Equivalência de Programas e o Teste de Software: Resultados de um Experimento de Aplicação do Critério Análise de Mutantes

Inali Wisniewski Soares
inali@unicentro.br
UNICENTRO, CP: 730, 85015-430
Guarapuava, PR

Silvia Regina Vergilio
silvia@inf.ufpr.br
DInf- UFPR, CP: 19081, 81531-970
Curitiba, PR

Resumo

O critério de teste Análise de Mutantes, baseado em erros, tem despertado grande interesse e se mostrado bastante efetivo em detectar erros, um dos principais objetivos da atividade de teste de software. Entretanto, a existência de mutantes equivalentes dificulta a sua completa automatização visto que, determinar se dois programas são equivalentes é uma questão indecidível. Essa tarefa necessita ser realizada manualmente e eleva o esforço e o custo da aplicação do critério. Esse trabalho explora a problemática de equivalência de programas na atividade de teste, apresentando resultados de um experimento de aplicação do critério Análise de Mutantes. Os resultados incluem a porcentagem de mutantes equivalentes gerados e a lista de operadores de mutação que mais geram equivalência. Esses resultados são importantes pois contribuem para o estabelecimento de uma estratégia de aplicação do critério Análise de Mutantes, bem como para implementação de mecanismos que auxiliem a identificação automática de mutantes equivalentes e conseqüentemente para reduzir os custos e esforço gastos com o teste.

Palavras-chave: teste de software**,** análise de mutantes, mutantes equivalentes**.**

1. Introdução

Dentre as atividades de verificação e validação, o teste é considerado fundamental para garantir a qualidade do software. Entretanto, essa atividade costuma custar mais caro do que deveria. Na tentativa de reduzir esse custo, diversas técnicas de teste têm sido propostas para selecionar os melhores casos de teste que possam revelar a maioria dos defeitos, conseguindo redução de tempo e esforço.

Às técnicas de teste, geralmente, estão associados critérios de teste, que podem ser utilizados tanto para auxiliar na geração de conjuntos de dados de teste como para auxiliar na avaliação da adequação desses conjuntos, ou seja para oferecer medidas a serem utilizadas para se considerar a atividade de teste encerrada.

A técnica baseada em erros deriva os casos de teste considerando os principais erros que geralmente ocorrem durante o processo de desenvolvimento de software. A Análise de Mutantes e a Semeadura de Erros são critérios típicos que se concentram em erros [2].

A Análise de Mutantes é um critério que utiliza um conjunto de programas ligeiramente modificados, denominados mutantes, obtidos a partir do programa P em teste. O conjunto é utilizado para selecionar e avaliar os dados de teste. O objetivo é encontrar um conjunto de casos de teste T capaz de revelar as diferenças de comportamento existentes entre P e seus mutantes [5]. Os mutantes gerados e executados com o conjunto de casos de teste devem ser mortos, isto é, apresentar resultados diferentes do programa original. Uma medida de cobertura dada pelo número de mutantes gerados e pelo número de mutantes mortos é utilizada para avaliar um dado conjunto de teste.

Resultados de comparações teóricas e empíricas entre diferentes critérios de teste

[10,11,12] apontam o critério Análise de Mutantes como um dos mais eficazes em revelar defeitos. Porém a sua aplicação efetiva requer a existência de uma ferramenta de teste automatizada. Dentre essas, podemos citar as ferramentas Mothra [6], Proteum [4]. A existência dessas ferramentas fez com que, nas últimas décadas, o critério Análise de Mutantes se tornasse cada vez mais popular e utilizado.

Um problema para completamente automatizar a aplicação do critério Análise de Mutantes é a existência de mutantes equivalentes. Um mutante é equivalente se ele apresenta o mesmo comportamento que o programa em teste P independentemente da entrada, ou seja, não existe um caso de teste que é capaz de diferenciar P de um mutante equivalente. Os mutantes equivalentes precisam ser identificados e descontados para calcular o escore de mutação e garantir a satisfação do critério.

A determinação de mutantes equivalentes é uma questão indecidível [2], ou seja não existe algoritmo de propósito geral para determinar a equivalência entre dois programas. Isso resulta em muitos problemas para a atividade de teste relacionados a automatização. Embora estudos teóricos tenham sido conduzidos [1,3,7,8] no intuito de propor heurísticas para determinar a equivalência de programas, a determinação de mutantes equivalentes é a atividade que exige maior esforço do testador, pois nas ferramentas citadas acima, ela é realizada manualmente, o que contribui para elevar o custo do teste de mutação.

Esse trabalho explora a problemática de equivalência de programas na atividade de teste, mais particularmente na aplicação do critério Análise de Mutantes. São apresentados resultados de um experimento realizado com a ferramenta Proteum. Os resultados incluem a porcentagem de mutantes equivalentes gerados e a lista de operadores de mutação dessa ferramenta que mais geraram equivalência. Esses resultados são importantes pois contribuem para o estabelecimento de uma estratégia para reduzir o custo de aplicação do critério Análise de Mutantes, assim como podem ser utilizados para implementar mecanismos na ferramenta Proteum para identificação de mutantes equivalentes.

Esse artigo está dividido do seguinte modo. A Seção 2 mostra resumidamente os principais conceitos relacionados ao critério Análise de Mutantes. A Seção 3 descreve como foi realizado o experimento. Na Seção 4 são analisados os resultados obtidos. A Seção 5 contém as conclusões.

2. Teste de Mutação e Equivalência de Programas

Entre os vários critérios propostos para se conduzir e avaliar a qualidade da atividade de teste, destaca-se o critério Análise de Mutantes [5], baseado em erros.

O critério Análise de Mutantes, proposto por DeMillo [5] fundamenta-se em dois pressupostos: 1) hipótese do programador competente: “programadores experientes escrevem programas muito próximos do correto"; e 2) efeito de acoplamento: “erros complexos são decorrentes de erros simples". Assim, espera-se que se existe um conjunto de testes que detecte erros simples em um programa então este poderá também revelar erros complexos.

A idéia básica do critério é gerar para um programa que será testado, vários programas mutantes diferentes desse programa por pequenas alterações sintáticas. Para satisfazer o critério é necessário gerar um dado de teste que faça com que cada mutante comporte-se diferentemente do programa em teste. Neste caso, o mutante é dito morto. Quando não existir nenhum dado de teste capaz de distinguir um mutante do programa em teste, o mutante é equivalente. O escore da cobertura obtida através do número de mutantes mortos e número de mutantes gerados, descontando-se o número de mutantes equivalentes é usada para avaliar o

conjunto de dados de teste sendo utilizado. Aliada ao alto custo computacional e espacial, a existência de mutantes equivalentes dificulta a utilização prática do critério.

A análise dos mutantes equivalentes é o passo que requer mais intervenção humana, e um dos maiores obstáculos para a aplicação do teste de mutação. Sem determinar todos os mutantes equivalentes o escore de mutação nunca será 100%. Assim o testador não terá completa confiança no programa e nos dados de teste. Pior, o testador será incapaz de saber se os mutantes restantes são equivalentes ou se o conjunto de teste é insuficiente. Determinar mutantes equivalentes manualmente consome muito tempo; é necessário um entendimento completo do programa, o que contribui para elevar o custo do teste de mutação.

A ferramenta Proteum [4] apóia o teste de mutação para programas C. A ferramenta Proteum oferece recursos ao testador para, através da aplicação do critério Análise de Mutantes, avaliar a adequação de, ou gerar um conjunto de casos de teste T para determinado programa P.

Em geral, o problema de resolver se dois programas são equivalentes é indecidível; essa limitação teórica não significa que o problema deva ser abandonado por não ter solução. Na verdade alguns métodos e heurísticas têm sido propostos para determinar a equivalência de programas em uma grande porcentagem dos casos de interesse [1,3,7,8].

Os programas equivalentes têm duas vantagens sobre o problema geral de equivalência, primeiro, programas mutantes são muito semelhantes aos programas originais. Budd e Angluin descrevem mutantes como vizinhos dos programas originais e pesquisadores utilizam esse fato para desenvolver técnicas e heurísticas para determinar mutantes equivalentes [7,8]. A segunda vantagem é que o teste de software é inerentemente uma ciência imperfeita; assim, os resultados parciais são de grande importância.

Um estudo teórico dos principais trabalhos que tratam da determinação de mutantes equivalentes foi realizado e uma descrição desses trabalhos está em [9].

3. Descrição do Experimento

Foram utilizados no experimento oito programas utilitários UNIX escritos na linguagem C, que são de domínio público: *cal, checkeq, col, comm, look, spline, tr e uniq*. Esses programas fazem parte de um benchmark e foram utilizados em diversos trabalhos da literatura com o objetivo de comparar diferentes critérios de teste [10,11,12].

Para realizar este experimento foi utilizada a ferramentas Proteum [4] para teste de programas no nível de unidade. A aplicação do experimento consistiu de duas fases principais: satisfazer o critério Análise de Mutantes e verificar os operadores que geraram o maior número de mutantes equivalentes.

I. Aplicação do Critério Análise de Mutantes

1. Geração dos programas mutantes: para geração dos mutantes foram utilizados todos os operadores disponíveis na ferramenta Proteum.

2. Geração de conjuntos AM-adequados: Para gerar os conjuntos AM-adequados fez-se uso dos conjuntos de dados de teste gerados de maneira "ad-hoc" e aleatória utilizados em [11], e submissão desses dados à ferramenta Proteum. Baseando-se nesses conjuntos e nas informações fornecidas pela ferramenta Proteum (mutantes vivos), foram adicionados novos casos de teste ao conjunto até obter-se um conjunto AM-adequado para cada programa. A

determinação de mutantes equivalentes é responsabilidade do testador, visto que a ferramenta Proteum utilizada não possui mecanismos que automatizem essa atividade. Os mutantes foram analisados para cada programa, e a equivalência foi determinada manualmente.

II. Operadores X Equivalência: para verificar a relação operador de mutação e número de mutantes gerados, foi utilizada a opção gerar relatórios da ferramenta Proteum.

4. Resultados

4.1 Porcentagem de Mutantes Equivalentes

Na Tabela 1 são apresentadas informações referentes à aplicação do critério Análise de Mutantes aos programas: número de mutantes gerados, mortos e equivalentes; e na última coluna é apresentada a cobertura, sem descontar os mutantes equivalentes. Esses dados demonstram o grau de complexidade dos programas considerados no experimento, onde é possível observar que:

- o programa spline teve o maior número de mutantes gerados 12560;
- o programa *uniq* teve o menor número de mutantes gerados 1623;
- os programas que tiveram o maior número de mutantes mortos e consequentemente o menor número de mutantes equivalentes (em porcentagem) foram os programas *cal* e *checkeq*;
- programa que teve o menor número de mutantes mortos e consequentemente o maior número de mutantes equivalentes foi o programa *tr*, devido ao grande número de variáveis e constantes existentes nesse programa;
- A média de mutantes equivalentes determinados no experimento representa 11,65% dos mutantes gerados e não deve ser desprezada pois grande quantidade de esforço foi gasto na atividade de determinação.

A determinação manual dos mutantes equivalentes consumiu uma grande quantidade de esforço e tempo. Em média foram gastas 240 horas para essa tarefa, foram analisados pelo menos 4.298 mutantes. Pode-se concluir que é muito importante que as ferramentas de teste ofereçam mecamismos que auxiliem o usuário nessa tarefa.

Tabela 1: Aplicação do critério AM

Programa	Número de Mutantes			Cobertura
	Gerados	Mortos	Equivalentes	
Cal	4334	4015	309 (7%)	93.00
Checkeq	3075	2861	214 (7%)	93.00
Col	6910	6023	887 (13%)	87.00
Comm	1938	1652	281 (15%)	85.00
Look	2030	1814	216 (11%)	89.00
Spline	12560	11134	1426 (11%)	89.00
Tr	4422	3632	790 (18%)	82.00
Uniq	1623	1463	160 (10%)	90.00
Total	36892	32594	4298 (11,65%)	88.35

4.2 Operadores e Equivalência

A Tabela 2 apresenta os totais de mutantes gerados e equivalentes determinados para os operadores da ferramenta Proteum, que geraram o maior e o menor número de mutantes equivalentes, para os programas em teste: *cal, checkeq, col, comm, look, spline, tr e uniq.* Um glossário de siglas e descrição dos operadores utilizados no trabalho é apresentado no apêndice. Através desses dados pode-se observar que:

- Os quatro operadores que geraram o maior número de mutantes equivalentes em relação ao número total de mutantes gerados em porcentagem, em ordem (do maior para o menor) foram: OCOR, VDTR, OLBN, e OEBA.
- Os quatro operadores que geraram o menor número de mutantes equivalentes em relação ao número total de mutantes gerados em porcentagem, em ordem (do menor para o maior) foram: STRP, STRI, SSWM e OABA.

A Tabela 3 apresenta os quatro operadores que mais geraram mutantes equivalentes por programa do experimento.

Tabela 2: Operadores que geraram o maior e o menor número de mutantes equivalentes

Maiores				Menores			
Operador	Mutantes			Operador	Mutantes		
	Gerados	equivalentes	%		gerados	Equivalentes	%
OCOR	312	300	96.15	STRP	941	12	1.28
VDTR	1818	692	38.06	STRI	312	5	1.59
OLBN	144	54	37.5	SSWM	62	1	1.61
OEBA	624	211	33.81	OABA	51	1	1.96

Tabela 3: Operadores que geraram o maior número de mutantes equivalentes por programa

Programa	**1º**	**2º**	**3º**	**4º**
Cal	VDTR	Cccr	OEBA	OEAA
Checkeq	VDTR	OEBA/ ORAN/ ORRN	OEAA	OLAN/ ORBN
Col	OCOR	Cccr	VDTR	Vsrr
Comm	Cccr	OEBA/ VDTR	OEAA	ORAN/ORBN
Look	VDTR	Cccr	OCOR	OEBA
Spline	Vsrr	VDTR	CRCR	OEAA
Tr	Vsrr	Cccr	VDTR	OEBA
Uniq	OCOR	VDTR	Cccr	OEBA/ORRN

A seguir são apresentados, com uma breve explicação, os operadores da ferramenta Proteum que geraram o maior número de mutantes equivalentes em relação ao total de mutantes gerados para os programas em teste através de exemplos das mutações equivalentes ocorridas nos programas do experimento para esses operadores. Maiores informações referentes ao experimento podem ser encontradas em [9].

OCOR - Cast Operator by Cast Operator

Este operador pertence ao grupo das mutações de operadores. Os operadores de *"cast"* envolvem os tipos primitivos da linguagem (*int, char, long, float, etc*) e são trocados por todos os outros tipos primitivos da linguagem. A Figura 1 exemplifica esse operador, onde é utilizada a rotina *canon* do programa *look.* O objetivo da função *tolower()* é devolver o equivalente minúsculo da variável *c* se esta for uma letra, caso contrário *c* será devolvida sem alteração. A mutação do valor de retorno da função *tolower()* que é um tipo *int* por um tipo *float* nesse caso produzirá uma saída do programa igual ao programa original pois um tipo

float pode armazenar sem problemas um tipo *int;* portanto, esse mutante é equivalente ao original.

VDTR - Domain Traps

Este operador pertence ao grupo de mutações de variáveis, onde cada referência escalar é substituída pelas chamadas às funções *trap_on_zero (e)*, *trap_on_positive (e)* e *trap_on_negative(e).* Estas funções abortam a execução dos mutantes caso a expressão tenha valor zero, positivo ou negativo, respectivamente. Caso contrário retornam o valor da expressão.

A Figura 2 exemplifica esse operador, onde é utilizada algumas linhas da rotina *main* do programa *cal.* Nesse exemplo a variável *argc* é um parâmetro que contém o número de argumentos da linha de comando e é um inteiro, *argc* é sempre pelo menos 1 porque o nome do programa é qualificado como primeiro argumento, e não é possível tornar a expressão *argc < 0* verdadeira, e fazer o mutante se comportar diferentemente do programa original.

OLBN - Logical Operator by Bitwise Operator

Este operador pertence ao grupo de mutações de operadores, onde cada operador lógico é substituído por um operador bitwise.

A Figura 3 exemplifica esse operador, onde é utilizada a rotina *main* do programa *cal.* O operador lógico || no comando if(m<1 || m>12) é trocado pelo operador bitwise ∧, independentemente de quais valores a variável *m* possuir o resultado será equivalente ao resultado do programa original; como exemplo, se a variável *m* possuir o valor -1 o comando será executado e a mensagem indicando " Bad month, -1" será impressa porque para os dois programas 0 || 1 e 0 ∧ 1 será 1.

OEBA - Plain Assignment by Bitwise assignment

Este operador pertence ao grupo das mutações de operadores e troca atribuição plana por atribuição bitwise.

A Figura 4 exemplifica esse operador, onde é utilizada a rotina *main* do programa cal. A atribuição plana = do comando for (i=0; i < 6*24; i+=24) é trocada pela atribuição bitwise &=, e o resultado dessa atribuição para a variável *i* continua sendo 0, portanto existe equivalência entre esses programas.

```
Canon(char * src, char * copia)
{  int cnt;  char c;
  for (cnt = len + 1; (c = *src++) && cnt; --cnt)
     if (!dict || isalnum(c))
          *copia++ = fold && isupper(c) ? (int)tolower(c) : c;
■         *copia++ = fold && isupper(c) ? (float)tolower(c) : c;
     *copia = EOS; }
```

Figura 1: Exemplo: Operador OCOR

```
main(int argc, char *argv[ ])
{       register y, i, j;  int m;
        if (argc == 2)
■       if ((TRAP_ON_NEGATIVE(argc) == 2))
            .....}
```

Figura 2: Exemplo: Operador VDTR

```
Main(int argc, char *argv[ ])
{                       register y, i, j; int m;
        . . .
         m = atoi(argv[1]);
         if(m<1 || m>12) {
■        if(m<1 ∧ m>12) {
             fprintf(stderr, "cal: %s: Bad month.\n", argv[1]);
                  exit(1);}}
```

Figura 3: Exemplo: Operador OLBN

```
Main(int argc, char *argv[ ])
{           .....
            for(i=0; i<6*24; i+=24)
■           for(i&=0; i<6*24; i+=24)
                    pstr(string+i, 24);
            .....
```

Figura 4: Exemplo: Operador OEBA

4.3 Estratégia de Aplicação do Critério Análise de Mutantes Baseada em Equivalência

A Proteum permite que o usuário selecione a porcentagem de mutantes a serem gerados por um determinado operador de mutação. Nesse sentido, os resultados aqui descritos podem ser utilizados para estabelecer uma estratégia de aplicação do Critério Análise de Mutantes. Essa estratégia estabelece que porcentagens baixas ou nulas sejam atribuídas a operadores que geraram um alto número (porcentagem) de mutantes equivalentes. São eles: OCOR, VDTR, OLBN, OEBA.

Os operadores que geraram o menor número de mutantes equivalentes foram incluídos na ferramenta Proteum para se garantir a inclusão do critério Todos-Ramos. Em [SOA02] foi proposta uma estratégia incremental de aplicação de diferentes critérios de teste que sugere seguir a relação de inclusão entre critérios para estabelecer uma ordem de aplicação dos mesmos. Parte-se primeiramente de critérios mais "fracos". Uma possível ordem é Critérios Baseados em Fluxo de Controle, em Fluxo de Dados, em Restrições e por último Baseados em Erros, tais como o critério Análise de Mutantes.

Durante a realização do experimento aqui descrito, observou-se que a maioria dos mutantes gerados por operadores que geraram o menor número de mutantes equivalentes, são facilmente cobertos. Se uma estratégia incremental de aplicação dos critérios tal como a descrita em [Soa00] for utilizada, esses operadores poderão ser desconsiderados, reduzindo o número de execuções do programa e consequentemente o custo do critério AM.

5. Conclusões

A determinação de mutantes equivalentes é uma questão indecidível e isso dificulta a completa automatização do critério baseado em erros Análise de Mutantes. Essa tarefa precisa, em geral, da intervenção humana e exige um grande esforço, aumentando os custos da atividade de teste.

Esse trabalho focalizou a problemática de mutantes equivalentes, apresentando resultados de um experimento de aplicação do critério Análise de Mutantes. Os resultados obtidos indicam, para os programas utilizados, que 11,65% dos mutantes gerados pelos operadores da ferramenta Proteum são equivalentes. Essa porcentagem pode indicar uma medida a ser utilizada para se estimar o número de mutantes equivalentes de um programa. Essa medida seria considerada para se encerrar a aplicação do critério Análise de Mutantes. Por exemplo, considerar durante o teste que uma cobertura real em torno de 90% é suficiente. Entretanto, novos experimentos deverão ser conduzidos com o objetivo de se obter um modelo empírico a ser utilizado em estimativas como essa, pois os programas utilizados nesse experimento constituem uma pequena amostra.

Os resultados observados sobre equivalência descritos na seção 4.2 servirão como base para implementar mecanismos e heurísticas para determinação de mutantes equivalentes, facilidades que poderão ser incorporadas à ferramenta Proteum, visto que a automatização completa é indecidível.

A estratégia proposta deverá ser avaliada em experimentos futuros. Sugere-se que a utilização ou não dos operadores que mais ou menos geraram mutantes equivalentes seja avaliada pelo usuário que poderá estabelecer a sua própria estratégia de aplicação para o critério Análise de Mutantes.

Referências Bibliográficas

[1] T.A. Budd, D. Angluin. *Two notions of correctness and their relation to testing*. Acta Informatica, Vol. 18(1):31-45, Nov., 1982.

[2] T.A. Budd. *Mutation Analysis: Ideas, Examples, Problems and Prospects, Computer Program Testing.* North-Holand Publishing Company, 1981.

[3] D. Baldwi, F. Sayward. *Heuristics for Determining Equivalence of Program Mutations*. CT, Res. Rep. 276, Dep. of Comp. Science-Yale University, New Haven, 1979.

[4] M.E. Delamaro, "*Proteum - Um ambiente de teste baseado na Análise de Mutantes*", Dissertação de Mestrado, ICMSC/USP - São Carlos, SP, Brasil, Oct., 1993.

[5] R.A. De Millo, R.J. Lipton, F.G. Sayward. *"Hints on Test Data Selection: Help for the Practicing Programmer"*. IEEE Computer, April, 1978.

[6] K.N. King, A.J. Offutt. *A Fortran Language System for Mutation-Based Software Testing. Software Practice and Experience.* Vol. 21(7):685-718, July 1991.

[7] A.J. Offutt, W. M. Craft. *Using compiler optimization techniques to detect equivalent mutants. The Journal of Software Testing, Verification, and Reliability*. 4(3):131-154. Sept. 1994.

[8] A.J. Offutt, Jie Pan. *Automatically Detecting Equivalent Mutants and Infeasible Paths. The Journal of Software Testing, Verification, and Reliability.* 7(3):165-192. Sept. 1997.

[9] I.W. Soares. *Análise de Mutantes e Critérios Restritos no Contexto de Teste de Software: Resultados de uma Avaliação Empírica*. Dissertação de Mestrado, UFPR, Curitiba, 2000.

[10] I.W. Soares, S.R. Vergilio. *Mutation Analysis and Constraint-Based Criteria: Results from an Empirical Evaluation in the Context of Software Testing.* III IEEE Latin-American Test Workshop, pg 33-38, vol 1. Montevideo, Feb. 2002.

[11] S.R. Vergilio. *Critérios Restritos de Teste de Software: Uma Contribuição para Gerar Dados de Teste mais eficazes. Doctorate Dissertation*, DCA/FEEC/Unicamp, Campinas – SP. July. 1997.

[12] W.E. Wong, A.P. Mathur, J.C. Maldonado. *Mutation versus all-usos: An Empirical evaluation of cost, strength and effectivenes.* In Software Quality and Productivity-Theory, Pratice, Education and Training. Hong Kong, Dec. 1994.

Apêndice – Descrição dos Operadores da Ferramenta Proteum

Operador	**Descrição**
Cccr	Troca constantes por todas as constantes do programa
CRCR	Troca constantes por 0, 1 e –1, dependendo do tipo de referência
OABA	Troca atribuição aritmética por operador de atribuição bitwise
OCOR	Troca o tipo primitivo do operador cast
OEAA	Troca atribuição plana por atribuição aritmética
OEBA	Troca atribuição plana por atribuição bitwise
OABA	Troca operador aritmético por operador booleano
OLAN	Troca operador lógico por operador aritmético
OLBN	Troca operador lógico por operador bitwise
ORAN	Troca operador relacional por operador aritmético
ORBN	Troca operador relacional por operador bitwise
ORRN	Troca operador relacional por operador relacional
SSWM	Força a execução de todos as alternativas do comando switch
STRI	Força a execução de true e false para cada if
STRP	Força a execução de todos os caminhos do programa
VDTR	Força cada referência escalar ser negativa, positiva e zero
Vsrr	Substitui referências a vetores pro variáveis escalares, globais e locais do programa