Geração de Dados de Teste: Uma Estratégia que Preserva a Hierarquia de Critérios

Silvia Regina Vergilio (DINF-UFPR)
José Carlos Maldonado (ICMSC-USP)
Mario Jino (FEE-UNICAMP)

DCA/FEE/UNICAMP
CP: 6101 CEP: 13081-970
email: silvia@dca.fee.unicamp.br

RESUMO

Comparações entre critérios de teste têm sido efetuadas baseadas principalmente em hierarquias definidas por uma relação de inclusão. Um critério C_1 inclui um critério C_2 se para qualquer programa, todo conjunto de dados de teste que satisfaz C_1 também satisfaz C_2. Pouca informação sobre a eficácia, definida como a capacidade dos critérios de revelar defeitos, é fornecida por uma hierarquia. A eficácia depende da estratégia utilizada para gerar conjuntos de teste adequados a cada critério, e por isso, a preservação da hierarquia não pode ser garantida. Uma estratégia chamada SINGER é proposta para ser utilizada juntamente com a estratégia proposta em [17], para gerar dados de teste com maior probabilidade de revelar defeitos e garante que a hierarquia seja sempre preservada, no que diz respeito a eficácia dos critérios, isto é, ao se aplicar os critérios C_1 e C_2, tal que C_1 inclui C_2, C_1 revelará pelo menos todos os defeitos revelados por C_2.

ABSTRACT

Comparisons of testing criteria have been based mainly on hierachies defined by an inclusion relation. Criterion C_1 includes criterion C_2 if, for every program, every test data set which satisfies C_1 also satisfies C_2. Little information on the efficacy, defined as the defect revealing ability of criteria, is provided by a hierarchy. Efficacy depends on the strategy for generation of test data sets adequate to each criterion and, hence, we can not guarantee that the hierarchy is maintained. A strategy called SINGER is proposed to be used in conjunction with the strategy proposed in [17], to generate test data with higher probability of revealing defects. SINGER garantees that hierarchy is always maintained with respect to the efficacy of criteria; that is, when applying ccriteria C_1 and C_2, such that C_1 includes C_2, C_1 will reveal at least all the defects revealed by C_2.

Palavras Chave: critérios de teste, relação de inclusão, eficácia.

1 Introdução

O teste é uma atividade fundamental para assegurar a qualidade e a confiabilidade de um software e representa a última revisão da especificação, projeto e codificação. Myers [12] afirma que o principal objetivo do teste é revelar a presença de defeitos no programa e que um bom caso de teste é aquele com alta probabilidade de revelar um defeito ainda não descoberto.

Diferentes técnicas de teste têm sido propostas com o objetivo de selecionar casos de teste para detectar a maioria dos defeitos, com um custo mínimo. Essas técnicas são vistas como complementares, pois cada uma delas testa o software sob uma perspectiva diferente. Elas estabelecem um critério de teste a ser satisfeito para se considerar a atividade de teste encerrada; esse critério poderá ser utilizado para avaliar a qualidade dos testes.

Vários estudos empíricos e teóricos [7], [4], [18], [1], [11] têm sido conduzidos com o objetivo de comparar diferentes critérios de teste. Segundo Wong [20], custo, eficácia e dificuldade de satisfação são os fatores básicos que devem ser considerados na escolha de um critério. 1) Custo: refere-se ao esforço necessário para utilizar o critério e é geralmente medido pelo número de casos de teste necessários para satisfazer o critério dado; 2) Eficácia: refere-se à capacidade de um critério em revelar um número maior de erros em relação a outro; 3) Dificuldade de Satisfação: refere-se à probabilidade de satisfazer um critério tendo satisfeito o outro. Esse último fator está associado a uma hierarquia de critérios dada por uma relação de inclusão. Um critério C_1 inclui um critério C_2 se, para qualquer programa, todo conjunto de casos de teste T que satisfaz C_1 também satisfaz C_2.

A relação de inlcusão é importante no que diz respeito ao estabelecimento de novos critérios. Ela estabelece certas propriedades dos critérios estruturais e requisitos mínimos que eles devem preencher, tais como, incluir o critério todos-ramos e do ponto de vista de fluxo de dados, ao menos um uso de todo resultado computacional (o que significa inclur o critério todas-definições [10].

A relação de inclusão tem sido bastante utilizada para comparar critérios quando considerados os fatores 1 e 3. Por outro lado, muitos autores não consideram a relação de inclusão um meio para avaliar a eficácia dos critérios (fator 2). Frankl [6] explora a relação de inclusão entre critérios e diz que o fato de C_1 incluir C_2 não garante que C_2 seja melhor para revelar defeitos que C_1. Segundo Weyuker et al [18], a relação de inclusão é útil para comparar o custo entre os critérios, mas não diz nada sobre a eficácia destes em revelar defeitos. Hamlet [7] diz que é possível um critério C_1 incluir um critério C_2, e que um conjunto de casos de teste T_2 (que satisfaz C_2) revele defeitos não revelados por um conjunto T_1 (que satisfaz C_1). Isso porque a eficácia de T_1 e de T_2 dependerá obviamente da estratégia utilizada para gerá-los.

Vergilio, Maldonado e Jino [17] propuseram uma estratégia de geração de dados de teste para satisfazer critérios estruturais baseados em fluxo de dados chamada SGER (Estratégia para Geração de Dados de Teste Sensíveis a Erros). Essa estratégia tem como objetivo aumentar a eficácia dos critérios baseados em fluxo de dados, permitindo

a utilização de técnicas complementares à técnica estrutural, selecionando dados de teste com alta probabilidade de revelar defeitos. Além disso, visa diminuir custos, reduzindo os efeitos causados por caminhos não executáveis nas atividades de teste

A utilização da estratégia SGER contribui para que a hierarquia entre critérios seja preservada, quando considerado o fator eficácia, mas existem casos para os quais a preservação não é garantida. Este trabalho tem como objetivo propor uma maneira de garantir que a hierarquia entre critérios seja preservada, ou seja, dado que C_1 inclui C_2, garante-se que T_1 (que satisfaz C_1) revelará pelo menos todos os erros que T_2 (que satisfaz C_2).

Na Seção 2 a hierarquia entre os critérios estruturais de teste e exemplos para os quais ela não é preservada com relação ao fator eficácia são apresentados. A Seção 3 mostra como e quando a aplicação da estratégia SGER poderá ou não preservar a hierarquia dada pela relação de inclusão. Na Seção 4 uma nova estratégia, chamada SINGER (Estratégia **I**ncremental para **G**eração de Dados de Teste Sensíveis a **Er**ros) é proposta com o objetivo de garantir que a hierarquia dada pela relação de inclusão seja sempre preservada. Na Seção 5 estão as conclusões e trabalhos futuros.

2 A Relação de Inclusão e a Eficácia dos Critérios

Os critérios estruturais de teste selecionam requisitos de teste baseados na implementação e no código fonte do programa em teste. Entre esses critérios, os mais conhecidos são os critérios baseados em fluxo de controle: critério todos-nós, todos-ramos, todos-caminhos, que exigem, respectivamente, que cada nó, ramo ou caminho do grafo de fluxo de controle associado ao programa seja executado pelo menos uma vez. Muitas vezes não é possível executar todos os caminhos de um programa (o número de caminhos pode ser infinito). Os critérios baseados em fluxo de dados [14], [8], [15], [9] foram introduzidos com o objetivo de estabelecer critérios mais exigentes e melhores que o critério todos-ramos e estabelecem uma hierarquia entre o critério todos-ramos e todos-caminhos. Eles procuram selecionar os caminhos mais interessantes de serem testados, aqueles que exercitam definições e conseqüentes usos (ou potenciais-usos) de variáveis do programa.

A Figura 1 extraída de [10] apresenta a hierarquia entre a família de critérios Potenciais-Usos e os critérios baseados em fluxo de controle. Esses critérios serão utilizados neste trabalho para ilustração.

A hierarquia é dada por uma relação de inclusão. Os critérios mais exigentes ocupam a parte superior da hierarquia. C_1 inclui C_2 (Notação: $C_1 \Rightarrow C_2$) se, para todo programa P, um conjunto de teste T que é C_1-adequado também é C_2-adequado, ou seja, a satisfação de C_1 garante a satisfação de C_2.

No entanto, a hierarquia não garante que ao se aplicar C_1 sejam revelados os mesmos defeitos revelados ao se aplicar C_2. Como dito anteriormente, isso é dependente da estratégia utilizada para gerar o conjunto de teste adequado aos critérios C_1 e C_2. Além

disso, muitos defeitos presentes em um programa são sensíveis somente a certos tipos de dados. A Figura 2a contém um programa para calcular o máximo entre dois números m e n. A Figura 2b contém a versão incorreta do programa, onde a atribuição $max = n$ é realizada incorretamente. A Figura 2c contém o grafo de fluxo de controle (GFC) associado as duas versões do programa. A Tabela 1 mostra os elementos requeridos para o critério todos-ramos e todos-potenciais-usos, sendo que todos-potenciais-usos $\Rightarrow$ todos-ramos.

Na Tabela 2 estão os dados de teste executados para satisfazer os dois critérios, utilizando-se a versão incorreta. Suponha que esses dados tenham sido gerados aleatoriamente. Por uma questão de sorte, um dos casos de teste gerados para o critério todos-ramos revelou o defeito. A hierarquia dada pela relação de inclusão não foi preservada, já que ao se aplicar todos-ramos, foi revelado um erro não revelado ao se aplicar o critério todos-potenciais-usos.

Um outro exemplo para o qual a hierarquia não é preservada é dado na Figura 3. A Figura 3a contém o programa correto. O programa da Figura 3b difere no comando de atribuição à variável z e está incorreto. A Tabela 3 mostra os elementos requeridos para as duas versões do programa. Para satisfazer os critérios foram selecionados casos de teste de tal maneira que todos eles contribuíssem para cobrir os elementos requeridos, ou seja, para que se executasse o menor conjunto de caminhos. Novamente, apenas um dos casos de teste selecionados para cobrir o critério todos-ramos revelou o defeito.

Para preservar a hierarquia entre os critérios, ou seja, dizer que C_1 é melhor que C_2 segundo o fator eficácia, é necessário que T_1 ($C_1 - adequado$) revele pelo menos todos os erros revelados por T_2 ($C_2 - adequado$). Nas próximas seções são dados exemplos de como a estratégia SGER pode aumentar a eficácia dos critérios baseados em fluxo de dados e também é apresentada uma nova estratégia, SINGER, que garante a preservação da hierarquia.

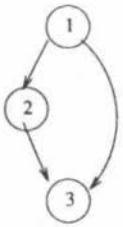
Figura 1: Hierarquia entre a Família de Critérios Potenciais-Usos e os Critérios Baseados em Fluxo de Controle

```
void prg1()
 { float m,max,n;
1   scanf("%f", &m);
1   scanf("%f ", &n);
1   max = m;
1   if (n>m)
2     max = n;
3   printf("%f", max);
  }
```

a) programa correto

```
void prg1()
 { float m,max,n;
1   scanf("%f", &m);
1   scanf("%f ", &n);
1   max = abs(m);
1   if (n>m)
2     max = n;
3   printf("%f", max);
  }
```

b) programa incorreto



c) GFC

Figura 2: Relação de Inclusão - Eficácia: Primeiro Exemplo

3 A estratégia SGER e a Eficácia dos Critérios Baseados em Fluxo de Dados

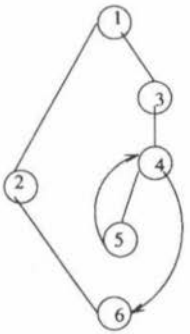
A estratégia SGER [17] visa facilitar a geração automática de dados de teste para cobrir critérios baseados em fluxo de dados; também poderá ser utilizada para cobrir critérios baseados em fluxo de controle. Ela permite reduzir os efeitos causados por caminhos não executáveis nas atividades de teste, utilizando a seguinte filosofia: entre dois caminhos candidatos a cobrir um elemento requerido por um critério, escolher o de menor número de predicados pois estudos estatísticos comprovam [16] [17] que, quanto maior o número de predicados de um caminho, maior a probabilidade de ele ser não executável. Ela tem o objetivo de responder à seguinte questão: uma vez selecionado o conjunto de caminhos necessários para cobrir os elementos requeridos, como gerar dados de teste que executem esses caminhos e que tenham maior probabilidade de revelar defeitos?

```
  void prg2();
 { double z,v;
   float i,x,y;
1  scanf("%d", &x);
1  scanf("%d", &y);
1  z = (x+x) + 6 - y *y;
1  if (z >= 0)
2     v =sqrt(z);
3  else
3  { v= x;
3    i = 1;
4    while (i <x)
5    {
5       v = v*x;
5       i++;
5    }
6  }
6  printf("%f",v) ;
6 }
```

a) programa correto

```
  void prg2();
 { double z,v;
   float i,x,y;
1  scanf("%d", &x);
1  scanf("%d", &y);
1  z = x*x + 3 - y*y;
1  if (z >= 0)
2     v =sqrt(z);
3  else
3  { v= x;
3    i = 1;
4    while (i <x)
5    {
5       v = v*x;
5       i++;
5    }
6  }
6  printf("%f",v) ;
6 }
```

b) programa incorreto



c) GFC

Figura 3: Relação de Inclusão - Eficácia: Segundo Exemplo

A todo caminho de um programa corresponde uma expressão chamada *condição do caminho* (termo usado pela execução simbólica) que é composta dos predicados que devem ser satisfeitos para que o caminho seja executado. O domínio de entrada para o caminho é definido para todos os valores que satisfazem essa condição. Para escolher entre esses valores, a estratégia SGER utiliza os fundamentos da técnica de teste funcional Análise do Valor Limite, que diz que um grande número de defeitos tende a ocorrer nos limites do domínio de entrada, e da técnica de Teste Baseado em Domínios proposta por White e Cohen [19]. Esta técnica requer a execução de pelo menos n+1 casos de teste, onde n é o número de variáveis de entrada do programa para cada borda (seja ela fechada ou aberta) do domínio de entrada de um caminho. Geralmente, n pontos pertencem à borda e um ponto não pertence.

Tabela 1: Elementos Requeridos para os Programas da Figura 2

crit. todos-ramos	crit. todos-pot-usos
arco (1,2)[1]	(1,(1,2),{m,max,n})
arco (1,3)	(1,(1,3),{m,max,n})
	(2,(2,3),{max})

Tabela 2: Casos de Teste Gerados para o Programa da Figura 2b

critério	dado de teste valor de (m,n)	saída esperada	saída obtida	caminho percorrido
todos-ramos	(-1,-3)	-1	1	1 3
	(3,9)	9	9	1 2 3
todos-pot-usos	(7,2)	7	7	1 3
	(1,3)	3	3	1 2 3

Tabela 3: Elementos Requeridos para os Programas da Figura 3

crit. todos-ramos	crit. todos-pot-usos
arco (1,2)	(1,(1,2),{x,y,z})
arco (4,5)	(1,(5,4),{x,y,z})
arco (4,6)	(1,(4,5),{x,y,z})
	(1,(4,6),{x,y,z})
	(2,(2,6),{v})
	(3,(4,6),{i,v})
	(3,(4,5),{i,v})
	(5,(4,6),{i,v})
	(5,(4,5),{i,v})

Tabela 4: Casos de Teste Gerados para o Programa da Figura 3b

critério	dado de teste valor de (x,y)	saída esperada	saída obtida	caminho percorrido
todos-ramos	(-1,2)	0	0	1 2 6
	(2,3)	1	4	1 2 3 4 5 4 6
todos-pot-usos	(-1,0)	2	2	1 2 6
	(-1,-5)	-1	-1	1 2 3 4 5 4 5 4 6
	(3,4)	27	27	1 3 4 6

Além disso, procura-se usar Teste Baseado em Restrições [3] para dar maior poder ao Teste Baseado em Domínios. Os pontos do domínio escolhidos deverão satisfazer algumas restrições que descrevem certos tipos de defeitos comuns em linguagens de programação. O caso de teste assim gerado pode mostrar a ausência ou presença do defeito associado à restrição; pode-se, assim, detectar outros tipos de defeitos, além dos determinados pelo Teste Baseado em Domínios e por técnicas estruturais.

Para o exemplo da seção anterior a SGER escolherá os caminhos 1 2 3 e 1 3 para cobrir os elementos requeridos pelo critério todos-potenciais-usos. Para facilitar, serão considerados valores de n e m restritos ao intervalo $[-10, 10]$. Ao caminho 1 3 está associada a condição $m >= n$. Os valores que satisfazem essa condição, ou seja, o domínio de entrada para o caminho é dado pelo triângulo da Figura 4. A restrição associada ao defeito da Figura 2b será $m < 0$, para que $abs(m) \neq m$.

A Tabela 5 apresenta dados de teste no limite do domínio de entrada do caminho. O último dado de teste, gerado aplicando-se a estratégia SGER, revela o defeito. Note que outros pontos, apesar de estarem no limite de entrada do caminho não revelam o defeito por não satisfazerem a restrição $abs(m) \neq m$.

Para este exemplo a aplicação da estratégia SGER garante a preservação da hierarquia porque a restrição gerada garantirá a revelação do defeito. A estratégia SGER, por utilizar, sempre que possível, pontos nos limites do domínio é adequada para revelar erros de domínio, causados por pequenas diferenças entre as bordas do programa em teste e a borda correta. Também é adequada para revelar outros tipos de defeitos que poderão ser descritos por restrições. Nesses casos, a estratégia auxilia na preservação da hierarquia, mas em outros nada poderá ser garantido. A SGER possui algumas limitações, principalmente no que diz respeito a sua implementação: condições de caminhos não lineares e dificuldade para resolver as restrições. Para o exemplo da Figura 3b, a condição do caminho 1 2 6 dada por $x^2+3-y^2 > 0$ é não linear. A restrição $x^2+3-y^2 \neq 2x+6-y^2$, associada ao defeito, também não é fácil de ser resolvida.

Pode ainda não ser possível determinar restrições necessárias e suficientes para se revelar um defeito. Uma restrição que descreva um estado intermediário incorreto de um programa, provocado por um defeito, é necessária mas não é suficiente para que o defeito seja revelado. Para se determinar condições suficientes é preciso ter conhecimento sobre a semântica do programa em teste e isso nem sempre é possível. Budd e Angluin [2] discutem o problema de suficiência, usando o termo "correção coincidente". Correção coincidente é uma limitação inerente às atividades de teste e ocorre quando um dado de teste gera um estado intermediário incorreto, mas mesmo assim uma saída correta é produzida. DeMillo [3] observa que na prática isso ocorre muito raramente.

Tabela 5: Casos de teste Gerados para o Programa da Figura 2b

dado de teste valor de (m,n)	saída esperada	saída obtida	pertence ao limite	satisfaz $(abs(m) \neq m)$
(0,0)	0	0	sim	não
(1,1)	1	1	sim	não
(2,0)	2	2	não	não
(-2,-2)	-2	2	sim	sim

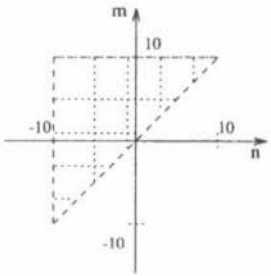


Figura 4: Domínio para o Caminho 1 3 do Programa da Figura 2b

4 A Estratégia SINGER e a Relação de Inclusão

Na seção anterior foi mostrado que a estratégia SGER pode auxiliar mas não garantir a preservação da hierarquia dada pela relação de inclusão quando considerada a eficácia entre os critérios. Uma maneira de garantir que a hierarquia entre os critérios seja preservada é gerar conjuntos de casos de teste T, de uma maneira incremental. Essa é a idéia de uma nova estratégia denominada SINGER (Estratégia **In**cremental para **G**eração de Dados de Teste Sensíveis a **Er**ros.

Utiliza-se a estratégia SGER, sempre que possível, inicialmente para gerar um conjunto de dados de teste T adequado ao critério menos exigente e depois para o critério superior até chegar ao mais exigente (o do topo da hierarquia). Dado o conjunto de critérios $(C_1, C_2, .., C_i, .., C_n)$ tal que $C_i \Rightarrow C_{i-1}$ (para $2 \leq i \leq n$), parte-se do critério C_1 (o mais abaixo na hierarquia), gera-se um conjunto T_1 de casos de teste, C_1-adequado, que revela um conjunto de defeitos F_1. Para o critério C_2 gera-se um conjunto T_2, C_2-adequado, tal que $T_1 \subset T_2$, que revelará um conjunto de defeitos F_2, tal que $F_1 \subset F_2$. Dessa maneira, se $C_i \Rightarrow C_j$, e $T_j \subset T_i$, preserva-se a hieraquia e garante-se que $F_j \subset Fi$, ou seja, que todos os defeitos revelados por C_j sejam revelados também por C_i.

A Tabela 6 mostra, para o exemplo da Figura 3, quais os dados de teste gerados pela SINGER. Inicialmente, o critério todos-ramos é aplicado, e um conjunto de casos de teste

Tabela 6: Executando Casos de Teste Incrementalmente

critério	dado de teste valor de (x,y)	saída esperada	saída obtida
todos-ramos	(-1,2)	0	0
	(2,3)	1	4
todos-pot-usos	(-1,2)	0	0
	(2,3)	1	4
	(3,4)	27	27

T_r (todos-ramos-adequado) é gerado. O conjunto T_r é avaliado em relação ao critério todos-potenciais-usos. Um caso de tese adicional será necessário para se obter o conjunto T_p (todos-potenciais-usos-adequado). Note que $T_p \subset T_r$ e tanto T_r como T_p revelam o erro.

Pode haver casos nos quais a aplicação da estratégia SINGER aumenta muito o número de casos de teste, elevando assim os custos. Nesses casos, se o tempo e custo forem limitados, a estratégia SGER deve ser aplicada. Por outro lado, quando uma alta confiabilidade for requerida, o custo de aplicação de todos os critérios é justificado pois, a aplicação da estratégia SINGER contribui para revelar um número maior de erros e para um aumento da qualidade dos testes gerados.

5 Conclusões

O trabalho explorou a utilização da hierarquia entre critérios estruturais dada pela relação de inclusão, para comparar critérios segundo a eficácia; ou seja, o que se pode dizer, dado que um critério C_1 inclui um critério C_2, sobre a capacidade de C_1 e C_2 de revelar defeitos. Dois exemplos mostrando que a hierarquia não se preserva quando considerada a eficácia dos critérios foram apresentados. Uma nova estratégia de geração de dados de teste denominada SINGER foi proposta. A utilização da SINGER garante sempre que dados C_1 e C_2, tal que C_1 inclui C_2, C_1 será tão eficaz quanto C_2.

A utilização da estratégia SGER leva à seleção de dados de teste com maior probabilidade de revelar erros e pode garantir, em muitos casos, que a hierarquia seja preservada; em outros casos nada é garantido, devido a limitações inerentes à própria técnica estrutural de teste. Além disso, não se garante que o programa foi testado para todos os tipos de defeitos que ele pode conter. Isso seria equivalente a provar a correção do programa. A diferença entre as saídas esperada e obtida pode ter sido causada por combinações de defeitos. Pretende-se explorar a utilização de combinações de restrições para resolver esta questão.

A estratégia SINGER garante que a hierarquia dada pela relação de inclusão seja sempre preservada e fornece, juntamente com a estratégia SGER, um roteiro a ser seguido para aumentar a eficácia dos critérios baseados em fluxo de dados. A obrigatoriedade de se

aplicar todos os critérios não se constitui uma desvantagem. Dependendo da criticalidade da aplicação, a utilização de todos os critérios se faz necessária; a SINGER contribuirá para reduzir os custos e aumentar a qualidade dos testes.

Será conduzido, como continuação desse trabalho, um experimento para estudar a validade da aplicação das estratégias SGER e SINGER. Será usado um conjunto de programas utilizado por Frate et al [5]. Esses programas possuem erros naturais encontrados durante a implementação. Conduzindo-se esse experimento poder-se-á verificar na prática a eficácia dos dados gerados e o custo e tempo gastos nessa geração. Pretende-se ainda, para realizar comparacões, repetir o experimento com outras técnicas tradicionalmente utilizadas para gerar dados de teste para satisfazer critérios estruturais, tais como a geração randômica. Outra proposta interessante é o estabelecimento de estratégias para gerar dados de teste com alta probabilidade de revelar defeitos para classes de programas específicos. A aplicação de tais estratégias levaria a um aumento da eficácia dos critérios para uma determinada classe.

Referências

[1] Beizer, B. Software Testing Techiniques, segunda edição, Van Nostrand Reinhold, N.Y. 1990.

[2] Budd, T.A.; Angluin, D., "Two Notions of Correctness and Their Relation to Testing", Acta Informatica, vol 18(1), pp 31-45, Nov. 1982.

[3] DeMillo, R.A.; Offutt J.; "Constraint-Based Automatic Test Data Generation", IEEE Trans. on Software Eng., 17(9), Set., 1991.

[4] Duran, J.W.; Ntafos, S., "An Evaluation of Random Testing", IEEE Trans. Software Eng., 10(7), pp 438-444, Julho 1984.

[5] Frate, F. et al, "Experiments to Investigate the Correlation Between Code Coverage and Software Reliability", Technical Report, Software Engineering Research Center, Purdue University, West Lafayette, Indiana, Abril, 19995.

[6] Frankl, P.G.; Weyuker, E.J., "A Formal Analysis of The Fault-Detecting Ability of Testing Methods", IEEE Trans. Software Eng., vol 19(3), pp 202-213, Março 1993.

[7] Hamlet, D.; Taylor, R., "Partition Testing Does Not Inspire Confidence", IEEE Trans. Software Eng., 16(12), Dez. 1990.

[8] Laski, J.W.; Korel, B., "A Data Flow Oriented Program Testing Strategy", IEEE Trans.on Software Eng., 9(3), 347-354, Maio, 1983.

[9] Maldonado, J.C; Chaim, M.L.; Jino, M., "Seleção de Casos de Teste Baseada nos Critérios Potenciais Usos", II Simpósio Brasileiro de Engenharia de Software, Canela, RS, Out., 1988.

[10] Maldonado, J.C., Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software. *Tese de Doutorado*, DCA/FEE/ UNICAMP - Campinas, SP, Julho, 1991.

[11] Mathur, A.P.; Wong, W.E., "An Empirical Comparison of Data Flow and Mutation-Based Test Adequacy Criteria", Software Testing, Verification and Confiability, vol 4, pp 9-31, 1994.

[12] Myers, G.J., The Art of Software Testing, Wiley, 1979.

[13] Pressman, R.B., Software Engineering: a Practitioner's Approach, Third Edition, New York, McGraw-Hill, 1992.

[14] Rapps, S.; Weyuker, E.J., "Data Flow Analysis Techniques for Test Data Selection", *in* Proc. Int. Conf. Software Engineering, Tokyo, Set., 1982.

[15] Ural, U.; Yang, B., "A Structural Test Selection Criterion", Information Processing Letters, Julho, 1988.

[16] Vergilio, S.R., Caminhos Não Executáveis: Caracterização, Previsão e Determinação para Suporte ao Teste de Programas. *Tese de Mestrado*, DCA/FEE/UNICAMP, Campinas, SP, Jan. 1992.

[17] Vergilio, S.R.; Maldonado, J.C; Jino, M., "Uma Estratégia de Geração de Dados de Teste", VII Simpósio Brasileiro de Engenharia de Software, Rio de Janeiro, RJ, Out. 1993.

[18] Weyuker, E.J.; Weiss, S.N.; Hamlet, R.G., "Comparison of Program Testing Strategies", in Proceedings of the Fourth Symposium on Software Testing, Analysis and Verification, Victoria, British Columbia, Canada, ACM Press, pp 154-164, 1991.

[19] White, L.J.; Cohen, E.I., "A Domain Strategy for Computer Program Testing", IEEE Trans. on Software Eng., 6(3), 247-257, Maio, 1980.

[20] Wong, W.E. et al, "Mutation Versus All-uses: An Empirical Evaluation of Cost, Strenght and Effectiveness", Software Quality and Productivity - Theory, practice, education and training, Hong Kong, Dez. 1994.