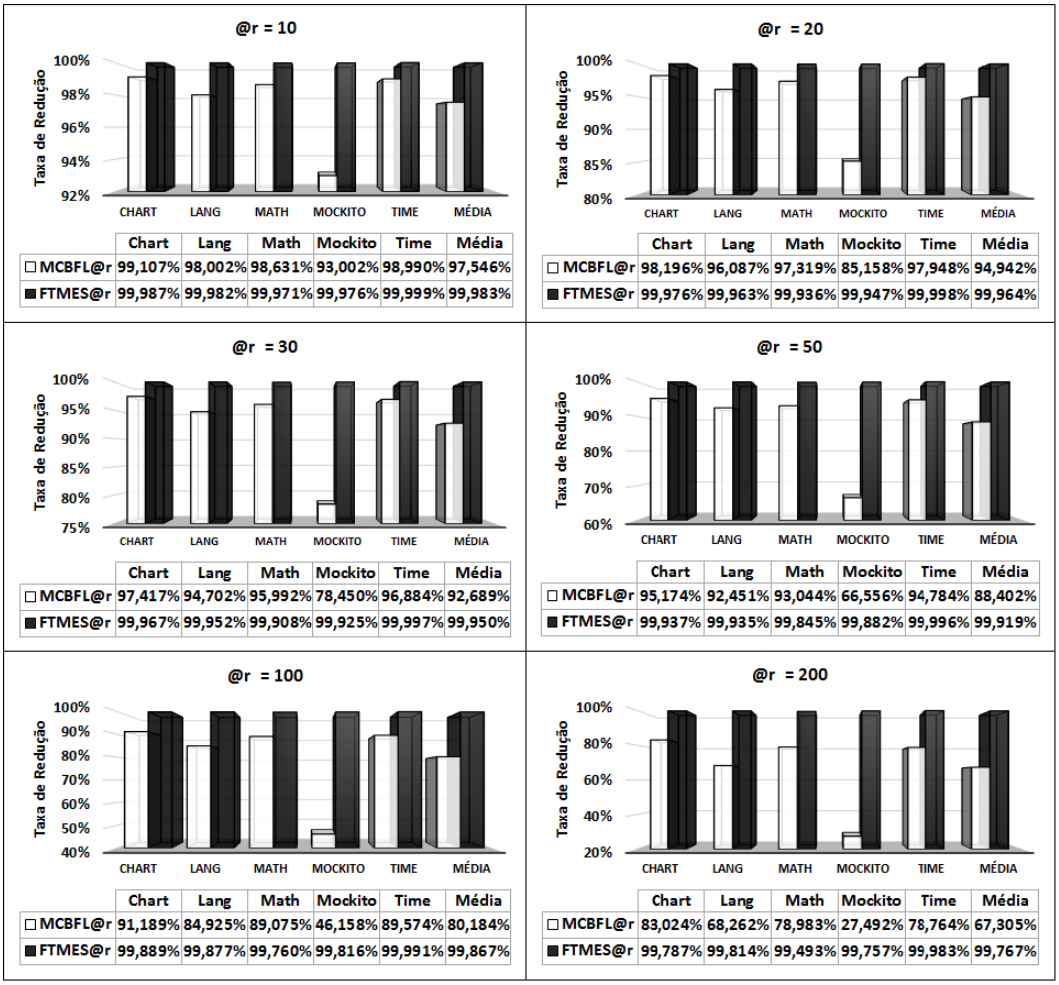


Figura 5.2: *Percentual de redução da quantidade de MTP empregado por FTMES@r e MCBFL@r por meio de SFilter@r em relação à MBFL tradicional.*

Na Figura 5.2, observe que, quanto maior o percentual, maior é a redução de custo obtida. FTMES@r produziu maiores reduções para todos os valores de MCBFL@r e em todos os valores de $@r$ na aplicação de SFilter@r. Além disso, é notável que valores de $@r$ menores ou iguais a 30 ($@r \leq 30$) apresentam reduções acima de 92% em relação à MBFL tradicional.

A análise de MCBFL@r revela que valores de $@r$ entre [50,100] produzem uma média de redução de 80%. A menor média de redução é de 67% quando $@r = 200$. Um resultado que chama atenção, na análise de MCBFL@r, é o fato do programa *Mockito* apresentar um resultado de redução consideravelmente menor comparado aos percentuais realizados nos outros programas. Para entender o que acontece, nesse caso, é necessário retomar os elementos da otimização SAM que podem aumentar ou diminuir a eficiência da MCBFL.

A otimização SAM descarta execuções de um mutante m com todo o conjunto T_p dos casos de teste que passam quando m não morre com nenhum $t \in T_f$ (casos de teste

que falham). Sendo assim, a otimização SAM não terá nenhum efeito de redução se todos os mutantes $m \in M$ forem "mortos" por algum $t \in T_f$. Em outras palavras, a otimização SAM não realiza redução se o escore de mutação do conjunto T_f for igual a 1.0, o que obrigará a execução de todo o conjunto T_p sobre todos os mutantes gerados. Assim, quanto maiores forem os escores de mutação de T_f menores serão as possibilidades de redução pela otimização SAM na execução de T_p sobre os mutantes gerados.

Nesse sentido, a técnica MCBFL@r emprega menores reduções de custo no programa *Mockito* porque o escore de mutação do conjunto T_f é superior ao escore de mutação obtido nos outros programas. A Tabela 5.3 apresenta o escore de mutação do conjunto T_f sobre os mutantes dos programas estudados. Observe que o escore de mutação do programa *Mockito* é superior aos obtidos nos outros programas.

FTMES@r produziu uma redução da quantidade de execuções superior a 99% para qualquer valor de $@r$ e em todos os programas considerados. Tal resultado demonstra a alta redução alcançada pela abordagem FTMES@r. A fórmula 5-3 descreve o cálculo realizado para obtenção do percentual de aproximação do tempo de uma técnica MBFL em relação ao tempo da técnica SBFL. Assim, se o percentual for mais próximo de 100% indica que o tempo da MBFL foi otimizado tornando-se mais próximo ao tempo ao tempo da abordagem SBFL. Similarmente, um menor percentual indica uma menor otimização e uma maior distância ao tempo da SBFL.

$$\%Aprox = \frac{SBFL_{tempo}}{MBFL_{tempo}} \tag{5-3}$$

Conforme descrito na avaliação da PP1 (Seção 5.5), a classe das técnicas SBFL possui um custo computacional muito menor do que a classe de técnicas MBFL. Em geral, o custo de uma técnica MBFL já inicia com o custo de uma técnica SBFL porque realiza a execução do programa defeituoso original sobre o conjunto de teste, seja para MBFL tradicional ou híbrida. Assim, o objetivo de uma técnica de redução de custo consiste em aproximar o custo da MBFL a esse custo inicial da técnica SBFL.

Nesse sentido, a Figura 5.3 mostra o percentual de aproximação ao tempo de execução da SBFL de FTMES@r e MCBFL@r alcançam ou quando empregada originalmente (coluna **Sem**). Quanto maior for o percentual, maior é a aproximação ao tempo de execução da técnica SBFL. Em outras palavras, um maior percentual representa uma maior redução no custo em FTMES e MCBFL, fato que gera uma maior aproximação ao tempo de execução da SBFL.

A Figura 5.3 demonstra que FTMES@r produz maior aproximação de tempo de execução do que MCBFL@r. A aproximação de tempo FTMES@r é superior a 50% quando $@r \geq 50$ para todos os programas. Por outro lado, MCBFL@r produz menores percentuais de aproximações nos programas *Mockito* e *Time*.

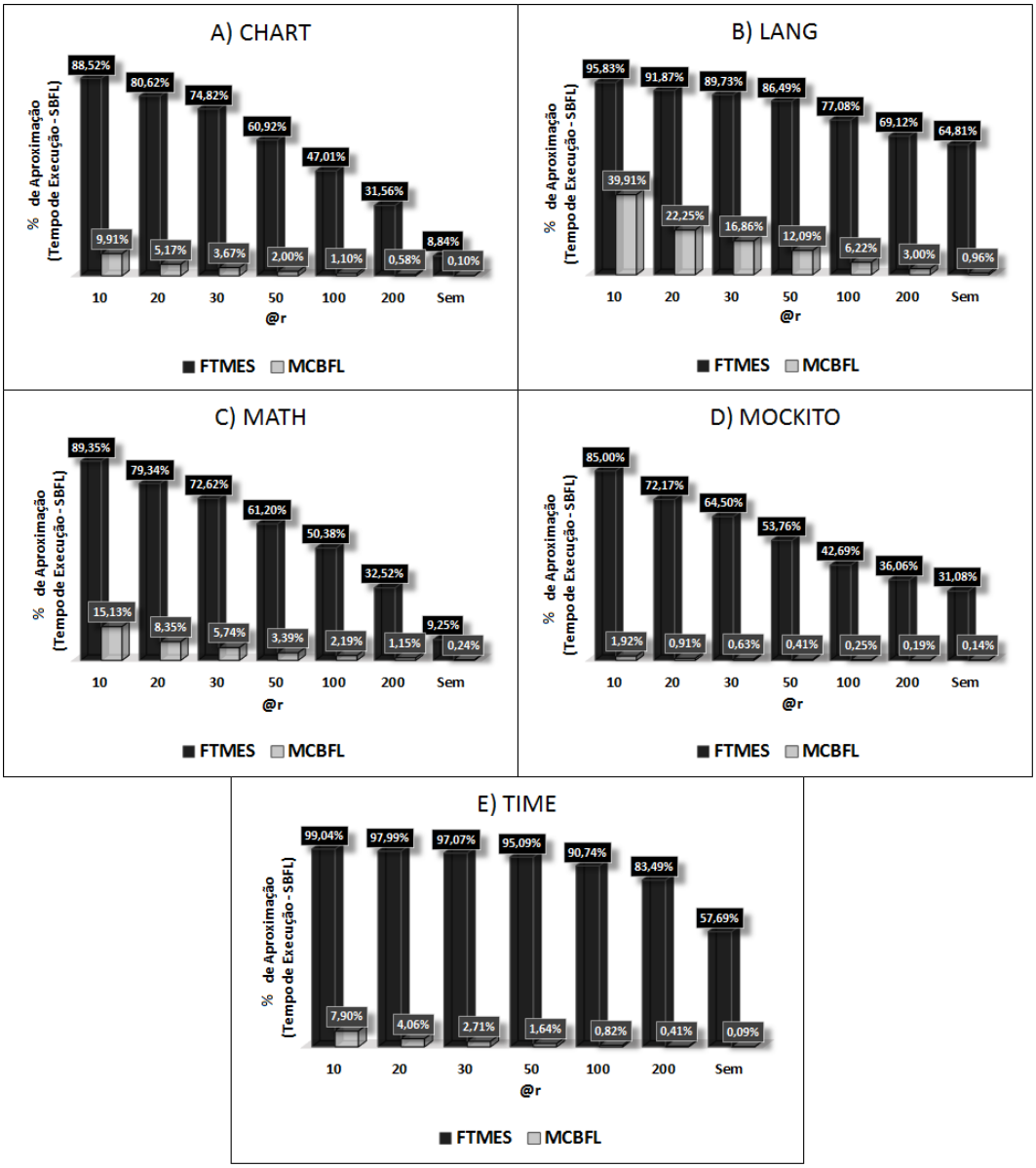


Figura 5.3: *Comparativo do percentual de aproximação do tempo de execução em relação à técnica SBFL.*

Resposta para PP3.1: FTMES@r produziu a maior redução de custo para a MBFL para todos os programas e tamanhos de ranking (@r) estudados. A interação SFilter@r com a técnica MCBFL também produziu reduções significativas. Todavia, as reduções produzidas pela abordagem FTMES@r foram consideravelmente superiores.

Aumento da eficácia de localização da SBFL

PP3.2: FTMES@r aprimora a localização de defeitos produzindo maior eficácia?

O objetivo da PP3.2 consiste em avaliar se a interação entre SFilter@r com FTMES (FTMES@r) produz maior eficácia de localização do que a interação entre SFilter@r com MCBFL-hybrid-avg (MCBFL@r) na priorização dos elementos defeituosos do Ranking@r obtido pela técnica SBFL.

A Tabela 5.10 mostra resultados de acurácia (acc@n) e esforço perdido (wef_med) dos Rankings originais obtidos pela SBFL comparados aos resultados de FTMES@r e MCBFL@r. A primeira coluna consiste nos valores de $@r$ que são $\{30, 50, 100, 200\}$ [7]. A segunda e a terceira coluna apresentam as técnicas e os programas estudados, respectivamente. A quarta coluna (Def.) representa os defeitos de cada versão de programa.

É importante destacar que a quantidade de defeitos (*bugs* ou versões de programa) selecionados para o experimento depende dos diferentes valores de $@r$, uma vez que existem algumas versões de programas com tamanhos de rankings menores dos que os valores de @r. Assim, a versão é desconsiderada quando a quantidade de elementos cobertos pelos casos de teste que falham for menor do que o valor do que o valor de @r. Isso acontece, por exemplo, com a versão 25 do programa *Chart* e o experimento e o parâmetro @r=30. Portanto, existe uma variação na quantidade de defeitos para valores de $@r$ que superam o tamanho dos rankings de algumas versões de programa. Observe que os valores de $@r$ iguais a 100 e 200 selecionaram menores quantidades de versões para o experimento (211 e 198 versões).

As colunas de 5 a 9 da Tabela 5.10 evidenciam os diferentes valores de n ($\{1, 3, 5, 10, 20\}$) para o cálculo da acurácia ($acc@n$). A partir da quarta coluna, a sexta linha de cada experimento apresenta os totais para cada técnica. Por exemplo, considerando o experimento com $@r = 50$, da técnica FTMES, existe um total de 220 defeitos considerados, $acc@1 = 28/5\%$, sendo 28 defeitos localizados, que representam 5% do total de 220 defeitos. A mesma leitura pode ser feita para o cálculo da acurácia das outras top n posições. Por fim, a coluna 10 destaca a média de esforço desperdiçado (*wef_med*) demandado por cada técnica.

No *layout* da Tabela 5.10, as células com fundo em cor cinza representam a performance de localização do ranking obtido originalmente pela técnica SBFL. As células com fundo em cor branca representam a atuação das técnicas MBFL, aprimorando o ranking original da SBFL com uma melhor priorização dos elementos defeituosos.

[7] Os valores de @r = 10 e @r = 20 não foram apresentados na Tabela 5.10 simplesmente por uma questão de apresentação dos dados.

Tabela 5.10: *Aplicação da SFilter@r e a eficácia de localização.*

@r	Técnica	Prog.	Def.	acc@1	acc@3	acc@5	acc@10	acc@20	wef_med
	Nome:	Chart	23	0	2	2	6	11	92,70
	SBFL	Lang	49	1	5	13	24	30	17,14
Sem	**Estratégia:**	Math	93	6	22	27	34	43	40,17
	sem	Mockito	29	4	9	13	13	15	193,45
		Time	26	0	4	8	11	13	61,00
	Total		**220**	**11/5%**	**42/19%**	**63/29%**	**88/40%**	**112/51%**	**404,46**
	Nome:	Chart	23	2	7	8	10	12	89,30
	MCBFL-hybrid-avg	Lang	49	7	16	20	31	34	13,67
	Estratégia:	Math	93	11	26	34	44	54	37,32
	sem	Mockito	29	5	8	9	13	17	193,10
		Time	26	5	11	12	14	16	57,23
@r = 30	**Total**		**220**	**30/14%**	**68/31%**	**83/38%**	**112/51%**	**133/60%**	**390,63**
	Nome:	Chart	23	0	4	6	9	12	90,00
	MCBFL-hybrid-avg	Lang	49	1	5	16	26	36	15,35
	Estratégia:	Math	93	12	28	30	40	52	37,48
	FTMES	Mockito	29	3	9	12	13	15	193,72
		Time	26	0	4	9	12	14	59,96
	Total		**220**	**16/7%**	**50/23%**	**73/33%**	**100/45%**	**129/59%**	**396,52**
	Nome:	Chart	23	0	2	2	6	11	127,04
	SBFL	Lang	49	1	5	13	24	30	23,96
Sem	**Estratégia:**	Math	93	6	22	27	34	43	44,02
	sem	Mockito	29	4	9	13	13	15	233,93
		Time	26	0	4	8	11	13	93,42
	Total		**220**	**11/5%**	**42/19%**	**63/29%**	**88/40%**	**112/51%**	**522,38**
	Nome:	Chart	23	2	7	8	10	12	123,22
	MCBFL-hybrid-avg	Lang	49	7	16	20	31	34	18,65
	Estratégia:	Math	93	10	27	35	46	55	38,55
	sem	Mockito	29	5	9	9	14	19	230,62
		Time	26	4	9	11	14	16	89,73
	Total		**220**	**28/13%**	**68/31%**	**83/38%**	**115/52%**	**136/62%**	**500,77**
@r=50	**Nome:**	Chart	23	0	4	6	9	12	124,09
	MCBFL-hybrid-avg	Lang	49	1	5	16	25	35	21,39
	Estratégia:	Math	93	12	28	31	41	50	39,83
	FTMES	Mockito	29	3	9	12	13	15	234,03
		Time	26	0	4	9	11	14	91,85
	Total		**220**	**16/7%**	**50/23%**	**74/34%**	**99/45%**	**126/57%**	**511,18**
	Nome:	Chart	23	0	2	2	6	11	208,57
	SBFL	Lang	41	0	4	10	17	22	33,32
Sem	**Estratégia:**	Math	92	6	22	27	34	43	50,87
	sem	Mockito	29	4	9	13	13	15	240,79
		Time	26	0	4	8	11	13	160,65
	Total		**211**	**10/5%**	**41/19%**	**60/28%**	**81/38%**	**104/49%**	**694,20**
	Nome:	Chart	23	2	7	8	10	13	203,87
	MCBFL-hybrid-avg	Lang	41	5	14	16	24	28	24,44
	Estratégia:	Math	92	10	27	34	45	53	42,57
	sem	Mockito	29	5	8	8	13	16	246,93
		Time	26	4	9	11	13	15	158,12
@r=100	**Total**		**211**	**26/12%**	**65/31%**	**77/36%**	**105/50%**	**125/59%**	**675,92**
	Nome:	Chart	23	0	4	6	9	12	205,00
	MCBFL-hybrid-avg	Lang	41	0	4	12	18	27	28,78
	Estratégia:	Math	92	12	28	31	41	49	45,04
	FTMES	Mockito	29	3	9	12	12	15	244,38
		Time	26	0	4	9	11	14	159,08
	Total		**211**	**15/7%**	**49/23%**	**70/33%**	**91/43%**	**117/55%**	**682,28**
	Nome:	Chart	21	0	2	2	6	9	146,76
	SBFL	Lang	39	0	4	10	17	21	37,36
Sem	**Estratégia:**	Math	84	4	18	22	28	35	61,06
	sem	Mockito	29	4	9	13	13	15	213,28
		Time	25	0	3	7	10	12	184,20
	Total		**198**	**8/4%**	**36/18%**	**54/27%**	**74/37%**	**92/46%**	**642,66**
	Nome:	Chart	21	1	6	7	9	11	139,33
	MCBFL-hybrid-avg	Lang	39	4	12	15	23	27	22,77
	Estratégia:	Math	84	8	20	27	36	46	52,33
	sem	Mockito	29	4	8	8	14	17	215,24
		Time	25	5	9	10	13	15	168,00
@r=200	**Total**		**198**	**22/11%**	**55/28%**	**67/34%**	**95/48%**	**116/59%**	**597,68**
	Nome:	Chart	21	0	4	5	8	10	143,33
	MCBFL-hybrid-avg	Lang	39	0	4	12	18	26	29,44
	Estratégia:	Math	84	10	23	26	34	42	53,32
	FTMES	Mockito	29	3	9	12	12	15	212,24
		Time	25	0	3	8	10	13	182,16
	Total		**198**	**13/7%**	**43/22%**	**63/32%**	**82/41%**	**106/54%**	**620,49**

Em geral, a aplicação das técnicas MBFL aumentou a acurácia ($acc@n$) de localização de defeitos da técnica SBFL para todos os valores de $@r$ e n. A melhoria também acontece na média do esforço desperdiçado (*wef_med*) uma vez que foram obtidos valores menores.

Considerando $n = 20$, o parâmetro $@r = 50$ configurou a melhor acurácia produzida de MCBFL@r: 136 defeitos localizados / 62% de 220. Para o mesmo valor de n, a melhor acurácia produzida por FTMES@r foi alcançada com o valor $@r = 30$: 129 defeitos localizados / 59% de 220. A Figura 5.4 exibe o percentual de melhoria que a aplicação da SFilter@r realizou em relação à SBFL.

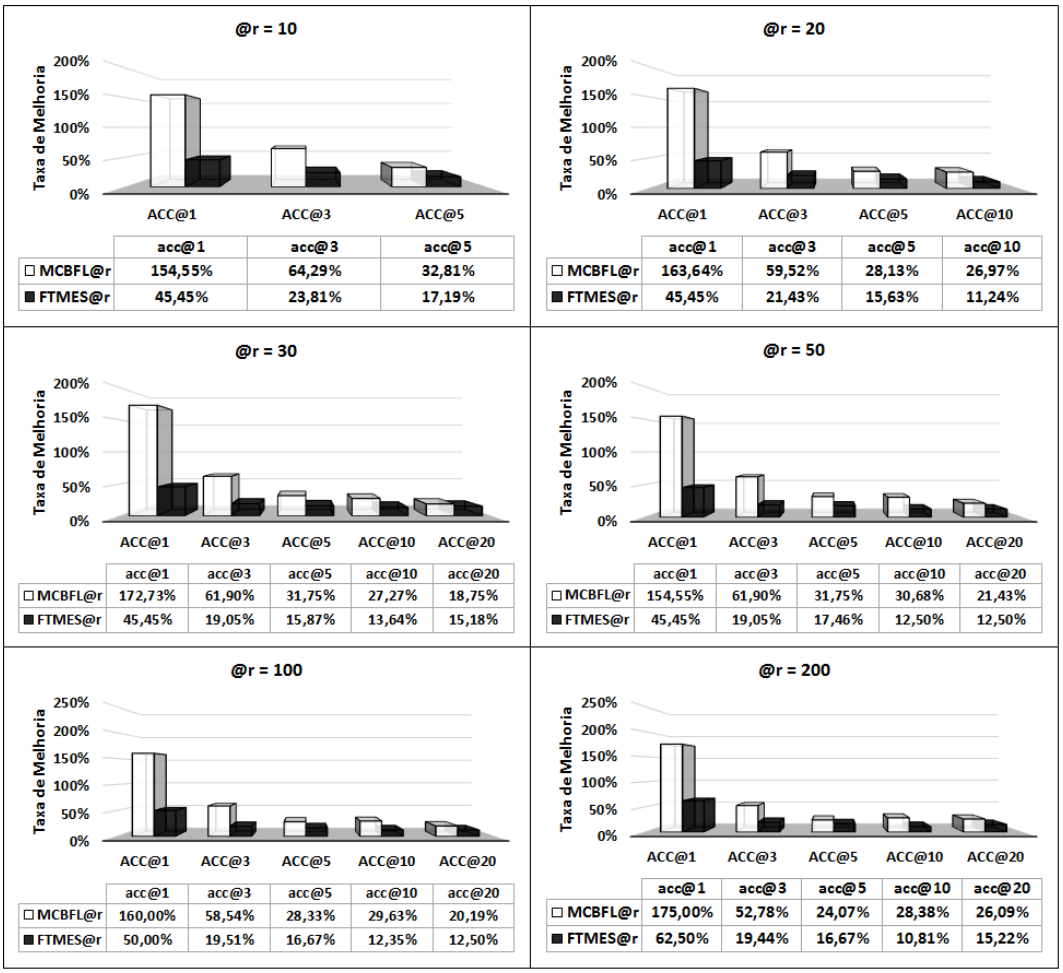


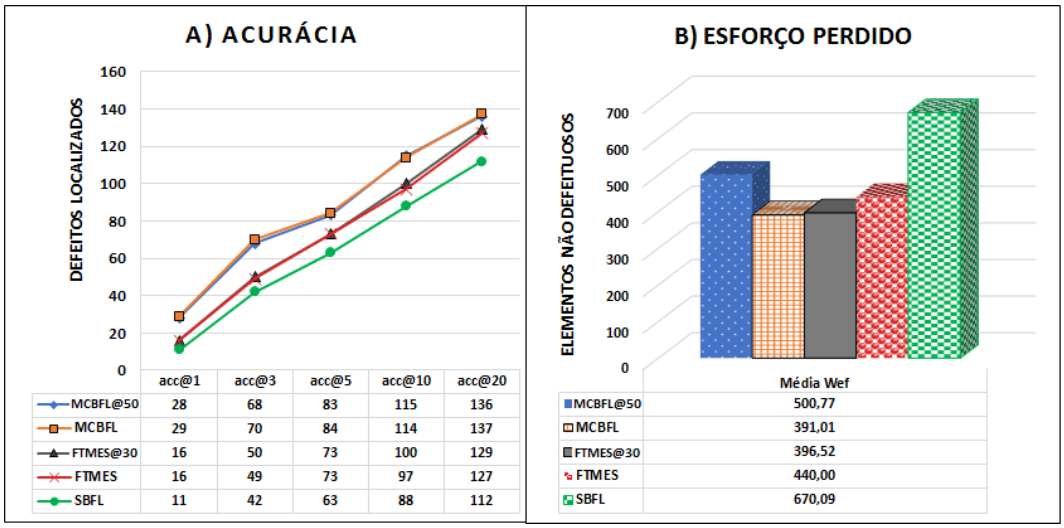
Figura 5.4: *Percentual de melhoria de acurácia de SFilter@r em relação à SBFL.*

A Figura 5.4 mostra que o valor de $@r = 200$ produz os maiores percentuais de melhoria na aplicação de SFilter@r em relação à SBFL. Todavia, as maiores quantidades de defeitos localizados da aplicação de SFilter@r com FTMES e MCBFL foram produzidos com os valores $@r = 30$ e $@r = 50$, respectivamente. A SFilter@r aplicada à MCBFL produziu maior acurácia em todos os experimentos em relação ao método FTMES@r. Por exemplo, com $n = 20$, a interação entre SFilter@r e MCBFL produziu percentuais de me-

lhoria de acurácia entre 18% e 26%, enquanto que a interação com FTMES produziu melhoria de acurácia entre 12% e 15%.

A Figura 5.5 apresenta resultados gerais de acurácia e esforço desperdiçado (*wef*) das abordagens, sem considerar a visão por programas. Em termos de acurácia (*acc@n*), a superioridade das técnicas MCBFL e MCBFL@50 é visível (interação *SFilter*@50 com MCBFL). Em contraste, a SBFL aparece com a pior performance.

As técnicas FTMES e FTMES@30 (interação *SFilter*@30 com FTMES) demonstram resultados entre o melhor e o pior desempenho. Em termos de esforço desperdiçado (*wef*), as técnicas MCBFL e FTMES@30 apresentam os melhores resultados, respectivamente. Considerando acurácia de localização (*acc@n*) e esforço (*wef*) como dois objetivos, a técnica MCBFL revela melhores resultados seguida da técnica FTMES@30. No entanto, essa análise refere-se apenas à dimensão de eficácia de localização.



	acc@1	acc@3	acc@5	acc@10	acc@20
MCBFL@50	28	68	83	115	136
MCBFL	29	70	84	114	137
FTMES@30	16	50	73	100	129
FTMES	16	49	73	97	127
SBFL	11	42	63	88	112

	Média Wef
MCBFL@50	500,77
MCBFL	391,01
FTMES@30	396,52
FTMES	440,00
SBFL	670,09

Figura 5.5: *Performance geral das técnicas em acurácia (acc@n) e esforço desperdiçado (wef).*

Resposta para PP3.2: FTMES@r produziu melhoria de eficácia de localização para SBFL em todos os cenários estudados. Além disso, FTMES@r produziu melhores valores de esforço desperdiçado (*wef*) do que MCBFL@r. Entretanto, a interação entre SFilter@r e MCBFL-hybrid-avg produziu melhores valores de *acc@n* em relação ao método FTMES@r, apesar da aproximação dos resultados.

Relação Custo-benefício

PP3.3: FTMES@r aprimora a localização de defeitos produzindo melhor relação custo-benefício (efetividade) [8]**?** Em alguns cenários, o ganho em eficácia de localização de uma técnica MBFL pode ser tão pequeno que não justifica o alto custo computacional demandado para o seu uso. Diante disso, é essencial a avaliação na perspectiva de custo-benefício. O objetivo da PP3.3 consiste em encontrar a técnica que apresenta melhor relação de custo-benefício levando em conta as diferentes possibilidades de otimizações produzidas pelas técnicas estudadas.

Por isso, além das técnicas elencadas para o estudo da PP3, as melhores foram selecionadas em interações entre SFilter@r com FTMES e MCBFL-hybrid-avg considerando: i) o valor de $n = 20$ para acurácia (acc@20) e ii) o valor de @r com melhor acurácia de localização. Assim, as seguintes interações foram selecionadas como técnicas nas análises seguintes:

1. **FTMES@30**: representando a melhor interação entre SFilter@r e FTMES com valor $@r = 30$;
2. **MCBFL@50**: representando a melhor interação entre SFilter@r e MCBFL com valor de $@r = 50$, em termos de eficácia de localização.

A exposição da Figura 5.6, a seguir, sugere a análise da Fronteira de Pareto observando os objetivos: tempo e acurácia (*acc@20*), uma avaliação similar à realizada entre técnicas multiobjetivo (YOO; HARMAN, 2010).

Considerando a Figura 5.6, uma técnica domina outra quando alcança menor tempo e maior acurácia. Portanto, quanto mais próxima de zero no eixo horizontal (tempo) e mais elevada no eixo vertical (acurácia), melhor é a técnica. Por exemplo, a técnica FTMES@30 domina as técnicas FTMES e MCBFL@50 no programa *Chart*. A ideia consiste em verificar a existência de uma relação de dominância analisando as diferentes performances das técnicas. Dependendo do contexto, algum objetivo pode ter mais importância do que outro. O ideal seria que a melhor técnica dominasse as outras técnicas em todos os objetivos. Todavia, nem sempre isso acontece em cenários reais. Apesar disso, a análise da Fronteira de Pareto também pode contribuir porque subsidia a escolha de alguma técnica específica quando o cenário (ou domínio do problema) configura alguma prioridade entre os objetivos.

Em termos gerais, a SBFL não é dominada por nenhuma outra técnica considerando o tempo. Contudo, é dominada por todas as outras na acurácia de localização. FTMES@30 domina FTMES na maioria dos cenários. Além disso, FTMES@30 domina

[8] O termo efetividade reflete o bom relacionamento entre custo e benefício de uma técnica.

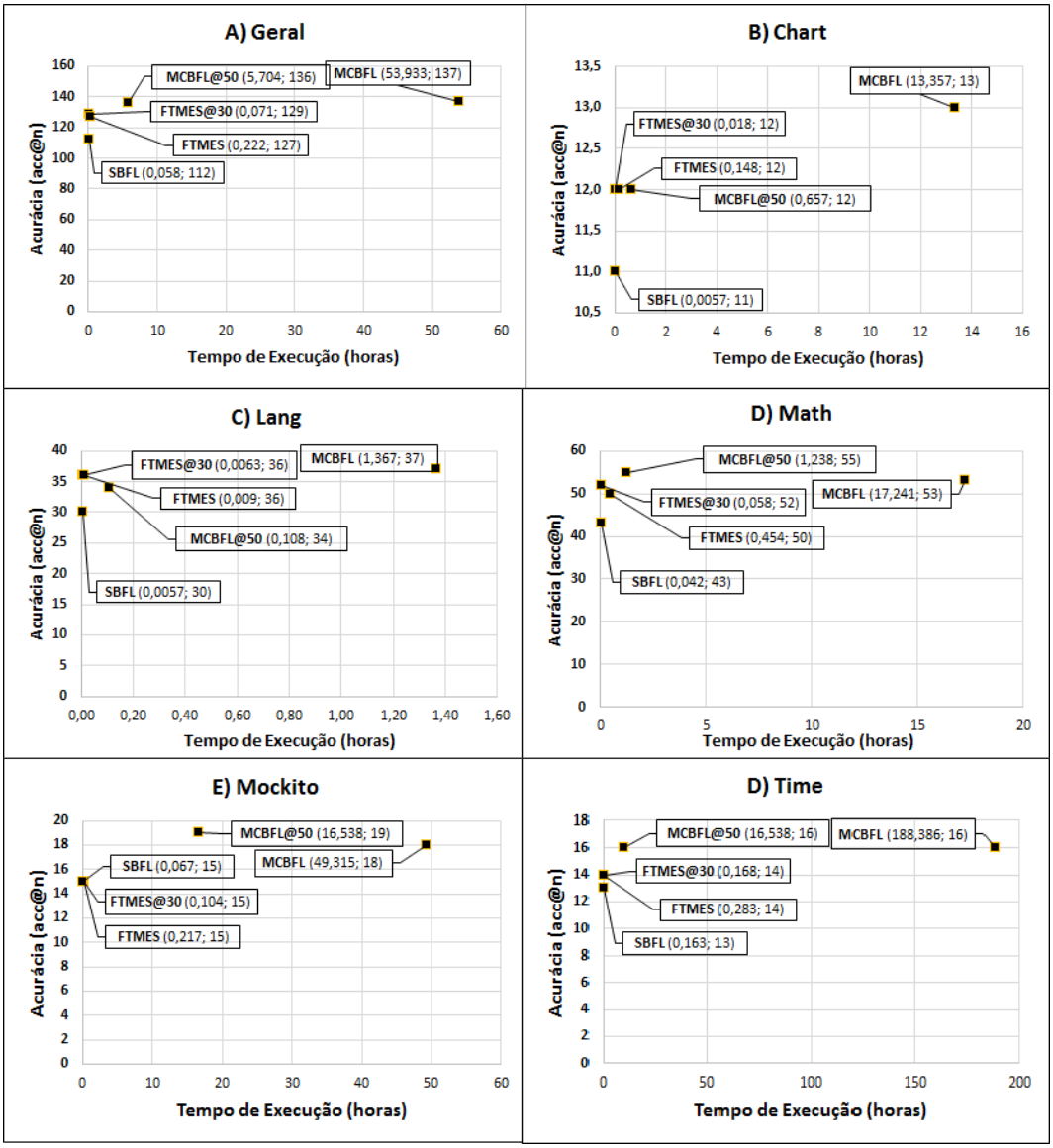


Figura 5.6: *Fronteira de Pareto: Tempo e Acurácia (acc@20).*

MCBFL@50 em dois programas: *Chart* e *Lang*. Esse resultado é motivador porque FTMES@30 é 70% e 99% menos custosa do que as técnicas FTMES e MCBFL@50, respectivamente. Isso comprova que a interação SFilter@r com FTMES não piorou a acurácia da FTMES original.

A mesma relação de dominância não é percebida entre MCBFL@50 e MCBFL. Em termos gerais, MCBFL@50 dominou MCBFL no tempo em todos os programas. No quesito acurácia, MCBFL domina MCBFL@50 nos programas *Chart* e *Lang*, e um empate acontece no programa *Time*. Esse resultado é esperado porque MCBFL representa a melhor técnica MBFL em termos de acurácia (acc@n), conforme Seção 5.6. Além disso, MCBFL considera todo o *ranking* enquanto MCBFL@50 considera apenas parte do *ranking* SBFL. Porém, um resultado interessante acontece na relação de dominância de MCBFL@50 em MCBFL, para os programas *Math* e *Mockito*.

Vale observar que a métrica *acc@n* contabiliza o primeiro elemento defeituoso até a posição *n*, tendo em vista que os defeitos (*bugs*) dos programas estudados contém múltiplas linhas (elementos), conforme explicado nas Seções 5.1.2 e 5.2. Além disso, acerca da dominância da técnica MCBFL@50 em MCBFL, é importante lembrar que a técnica MCBFL@50 é formada pela junção da técnica SFilter@50 com a técnica MCBFL. Em outras palavras, MCBFL@50 utiliza as 50 primeiras posições selecionadas pela abordagem SBFL como *ranking*.

Diante disso, MCBFL@50 domina MCBFL porque o *ranking* selecionado pela abordagem SBFL consegue posicionar melhor mais elementos defeituosos até a posição 50, embora a MCBFL posicione, em média, o primeiro elemento defeituoso antes da SBFL até a posição 20. Após a aplicação da técnica MCBFL sobre o *ranking* selecionado por SFilter@50, MCBFL@50 supera a técnica MCBFL reposicionando mais elementos defeituosos até as primeiras 20 posições (acc@20) para uma maior quantidade de versões de programa. A comprovação disso pode ser expressa pela métrica MAP (*Mean Average Precision*) que calcula o quão bem posicionados estão os todos os elementos defeituosos na comparação dos *rankings* das técnicas SBFL e MCBFL para todos os programas estudados. A Tabela 5.11 apresenta esses valores. Os programas *Math* e *Mockito* foram os únicos programas em que o valor MAP do *ranking* SBFL foi maior do que o valor MAP do ranking *MCBFL* explicando a dominância de acurácia dda técnica MCBFL@50 sobre MCBFL justamente nesses programas.

Tabela 5.11: *Valor MAP das técnicas SBFL e MCBFL-hybrid-avg (MCBFL).*

	MAP	
	MCBFL	*SBFL*
Chart	0,1858	0,1800
Lang	0,2299	0,1570
Math	0,2113	**0,2322**
Mockito	0,1823	**0,2161**
Time	0,1634	0,1163

O aprimoramento do *ranking* da técnica SBFL é notável, nos programas *Math* e *Mockito*, porque, somente após a aplicação da MCBFL aos elementos de programa selecionados por SFilter@50, a técnica MCBFL@50 passou a ter mais acurácia do que a técnica MCBFL indicando que a acurácia do *ranking* SBFL foi aprimorada. Conforme experimento da PP2 na Seção 5.6, antes da aplicação da técnica MCBFL nos elementos selecionados por SFilter@50, a técnica MCBFL possuía maior acurácia (acc@20). Ponderando-se o alto custo computacional da MCBFL original, a técnica MCBFL@50 parece ser uma alternativa melhor do a técnica MCBFL.

A Figura 5.7 sugere a mesma análise da Figura 5.6, porém fazendo uso dos objetivos: tempo e esforço desperdiçado. FTMES@30 domina FTMES em termos gerais, e MCBFL@50 em todos os programas. FTMES@30 também domina a MCBFL original em três dos cinco programas (*Lang*, *Math* e *Time*). Com base nos objetivos tempo e esforço desperdiçado, FTMES@30 demonstra-se como uma boa alternativa.

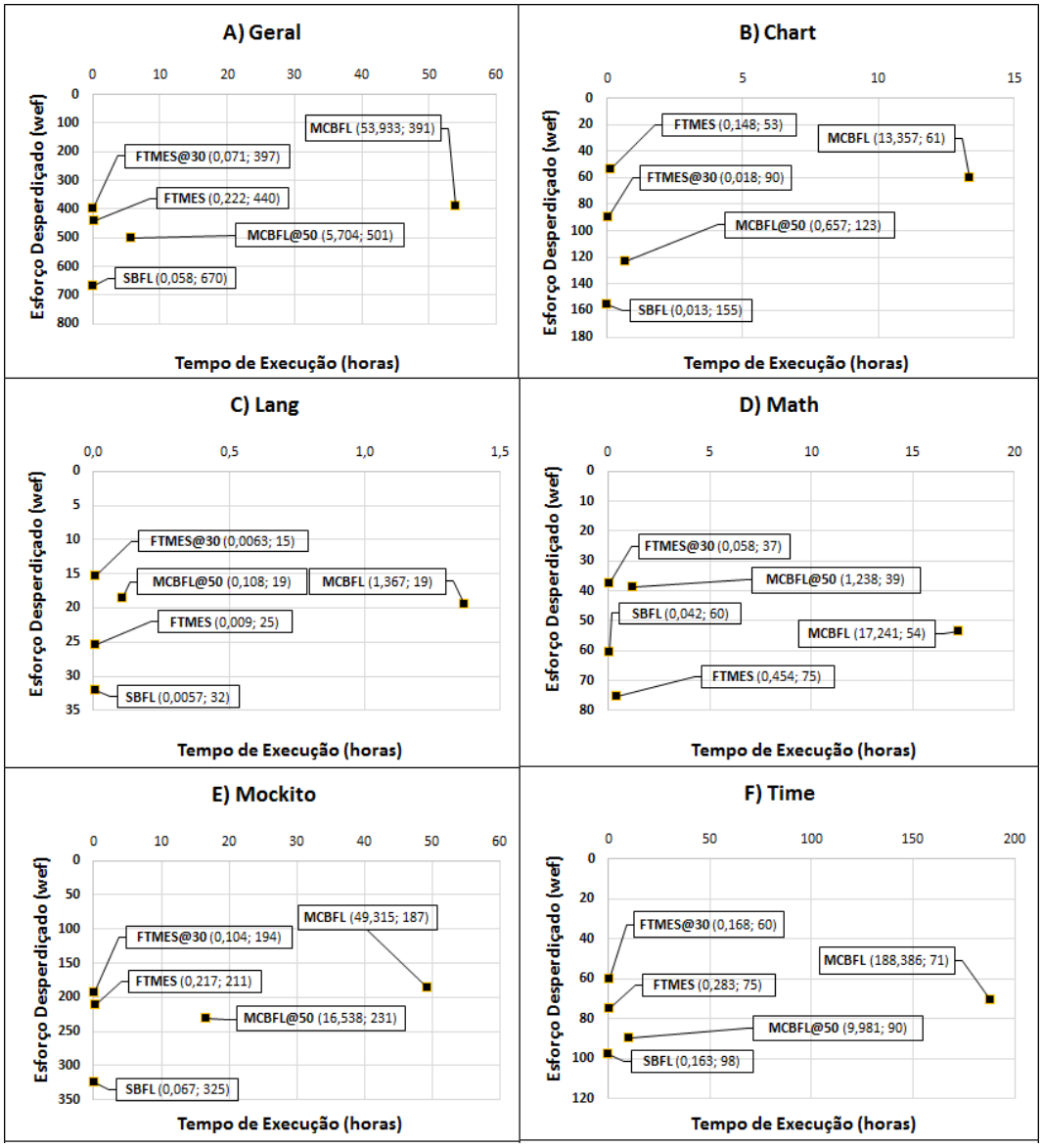


Figura 5.7: *Fronteira de Pareto: Tempo e Wef.*

FTMES@30 dominou todas as outras abordagens que executam mutantes, considerando o objetivo tempo de execução, com alta margem de diferença. FTMES@30 dominou MCBFL@50 em todos os programas observando os objetivos: tempo e Esforço Perdido (wef). Acerca da Acurácia (acc@20), FTMES@30 dominou MCBFL@50 em dois dos programas considerados. Todavia, tendo em vista apenas um objetivo, Acurácia (acc@20) ou Esforço Perdido (wef), FTMES@30 foi dominada por MCBFL com baixa margem de diferença.

A Figura 5.8 apresenta a distância de performance do método FTMES@30 em relação às outras técnicas. Na figura quanto maior o percentual melhor foi o ganho de FTMES@30 em relação à técnica de comparação para as métricas de tempo, acc@20 e wef. Valores de percentual negativos, reportam que FTMES@30 obteve um pior resultado em relação à técnica e métrica comparada.

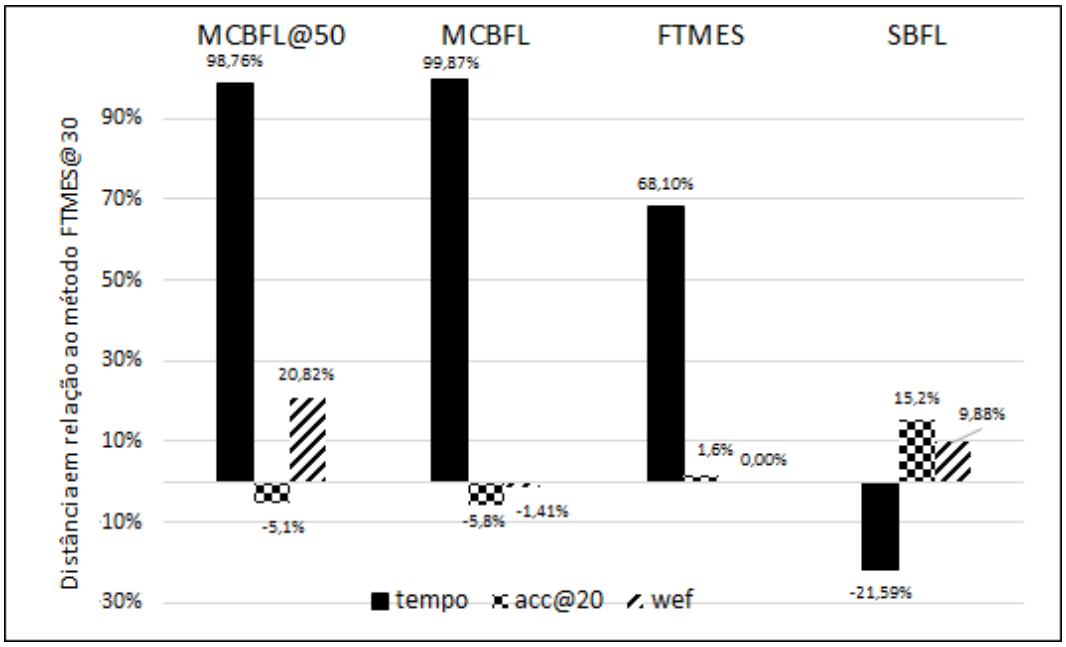


Figura 5.8: *Distância geral de performance de FTMES@30 em relação às técnicas MCBFL, MCBFL@50 e SBFL. A comparação da performance é descrita em termos de tempo, acc@20 e wef.*

A Figura 5.8 ajuda em uma visualização numérica do quanto FTMES@30 ou domina ou perde para alguma das técnicas os objetivos de tempo, acc@20 e wef. A escolha de uma técnica depende da prioridade dada a cada objetivo quando a relação de dominância não acontece sobre todos os objetivos. Em cenários reais, podem existir diversos contextos. Por exemplo, a acurácia deve ser priorizada ou o esforço desperdiçado pode ser considerado um critério mais importante. Além das priorizações por benefício, existem as priorizações por custo, em que o tempo possui maior prioridade.

Diante do exposto, nesta Tese foi defendido que uma técnica MBFL deve ser mais eficaz na localização e se aproximar do tempo de uma técnica SBFL quando empregada em conjunto com alguma estratégia de redução de custo. Todavia, quando uma técnica MBFL não domina outra em todos os objetivos, a melhor técnica vai depender da prioridade que se pretende dar para o benefício, seja para a acurácia ou para o esforço desperdiçado e as suas respectivas relações com o tempo.

Nesse sentido, as Equações 5-4 e 5-5 representam percentuais de priorização para subsidiar uma escolha baseada na importância do custo e dos diferentes benefícios

pretendidos. O valor α consiste na taxa de priorização para aproximação de custo da MBFL ao tempo de execução da SBFL. Por sua vez, o valor ω corresponde à taxa de priorização para melhoria do benefício, ora medido pela acurácia, ora medido pelo esforço desperdiçado. As Equações podem ser aplicadas em cenários com diferentes prioridades, por exemplo, algum cenário pode estabelecer pesos diferentes para custo e acurácia (ex.: $\alpha = 30\%$ e $\omega = 70\%$) e os mesmos pesos para custo e esforço desperdiçado (ex.: $\alpha = 50\%$ e $\omega = 50\%$).

$$p_{acc} = \alpha * custo_{aprox} + \omega * acc@n_{melhoria} \tag{5-4}$$

$$p_{wef} = \alpha * custo_{aprox} + \omega * wef_{melhoria} \tag{5-5}$$

A Figura 5.9 sugere a análise da Fronteira de Pareto sobre esses dois percentuais como objetivos: i) o percentual p_{acc} e ii) o percentual p_{wef}. A Figura mostra o desempenho das técnicas considerando diferentes pesos para priorização do custo-benefício em cada percentual: i) custo: $\alpha = 50\%, 30\%, 20\%, 10\%$) e ii) benefício: $\alpha = 50\%, 70\%, 80\%, 90\%$. Nos gráficos, quanto maior os valores percentuais sobre os dois eixos, melhor é a técnica, com base nos dois objetivos.

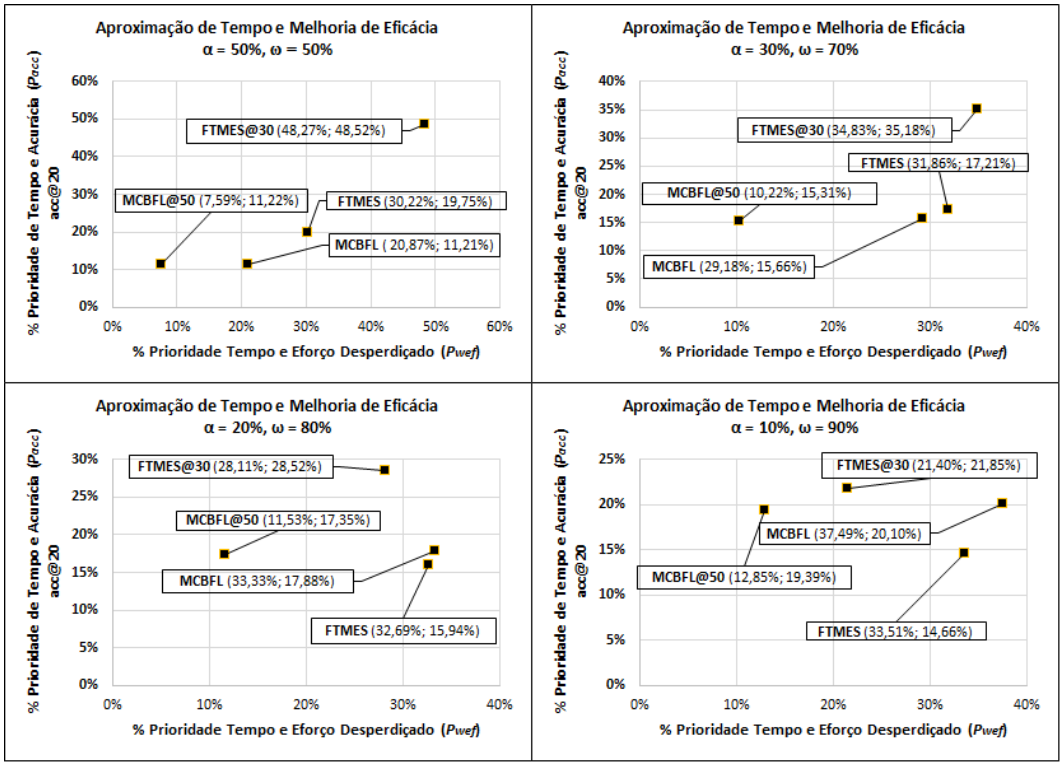


Figura 5.9: *Fronteira de Pareto com Pesos para Priorizar Custo e Benefício.*

No cenário que requer prioridades iguais para custo e benfício ($\alpha = 50\%$ e $\omega = 50\%$), FTMES@30 domina todas as técnicas MBFL com uma distância proeminente no gráfico 5.9. Similarmente, o componente FTMES domina MCBFL e MCBFL@50 nos dois objetivos também. Essa mesma conclusão vale para $\alpha = 30\%$ e $\omega = 70\%$.

No contexto em que $\alpha = 20\%$ e $\omega = 80\%$, FTMES@30 domina somente a técnica MCBFL@50 nos dois objetivos. MCBFL domina FTMES nos dois objetivos também, mas não domina FTMES@30. Tal conclusão, vale para o cenário $\alpha = 10\%$ e $\omega = 90\%$.

Contudo, para todos cenários cuja a importância do custo computacional é maior do que o benefício ($\alpha > \omega$), a técnica FTMES@r é mais indicada do que a técnica MCBFL. FTMES@r continua sendo mais indicada até $\alpha = 30\%$ $\omega = 70\%$, coforme Figura 5.9.

Resposta para PP3.3: FTMES@30 apresenta a melhor relação de custo-benefício dentre as técnicas que executam mutantes quando a mesma prioridade é estabelecida para tempo e eficácia de localização. FTMES@30 demonstrou uma melhor relação custo-benefício mesmo reduzindo a prioridade de custo para 30% e aumentando a prioridade do benefício para 70%. Assim, FTMES@r revela-se como a técnica com melhor relação de custo-benefício em termos de aproximação do tempo de execução e aprimoramento da eficácia em relação ao desempenho da técnica SBFL.

5.8 Ameaças à Validade

A qualidade da abordagem FTMES@r está relacionada à eficácia da técnica SBFL. Ademais, se o valor @r fornecido não for suficiente para ranquear o defeito entre os elementos de programas selecionados (Ranking@r), o método FTMES@r não tem poder de melhoria. Nesse sentido, se existir a necessidade de um alto valor para @r, a diferença na performance entre FTMES@r e FTMES pode ser insignificante.

Além disso, a abordagem proposta usa o componente FTMES para aprimorar o custo de execução das técnicas MBFL. A qualidade da abordagem proposta está ligada à qualidade do conjunto de teste, similarmente a muitas técnicas de teste e *debugging*. O método proposto apresentou boas soluções quando o tamanho do conjunto dos casos de teste que falham é menor do que o tamanho do conjunto dos casos de teste que passam. Essa característica é percebida na maioria dos cenários reais e *benchmarks* da área. Apesar de incomum, podem haver cenários em que tal característica não acontece.

Para o desenvolvimento da pesquisa, todas as técnicas estudadas foram implementadas e experimentadas em defeitos reais, seguindo os artigos seminais de cada pro-

posta, conforme Seção 5.3. Uma potencial ameaça interna, consiste na validade da implementação realizada. As implementações das técnicas Dstar (B1), MRSBFL-hybrid-max (B2), Metallaxis (B3), MCBFL-hybrid-avg (B4), foram validadas em termos de eficácia e eficiência mediante a implementação de Pearson et al. (2017) que disponibilizam o código fonte das técnicas e os resultados pelo seguinte link: "https://bitbucket.org/rjust/fault-localization-data/src/master/". Os resultados apresentados nas Seções 5.5, 5.6 e 5.7, entre a implementação da Tese com a implementação de Pearson et al. (2017), foram comparados e confirmados. Assim, foi mitigada a ameaça da validade das implementações de Dstar (B1), MRSBFL-hybrid-max (B2), Metallaxis (B3), MCBFL-hybrid-avg (B4).

Gong, Zhao e Li (2015) não disponibilizaram o código fonte executável da estratégia de execução de mutantes DMES (B5) (GONG; ZHAO; LI, 2015). Por esse motivo, DMES (B5) foi implementada seguindo o pseudocódigo disponibilizado no artigo (GONG; ZHAO; LI, 2015). Apesar desse pseudocódigo estar muito bem descrito e detalhado, ainda existe a ameaça da corretude da implementação de DMES (B5) realizada na presente Tese. Para amenizar essa ameaça, uma validação da implementação de DMES (B5) com a implementação de Gong, Zhao e Li (2015) foi realizada comparando os resultados de eficiência e eficácia (Seções 5.5 e 5.6) com os resultados da implementação de DMES (B5) publicados no artigo (GONG; ZHAO; LI, 2015). Na Seção VI desse artigo, Gong, Zhao e Li (2015) afirmam que DMES (B5) é capaz de reduzir o custo da MBFL tradicional (B3) em até 87%. Conforme Seção 5.5 da presente Tese, esse resultado foi confirmado, uma vez que DMES (B5) conseguiu reduzir 85% das quantidade de execuções MTPs em relação a MBFL tradicional (B3), em média para todos os programas estudados (Tabela 5.5). Porém, a Seção 5.6 descreve perda de eficácia de localização por DMES (B5). Esse resultado contrasta com a publicação de Gong, Zhao e Li (2015) que afirmam manter a eficácia de localização.

Nos resultados descritos na Seção 5.6 da presente Tese, foi observada perda de eficácia por todas as estratégias de execução, inclusive pelo método proposto nesta Tese (Seção 4.2.1), mesmo que em menor proporção e apresentando uma menor significância estatística do que DMES (B5). Essa diferença de eficácia entre os resultados encontrados na presente Tese com os resultados de eficácia publicados por Gong, Zhao e Li (2015), podem ter acontecido por outros dois fatores externos à implementação da presente pesquisa: i) linguagem de programação e ii) classe de defeitos.

Acerca da linguagem de programação, DMES (B5) foi avaliada por Gong, Zhao e Li (2015) em programas escritos em linguagem C, diferentemente da avaliação da presente Tese que considera programas escritos em linguagem Java. Por esse motivo, os operadores de mutação considerados por Gong, Zhao e Li (2015) são diferentes dos operadores de mutação considerados na presente Tese. Ao todo, Gong, Zhao e Li (2015) consideraram os 77 operadores de mutação oferecidos pela ferramenta de

mutação Proteum (DELAMARO; MALDONADO, 2001) específica para linguagem C. A presente Tese, considerou os 8 operadores de mutação da ferramenta Major (JUST, 2014) específicos para a linguagem Java. Como trabalhos futuros, pretendemos avaliar o método proposto em programas escritos em linguagem C (conforme Capítulo 6) para avaliar se o componente FTMES também consegue ser melhor do que DMES (B5) em termos de eficácia e eficiência.

Acerca da classe de defeitos, DMES (B5) foi avaliada por Gong, Zhao e Li (2015) nos programas da Siemens e Space que contém defeitos artificiais (DO; ELBAUM; ROTHERMEL, 2005). A presente Tese, avaliou DMES (B5) considerando os programas do benchmark Defects4J (JUST; JALALI; ERNST, 2014) que contém defeitos reais. Dessa forma, as diferentes classes de defeitos podem ter influenciado na eficácia de localização demonstrada por DMES (B5). Nesse mesmo contexto, um dos principais resultados de Pearson et al. (2017) que corroboram com essa possibilidade, foi a diferença na eficácia de localização quando defeitos artificiais e reais são considerados para avaliação de técnicas de localização de defeitos. Assim, como trabalhos futuros pretende-se avaliar o componente FTMES em programas contendo defeitos artificiais em comparação com DMES (B5) para constatar se a conclusão de Pearson et al. (2017) também se aplica no contexto da presente pesquisa.

CAPÍTULO 6

Conclusão

Desenvolvedores lidam constantemente com falta de tempo e de recursos no processo de desenvolvimento de software. Esse cenário dificulta a escolha e a aplicação das técnicas de localização de defeitos, uma vez que eficácia e eficiência são requisitos que nem sempre caminham juntos. Conforme apresentado nas Seções 1.1 e 3.1 dos Capítulos de Introdução e Trabalhos Correlatos, as técnicas de Localização de Defeitos Baseada em Mutação (MBFL) possuem maior eficácia de localização, porém, menor eficiência do que as abordagens de Localização de Defeitos Baseada no Espectro do Programa (SBFL). Os estudos existentes sobre redução de custo da MBFL ou se limitam a aplicações de técnicas de redução da Análise de Mutantes, ou exploram apenas a etapa de execução dos mutantes como possibilidade de otimização.

Nesta Tese, o método FTMES@r foi proposto para aprimorar a localização de defeitos explorando a eficiência das técnicas SBFL e a eficácia de localização da abordagem MBFL. Diferindo-se de todas as técnicas propostas em apenas uma das etapas do processo de localização, o mecanismo de FTMES@r explora duas etapas de redução de custo: i) na seleção dos elementos de programa (SFilter@r) e ii) na otimização da execução dos mutantes (FTMES). SFilter@r seleciona os elementos de programa em um ranking menor de tamanho @r. Esse ranking é posteriormente aprimorado pela aplicação da MBFL, que trabalha eficientemente na execução dos mutantes por meio do componente FTMES (veja Capítulo 4).

A avaliação da abordagem proposta compreende um conjunto de experimentos que consideram 221 defeitos reais, 10 técnicas de localização de defeitos e 5 métricas para avaliação das soluções. A experimentação foi realizada visando responder às seguintes perguntas de pesquisa:

(PP1) - FTMES supera a eficiência das estratégias de execução de mutantes existentes?

(PP2) - FTMES mantém a eficácia de localização das técnicas MBFL apresentando maior eficiência?

(PP3) - FTMES@r melhora a performance da localização de defeitos produzindo:

- **(PP3.1)** - menor custo computacional em relação às técnicas MBFL?

(PP3.2) - maior eficácia?

(PP3.3) - melhor relação custo-benefício (efetividade) entre as técnicas de localização de defeitos?

Para responder PP1 e PP2, inicialmente o componente FTMES foi avaliado considerando: i) duas técnicas que não executam mutantes (Dstar representando a SBFL e MRSBFL-hybrid-max); ii) a técnica Metallaxis retratando a MBFL tradicional; iii) a técnica MCBFL-hybrid-avg simbolizando a técnica MBFL com maior eficácia de localização encontrada, segundo Pearson et al. (2017) e ; iv) as estratégias de execução de mutantes (SAM, DMES e DMES_SAM). A aplicação de FTMES foi comparada com as outras técnicas em termos de eficácia e eficiência. Conclui-se que FTMES obteve melhores resultados em todos os programas estudados, sendo posicionada na literatura como a atual melhor técnica de execução de mutantes.

Para responder a PP3, as seguintes técnicas foram consideradas: i) Dstar, por ser a melhor técnica da abordagem SBFL; ii) a interação de SFilter@r com a melhor técnica MBFL (MCBFL-hybrid-avg) e; iii) FTMES@r, representa a abordagem proposta sugerindo a interação de SFilter@r com o componente FTMES (veja Capítulo 4). A resposta da PP3 foi alcançada em três perspectivas: PP3.1) redução de custo da MBFL; PP3.2) melhoria de eficácia em relação a SBFL e; PP3.3) relação custo-benefício.

O método FTMES@r realizou uma maior redução de custo revelando a vantagem de se utilizar uma técnica MBFL otimizada em duas etapas da localização de defeitos. FTMES@r produziu melhoria de eficácia de localização para SBFL em todos os cenários estudados. Entretanto, a interação entre SFilter@r e MCBFL-hybrid-avg produziu melhores valores de *acc@n* e *wef*, apesar de uma aproximação considerável de FTMES@r. Todavia, FTMES@r foi considerada como a técnica de melhor custo-benefício em termos de aproximação do tempo de execução e aprimoramento da eficácia em relação ao desempenho da técnica SBFL.

Além da proposta de um novo método de localização de mutantes que considera elementos importantes das técnicas SBFL e MBFL, outras contribuições e produções geradas durante o estudo realizado:

1. A nova estratégia de execução de mutantes orientada à execução de casos de teste que falham para tornar a aplicação da MBFL mais eficiente;
2. Uma proposta de localização de defeitos com redução de custo da MBFL nos níveis de seleção dos elementos de programa e na execução de programas mutantes;
3. Uma avaliação de eficiência e eficácia de diferentes classes de técnicas de localização e estratégias de redução de custo considerando defeitos reais;

Para trabalhos futuros, pode ser realizado um estudo de caso com maiores informações na quantidade de técnicas SBFL, em outros *benchmarks* que representem

defeitos reais, em outras linguagens de programação, visando confirmar os resultados e as limitações do método proposto. Além disso, motivados pelas possibilidades de redução de custo, existe o interesse em investigar a interação do método em diferentes dimensões da Análise de Mutantes, como, por exemplo, no contexto de Mutação Fraca (*Weak Mutation*) (HOWDEN, 1982; OFFUTT; LEE, 1994) e na Mutação de Alta Ordem (*High Order Mutation*) (JIA; HARMAN, 2009). Ainda não existe nenhum trabalho que propõe localização de defeitos em Mutação Fraca e Mutação de Alta Ordem. Métodos de localização de defeitos aplicados a tais dimensões seriam inovadores. Investigar FTMES@r nesses contextos seria, de fato, uma contribuição singular.

Por fim, pode ser realizada a investigação da aplicação de algoritmos evolucionários no problema multiobjetivo das estratégias de execução de mutantes considerando: i) a seleção de técnicas SBFL baseada em métricas e diferentes tamanhos de ranking; ii) a seleção de operadores de mutação; e iii) a minimização de casos de teste.

Referências Bibliográficas

ABREU, R.; ZOETEWEIJ, P.; GEMUND, A. J. c. V. An evaluation of similarity coefficients for software fault localization. In: *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. [S.l.: s.n.], 2006. p. 39–46.

ABREU, R.; ZOETEWEIJ, P.; GEMUND, A. J. C. v. Spectrum-based multiple fault localization. In: *2009 IEEE/ACM International Conference on Automated Software Engineering*. [S.l.: s.n.], 2009. p. 88–99. ISSN 1938-4300.

ABREU, R.; ZOETEWEIJ, P.; GEMUND, A. J. C. van. On the accuracy of spectrum-based fault localization. In: *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*. Washington, DC, USA: IEEE Computer Society, 2007. (TAICPART-MUTATION '07), p. 89–98. ISBN 0-7695-2984-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1308173.1308264>.

ACREE, A. T. *On Mutation*. Tese (Doutorado) — Georgia Institute of Technology, 1980.

ADRAGNA, P. Software debugging techniques. *Queen Mary University of London*, 2010.

ARAKI, L. *Um algoritmo evolutivo de geração de dados de teste para satisfazer critérios baseados em códigos objeto Java*. 102 p. Tese (Doutorado) — Universidade Federal do Paraná, 2009.

BANZI, A. S. et al. Selecting mutation operators with a multiobjective approach. *Expert Syst. Appl.*, Pergamon Press, Inc., Tarrytown, NY, USA, v. 39, n. 15, p. 12131–12142, nov. 2012. ISSN 0957-4174. Disponível em: <http://dx.doi.org/10.1016/j.eswa.2012.04.041>.

BORGES, W. S. et al. Um algoritmo genético no modelo de ilhas para seleção de casos de teste na análise de mutantes. *Worshop de Engenharia de Software baseada em Busca (WESB)*, Congresso Brasileiro de Software Teoria e Prática (CBSoft), 2014. Disponível em: <http://www.ic.ufal.br/evento/cbsoft2014/anais/wesb_v1_p.pdf>.

BUDD, T. A. *Mutation Analysis of Program Test Data*. Tese (Doutorado) — Yale University, 1980.

BUDD, T. A. Mutation Analysis: Ideas, Examples, Problems and Prospects. In: *Proceedings of the Summer School on Computer Program Testing*. Sogesta: [s.n.], 1981. p. 129–148.

CAMPOS, E. C.; MAIA, M. d. A. Common bug-fix patterns: A large-scale observational study. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. [S.l.: s.n.], 2017. p. 404–413.

CAMPOS, J. et al. Gzoltar: an eclipse plug-in for testing and debugging. In: *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. [S.l.: s.n.], 2012. p. 378–381.

DEBROY, V.; WONG, W. E. Combining mutation and fault localization for automated program debugging. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 90, p. 45–60, abr. 2014. ISSN 0164-1212.

DELAMARO, M. E.; JINO, M.; MALDONADO, J. C. *Introdução ao Teste de Software*. [S.l.: s.n.], 2007. 408 p.

DELAMARO, M. E.; MALDONADO, J. C. Mutation testing for the new century. In: WONG, W. E. (Ed.). Norwell, MA, USA: Kluwer Academic Publishers, 2001. cap. Proteum/IM 2.0: An Integrated Mutation Testing Environment, p. 91–101. ISBN 0-7923-7323-5.

DEMILLI, R. A.; OFFUTT, A. J. Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering*, v. 17, n. 9, p. 900–910, Sep. 1991. ISSN 0098-5589.

DEMILLO, R.; LIPTON, R.; SAYWARD, F. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer*, v. 11, n. 4, p. 34–41, abr. 1978. ISSN 0018-9162.

DO, H.; ELBAUM, S. G.; ROTHERMEL, G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering: An International Journal*, v. 10, n. 4, p. 405–435, 2005.

FERRARI, F. C. *A contribution to the fault-based testing of aspect-oriented software*. 228 p. Tese (Doutorado) — Universidade de São Paulo, 2010.

FERRARI, F. C.; PIZZOLETO, A. V.; OFFUTT, J. A systematic review of cost reduction techniques for mutation testing: Preliminary results. In: *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. [S.l.: s.n.], 2018. p. 1–10.

FICICI, S. G.; BUCCI, A. *Advanced tutorial on coevolution*. New York, New York, USA: ACM Press, 2007. 3172 p. ISBN 9781595936981. Disponível em: <http://portal.acm.org-/citation.cfm?doid=1274000.1274110>.

GONG, P.; ZHAO, R.; LI, Z. Faster mutation-based fault localization with a novel mutation execution strategy. In: *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. [S.l.: s.n.], 2015. p. 1–10.

GUIDETTI, S. A. *Aplicação de Análise de Mutantes à Geração de Dados de Teste para Detecção de Vulnerabilidade do Tipo Buffer Overflow*. 131 p. Tese (Doutorado) — Universidade Estadual de Campinas, 2005.

HOWDEN, W. E. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, SE-8, n. 4, p. 371–379, July 1982. ISSN 0098-5589.

HUSSAIN, S. *Mutation Clustering*. 41 p. Tese (Doutorado) — King's College London, 2008.

IEEE. Iso/iec/ieee international standard - systems and software engineering – vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, p. 1–418, Dec 2010.

JIA, Y.; HARMAN, M. Constructing Subtle Faults Using Higher Order Mutation Testing. *2008 Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, Ieee, p. 249–258, set. 2008. Disponível em: <http://ieeexplore.ieee.org-/lpdocs/epic03/wrapper.htm?arnumber=4637557>.

JIA, Y.; HARMAN, M. Higher order mutation testing. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 51, n. 10, p. 1379–1393, out. 2009. ISSN 0950-5849. Disponível em: <http://dx.doi.org/10.1016/j.infsof.2009.04.016>.

JIA, Y.; HARMAN, M. An analysis and survey of the development of mutation testing. *Software Engineering, IEEE Transactions on*, v. 37, p. 649 – 678, 2011.

JIA, Y.; HARMAN, M. An analysis and survey of the development of mutation testing. *Software Engineering, IEEE Transactions on*, v. 37, n. 5, p. 649 –678, sept.-oct. 2011. ISSN 0098-5589.

JONES, J. A.; HARROLD, M. J. Empirical evaluation of the tarantula automatic fault-localization technique. In: *in Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. [S.l.: s.n.], 2005. p. 273–282.

JUNIOR, P. S. L. *Teste baseado na interação entre regras ativas escritas em SQL*. Tese (Doutorado) — Universidade Estadual de Campinas (UNICAMP), 2005.

JUST, R. The major mutation framework: Efficient and scalable mutation analysis for java. In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2014. (ISSTA 2014), p. 433–436. ISBN 978-1-4503-2645-2. Disponível em: <http://doi.acm.org/10.1145/2610384.2628053>.

JUST, R.; JALALI, D.; ERNST, M. D. Defects4J: A Database of existing faults to enable controlled testing studies for Java programs. In: *ISSTA 2014, Proceedings of the 2014 International Symposium on Software Testing and Analysis*. San Jose, CA, USA: [s.n.], 2014. p. 437–440. Tool demo.

JUST, R.; SCHWEIGGERT, F. Higher accuracy and lower run time: efficient mutation analysis using non-redundant mutation operators. *Software Testing, Verification and Reliability*, v. 25, n. 5-7, p. 490–507, 2015. Disponível em: <https://onlinelibrary.wiley-.com/doi/abs/10.1002/stvr.1561>.

KRAUSER, W. et al. High Performance Software Testing on SIMD Machines. v. 17, n. 5, 1991.

LAENEN, F. V. Mutation testing. *ACCU magazine Overload*, ACCU, v. 104, abr. 2012. ISSN 1354-3172.

LIU, Y. et al. Statement-oriented mutant reduction strategy for mutation based fault localization. In: *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. [S.l.: s.n.], 2017. p. 126–137.

LIU, Y. et al. An optimal mutation execution strategy for cost reduction of mutation-based fault localization. *Inf. Sci.*, Elsevier Science Inc., New York, NY, USA, v. 422, n. C, p. 572–596, jan. 2018. ISSN 0020-0255. Disponível em: <https://doi.org/10.1016/j.ins.2017-.09.006>.

LUIZ, A. M. et al. Uso de algoritmo genético distribuído na seleção de casos de teste para o teste de mutação. *Worshop de Engenharia de Software baseada em Busca (WESB)*, Congresso Brasileiro de Software Teoria e Prática (CBSoft), 2014. Disponível em: <http://www.ic.ufal.br/evento/cbsoft2014/anais/wesb_v1_p.pdf>.

MA, Y.-S.; OFFUTT, J.; KWON, Y. R. MuJava: an automated class mutation system. *Software Testing, Verification and Reliability*, v. 15, n. 2, p. 97–133, jun. 2005. ISSN 0960-0833.

MACHADO, B. N. et al. Sbstframe: uma proposta de framework para o teste de software baseado em busca. *Worshop de Engenharia de Software baseada em Busca (WESB)*, Congresso Brasileiro de Software Teoria e Prática (CBSoft), 2014. Disponível em: <http://www.ic.ufal.br/evento/cbsoft2014/anais/wesb_v1_p.pdf>.

MARTINS, B. P. et al. Uso de algoritmo genético distribuído na seleção de casos de teste para o teste de mutação. *Encontro Anual de Tecnologia da Informação*, 2014. Disponível em: <http://www.eati.info/eati/2014/assets/anais/artigo19.pdf>.

MATHUR, A. Performance, effectiveness, and reliability issues in software testing. *Computer Software and Applications Conference, ...*, p. 0–1, 1991. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=170248>.

MOON, S. et al. Ask the mutants: Mutating faulty programs for fault localization. In: *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation*. Washington, DC, USA: IEEE Computer Society, 2014. (ICST '14), p. 153–162. ISBN 978-1-4799-2255-0.

MOTWANI, M. et al. Do automated program repair techniques repair hard and important bugs? *Empirical Software Engineering*, v. 23, n. 5, p. 2901–2947, Oct 2018. ISSN 1573-7616. Disponível em: <https://doi.org/10.1007/s10664-017-9550-0>.

MYERS, G.; SANDLER, C. *The Art of Software Testing*. [S.l.]: John Wiley & Sons, Inc., 2004. ISBN 0471469122.

MYERS, G. J. *The Art of Software Testing*. [S.l.]: Wiley, New York, 1979.

NAISH, L.; LEE, H. J.; RAMAMOHANARAO, K. A model for spectra-based software diagnosis. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 20, n. 3, p. 11:1–11:32, ago. 2011. ISSN 1049-331X. Disponível em: <http://doi.acm.org/10.1145-/2000791.2000795>.

NAMIN, A. S.; ANDREWS, J.; MURDOCH, D. Sufficient mutation operators for measuring test effectiveness. In: *2008 ACM/IEEE 30th International Conference on Software Engineering*. [S.l.: s.n.], 2008. p. 351–360. ISSN 0270-5257.

NAMIN, A. S.; ANDREWS, J. H. Finding sufficient mutation operators via variable reduction. In: *Second Workshop on Mutation Analysis (Mutation 2006 - ISSRE Workshops 2006)*. [S.l.: s.n.], 2006. p. 5–5.

NETO, E. A. R. et al. Avaliação de técnicas para redução de base de dados de produção. *Encontro Anual de Tecnologia da Informação*, 2014. Disponível em: <http://www.eati.info/eati/2014/assets/anais/artigo18.pdf>.

NOBRE, T. *Uma Abordagem Baseada em Algoritmos de Otimização Multiobjetivos para Reduzir o Custo do Critério de Teste Análise de Mutantes*. 66 p. Tese (Doutorado) — Universidade Federal do Paraná, 2011.

OFFUTT, A. J.; LEE, S. D. An empirical evaluation of weak mutation. *IEEE Transactions on Software Engineering*, v. 20, n. 5, p. 337–344, May 1994. ISSN 0098-5589.

OLIVEIRA, A. A. L. d. et al. Uso de algoritmo genético distribuído na seleção de casos de teste para o teste de mutação. *Encontro Anual de Tecnologia da Informação*, 2014. Disponível em: <http://www.eati.info/eati/2014/assets/anais/artigo25.pdf>.

OLIVEIRA, A. A. L. de et al. Ftmes: A failed-test-oriented mutant execution strategy for mutation-based fault localization. In: *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. [S.l.: s.n.], 2018.

OLIVEIRA, A. A. L. de et al. Ftmes@r: A mutation execution strategy method to improve the top@r positions of spectrum-based fault localization techniques. *Software Quality Journal*, 2018.

OLIVEIRA, A. A. L. de et al. Ftscmes: A new mutation execution strategy based on failed tests' mutation score for fault localization. In: CZACHÓRSKI, T. et al. (Ed.). *Computer and Information Sciences*. Cham: Springer International Publishing, 2018. p. 177–187. ISBN 978-3-030-00840-6.

PAPADAKIS, M. et al. Mutation testing advances: An analysis and survey. *Advances in Computers*, 2018.

PAPADAKIS, M.; TRAON, Y. L. Using mutants to locate "unknown"faults. In: *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. [S.l.: s.n.], 2012. p. 691–700.

PAPADAKIS, M.; TRAON, Y. L. Effective fault localization via mutation analysis: A selective mutation approach. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2014. (SAC '14), p. 1293–1300. ISBN 978-1-4503-2469-4.

PAPADAKIS, M.; TRAON, Y. L. Metallaxis-fl: Mutation-based fault localization. *Softw. Test. Verif. Reliab.*, John Wiley and Sons Ltd., Chichester, UK, v. 25, n. 5-7, p. 605–628, ago. 2015. ISSN 0960-0833.

PARNIN, C.; ORSO, A. Are automated debugging techniques actually helping programmers? In: *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2011. (ISSTA '11), p. 199–209. ISBN 978-1-4503-0562-4. Disponível em: <http://doi.acm.org/10.1145/2001420.2001445>.

PEARSON, S. Evaluating and improving fault localization techniques. In: *Technical report UW-CSE-16-08-03, August 2016*. [S.l.: s.n.], 2016.

PEARSON, S. et al. Evaluating and improving fault localization. In: *Proceedings of the 39th International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2017. (ICSE '17), p. 609–620. ISBN 978-1-5386-3868-2. Disponível em: <https://doi.org/10.1109/ICSE.2017.62>.

PRADO, M. P. *Contribuições ao Suporte Cognitivo em Teste de Software Unitário: Um Framework de Tarefas e uma Agenda de Pesquisa*. 155 p. Tese (Doutorado) — Universidade Federal de Goiás, 2018.

PRESSMAN, R. S. *Engenharia de Software*. 6. ed. São Paulo: Makron Books do Brasil, 2006.

Ré, R. *Uma contribuição para a minimização do número de stubs no teste de integração de programas orientados a aspectos*. 172 p. Tese (Doutorado) — Universidade de São Paulo, 2009.

SANTELICES, R. et al. Lightweight fault-localization using multiple coverage types. In: *2009 IEEE 31st International Conference on Software Engineering*. [S.l.: s.n.], 2009. p. 56–66. ISSN 0270-5257.

SINGH, B.; GAUTAM, S. The impact of software development process on software quality: A review. In: . [S.l.: s.n.], 2016. p. 666–672.

SOHN, J.; YOO, S. Fluccs: Using code and change metrics to improve fault localization. In: *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2017. (ISSTA 2017), p. 273–283. ISBN 978-1-4503-5076-1. Disponível em: <http://doi.acm.org/10.1145/3092703.3092717>.

SRIVASTVA, S.; DHIR, S. Debugging approaches on various software processing levels. In: *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*. [S.l.: s.n.], 2017. v. 2, p. 302–306.

STEIMANN, F.; FRENKEL, M.; ABREU, R. Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators. In: *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2013. (ISSTA 2013), p. 314–324. ISBN 978-1-4503-2159-4.

TASSEY, G. The economic impacts of inadequate infrastructure for software testing. *National Institute ofStandards and Technolog*, maio 2002.

TRICENTS. Software fail watch: 5th edition. 2018. Disponível em: <https://www.tricentis-.com/wp-content/uploads/2018/01/20180119_Software-Fails-Watch_Small_Web.pdf>.

VINCENZI, A. M. R. *Subsídios para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação*. 138 p. Tese (Doutorado) — Universidade de São Paulo, 1998.

WONG, E. et al. A crosstab-based statistical method for effective fault localization. In: *2008 1st International Conference on Software Testing, Verification, and Validation*. [S.l.: s.n.], 2008. p. 42–51. ISSN 2159-4848.

WONG, W. E.; DEBROY, V.; CHOI, B. A family of code coverage-based heuristics for effective fault localization. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 83, n. 2, p. 188–208, fev. 2010. ISSN 0164-1212.

WONG, W. E. et al. The dstar method for effective software fault localization. *IEEE Transactions on Reliability*, v. 63, n. 1, p. 290–308, March 2014. ISSN 0018-9529.

WONG, W. E. et al. Software fault localization using dstar (d*). In: *2012 IEEE Sixth International Conference on Software Security and Reliability*. [S.l.: s.n.], 2012. p. 21–30.

WONG, W. E. et al. A survey on software fault localization. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 42, n. 8, p. 707–740, ago. 2016. ISSN 0098-5589.

YOO, S.; HARMAN, M. Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 83, n. 4, p. 689–701, abr. 2010. ISSN 0164-1212.

YU, K. et al. Locating faults using multiple spectra-specific models. In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2011. (SAC '11), p. 1404–1410. ISBN 978-1-4503-0113-8. Disponível em: <http://doi.acm.org/10-.1145/1982185.1982490>.

ZHANG, J. et al. Predictive mutation testing. In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2016. (ISSTA 2016), p. 342–353. ISBN 978-1-4503-4390-9. Disponível em: <http://doi.acm.org/10-.1145/2931037.2931038>.