Anais

Volume 02
ISSN 2178-6097

WTDSoft 2014

IV WORKSHOP DE TESES E DISSERTAÇÕES DO CBSOFT

COORDENADOR DO COMITÊ DE PROGRAMA
Eduardo Santana de Almeida - Universidade Federal da Bahia (UFBA)

COORDENAÇÃO DO CBSOFT 2014
Baldoino Fonseca - Universidade Federal de Alagoas (UFAL)
Leandro Dias da Silva - Universidade Federal de Alagoas (UFAL)
Márcio Ribeiro - Universidade Federal de Alagoas (UFAL)

REALIZAÇÃO
Universidade Federal de Alagoas (UFAL)
Instituto de Computação (IC/UFAL)

PROMOÇÃO
Sociedade Brasileira de Computação (SBC)

PATROCÍNIO
CAPES, CNPq, INES, Google

APOIO
Instituto Federal de Alagoas, Aloo Telecom, Springer, Secretaria de Estado do Turismo AL, Maceió Convention & Visitors Bureau, Centro Universitário CESMAC e Mix Cópia

Search-Based Mutation Testing para Programas Concorrentes

Rodolfo Adamshuk Silva[1], **Simone do Rocio Senger de Souza**[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (ICMC/USP)
Caixa Postal 668 – 13.560-970 – São Carlos – SP – Brazil

{adamshuk,srocio}@icmc.usp.br

Projeto de Doutorado

Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional do ICMC-USP

Ano de ingresso no programa: 03/2013

Época prevista de conclusão: 02/2017

Data da aprovação da proposta de tese: Previsão de 15/09/2014

Sigla do evento do CBSoft relacionado: SBES

Resumo. *O teste de software é uma atividade que busca garantir a qualidade por meio da identificação de falhas no produto. Para a aplicação do teste de software, algumas técnicas são utilizadas como, por exemplo, a técnica estrutural, a funcional e a baseada em defeitos. Cada técnica possui seus critérios para auxiliar na criação dos casos de teste. Um dos critérios da técnica baseada em defeitos é o teste de mutação. Este critério de teste baseia-se nos enganos que podem ser cometidos pelos desenvolvedores de software. O teste de mutação tem se mostrado eficaz para revelar defeitos, porém, o seu alto custo compromete sua utilização, principalmente quando são considerados programas complexos. A Programação Concorrente é um paradigma de desenvolvimento essencial para a construção de aplicações com o intuito de reduzir o tempo computacional em muitos domínios. Estas aplicações têm novas características como comunicação, sincronização e não determinismo que precisam ser considerados durante a atividade de teste. No contexto de programas concorrentes, novos desafios são impostos e precisam ser adequadamente tratados. Isso ocorre devido ao não determinismo e as diferentes sincronizações entre processos. Search-Based Software Testing é uma abordagem que vem sendo empregada para otimizar a atividade de teste, aplicando meta-heurísticas para a solução de problemas complexos, como por exemplo, geração de dados de teste. Este projeto de doutorado está inserido nesse contexto, no qual se pretende investigar o uso de Search-Based Software Testing para redução do custo da aplicação do teste de mutação no contexto de aplicações concorrentes.*

Palavras-chave. *Teste de mutação, Search-Based Testing, Programação concorrente.*

1. Introdução

Como um resultado dos avanços tecnológicos na área de hardware, por exemplo, processadores *multicore*, a computação distribuída faz-se cada vez mais presente nos sistemas atuais. Com isso, a programação concorrente está tornando-se cada vez mais popular no desenvolvimento de softwares modernos. Esse paradigma de desenvolvimento é essencial para a construção de aplicações capazes de reduzir o tempo computacional em vários domínios, por exemplo softwares para processamento de imagens, dinâmica de fluidos, entre outros. Diferentemente dos programas com características sequenciais, a computação distribuída envolve programas (ou processos) concorrentes (ou paralelos) que interagem para resolver um determinado problema. Essa interação pode ocorrer de forma sincronizada ou não, sendo que esses programas podem ou não concorrerem pelos mesmos recursos computacionais.

Teste de software é uma atividade de garantia de qualidade que visa a identificar defeitos no produto em teste. A atividade de teste consiste em uma análise dinâmica do produto e é uma atividade relevante para a identificação e eliminação de defeitos que persistem. O teste de produtos de software envolve basicamente quatro etapas: planejamento de testes, projeto de casos de teste, execução e avaliação dos resultados dos testes [Myers et al. 2011]. Essas atividades devem ser desenvolvidas ao longo do próprio processo de desenvolvimento de software e, para isso, um ponto crucial é a seleção dos casos de teste que serão utilizados. Sabe-se que executar todos os casos de teste possíveis a partir do domínio de entrada do produto em teste é impraticável e, portanto, critérios de teste são definidos.

Os critérios de teste procuram estabelecer como selecionar casos de teste que possuam alta probabilidade de encontrar a maioria dos defeitos com um mínimo de tempo e esforço. Um teste bem sucedido é aquele que consegue determinar casos de teste para os quais o programa em teste falhe. Um dos critérios de testes da técnica baseada em defeitos é o teste de mutação. Este critério utiliza informações sobre os enganos típicos que podem ser cometidos no processo de desenvolvimento para derivar casos de teste [Jia and Harman 2011]. Assim, os defeitos típicos de um domínio ou paradigma de desenvolvimento são caracterizados e implementados como operadores de mutação que, durante a atividade de teste, geram versões modificadas (mutantes) do produto em teste (especificação ou implementação, por exemplo). A intenção é auxiliar a seleção de casos de teste que demonstrem que os defeitos modelados pelos operadores de mutação não estão presentes no produto em teste. Um problema do teste de mutação que impede que ele seja aplicado amplamente é o alto custo computacional de executar o enorme número de mutantes com os casos de teste, além do alto número de casos de teste necessários para alcançar um escore de mutação mais próximo de 1.0 [Jia and Harman 2011].

A atividade de teste no contexto de programas concorrentes é considerada mais complexa quando comparada com a atividade de teste de programas sequenciais. Além das dificuldades inerentes à atividade de teste, outras ocorrem devido, principalmente, ao comportamento não determinístico dessas aplicações, no qual múltiplas execuções de um programa concorrente com a mesma entrada podem executar diferentes sequências de sincronização e podem produzir diferentes resultados. Cabe à atividade de teste, nesse cenário, identificar se todas as sequências de sincronização possíveis foram executadas e se as saídas obtidas estão corretas. Essa característica difere os programas concorrentes dos programas sequenciais e precisa

ser considerada durante a atividade de teste de software. Assim como os demais critérios de teste, o teste de mutação tem sido investigado para programas concorrentes [Offutt et al. 1996, Silva-Barradas 1998, Bradbury et al. 2006, Silva 2013].

2. Motivação e Objetivo

A busca por estratégias que visem a automatização de teste de software é constante não só para o critério de mutação, mas também para os outros critérios. Isso vem acontecendo porque os programas a serem testados estão cada vez maiores e mais complexos, tornando a atividade de teste mais custosa, em termos financeiros e computacionais, e mais demorada. Até mesmo programas simples são custosos de serem testados devido ao problema de explosão de casos de testes [Jia and Harman 2011]. Todavia, alguns desses problemas podem ser modelados matematicamente a fim de serem resolvidos por meio da otimização matemática utilizando-se meta-heurísticas. É nesse contexto que surgiu o conceito de *Search-Based Software Testing* (SBST) que consiste no uso de técnicas de busca utilizando meta-heurística (algoritmo genético, por exemplo) para automatizar total ou parcialmente a atividade de teste [McMinn 2011]. Técnicas de busca meta-heurística são arcabouços de alto nível que utilizam heurísticas para encontrar soluções à problemas que envolvem combinação de resultados a um custo computacional aceitável [McMinn 2011]. Essas técnicas não são algoritmos prontos, mas estratégias para a adaptação para problemas específicos.

Pesquisas nessa área mostram que é promissora a aplicação de *Search-Based* no contexto de teste de software. As áreas de aplicação são cinco: geração de dados de teste, seleção de casos de teste, priorização de casos de teste, testes não funcionais e testes funcionais. A geração de dados de teste consiste em identificar um conjunto de dados de entrada válido para um programa, que satisfaça um determinado critério de teste com o objetivo de encontrar um dado de teste que faça o programa se comportar diferentemente do esperado, revelando, assim, um defeito. A seleção de casos de teste consiste em escolher quais casos de teste devem ser aplicados em um sistema [Maia 2009]. A priorização de casos de teste trata da ordenação dos casos de teste de modo que os mais significativos sejam executados primeiro [Maia et al. 2010]. Os testes não funcionais verificam características do sistema não relacionadas a funcionalidades [Briand et al. 2004]. Por sua vez, os testes funcionais verificam se a implementação de uma aplicação está de acordo com sua especificação [Cohen et al. 2003].

O principal problema associado ao teste de mutação está no número de mutantes gerados, sendo, na maioria das vezes, alto, fazendo com o que o custo computacional também seja alto [Jia and Harman 2011]. Esse problema é agravado quando se está tratando de programas concorrentes, uma vez que, por causa do não determinismo, o programa possui diferentes sequências de sincronização que devem ser exercitadas durante a atividade de teste. Isso faz com que a complexidade computacional da execução de todos os mutantes, a geração de dados de teste para matar todos os mutantes sejam altos [Jia and Harman 2011]. Outro problema está relacionado com a identificação dos mutantes equivalentes que é realizada pelo testador e leva certo tempo para ser concluída, pois se deve analisar o programa mutante, comparar com o programa original, para então decidir se o mutante é equivalente ou não. Então, quanto mais mutantes forem gerados, mais mutantes terão que ser analisados para identificar os equivalentes.

Devido ao alto custo da aplicação do teste de mutação, várias estratégias vêm sendo utilizadas para fazer com que o critério de mutação possa ser utilizado de maneira mais eficiente. As técnicas de redução do custo do teste de mutação podem ser divididas em três tipos: *do fewer* (redução de número de mutantes sem afetar desempenho), *do faster* (procurar executar os mutantes o mais rápido possível) e *do smarter* (dividir o custo computacional ou evitar a execução completa ou guardar informações do estado da execução) [Offutt and Untch 2001].

Levando em consideração os resultados já obtidos em [Silva 2013], surgiu a motivação de investigar abordagens que proporcionem a diminuição dos custos de aplicação do teste de mutação (*do fewer*) para programas concorrentes. O objetivo do projeto de doutorado é investigar e propor uma abordagem para a otimização do teste de mutação utilizando técnicas de *Search-Based Software Testing*. Os desafios científicos identificados neste projeto de doutorado estão relacionados ao uso de SBST aplicado ao teste de mutação. Além disso, tem-se a aplicação de técnicas de SBST no contexto de programas concorrentes, o que é considerado inovador, pois essa técnica é pouco aplicada para resolver os problemas envolvendo teste nesse contexto.

3. Resultados e Contribuições

Com o objetivo de se ter uma visão ampla do contexto a ser estudado, uma revisão sistemática foi conduzida com o objetivo encontrar trabalhos que aplicam técnicas de meta-heurística como forma de automatizar o teste de mutação. Como a finalidade de alcançar esse objetivo, foram formadas as seguintes questões de pesquisa: Q1) Em quais etapas do teste de mutação SBST está sendo aplicada? Q2) Quais as técnicas de meta-heurística estão sendo utilizadas no teste de mutação? A condução da revisão foi realizada levando em consideração cinco máquinas de busca: IEEE Xplore, ACM Digital Library, Science Direct, Springerlink e ISI Web of Knowledge. A pesquisa retornou 195 trabalhos dos quais foram selecionados 55 artigos para a leitura completa. Isso demonstrou a falta de trabalhos nessa área. Vale ressaltar que nenhum dos artigos encontrados aplica SBST para teste de mutação em programas concorrentes, o que faz desta proposta de doutorado inovadora. A *string* de busca e os critérios de exclusão com o número de trabalhos excluídos são mostrados na Figura 1.

String de Busca

("mutation test" **OR** "mutation testing" **OR** "mutation-based test" **OR** "mutation based test" **OR** "mutation analysis" **OR** "program mutation") **AND** ("evolutionary" **OR** "heuristic" **OR** "search-based" **OR** "search based" **OR** "metaheuristic" **OR** "meta-heuristic" **OR** "optimization" **OR** "hill-climbing" **OR** "hill climbing" **OR** "simulated annealing" **OR** "tabu search" **OR** "genetic algorithms" **OR** "genetic programming" **OR** "ant colony")

Critério de Exclusão	Trabalhos Excluídos
Trabalhos que apresentam automatização do teste de mutação, mas sem a utilização de uma meta-heurística	49
Trabalhos que apresentam a utilização de meta-heurística para outros tipos de teste de software	20
Trabalhos que não apresentam teste de mutação nem SBST	11
Trabalhos que apresentam conceitos de teste de mutação aplicados em experimentos utilizando SBST	12
Trabalhos que sejam anais de eventos ou que descrevam eventos	19
Total	**111**

Figura 1. String de busca e critérios de exclusão.

Respondendo Q1, SBST está sendo aplicado para otimizar: a seleção de operador de mutação (1 trabalho), a geração de dado de teste (39 trabalhos),

a geração de mutante (12 trabalhos) e a geração de dado de teste e mutante simultaneamente (3 trabalhos). Respondendo a segunda questão de pesquisa, dentre os 15 trabalhos que apresentam abordagens para a redução do número dos mutantes utilizando SBST, as seguintes meta-heurísticas são utilizadas: *Genetic Algorithm* (15 trabalhos), NSGA-II (4 trabalhos), *Greedy Algorithm* (2 trabalhos), *Hill Climbing* (2 trabalhos), *Immune Algorithm* (1 trabalho) e *Local Search* (1 trabalho).

3.1. Trabalhos Relacionados

[May et al. 2003] apresenta uma abordagem inspirada no sistema imunológico do corpo humano chamada *Immune Algorithm*. A função de aptidão é calculada para o mutante observando o quão diferente foi a saída dada por ele em relação à saída do programa original. Para o dado de teste, a função de aptidão é calculada pelo número de mutantes que foram mortos. Nos trabalhos de [Adamopoulos et al. 2004, Assis Lobo de Oliveira et al. 2013] é apresenta uma metodologia que utiliza o conceito de co-evolução para gerar mutantes e dados de teste utilizando *Genetic Algorithm*. Em [Adamopoulos et al. 2004] a função de aptidão é calculada a partir do escore de mutação e em [Assis Lobo de Oliveira et al. 2013] é a divisão entre o número de mutantes mortos e o total de mutantes.

Em [Jia and Harman 2008] *Genetic Algorithm* e *Hill Climbing* são utilizadas para geração de mutantes. A função de aptidão é calculada pela união do conjunto de casos de teste que mata o mutante, dividido pelo conjunto de casos de teste, chamada de fragilidade do mutante. Nos trabalhos de [Domínguez-Jiménez et al. 2009b, Domínguez-Jiménez et al. 2009a, Domínguez-Jiménez et al. 2010, Domínguez-Jiménez et al. 2011] é apresentada uma técnica utilizando *Genetic Algorithm* para a geração de mutantes. A função de aptidão é calculada levando em consideração se o mutante é morto ou não pelo caso de teste e quantos mutantes foram mortos pelo mesmo caso de teste.

Em [Blanco-Munoz et al. 2011] *Genetic Algorithm* é utilizado para a geração de mutantes e a função de aptidão é calculada pela fragilidade do mutante. Em [Schwarz et al. 2011] é definida uma abordagem que utiliza *Genetic Algorithm* para encontrar um conjunto de mutantes que tenham um alto impacto e que estejam espalhados por todo o código. A função de aptidão é calculada pelo impacto que o mutante tem com relação aos outros mutantes.

No trabalho de [Omar et al. 2013] são aplicadas as meta-heurísticas *Genetic Algorithm* e *Local Search* para encontrar mutantes de ordem superior (HOM) mais difíceis de serem mortos. Em [Harman et al. 2010] são aplicados *Genetic Algorithm*, *Hill Climbing* e *Greedy Algorithm* para geração de HOMs. Para ambos os trabalhos, a função de aptidão é calculada pela diferença entre as falhas encontradas no HOM e as falhas encontradas nos mutantes que compõe o HOM. Em [Langdon et al. 2009a, Langdon et al. 2009b, Langdon et al. 2010] é apresentada uma abordagem que utiliza NSGA-II para a geração de mutantes. NSGA-II não calcula função de aptidão.

3.2. Abordagem Proposta

O objetivo deste trabalho é diminuir o custo relacionado à aplicação do teste de mutação no contexto de programas concorrentes. Para isso, está sendo investigado o uso de meta-heurísticas para a geração de mutantes. A partir desse objetivo,

duas perguntas puderam ser feitas: 1) Como o teste de mutação pode ser otimizado? e 2) Quais as possibilidades de utilização de SBST no contexto de teste de mutação e programação concorrente? Para responder essas perguntas, primeiramente foi conduzida uma revisão sistemática com o objetivo de identificar trabalhos que utilizam SBST para otimizar o teste de mutação sem focar em programas concorrentes. Com isso, pôde-se ter uma ideia das pesquisas na área de teste de mutação.

Com os trabalhos publicados nessa área, pode-se observar que há a motivação de diminuir o custo do teste de mutação por meio de três abordagens diferentes: 1) seleção de operador de mutação, 2) geração de dado de teste, 3) geração de mutante. Observou-se que grande parte dos trabalhos utilizavam a meta-heurística *Genetic Algorithm*, que utiliza uma função de aptidão para atribui valores aos mutantes dependendo de quão bons eles são com relação ao resultado de otimização que se quer alcançar. Levando em consideração as funções de aptidão, não foi encontrada nenhuma que utilizasse a frequência de mutação para avaliar um mutante. Então, essa foi a motivação para desenvolver um experimento para avaliar se a frequência de execução de mutantes pode ser uma boa alternativa para ser utilizada, posteriormente como função de aptidão em um *Genetic Algorithm*.

A frequência de execução calcula a quantidade de dados de teste que fazem com que o ponto onde há a mutação no programa mutante seja executado. Um importante ponto a se observar é a quantidade de mutantes vivos com uma alta frequência de execução. Segundo [Bottaci 2001], para demonstrar a presença de uma falha em um mutante (matar o mutante) é necessário que a execução do caso de teste alcance o ponto de mutação (condição de alcançabilidade), o valor da expressão modificada deve mudar o estado do programa mutante quando comparado com o programa original (condição de necessidade) e essa diferença no estado precisa ser propagada até a saída (condição de suficiência). Com isso, um mutante que possui uma grande frequência de execução pode ser um possível mutante equivalente (não satisfazendo a condição de necessidade) ou não há um caso de teste que o faça gerar uma saída diferente da apresentada pelo programa original (não satisfazendo a condição de alcançabilidade).

O experimento tem como objetivo analisar os mutantes mais executados com o propósito de verificar quantos são equivalentes ao programa original. A hipótese nula diz que todos os mutantes com uma maior frequência de execução e que permanecem vivos são equivalentes. A hipótese alternativa diz que há um conjunto de mutantes que possuem uma frequência alta, porém não são equivalentes ao programa original. Esse experimento está em desenvolvimento e consiste em aplicar o teste de mutação utilizando a ferramenta *Proteum* em programas escritos na linguagem C e identificar os mutantes equivalentes. O objetivo é verificar se os mutantes equivalentes são os mutantes com maior frequência de execução e quantos mutantes mortos tem uma alta frequência de execução. Com isso, espera-se refutar a hipótese nula e poder afirmar que há um conjunto de mutantes que possuem uma frequência alta, porém não são equivalentes ao programa original, podendo usar essa medida como função de aptidão no *Genetic Algorithm*. Os resultados desse estudo irão propiciar o estabelecimento de uma proposta de redução de custo do teste de mutação, empregando técnicas de meta-heurística que posteriormente será aplicada no contexto de programação concorrente.

Referências

[Adamopoulos et al. 2004] Adamopoulos, K., Harman, M., and Hierons, R. M. (2004). How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution. In *IN GECCO (2), VOLUME 3103 OF LECTURE NOTES IN COMPUTER SCIENCE*, pages 1338–1349. Springer.

[Assis Lobo de Oliveira et al. 2013] Assis Lobo de Oliveira, A., Gonyalves Camilo-Junior, C., and Vincenzi, A. (2013). A coevolutionary algorithm to automatic test case selection and mutant in mutation testing. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 829–836.

[Blanco-Munoz et al. 2011] Blanco-Munoz, E., Garcia-Dominguez, A., Dominguez-Jimenez, J., and Medina-Bulo, I. (2011). Towards higher-order mutant generation for ws-bpel. In *e-Business (ICE-B), 2011 Proceedings of the International Conference on*, pages 1–6.

[Bottaci 2001] Bottaci, L. (2001). A genetic algorithm fitness function for mutation testing. In *SEMINAL'2001 – First International Workshop on Software Engineering using Metaheuristic INnovative ALgorithms*, Toronto, Ontario, Canada.

[Bradbury et al. 2006] Bradbury, J. S., Cordy, J. R., and Dingel, J. (2006). Mutation operators for concurrent Java (J2SE 5.0). In *Proceedings of the Second Workshop on Mutation Analysis*, pages 11–20, Washington, DC, USA.

[Briand et al. 2004] Briand, L. C., Labiche, Y., and Shousha, M. (2004). Performance stress testing of real-time systems using genetic algorithms. Technical report, Carleton University.

[Cohen et al. 2003] Cohen, M. B., Gibbons, P. B., Mugridge, W. B., and Colbourn, C. J. (2003). Constructing test suites for interaction testing. In *Proceedings of the 25th International Conference on Software Engineering*, pages 38–48.

[Domínguez-Jiménez et al. 2011] Domínguez-Jiménez, J., Estero-Botaro, A., García-Domínguez, A., and Medina-Bulo, I. (2011). Evolutionary mutation testing. *Information and Software Technology*, 53(10):1108 – 1123.

[Domínguez-Jiménez et al. 2009a] Domínguez-Jiménez, J. J., Estero-Botaro, A., García-Domínguez, A., and Medina-Bulo, I. (2009a). Gamera: An automatic mutant generation system for ws-bpel compositions. *Web Services, European Conference on*, pages 97–106.

[Domínguez-Jiménez et al. 2010] Domínguez-Jiménez, J. J., Estero-Botaro, A., García-Domínguez, A., and Medina-Bulo, I. (2010). Gamera: A tool for ws-bpel composition testing using mutation analysis. volume 6189 of *Lecture Notes in Computer Science*, pages 490–493.

[Domínguez-Jiménez et al. 2009b] Domínguez-Jiménez, J. J., Estero-Botaro, A., and Medina-Bulo, I. (2009b). A framework for mutant genetic generation for ws-bpel. volume 5404 of *Lecture Notes in Computer Science*, pages 229–240.

[Harman et al. 2010] Harman, M., Jia, Y., and Langdon, W. (2010). A manifesto for higher order mutation testing. In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, pages 80–89.

[Jia and Harman 2008] Jia, Y. and Harman, M. (2008). Constructing subtle faults using higher order mutation testing. In *Source Code Analysis and Manipulation, 2008 Eighth IEEE International Working Conference on*, pages 249–258.

[Jia and Harman 2011] Jia, Y. and Harman, M. (2011). An analysis and survey of the development of mutation testing. *Software Engineering, IEEE Transactions on*, 37(5):649–678.

[Langdon et al. 2009a] Langdon, W., Harman, M., and Jia, Y. (2009a). Multi objective higher order mutation testing with genetic programming. In *Testing: Academic and Industrial Conference - Practice and Research Techniques*, pages 21–29.

[Langdon et al. 2009b] Langdon, W. B., Harman, M., and Jia, Y. (2009b). Multi objective higher order mutation testing with gp. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1945–1946.

[Langdon et al. 2010] Langdon, W. B., Harman, M., and Jia, Y. (2010). Efficient multi-objective higher order mutation testing with genetic programming. *J. Syst. Softw.*, 83(12):2416–2430.

[Maia et al. 2010] Maia, C. L. B., do Carmo, R. A. F., de Freitas, F. G., de Campos, G. A. L., and de Souza, J. T. (2010). Automated test case prioritization with reactive GRASP. *Adv. Soft. Eng.*, pages 2:1–2:13.

[Maia 2009] Maia, Camila Loiola Brito, d. C. R. A. F. d. F. F. G. d. C. G. A. L. d. S. J. T. (2009). A multi-objective approach for the regression test case selection problem. In *Proceedings of Anais do XLI Simpósio Brasileiro de Pesquisa Operacional*, pages 1824–1835.

[May et al. 2003] May, P., Mander, K., and Timmis, J. (2003). Software vaccination: An artificial immune system approach to mutation testing. In Timmis, J., Bentley, P., and Hart, E., editors, *Artificial Immune Systems*, volume 2787 of *Lecture Notes in Computer Science*, pages 81–92. Springer Berlin Heidelberg.

[McMinn 2011] McMinn, P. (2011). Search-based software testing: Past, present and future. In *International Workshop on Search-Based Software Testing (SBST 2011)*, pages 153–163.

[Myers et al. 2011] Myers, G. J., Sandler, C., and Badgett, T. (2011). *The Art of Software Testing*. Wiley Publishing, 3rd edition.

[Offutt and Untch 2001] Offutt, A. J. and Untch, R. H. (2001). Mutation testing for the new century. chapter Mutation 2000: Uniting the Orthogonal, pages 34–44. Kluwer Academic Publishers, Norwell, MA, USA.

[Offutt et al. 1996] Offutt, A. J., Voas, J., and Payne, J. (1996). Mutation operators for Ada. Technical report.

[Omar et al. 2013] Omar, E., Ghosh, S., and Whitley, D. (2013). Constructing subtle higher order mutants for java and aspectj programs. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pages 340–349.

[Schwarz et al. 2011] Schwarz, B., Schuler, D., and Zeller, A. (2011). Breeding high-impact mutations. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, pages 382–387.

[Silva 2013] Silva, R. A. (2013). Teste de mutação aplicado a programas concorrentes em mpi. Mestrado, ICMC, Instituto de Computação e Matemática Computacional, USP.

[Silva-Barradas 1998] Silva-Barradas, S. (1998). *Mutation analysis of concurrent software*. PhD dissertation, Dottorato di Ricerca in Ingegneria Informatica e Automatica, Politecnico di Milano.