Avaliando Diferentes Estratégias de Redução de Custo do Teste de Mutação

Adam Banzi, Diego Antunes, Gabriel Pinheiro, João Carlos Árias, Rafael Hornun, Rafael Cabral, Regiane Friedemann, Silvia Vergilio, Tiago Nobre

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-990 – Curitiba – PR – Brazil

{adam,diegor,gabrielb,joao}@inf.ufpr.br,
{rafaelh,rafaelv,regiane,silvia,tiagon}@inf.ufpr.br

Resumo. *O teste baseado em mutação tem sido aplicado em diferentes contextos nos quais tem se mostrado um dos mais eficazes em revelar defeitos. Entretanto, uma disvantagem desse critério é o alto custo computacional. Por isso diferentes estratégias têm sido propostas na literatura com o objetivo de reduzir esse custo. A utilização de tais estratégias é fundamental para permitir a aplicação do teste de mutação em aplicações reais, entretanto, elas têm sido avaliadas somente para um conjunto de programas pequenos e com poucas estruturas de dados. Para fornecer indícios sobre o efeito dessas estratégias em programas mais complexos, este trabalho apresenta resultados de um experimento conduzido com a ferramenta Proteum e com quatro estratégias: Mutação Aleatória, Mutação Seletiva, Mutação Restrita e a Estratégia Essencial. As estratégias são avaliadas segundo o escore de mutação e custo, dado pela redução no número de mutantes gerados.*

Abstract. *Mutation based testing has been applied in diverse contexts and has been considered by many authors as the most efficacious criterion in terms of revealed faults. However, it presents high computational costs. Because of this, different strategies for reducing mutation test costs have been proposed in the literature. The evaluation of such strategies is fundamental to allow the mutation test application in large systems, however, they have been evaluated only with small programs, with few structures. To provide information about the effects of those strategies in complex programs, this work presents results from an experiment conducted with Proteum tool and four strategies: Random Mutation, Selective Mutation, Constraint Mutation and the Essential Strategy. The strategies are evaluated according to the mutation score and cost, given by the number of generated mutants.*

1. Introdução

A atividade de teste possui um papel fundamental para a garantia da qualidade de software. Para auxiliar nessa atividade, técnicas e critérios de teste foram propostos ao longo dos anos, destacando-se o teste de mutação, que têm sido objeto de estudo em diferentes contextos [Franzotte and Vergilio 2006, Lee and Offutt 2001, Tuya et al. 2007]. Originalmente proposto em [DeMillo et al. 1978], o critério Análise de Mutantes realiza o

teste de um programa P utilizando um conjunto de mutantes, que são gerados a partir de uma modificação sintática simples em P, introduzida por uma regra, chamada operador de mutação, que descreve erros típicos associados aos principais enganos que os programadores podem cometer. O objetivo é derivar um conjunto de dados de teste T que seja capaz de matar todos os mutantes gerados, ou seja, diferenciar o comportamento de P e de seus mutantes. Nesse caso T é dito adequado e o programa considerado suficientemente testado.

Para aplicação efetiva do critério, algumas ferramentas estão disponíveis [DeMillo et al. 1988, Delamaro and Maldonado 1996, Ma et al. 2005]. Os experimentos realizados com essas ferramentas mostram que o critério Análise de Mutantes é um dos critérios mais eficazes em termos do número de defeitos revelados, entretanto, necessita de um número grande de casos de teste e apresenta elevado custo computacional para executar e analisar todos os mutantes gerados [Souza 1996, Souza and et al 2007].

Para permitir que a utilização do teste de mutação saia do estado da arte para o da prática, alternativas têm sido criadas para reduzir o custo sem, entretanto, diminuir o escore de mutação (cobertura do teste). Essas alternativas visam a resolver o problema da suficiência de operadores: minimizar o número de mutantes e dados de teste gerados através da redução do número de operadores utilizados, mantendo a mesma cobertura que seria obtida utilizando-se o conjunto total de operadores. Este problema tem sido investigado por muitos autores e diferentes estratégias foram propostas, destacando-se: a Mutação Aleatória [Acree et al. 1979], a Mutação Seletiva [Offutt et al. 1996], a Mutação Restrita [Mathur 1991, Wong et al. 1994] e a Abordagem Essencial [Barbosa 1998, Barbosa et al. 2001, Vincenzi et al. 1999].

Estas estratégias também foram avaliadas e comparadas. Os primeiros trabalhos [Mathur 1991, Offutt et al. 1996, Wong et al. 1994] reportam resultados de experimentos conduzidos com programas Fortran. No contexto de teste de unidade de programas C destacam-se [Barbosa 1998, Wong et al. 1997]. Outros experimentos consideram o teste de integração [Barbosa et al. 2001, Vincenzi et al. 1999] ou tentam prever a cobertura considerando o conjunto total de operadores de um conjunto de dados de teste adequado a um conjunto essencial de operadores [Namin et al. 2008]. Entretanto, os experimentos de avaliação nem sempre consideram todas as estratégias mencionadas acima e na maioria das vezes têm sido conduzidos com programas pequenos. Por exemplo, os trabalhos [Barbosa 1998, Barbosa et al. 2001, Mathur 1991, Vincenzi et al. 1999, Wong et al. 1994, Wong et al. 1997] utilizam programas pequenos cujas estruturas envolvem apenas vetores. Portanto, os resultados desses experimentos podem ser diferentes no contexto real de aplicação do teste de mutação, o qual envolve grandes sistemas e no qual as estratégias são realmente importantes.

No sentido de contribuir para essa avaliação, este artigo apresenta resultados de um experimento conduzido com todas as estratégias mencionadas acima para o teste de unidade de programas da Siemens [Hutchins et al. 1994]. O experimento foi conduzido com a versão Linux da ferramenta Proteum [Delamaro and Maldonado 1996], que possui 71 operadores e realiza o teste de programas C. As estratégias Mutação Aleatória e Essencial são as que apresentam melhores resultados considerando os fatores de comparação: custo em termos do número de mutantes e dados de teste, e escore de mutação.

O trabalho está organizado da seguinte maneira. A Seção 2 trata do problema de suficiência de operadores descrevendo os principais trabalhos relacionados e as estratégias comparadas. A Seção 3 descreve os principais passos do experimento realizado. A Seção 4 discute os resultados obtidos. A Seção 5 contém as conclusões e desdobramentos deste trabalho.

2. Suficiência de operadores de mutação

A ferramenta Proteum [Delamaro and Maldonado 1996], utilizada neste trabalho, realiza o teste de programas C e implementa 71 operadores de mutação agrupados em quatro classes: mutação em comandos, em constantes, em variáveis e em operadores. Esses mutantes são então executados para um conjunto T de dados de teste fornecidos pelo usuário. Caso a saída da execução de T para um determinado mutante $M_i, i = 1, 2, \ldots, n$ seja diferente da saída obtida pela execução de T em P, diz-se que o mutante M_i foi morto. O objetivo é matar todos os mutantes não-equivalentes. Um mutante é dito equivalente quando a saída do programa original P e a do mutante são iguais para todo dado de teste possível para P. Para T é calculado uma medida de cobertura de mutantes, denominada *escore de mutação*, de acordo com a Equação 1 [Delamaro and Maldonado 1996]:

$$escore(T, M) = \frac{|\text{mutantes mortos}|}{|\text{total de mutantes}| - |\text{mutantes equivalentes}|} \quad (1)$$

Se um conjunto T de casos de teste matou todos os mutantes não-equivalentes, T é considerado *adequado*.

O custo do teste de mutação é bastante alto, pois ele requer a execução e avaliação de um número geralmente grande de mutantes. Então, se cada operador descreve um tipo de defeito, não seria possível identificar um conjunto pequeno de operadores de tal maneira que, revelando os defeitos descritos por esses operadores, também garantidamente seriam revelados os defeitos descritos pelo conjunto total de operadores? Esse problema é conhecido como o problema de suficiência de operadores, que pode ser definido mais formalmente como [Barbosa 1998]:

Entrada: um conjunto M de mutantes para um programa P e um conjunto T, M-adequado.

Saída: um conjunto $M' \subseteq M$, tal que o $escore(T', M)$ seja igual ou muito próximo ao $escore(T, M)$, sendo T' um conjunto M'- adequado.

Um dos primeiros trabalhos abordando o problema de suficiência foi o de Acree et al. [1979] que introduziu a idéia de Mutação Aleatória ("*Randomly Selected* $X\%$"). Nesta estratégia é selecionada aleatoriamente uma pequena porcentagem ($X\%$) do total de operadores e o teste é realizado sobre essa porcentagem. Segundo [DeMillo et al. 1988], mesmo com uma pequena amostragem do total de mutantes é possível construir bons casos de teste. Entretanto, a Mutação Aleatória possui a desvantagem de não levar em consideração os tipos de operador de mutação e a classe à qual pertencem.

Mathur et al. [1991] propuseram uma estratégia baseada na idéia de [Acree et al. 1979] que ficou conhecida como Mutação Restrita. O objetivo é selecionar um pequeno subconjunto de operadores de mutação para serem aplicados no programa a

ser testado. Entretanto, uma das dificuldades desta estratégia diz respeito à identificação de quais serão os operadores a serem utilizados.

Uma maneira de selecionar os operadores, chamada a Mutação N Seletiva ("*N Selective Mutation*"), foi proposta em [Offutt et al. 1993], e tem o objetivo de gerar um subconjunto de operadores de mutação que não contenham os operadores responsáveis pela geração da maioria dos mutantes. Ou seja, a Mutação N Seletiva é aquela que deixa de gerar mutantes para N operadores de mutação. Em estudo posterior, Offut et al. [1996] separou os 22 operadores de mutação da ferramenta Mothra [Choi et al. 1989] em quatro classes: ES-selective, RS-selective, RE-selective e E-selective e obteve um conjunto restrito com 5 operadores e uma redução de 77,56% de mutantes gerados com um escore de mutação superior a 98%.

Um conjunto restrito de operadores para a linguagem C foi proposto em [Wong et al. 1994, Wong and Mathur 1995] Um outro experimento [Wong et al. 1997] forneceu indícios de que os operadores essenciais da ferramenta Proteum (linguagem C) possuem forte relação com os operadores essenciais da ferramenta Mothra (linguagem Fortran) obtidos por Offut et al. Com isso pôde-se concluir que um conjunto similar de operadores essenciais é obtido para ferramentas que realizam o teste de programas escritos em um mesmo paradigma (neste caso o paradigma procedural e as lingugens C e Fortran), pois geralmente essas ferramentas implementam operadores com a mesma função.

Outro trabalho relevante nessa direção foi o de Barbosa et al [Barbosa 1998, Barbosa et al. 2001, Vincenzi et al. 1999] que propuseram uma estratégia incremental chamada *Essencial* para determinar um conjunto de operadores essenciais. Como resultado da aplicação dessa estratégia, um conjunto, denominado conjunto essencial de operadores, foi proposto para a ferramenta Proteum. O conjunto inclui pelo menos um operador de cada classe e foi determinado considerando operadores com alto escore de mutação e a inclusão empírica entre operadores. Esta característica é o principal diferencial da abordagem Essencial com relação às abordagens aleatória e seletiva. Em estudos realizados a utilização dos operadores essenciais proporcionou uma redução de aproximadamente 65% no número de mutantes gerados, mantendo um alto grau de adequação (0,99616) em relação ao teste de mutação. Além disso, a estratégia Essencial, juntamente com a estratégia aleatória, alcançou o melhor resultado em termos de escore, apesar de apresentar o maior custo.

Os resultados dos trabalhos mencionados acima demonstram que é possível reduzir o custo do teste de mutação sem que haja uma redução significativa da eficácia em revelar a presença de defeitos. Entretanto, esses experimentos têm sido conduzidos com programas pequenos e com poucas estruturas de dados. Os programas Fortran utilizados por Offutt et al tinham em média 20,6 comandos; os utilizados por Wong et al, 57,9 linhas de código em média; e os de Barbosa et al 22,89 linhas [Namin et al. 2008]. Os programas utilizados não utilizam estruturas de dados complexas. Por exemplo, os programas C utilizados não contêm estruturas ou alocação dinâmica de memória.

Portanto, avaliar o desempenho dessas estratégias considerando programas mais complexos é importante para fornecer indícios de qual estratégia utilizar em ambientes reais e com grandes sistemas, e esse é o objetivo do experimento descrito na próxima

seção.

3. Decrição do Experimento

Nesta seção são descritos o experimento realizado, os programas utilizados, como foram aplicadas as estratégias e resultados obtidos.

3.1. Programas Utilizados

Foi utilizado um conjunto de programas da Siemens do repositório disponível na Universidade de Nebraska-Lincoln (Subject Infraestructure Repository (SIR[1]) [Hutchins et al. 1994]. Nesse repositório estão disponíveis, além de outras informações, o código C e um conjunto de casos de teste que foi utilizado no experimento. Estes programas contêm em média 270 LOC e incluem a maioria das estruturas C usadas em grandes sistemas, tais como: struct, apontador, alocação de memória, comandos *switch*, expressões condicionais complexas, e etc. Maiores informações sobre os programas utilizados encontram-se na Tabela 1, tais como o número de linhas de código (LOC) e o número de casos de teste disponíveis no repositório (NCR).

Cada programa e cada caso de teste disponível no diretório foram submetidos à versão Linux da ferramenta Proteum 1.4. Foram utilizados os 71 operadores da ferramenta, mas dependendo da característica do programa nem todo operador pôde ser aplicado. O número de operadores utilizados (O) e mutantes gerados (M) para cada programa pode ser visualizado na Tabela 1.

Pode-se observar que nem todos os mutantes puderam ser mortos, os mutantes que não foram cobertos pelo conjunto disponível foram considerados equivalentes (MEq), assim como foram descartados os casos de teste que não foram efetivos em matar mutantes, ou seja, aqueles que, considerada a ordem de execução, não contribuíram para aumentar o escore, restando apenas os dados de teste efetivos que compõem o conjunto T-adequado a M.

Tabela 1. Programas Utilizados

Programa	LOC	M	MEq	O	NCR	T
totinfo	281	5824	827	48	1052	31
schedule	296	2134	286	39	2650	39
schedule2	263	2815	588	39	2710	35
printtokens	243	4133	470	40	3775	252

3.2. Aplicação das Estratégias

De acordo com cada estratégia, um subconjunto de operadores $O' \subseteq O$ foi selecionado, assim como os respectivos mutantes M'. A partir desses mutantes foram determinados conjuntos de dados de teste adequados $T' \subseteq T$, ou seja, conjuntos formados por dados de teste de T que realmente contribuíram para aumentar o escore com relação a M' na mesma ordem inicial de execução. O escore de T' para M foi então calculado. O número de elementos dos conjuntos M' e T' e o escore para cada estratégia e cada programa podem ser visualizados na Tabela 2. A seguir é fornecida uma breve descrição de como foram obtidos os conjuntos O' e M' para cada estratégia.

[1] `http://sir.unl.edu/php/index.php`

Tabela 2. Resultados dos experimentos

Programa	Estratégia	M'	T'	Escore
totinfo	Aleatoria ($N = 10\%$)	480	11	0.9895
	Aleatoria ($N = 20\%$)	980	17	0.9955
	Aleatoria ($N = 40\%$)	1978	19	0.9959
	Seletiva ($N = 5$)	1908	20	0.9977
	Seletiva ($N = 10$)	1088	16	0.9959
	Seletiva ($N = 20$)	275	9	0.9761
	Essencial restrito	655	22	0.9975
	Essencial Barbosa et al	1283	21	0.9971
schedule	Aleatoria ($N = 10\%$)	186	18	0,9918
	Aleatoria ($N = 20\%$)	369	19	0,9945
	Aleatoria ($N = 40\%$)	741	24	0,9989
	Seletiva ($N = 5$)	907	1	0,6493
	Seletiva ($N = 10$)	485	1	0,6493
	Seletiva ($N = 20$)	395	1	0,6493
	Essencial restrito	743	20	0,9929
	Essencial Barbosa et al	1283	21	0,9967
schedule2	Aleatoria ($N = 10\%$)	229	14	0.9789
	Aleatoria ($N = 20\%$)	456	20	0.9934
	Aleatoria ($N = 40\%$)	907	27	0.9991
	Seletiva ($N = 5$)	1284	1	0.7483
	Seletiva ($N = 10$)	737	1	0.7483
	Seletiva ($N = 20$)	234	1	0.7483
	Essencial restrito	793	20	0.9925
	Essencial Barbosa et al	466	22	0.9881
printtokens2	Aleatoria ($N = 10\%$)	365	38	0.9617
	Aleatoria ($N = 20\%$)	732	45	0.9620
	Aleatoria ($N = 40\%$)	1467	81	0.9762
	Seletiva ($N = 5$)	2171	1	0.4520
	Seletiva ($N = 10$)	1179	1	0.4520
	Seletiva ($N = 20$)	340	1	0.4200
	Essencial restrito	1127	37	0.9585
	Essencial Barbosa et al	751	79	0.9737

Tabela 3. Resultados totais para cada estratégia

Estratégia	M'	T'	Escore (média)
Todos operadores/mutantes	12735	357	1.0000
Aleatoria ($N = 10\%$)	1260	81	0,9804
Aleatoria ($N = 20\%$)	2537	101	0,9863
Aleatoria ($N = 40\%$)	5093	151	0,9925
Seletiva ($N = 5$)	6270	23	0,7118
Seletiva ($N = 10$)	3489	19	0,7113
Seletiva ($N = 20$)	1244	12	0,7064
Essencial restrito	3318	99	0,9853
Essencial Barbosa et al	3783	143	0,9889

Mutação aleatória: foram utilizadas três porcentagens (X%): $10\%, 20\%$ e 40%. Ou seja foram selecionados aleatoriamente 10%, 20% e 40% dos mutantes gerados por cada operador de mutação. Portanto $O' = O$.

Mutação seletiva: foram considerados três valores para N: 5, 10 e 20. Então, os mutantes gerados pelos $N = 5$ operadores que mais geraram mutantes foram excluídos do conjunto total M e o conjunto de operadores O' utilizado ficou com $(O - 5)$ operadores. Analogamente com $(O - 10)$ e $(O - 20)$ para $N = 10$ e $N = 20$.

Conjunto essencial restrito: o conjunto foi obtido selecionando-se de cada classe de operadores da Proteum (comandos, constantes, operadores e variáveis) o operador com maior escore. Ou seja, o operador da classe cujos dados de teste adequados para matar todos os mutantes gerados por ele alcançaram o maior escore com relação a todos os mutantes gerados por todos os operadores. Portanto, o conjunto essencial restrito O' ficou com 4 operadores.

Conjunto essencial Barbosa et al: foi utilizado o conjunto determinado em [Barbosa 1998, Vincenzi et al. 1999] e formado pelos seguintes operadores: SWDD, SMTC, SSDL, OLBN, ORRN, VTWD, VDTR, CCCR, CCSR, com $O' = 9$.

4. Análise dos Resultados

Nesta seção a análise dos resultados é efetuada considerando dois fatores principais. O primeiro é o custo dado pelo número de mutantes gerados e que precisam ser executados. O segundo é o escore de mutação obtido, que representa uma maior cobertura no número de defeitos descritos pelos operadores a serem revelados pelos dados de teste. Pode-se também considerar como um fator adicional o número de dados de teste necessários, pois isso pode representar um esforço adicional para manter e lidar com tais dados no teste de regressão.

A análise está baseada na Tabela 3 e no conjunto M e T originais, apresentados na primeira linha dessa tabela (“Todos operadores/mutantes”).

Percebe-se na tabela que as estratégias Mutação Aleatória e Restrita alcançaram os melhores resultados em termos de escore, sendo o melhor resultado alcançado pela estratégia Aleatória 40%, seguido da estratégia Essencial proposta por Barbosa et al. Entretanto, quanto maior o escore obtido pela estratégia maior o número de dados de teste

utilizados. Pode-se observar que a ordem das estratégias quando considerado esse fator é inversa. Resultados semelhantes foram obtidos na literatura [Barbosa 1998].

Um resultado que mostra a influência do número de casos de teste é o obtido com a estratégia Seletiva. Os melhores escores foram obtidos com conjuntos maiores de casos de teste associados, tais como o do programa *totinfo*. Com relação ao número de mutantes gerados, também percebe-se que a estratégia Seletiva com $N = 5$ apresentou o maior valor. Os maiores valores seguintes são os das estratégia com maiores escores: Aleatória 40%, seguido da estratégia Essencial proposta por Barbosa et al.

O gráfico da Figura 1 mostra a porcentagem de redução do número de mutantes de cada estratégia. Tanto a estratégia Seletiva com $N = 20$ como a Aleatória 10% apresentaram uma redução em torno dos 90%. A menor redução ficou para a Seletiva $N = 5$ com 50%. As estratégias Aleatória 40% e Essencial de Barbosa et al, que apresentaram os maiores escores, tiveram as porcentagens de redução mais baixas, respectivamente 60% e 70%.

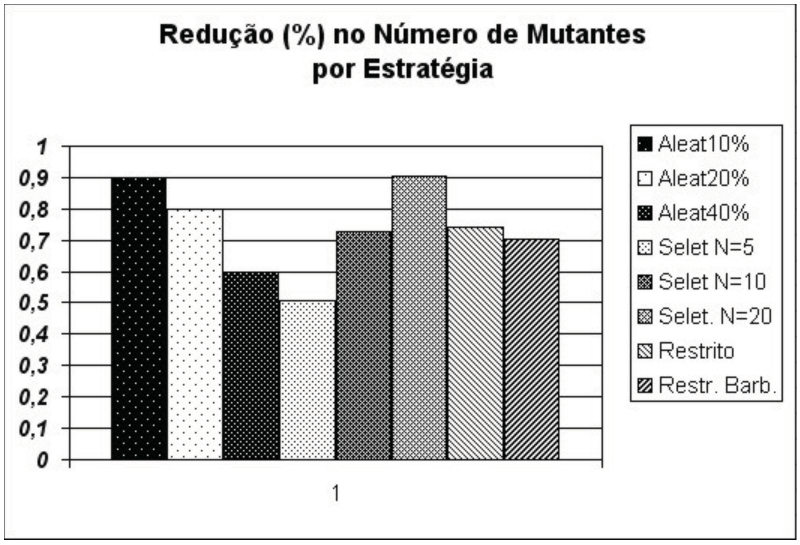

Figura 1. Redução no número de mutantes (Porcentagem)

Em uma avaliação geral, considerando todos os fatores, a estratégia Aleatória 10% apresenta um escore de 98% com um dos menores número de mutantes (uma redução de 90%) e um número de dados de teste intermediário, e, por isso parece ser uma ótima opção. Além disso, esta estratégia tem a vantagem de ser de fácil aplicação e de considerar todas as classes de operadores na redução.

5. Conclusão

O teste de mutação vem sendo aplicado nos mais diferentes e variados contextos. A existência de diversas ferramentas de suporte permite sua aplicação fora do ambiente acadêmico. O critério possui uma alta eficácia em termos do número de defeitos revelados, entretanto um alto custo computacional. Portanto a avaliação de estratégias para reduzir esse custo, tema deste artigo, é fundamental.

Os programas utilizados no experimento aqui descrito representam a maioria das estruturas utilizadas em grandes sistemas e os resultados obtidos podem apontar a me-

lhor estratégia em aplicações reais. Além disso, assim como os resultados reportados em [Wong et al. 1997], espera-se que resultados similares sejam obtidos utilizando uma outra ferramenta que implemente operadores com a mesma função que os da Proteum, por exemplo a ferramenta Mothra. Isso deverá ser investigado em experimentos futuros.

Quando considerado o fator escore de mutação relacionado ao tipo de defeito coberto, as estratégias Aleatória 40% e Essencial de Barbosa et al, apresentaram os melhores valores (0,99), com redução de custo, dado pelo número de casos de teste com respectivamente 60% e 70%. Se forem considerados ambos os fatores, a melhor estratégia é a Aleatória 10% com escore de 0,98 e redução de custo de 90%.

Outros experimentos semelhantes poderão ser conduzidos também com programas complexos da indústria para confirmar os resultados aqui encontrados. Pretende-se ainda avaliar essas estratégias no teste de software orientado a objetos utilizando-se a ferramenta MuJava [Ma et al. 2005].

Nesses experimentos poderão ser avaliados outros fatores, tais como a cobertura de critérios estruturais dos conjuntos T' de dados de teste associados a cada estratégia. Um fator que merece ser avaliado e não foi abordado nesse trabalho é o número de mutantes equivalentes gerados por operador. Se a equivalência tivesse sido determinada, talvez um resultado ligeiramente diferente tivesse sido obtido. Os mutantes equivalentes representam um esfoço adiicional e constituem-se um outro fator a ser minimizado. Buscar um conjunto essencial de operadores que também reduza o número de mutantes equivalentes gerados deve ser objetivo de um próximo estudo no qual se pretende investigar a aplicação de técnicas de minimização multi-objetivo para encontrar os melhores conjuntos de operadores para um dado programa.

Referências

Acree, A., Budd, T., DeMillo, R., Lipton, R., and Sayward, F. (1979). *Mutation analysis*. School of Information and Computer Science, Georgia Institute of Technology, Atlanta.

Barbosa, E. (1998). *Uma contribuição para determinação de um conjunto essencial de operadores de mutação no teste de programas C*. Biblioteca Digital de Teses e Dissertações da USP.

Barbosa, E., Maldonado, J., and Vincenzi, A. (2001). Towards the determination of sufficient mutant operators for C. *Software Testing Verification and Reliability*, 11:113–136.

Choi, B., DeMillo, R., Krauser, E., Martin, R., Mathur, A., Offutt, A., Pan, H., and Spafford, E. (1989). The Mothra tool set (software testing). In *System Sciences, 1989. Vol. II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on*, volume 2.

Delamaro, M. and Maldonado, J. (1996). Proteum–a tool for the assessment of test adequacy for C programs. In *Proceedings of the Conference on Performability in Computing Systems (PCS 96)*, pages 79–95.

DeMillo, R., Guindi, D., McCracken, W., Offutt, A., and King, K. (1988). An extended overview of the Mothra software testing environment. In *Software Testing, Verification, and Analysis, 1988., Proceedings of the Second Workshop on*, pages 142–151.

DeMillo, R., Lipton, R., and Sayward, F. (1978). Hints on test data selection: Help for the practicing programmer. *IEEE Software*, 11:34–41.

Franzotte, L. and Vergilio, S. (2006). Applying Mutation Testing to XML Schemas . In *18th International Conference on Software Engineering and Knowledge Engineering (SEKE'06)*.

Hutchins, M., Foster, H., Goradia, T., and Ostrand, T. (1994). Experiments of the effectiveness of data data flow and control flow-based w-test adequacy criteria. In *16th International Conference on Software Engineering (ICSE 1994)*, pages 191–200.

Lee, S. and Offutt, A. (2001). Generating Test Cases for XML-based Web Component Interaction Using Mutation Analysis . In *12th International Symposium o Software Reliability Engineering*, pages 200–209.

Ma, Y.-S., Offutt, A., and Kwon, Y. (2005). MuJava : An automated class mutation system . *Software Testing Verification and Reliability*, 2(15):97–133.

Mathur, A. (1991). Performance, effectiveness, and reliability issues in software testing. In *the Fifteenth Annual International Computer Software and Applications Conference, COMPSAC'91*, pages 604–605.

Namin, A. S., Andrews, J. H., and Murdoch, D. (2008). Sufficient Mutation Operators for Measuring Test Effectiveness . In *International Conference on Software Engineering (ICSE'2008)*, pages 351–360.

Offutt, A., Lee, A., Rothermel, G., Untch, R., and Zapf, C. (1996). An experimental determination of sufficient mutant operators . *ACM Transactions on Software Engineering and Methodology*, 2(5):99–118.

Offutt, A., Rothermel, G., and Zapf, C. (1993). An experimental evaluation of selective mutation. In *Proceedings of the 15th international conference on Software Engineering*, pages 100–107. IEEE Computer Society Press Los Alamitos, CA, USA.

Souza, S. (1996). *Avaliação do custo e eficácia do critério análise de mutantes na atividade de teste de software*. PhD thesis, Dissertação de Mestrado, ICMC/USP, São Carlos, SP.

Souza, S. and et al (2007). Estudos Teóricos e Experimentais. In *Introdução ao Teste de Software*. Editora Campus-Elsevier.

Tuya, J., Suárez-Cabal, M., and Riva, C. d. l. (2007). Mutating database queries. *Information and Software Technology*, 4(49):398–417.

Vincenzi, A., Maldonado, J., Barbosa, E., and Delamaro, M. (1999). Operadores Essenciais de Interface: Um Estudo de Caso. In *13th Simposio Brasileiro de Engenharia de Software, Florianopolis, SC*, pages 373–391.

Wong, W., Delamaro, M., Maldonado, J., and Mathur, A. (1994). Constrained mutation in C programs. In *Proceedings of the 8th Brazilian Symposium on Software Engineering*, pages 439–452.

Wong, W., Maldonado, J., Delamaro, M., and Souza, S. (1997). A comparison of selective mutation in C and fortran. In *Workshop do Projeto Validação e Teste de Sistemas de Operação*, pages 71–84.

Wong, W. and Mathur, A. (1995). Reducing the cost of mutation testing: An empirical study. *The Journal of Systems & Software*, 31(3):185–196.