Operadores Essenciais de Interface: Um Estudo de Caso

Auri Marcelo Rizzo Vincenzi[1]
José Carlos Maldonado[1]
Ellen Francine Barbosa[1]
Márcio Eduardo Delamaro[2]

[1] Instituto de Ciências Matemáticas e de Computação – ICMC
Universidade de São Paulo – USP
Av. Dr. Carlos Botelho, 1465
13560-970, C.P. 668 – São Carlos (SP) – Brasil
{auri, jcmaldon, francine}@icmc.sc.usp.br

[2] Departamento de Informática – DIN
Universidade Estadual de Maringá – UEM
Av. Colombo, 5790
87020-900 – Maringá (PR) – Brasil
delamaro@din.uem.br

Abstract

Empirical studies have provided evidences that Mutation Analysis is effective on revealing faults. However, its high cost, due to the high number of mutants created, has motivated the proposition of alternative approaches for its application [1, 2, 3, 16, 19, 20, 27], among them the determination of essential mutation operators sets [2, 3, 20, 27]. In this context, some studies were already conducted for the determination of essential mutation operators sets for the Fortran and C languages [2, 3, 20, 27]. Mutation Testing application in the unit level for C language is supported by Proteum tool, which implements 71 mutation operators [4]. In the empirical study conducted by Barbosa [2, 3], using the Proteum tool, the Essencial procedure was proposed, contributing to the determination of essential mutation operators set for C language. Using the mutation concept in the integration testing the Interface Mutation criterion was proposed [5]. The PROTEUM/IM tool, developed to support the application of Interface Mutation, has 33 mutation operators built to model typical integration errors [5]. This work investigates the Essencial procedure application for the determination of an essential mutation operators set in the context of integration testing, contributing to the establishment of incremental testing strategies based on the mutation testing, including both the unit and integration phases.

Resumo

Apesar de evidências obtidas em estudos empíricos da eficácia do Teste de Mutação, seu alto custo, decorrente principalmente do grande número de mutantes gerados, tem motivado a proposição de diversas abordagens alternativas para a sua aplicação [1, 2, 3, 16, 19, 20, 27], dentre essas a determinação de conjuntos essenciais de operadores de mutação. Nesse contexto, alguns estudos já foram conduzidos para a determinação de um conjunto essencial de operadores de mutação para as linguagens Fortran e C [2, 3, 20, 27]. A aplicação do teste de mutação no nível de unidade para a linguagem C é apoiada pela ferramenta Proteum, que implementa 71 operadores de mutação [4]. No estudo empírico conduzido por Barbosa [2, 3], utilizando a ferramenta Proteum, foi proposto o procedimento Essencial, contribuindo para a determinação de um conjunto essencial de operadores de mutação para a linguagem C. Explorando o conceito de mutação no teste de integração foi proposto o critério Mutação de Interface [5]. A ferramenta PROTEUM/IM, desenvolvida para apoiar a aplicação do critério Mutação de Interface, possui 33 operadores de mutação desenvolvidos para modelar erros típicos de integração [5]. Este trabalho investiga a aplicação do procedimento Essencial para a determinação de um conjunto essencial de operadores para o teste de integração, contribuindo para o estabelecimento de estratégias de teste incrementais baseadas no teste de mutação que englobe as fases de unidade e de integração.

1. Introdução

A atividade de teste é um elemento crítico para a garantia da qualidade de software. Técnicas que auxiliam na condução dessa atividade têm sido propostas; a diferença entre essas técnicas está, basicamente, na origem da informação que é utilizada para avaliar ou construir conjuntos de casos de teste, sendo que cada técnica possui uma variedade de critérios de teste para esse fim. Na técnica funcional (caixa preta) os requisitos de testes são obtidos a partir da especificação; na técnica estrutural (caixa branca) derivam-se os requisitos a partir dos aspectos de implementação do software; e na técnica baseada em erros os elementos requeridos para caracterizar a atividade de teste são baseados em informações sobre erros comuns que podem ocorrer durante o desenvolvimento de software. É importante notar que nenhuma das técnicas de teste é completa, devendo ser aplicadas em conjunto para tentar assegurar um teste de melhor qualidade [14, 21].

A atividade de teste inicia-se com os testes em nível de unidade os quais têm como objetivo identificar erros de lógica e de implementação em cada unidade do software, separadamente. Em seguida, são realizados os testes de integração que visam a identificar erros de interação entre unidades que já foram testadas individualmente. Após a integração do sistema são realizados os testes de sistema que visam a garantir que o software atenda a todas as exigências funcionais, comportamentais e de desempenho descritas na especificação [21].

Embora exista uma grande variedade de critérios que podem ser aplicados em nível de unidade, o mesmo não ocorre para o teste de integração, no qual, em geral, utilizam-se os critérios funcionais. Desse modo, estudos empíricos vêm sendo conduzidos objetivando estender alguns critérios estruturais e baseados em erros para o teste de integração, permitindo com isso que critérios mais sistemáticos e rigorosos possam ser utilizados nessa fase do teste [5, 11, 13, 12].

Devido à diversidade de critérios de teste existentes surge a questão de qual critério utilizar para se obter o melhor resultado com o menor custo. Na tentativa de responder a esta e outras dúvidas que surgem no momento de decidir se um programa está ou não suficientemente testado, estudos teóricos [5, 10, 14, 22] e empíricos [1, 5, 6, 7, 16, 19, 23, 27] vêm sendo realizados. Através da comparação entre os critérios de teste procura-se obter uma estratégia que seja eficaz para revelar a presença de erros no programa e que apresente um baixo custo de aplicação [15, 23].

Ressalta-se que para a aplicação efetiva de um critério é necessária a existência de uma ferramenta de teste automatizada. A utilização de ferramentas possibilita reduzir o esforço para a realização do teste, bem como diminuir os erros causados pela intervenção humana nessa atividade. Além disso, a existência de ferramentas de teste viabiliza a realização de estudos comparativos entre as técnicas e critérios de teste.

O teste de mutação ou Análise de Mutantes, um dos critérios da técnica baseada em erros, tem se mostrado, através de estudos teóricos e empíricos [1, 16, 19, 23, 27], ser eficaz em revelar a presença de erros. Entretanto, existem alguns problemas na sua utilização devido ao alto custo computacional exigido na execução e análise do grande número de mutantes gerados. Na tentativa de solucionar esses problemas algumas abordagens, tais como a Mutação Aleatória [1] e a Mutação Restrita [16] vêm sendo propostas. Essas abordagens procuram reduzir a quantidade de mutantes gerados através da redução do número operadores de mutação utilizados durante o teste. A questão reside em quais operadores selecionar de modo a obter uma maior cobertura com um menor custo. Uma linha de trabalho relevante nesse sentido é a determinação de conjuntos essenciais de operadores de mutação e, nesse contexto, alguns estudos já foram conduzidos para as linguagens Fortran e C [2, 3, 20, 27]. A

aplicação do teste de mutação no nível de unidade para a linguagem C é apoiada pela ferramenta *Proteum*, que implementa 71 operadores de mutação, divididos em 4 classes, de acordo com o local em que as mutações são realizadas: Mutação de Comandos, Operadores, Constantes e Variáveis [4]. Nos estudos empíricos conduzidos por Barbosa [2, 3] utilizando a ferramenta *Proteum*, foi proposto o procedimento Essencial, o qual representa uma contribuição na determinação de um conjunto essencial de operadores de mutação para a linguagem C.

Procurando estender a aplicação do teste de mutação para a fase de integração, um novo critério denominado Mutação de Interface (IM – *Interface Mutation*) foi desenvolvido [5]. Com esse critério é possível testar a interação entre as unidades que compõem o software, ao contrário do teste de mutação original, que explora as características das unidades, separadamente. Para apoiar a aplicação do critério Mutação de Interface foi desenvolvida no Instituto de Ciências Matemáticas e de Computação (ICMC/USP) a ferramenta PROTEUM/IM [5]. A arquitetura da ferramenta PROTEUM/IM é similar à da *Proteum* [4] e ambas são destinadas ao teste de programas escritos na linguagem C. A diferença básica entre elas é que a PROTEUM/IM possui 33 operadores de mutação de interface desenvolvidos para modelar erros típicos de integração[1] [5] ao passo que a *Proteum* implementa 71 operadores que modelam erros típicos de unidade.

Com a proposição do critério Mutação de Interface é evidente o aspecto positivo de se utilizar o mesmo conceito de mutação nas diversas fases do teste; é também evidente a indagação sobre qual estratégia utilizar para se obter a melhor relação custo/eficácia no teste de um produto. Este trabalho tem por objetivo o desenvolvimento de estratégias de teste incrementais baseadas no teste de mutação que englobem as fases de unidade e de integração. Nesse sentido, o conjunto essencial de operadores de mutação de interface é determinado utilizando o procedimento Essencial; além disso, uma estratégia de aplicação do teste de mutação em nível de unidade e de integração é apresentada, considerando os conjuntos essenciais de unidade [2, 3] e de interface, respectivamente.

O artigo está organizado da seguinte forma: na Seção 2 são apresentados os conceitos relacionados ao critério Mutação de Interface. Na Seção 3 são discutidos alguns pontos relevantes que devem ser considerados durante a caracterização de um conjunto essencial de operadores de mutação. Um estudo de caso da aplicação do procedimento Essencial em nível de integração é apresentado na Seção 4. Na Seção 5 é feita a análise de uma estratégia incremental de aplicação do teste de mutação, utilizando os conjuntos essenciais de unidade e de interface, considerando a redução de custo, em termos do número de mutantes gerados, e o escore de mutação obtido em relação ao teste de mutação. A Seção 6 sintetiza os principais resultados e desdobramentos deste trabalho.

2. Mutação de Interface

A idéia do teste de mutação é avaliar a qualidade de um determinado conjunto de casos de teste, identificando sua capacidade em revelar a presença de erros simples que foram introduzidos no programa em teste. Um bom conjunto de casos de teste, além de revelar erros simples, também deve ser capaz de revelar a presença de erros mais complexos [18].

Os erros simples introduzidos no programa em teste são modelados através de operadores de mutação (*mutant operators*). Os operadores de mutação são construídos para

[1] Erros de integração são erros causados pela passagem de valores incorretos entre unidades, ou seja, pela comunicação entre as mesmas (parâmetros, variáveis globais, comandos de retorno, etc.).

satisfazer a um entre dois propósitos: 1) induzir mudanças sintáticas simples com base nos erros típicos cometidos durante o processo de desenvolvimento (como trocar o nome de uma variável); ou 2) forçar determinados objetivos de teste (como executar cada arco do programa) [20].

Embora o teste de mutação tenha se mostrado, através de estudos teóricos e empíricos, ser um dos mais eficazes para revelar a presença de erros [1, 8, 23, 25, 26], um dos problemas para a sua utilização é o alto custo computacional exigido na execução do grande número de mutantes gerados. Além disso, o teste de mutação em nível de unidade gera sempre o mesmo conjunto de mutantes independentemente de como as unidades interagem entre si para compor o programa. Desse modo, se a unidade é chamada em dois ou mais pontos do programa, não é possível garantir que todos esses pontos de chamada sejam testados: pode existir um conjunto de casos de teste que seja capaz de distinguir todos os mutantes gerados para uma dada unidade, passando por um único ponto de chamada [5].

Ao contrário, o teste de integração preocupa-se em assegurar que as interações entre unidades sejam testadas. Assim, o objetivo do critério Mutação de Interface é inserir perturbações nas conexões entre duas unidades, de modo que, para obter um conjunto de casos de teste adequado à Mutação de Interface, o testador deve criar casos de teste que revelem as diferenças de comportamento existentes entre o programa original e seus mutantes de interface não equivalentes. Utilizando o raciocínio do teste de mutação, casos de teste capazes de distinguir mutantes de interface também devem ser capazes de revelar grande parte dos erros de integração. Essa afirmação depende, evidentemente, de quais mutantes são utilizados ou, em outras palavras, quais operadores de mutação são aplicados [5].

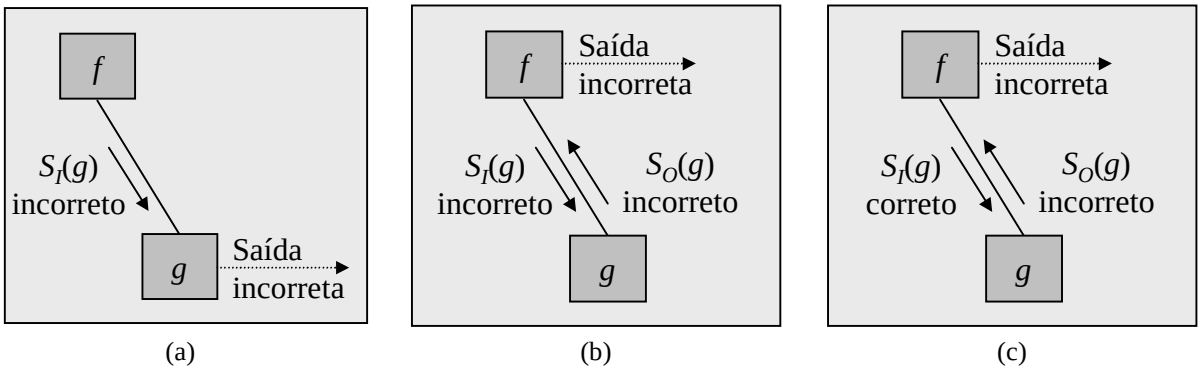
Segundo Haley e Zweben [11], os erros de integração podem ser classificados em erros de integração de domínio e computacional. Dada uma função *f* que chama *g*, o primeiro ocorre quando um erro de domínio[2] em *g* causa uma saída incorreta em *f*. O segundo ocorre quando um erro computacional em *g* produz um valor incorreto que é passado para *f* que, por sua vez, produz uma saída incorreta. Em ambos os casos existe algum valor incorreto sendo passado entre as unidades, o que resulta em uma saída incorreta. Considerando esses aspectos é possível classificar os erros de integração em três categorias.

Considere um programa *P* e um caso de teste *t* para *P*. Suponha que em *P* existam funções *f* e *g* tal que *f* chama *g*. Considere $S_I(g)$ como o conjunto de valores passados para *g* e $S_O(g)$ os valores retornados por *g*. Ao executar *P* com o caso de teste *t*, um erro de integração é identificado na chamada de *g* a partir de *f* quando [5]:

- Erro Tipo 1 (Figura 1 (a)): os valores contidos em $S_I(g)$ não são os esperados por *g*, influenciando a produção de saídas erradas antes do retorno de *g*. Esse tipo de erro ocorre, por exemplo, quando uma função é chamada com parâmetros incorretos fazendo com que a função chamada produza uma saída incorreta;
- Erro Tipo 2 (Figura 1 (b)): os valores contidos em $S_I(g)$ não são os esperados por *g*, desse modo, $S_O(g)$ assume valores errados fazendo com que *f* produza uma saída incorreta após o retorno de *g*. Um erro desse tipo pode ocorrer, por exemplo, quando um parâmetro incorreto passado para a função é utilizado para calcular o valor de retorno; e
- Erro Tipo 3 (Figura 1 (c)): os valores contidos em $S_I(g)$ são os esperados por *g*, mas valores incorretos em $S_O(g)$ são produzidos dentro de *g* e esses valores

[2] Um erro de domínio ocorre quando um caminho incorreto é executado; um erro computacional ocorre quando o caminho correto é executado mas o valor computado é incorreto.

fazem com que *f* produza um resultado incorreto após o retorno de *g*. Esse tipo de erro pode ocorrer se uma função é chamada com todos os parâmetros corretos, mas internamente ela realiza um cálculo incorreto produzindo um valor de retorno não esperado que, posteriormente, leva a um resultado incorreto.



(a) (b) (c)

Figura 1 – Tipos de Erros de Integração [5]: (a) Erro Tipo 1, (b) Erro Tipo 2, (c) Erro Tipo 3

Percebe-se que esta classificação dos tipos de erros é abrangente e não especifica o local do defeito que causa o erro. Ela simplesmente considera a existência de um valor incorreto entrando ou saindo de uma função chamada. Isso exclui, por exemplo, o caso em que $S_I(g)$ tem os valores esperados mas um erro dentro de *g* produz uma saída incorreta antes do retorno de *g*. Neste caso, não existe nenhuma propagação de erro através da conexão *f-g* e esse tipo de erro deveria ser detectado no teste de unidade.

Operadores de mutação destinados ao teste de unidade possuem semelhanças e diferenças em relação aos operadores de mutação destinados ao teste de integração. A idéia básica de ambos é a mesma, ou seja, introduzir modificações sintáticas no programa em teste transformando-o em programas similares: mutantes. Por outro lado, os operadores de mutação de interface estão relacionados a uma conexão entre duas unidades. Desse modo, os operadores, quando utilizados para testar uma conexão *f-g*, são aplicados de dois modos diferentes: 1) nos pontos onde *f* chama *g*; e 2) nos pontos dentro de *g* relacionados com a interface de comunicação entre as unidades. No segundo caso é necessário um mecanismo adicional que permita identificar o ponto a partir do qual *g* foi chamada. Visto que somente a conexão *f-g* está sendo testada, a mutação só deve estar ativa se *g* foi chamada a partir de *f*. De outro modo, *g* deve se comportar da mesma forma que no programa original. Essa característica pode requerer, em linguagens tais como C, que a decisão de aplicar ou não a mutação seja tomada em tempo de execução [5].

No experimento descrito neste artigo foi utilizada a ferramenta PROTEUM/IM [5]. Os 33 operadores de mutação implementados na ferramenta estão divididos em dois grupos, de acordo com o modo com que as mutações são realizadas em nível de integração. Dada uma conexão *f-g*, o Grupo-I aplica mutações dentro do corpo da unidade *g*; e o Grupo-II realiza mutações nos pontos onde a unidade *f* faz chamadas à *g*. Procurando estabelecer uma relação entre os dois grupos de operadores de mutação de interface e os quatro grupos de operadores de mutação de unidade (Comandos, Operadores, Variáveis e Constantes), os operadores dos grupos I e II foram subdivididos em cinco classes conforme apresentado na Tabela 1 [24]. Dentre as funcionalidades fornecidas pela ferramenta destacam-se os recursos para selecionar os operadores a serem utilizados, bem como a porcentagem de aplicação de cada um deles, o que viabiliza o estudo e avaliação de critérios alternativos [23, 26, 27]. Uma visão mais

detalhada sobre o critério Mutação de Interface e a ferramenta ROTEUM/IM pode ser encontrada em [5].

Tabela 1 – Operadores de Mutação de Interface[3]

Grupo-I-comandos	
I-CovAllEdg	Garante cobertura de desvios.
I-CovAllNod	Garante cobertura de nós.
I-RetStaDel	Elimina comando *return*.
I-RetStaRep	Troca comando *return*.
Grupo-I-variáveis	
I-DirVarAriNeg	Acrescenta negação aritmética em variáveis de interface.
I-DirVarBitNeg	Acrescenta negação de bit em variáveis de interface.
I-DirVarLogNeg	Acrescenta negação de lógica em variáveis de interface.
I-IndVarAriNeg	Acrescenta negação aritmética em variáveis não de interface.
I-IndVarBitNeg	Acrescenta negação de bit em variáveis não de interface.
I-IndVarLogNeg	Acrescenta negação de lógica em variáveis não de interface.
Grupo-I-variáveis	
I-DirVarIncDec	Acrescenta incremento (++) e decremento (--) em variável de interface.
I-DirVarRepCon	Troca variáveis de interface por elementos de C.
I-DirVarRepExt	Troca variáveis de interface por elementos de E.
I-DirVarRepGlo	Troca variáveis de interface por elementos de G.
I-DirVarRepLoc	Troca variáveis de interface por elementos de L.
I-DirVarRepPar	Troca variáveis de interface por elementos de P.
I-DirVarRepReq	Troca variáveis de interface por elementos de R.
I-IndVarIncDec	Acrescenta incremento (++) e decremento (--) em variável não de interface.
I-IndVarRepCon	Troca variáveis não de interface por elementos de C.
I-IndVarRepExt	Troca variáveis não de interface por elementos de E.
I-IndVarRepGlo	Troca variáveis não de interface por elementos de G.
I-IndVarRepLoc	Troca variáveis não de interface por elementos de L.
I-IndVarRepPar	Troca variáveis não de interface por elementos de P.
I-IndVarRepReq	Troca variáveis não de interface por elementos de R.
Grupo-II-comandos	
II-FunCalDel	Elimina chamada de função.
Grupo-II-argumentos	
II-ArgAriNeg	Acrescenta negação aritmética antes de argumento.
II-ArgBitNeg	Acrescenta negação de bit antes de argumento.
II-ArgLogNeg	Acrescenta negação lógica antes de argumento.
II-ArgDel	Elimina argumento.
II-ArgIncDec	Incrementa e decrementa argumento.
II-ArgRepReq	Troca argumentos por elementos de R.
II-ArgStcAli	Troca posição de argumentos de tipos compatíveis.
II-ArgStcDif	Troca posição de argumentos de tipos diferentes.

3. Conjunto Essencial de Operadores de Mutação

Como mencionado anteriormente, um dos maiores problemas relacionados ao teste de mutação é o grande número de mutantes gerados. Originalmente, o teste de mutação, em nível de unidade, foi denominado Análise de Mutantes (AM) e refere-se à aplicação de todos os

[3] Abreviações utilizadas na descrição dos operadores:
P – parâmetros formais utilizados na chamada da função; *L* – variáveis declaradas na função chamada;
G – variáveis globais utilizadas na função chamada; *C* – constantes utilizadas na função chamada; e
E – variáveis globais não utilizadas na função chamada; *R* – constantes requeridas.

operadores de mutação para o teste de um produto. Procurando solucionar o problema do grande número de mutantes gerados, algumas estratégias foram desenvolvidas, dentre elas o desenvolvimento de abordagens alternativas para sua utilização. Mutação Aleatória (*Randomly Selected X% Mutation*) [1], Mutação Restrita (*Constrained Mutation*) [16] e Mutação Seletiva (*Selective Mutation*) [19] são algumas dessas abordagens. O efeito da aplicação dessas abordagens tem sido investigado empiricamente [5, 17, 23, 26, 27] e os resultados demonstram que é possível reduzir o custo do teste de mutação sem que haja uma redução significativa de sua eficácia em revelar a presença de erros.

Concretamente, o que se almeja é a determinação de um subconjunto *SC* de operadores de mutação de forma que, conseguindo-se um conjunto de casos de teste *T* capaz de distinguir os mutantes gerados pelos operadores de *SC* (conjunto *T SC*-adequado), *T* seja capaz de distinguir todos os mutantes não equivalentes gerados pelo conjunto total de operadores de mutação definidos [3], ou seja, *T* seja AM-adequado. Nessa perspectiva, alguns trabalhos que vêm sendo desenvolvidos dizem respeito à determinação de conjuntos essenciais de operadores de mutação. Offutt *et al*. [20], a partir dos operadores de mutação da ferramenta *Mothra*[4] [9], determinaram um conjunto essencial de operadores de mutação para a linguagem Fortran. Wong *et al*. [27] compararam a Mutação Restrita no contexto das linguagens C e Fortran, e os resultados obtidos forneceram indícios de que os operadores da ferramenta *Proteum* [4] utilizados no experimento possuem uma forte relação com os operadores essenciais obtidos por Offutt *et al*. [20] para a linguagem Fortran.

Outro trabalho relevante nessa direção foi o desenvolvimento de um procedimento, denominado Essencial, para a determinação de conjuntos essenciais de operadores de mutação [2, 3]: a partir dos operadores de mutação implementados na ferramenta *Proteum*, foi conduzido um experimento no qual o procedimento desenvolvido foi aplicado a um grupo de programas, resultando em um conjunto essencial de operadores de mutação para a linguagem C em nível de unidade. O conjunto essencial obtido (CE_{AM}), ilustrado na Tabela 2, é formado por 9 dos 71 operadores de mutação de unidade. Mais detalhes a respeito do procedimento e do conjunto essencial de operadores de mutação de unidade para a linguagem C podem ser encontrados em [2, 3]. Os resultados obtidos com a aplicação do procedimento em nível de unidade mostraram-se bastante satisfatórios: 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.

Tabela 2 – Operadores Essenciais de Unidade da Linguagem C [2]

Operador	Descrição
SWDD	Substitui o comando *while* por *do-while*.
SMTC	Interrompe a execução do laço após duas execuções.
SSDL	Retira um comando de cada vez do programa.
OLBN	Substitui operador lógico por operador *bitwise*.
ORRN	Substitui operador relacional por operador relacional.
VTWD	Substitui referência escalar pelo seu valor sucessor e predecessor.
VDTR	Requer os valores negativo, positivo e zero para cada referência escalar.
Cccr	Substitui constantes por constantes.
Ccsr	Substitui referências escalares por constantes.

[4] A ferramenta *Mothra* [9] apóia a aplicação do teste de mutação em nível de unidade para programas escritos em Fortran.

4. Conjunto Essencial de Operadores de Mutação de Interface: Um Estudo de Caso

Para avaliar a aplicação do procedimento Essencial em nível de integração foram selecionados 5 programas utilitários do UNIX, já utilizados em outros experimentos [3, 24, 25, 27]. Para cada programa foram elaborados 11 conjuntos de casos de teste adequados ao critério Mutação de Interface (*IM*-adequados). Além disso, considerou-se a divisão dos operadores proposta por [24], apresentada na Tabela 1 da Seção 2. A Tabela 3 mostra a descrição e a complexidade dos programas utilizados em termos do total de linhas de código (LOC) e do número de conexões. Como pode ser observado, os resultados foram obtidos a partir de programas simples e pequenos. Sendo assim, mais experimentos nessa direção são necessários para que os resultados possam ser comparados e um parâmetro prático para a condução da atividade de teste seja obtido.

Tabela 3 – Conjunto de Programas do Experimento

Programas	Descrição	LOC	Conexões
Cal	Apresenta um calendário para o ano ou mês especificado.	119	4
Checkeq	Informa os delimitadores ausentes ou desbalanceados e pares .EQ/.EN.	76	1
Comm	Seleciona ou rejeita linhas comuns entre dois arquivos.	119	7
Look	Procura palavras em um dicionário ou linhas em uma lista ordenada.	107	3
Uniq	Informa ou remove linhas adjacentes duplicadas.	103	5

Uma informação fundamental para a aplicação do procedimento Essencial diz respeito à capacidade de conjuntos de casos de teste adequados a cada operador em distinguir os mutantes gerados pelos demais. A Tabela 4 apresenta parte desta informação coletada para um dos conjuntos de casos de teste *IM*-adequados do programa *Cal*. Por limitações de espaço, em vez do nome dos operadores de mutação, cada coluna faz referência ao número da linha na qual o operador se encontra. Por exemplo, a coluna 1 refere-se ao operador da primeira linha I-CovAllEdg, a coluna 2 ao operador I-CovAllNod, e assim sucessivamente até a coluna 33 que se refere ao operador da última linha da tabela II-FunCalDel. Considerando a quarta linha da Tabela 4, tem-se que um conjunto de casos de teste I-DirVarBitNeg-adequado determina escores de mutação de 0,91, 0,95, 0,97, 1,00, 1,00, 1,00 e 1,00 em relação aos operadores I-CovAllEdg, I-CovAllNod, I-DirVarAriNeg, I-DirVarBitNeg, II-ArgStcAli, II-ArgStcDif e II-FunCalDel, respectivamente.

Para um dado programa, a Tabela 4 foi gerada 11 vezes, correspondendo aos 11 conjuntos de casos de teste *IM*-adequados. Em seguida, foi gerada a Tabela 5, contendo o escore médio obtido pelos os 11 conjuntos de casos de teste.

Tabela 4 – Escore de Mutação que cada Operador Determina em Relação aos Demais: Programa *Cal*

Linha	Operador	1	2	3	4	...	31	32	33
1	I-CovAllEdg	1,000	1,000	1,000	1,000	...	1,000	1,000	1,000
2	I-CovAllNod	1,000	1,000	1,000	1,000	...	1,000	1,000	1,000
3	I-DirVarAriNeg	0,930	0,970	1,000	1,000	...	1,000	1,000	1,000
4	I-DirVarBitNeg	0,910	0,950	0,970	1,000	...	1,000	1,000	1,000
...	...	...	...	...	...	...	...	...	...
31	II-ArgStcAli	0,790	0,880	0,800	0,850	...	1,000	1,000	1,000
32	II-ArgStcDif	0,790	0,880	0,800	0,850	...	1,000	1,000	1,000
33	II-FunCalDel	0,790	0,880	0,800	0,850	...	1,000	1,000	1,000

Considerando-se que o *benchmark* é composto de cinco programas, cinco tabelas semelhantes à Tabela 5 sintetizando os resultados de cada programa foram geradas. A Tabela 6 apresenta a média obtida para os cinco programas. Observe que os operadores aparecem

ordenados de forma decrescente pela média, ou seja, do operador que mais inclui os demais para o que menos inclui. As linhas da Tabela 6 apresentam uma visão parcial sobre a capacidade de conjuntos de casos de teste adequados a determinado operador em distinguir os mutantes gerados pelos demais. Por exemplo, considerando o operador I-IndVarIncDec (segunda linha) tem-se que, em geral, o escore de mutação médio do conjunto I-IndVarIncDec-adequado em relação a I-IndVarBitNeg (terceira coluna) é de 0,981, indicando que o conjunto I-IndVarIncDec-adequado é capaz de distinguir, em média, 98,10% dos mutantes de I-IndVarBitNeg. O escore médio de um operador em relação ao total é obtido através da média geral de uma determinada linha da tabela. Por exemplo, o escore médio que conjuntos de casos de teste I-IndVarIncDec-adequados determinam em relação aos demais operadores é de 0,970.

Tabela 5 – Escore Médio dos 11 Conjuntos *IM*-adequados: Programa *Cal*

Linha	Operador	1	2	3	4	...	31	32	33
1	I-CovAllEdg	1,000	1,000	1,000	1,000	...	1,000	1,000	1,000
2	I-CovAllNod	1,000	1,000	1,000	1,000	...	1,000	1,000	1,000
3	I-DirVarAriNeg	0,901	0,944	1,000	0,995	...	1,000	1,000	0,997
4	I-DirVarBitNeg	0,897	0,940	0,995	1,000	...	1,000	1,000	0,995
...	...	...	...	...	...	...	...	...	...
31	II-ArgStcAli	0,773	0,864	0,836	0,840	...	1,000	1,000	0,978
32	II-ArgStcDif	0,773	0,864	0,836	0,840	...	1,000	1,000	0,978
33	II-FunCalDel	0,796	0,880	0,845	0,845	...	1,000	1,000	1,000

Tabela 6 – Média dos Escores e *Strength* dos Operadores para o Total de Programas do Experimento[5]

Linha	Operador	1	2	3	4	...	31	32	33	Média Geral	Desvio Padrão
1	I-IndVarRepReq	0,968	0,967	0,981	0,925	...	1,000	1,000	0,876	0,973	0,030
2	I-IndVarIncDec	0,959	0,964	0,981	0,924	...	1,000	1,000	0,876	0,970	0,030
3	I-IndVarBitNeg	0,958	0,961	0,989	0,926	...	1,000	1,000	0,875	0,968	0,029
4	I-IndVarLogNeg	0,959	0,965	0,963	0,915	...	1,000	1,000	0,876	0,964	0,033
...	...	...	...	...	...	...	...	...	...	...	...
31	I-RetStaDel	0,754	0,790	0,604	0,610	...	0,676	0,836	0,634	0,667	0,131
32	II-ArgDel	0,686	0,774	0,417	0,444	...	1,000	1,000	0,854	0,657	0,212
33	II-ArgStcAli	0,325	0,363	0,284	0,305	...	1,000	0,696	0,401	0,393	0,202
Média Geral		0,871	0,898	0,850	0,812	...	0,903	0,961	0,841	-	-
Desvio Padrão		0,129	0,114	0,185	0,176	...	0,129	0,082	0,107	-	-

Analisando-se uma determinada coluna da Tabela 6 obtém-se o escore de mutação de conjuntos adequados a cada operador em relação a um determinado operador *op*, ou seja, a capacidade de conjuntos de casos de teste adequados a cada operador em distinguir os mutantes de *op*. A partir desta informação é calculado o *strength* de *op*:

strength = (1 - escore de mutação em relação a *op*).

Por exemplo, considerando o operador I-RetStatDel (31ª coluna) tem-se que o *strength* deste operador em relação a I-IndVarRepReq (1ª linha) é zero (1 - 1,000), indicando que o conjunto I-IndVarRepReq-adequado é também I-RetStaDel-adequado. Já o operador I-IndVarLogNeg (4ª coluna) apresenta um *strength* em relação ao operador II-ArgDel de 0,556 (1 - 0,444), ou seja, o conjunto II-ArgDel-adequado não é capaz de distinguir 55,60% dos mutantes gerados por I-IndVarLogNeg. Da mesma forma que o escore, tem-se que o *strength*

[5] Os valores da média geral e desvio padrão referem-se a todos os operadores de mutação e não somente aos que estão representados na tabela.

médio de determinado operador em relação aos demais é obtido através da média geral de determinada coluna. Por exemplo, o *strength* médio do operador I-IndVarLogNeg (quarta coluna) é de 0,216 (1 - 0,812).

Os operadores da Tabela 6 foram ordenados segundo a média geral de escore e *strength*, dando origem às tabelas 7(a) e 7(b), respectivamente. Além do escore de mutação e do *strength*, outra informação requerida para a aplicação do procedimento Essencial [2, 3] é o custo (dado pelo número de mutantes gerados) associado a cada operador; essa informação também foi coletada e é apresentada Tabela 7(c).

Tabela 7 – Operadores Ordenados Segundo: (a) Escore, (b) *Strength* e (c) Custo

(a)

Operador	Escore
I-IndVarRepReq	0,973
I-IndVarIncDec	0,970
I-IndVarBitNeg	0,968
I-IndVarLogNeg	0,964
I-IndVarRepCon	0,961
I-DirVarIncDec	0,951
I-DirVarRepReq	0,949
I-DirVarRepCon	0,945
I-IndVarRepPar	0,943
...	...
I-RetStaDel	0,667
II-ArgDel	0,657
II-ArgStcAli	0,393

(b)

Operador	*Strength*
I-IndVarRepLoc	0,249
I-IndVarAriNeg	0,235
I-IndVarLogNeg	0,216
I-IndVarRepPar	0,214
I-DirVarRepCon	0,210
I-IndVarRepCon	0,203
I-DirVarRepPar	0,199
I-IndVarRepReq	0,196
I-IndVarBitNeg	0,194
...	...
II-ArgStcDif	0,039
II-ArgLogNeg	0,031
II-ArgRepReq	0,010

(c)

Operador	Custo
II-ArgStcAli	21
II-ArgAriNeg	26
II-ArgBitNeg	26
II-ArgLogNeg	26
I-RetStaRep	42
II-ArgStcDif	43
II-ArgRepReq	55
I-RetStaDel	65
II-ArgDel	92
...	...
I-DirVarRepCon	1380
I-IndVarRepReq	1391
I-IndVarRepCon	2555

4.1. Aplicação do Procedimento Essencial

A seguir, os passos do procedimento são descritos e aplicados com o objetivo de determinar o conjunto essencial de operadores de mutação de interface da ferramenta PROTEUM/IM. Informações mais detalhadas podem ser obtidas em [24].

Passo 1: Selecionar operadores que determinem alto escore de mutação

A seleção dos operadores que determinam alto escore de mutação depende de uma variável que deve ser instanciada pelo testador em função do conjunto de dados coletados. Tal variável, definida como Índice Médio de Escore de Mutação (*IMES*), estabelece o limite máximo que define se um operador apresenta ou não um alto escore médio em relação aos demais.

Para o experimento em questão considerou-se *IMES* = 0,960, sendo que os cinco primeiros operadores da Tabela 7(a) foram selecionados para compor CE_{pre} (Conjunto Essencial Preliminar). Ao final deste passo CE_{pre} = {I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon}.

Passo 2: Procurar selecionar um operador de cada classe de mutação

Deve-se procurar adicionar a CE_{pre} um operador de cada classe de mutação que ainda não possua representantes em CE_{pre}. Entretanto, como determinado operador pode ser incluído empiricamente[6] por um subconjunto de operadores, um operador *op* pertencente a

[6] Como define Barbosa [2], na prática, por limitações de tempo e custo, obter-se um escore de mutação próximo de 1 pode ser satisfatório. Seja *ms* um escore de mutação definido pelo testador. Dados um critério *C* e um conjunto de casos de teste *T*, diz-se que *T* é empiricamente adequado a *C* (*T* é *C*-adequado*) se *T* obtiver um escore de mutação igual ou superior a *ms*. Diz-se que C_1 inclui

uma determinada classe de mutação que ainda não possui representantes em CE_{pre} só deve ser adicionado à CE_{pre} se o conjunto de casos de teste T CE_{pre}-adequado apresentar um escore de mutação inferior a *ms* (escore de mutação definido pelo testador) em relação a *op*.

A seleção do operador de mutação de determinada classe é feita respeitando a ordem estabelecida na Tabela 7(a). Por exemplo, o conjunto CE_{pre} do Passo 1 só apresenta operadores das classes G-I-operadores e G-I-variáveis. Respeitando a ordem em que os operadores aparecem na Tabela 7(a), o primeiro operador que não pertence às classes G-I-operadores e G-I-variáveis é o operador I-CovAllNod da classe G-I-comandos. Considerando *ms* = 0,99, tem-se que I-CovAllNod não é incluído empiricamente pelos operadores de CE_{pre} – o escore de mutação que conjuntos de casos de teste CE_{pre}-adequados determinam em relação a I-CovAllNod é de 0,968 – e, desse modo, I-CovAllNod passa a fazer parte de CE_{pre}.

Este processo foi repetido procurando adicionar a CE_{pre} operadores das demais classes. Ao final deste passo CE_{pre} = {I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon, I-CovAllNod, II-ArgAriNeg}.

Passo 3: Avaliar inclusão empírica entre operadores

À medida que novos operadores são adicionados a CE_{pre}, pode ocorrer que a combinação de conjuntos de casos de teste adequados a cada operador passe incluir empiricamente outro operador presente em CE_{pre}, ou seja, podem existir em CE_{pre} operadores que não estejam colaborando para a melhoria do conjunto de casos de teste (o operador seleciona o mesmo conjunto de casos de teste dos demais), elevando o custo de aplicação do conjunto em termos do número de mutantes gerados. Desse modo, neste passo, cada operador pertencente a CE_{pre} é avaliado em relação aos demais operadores pertencentes a esse conjunto de modo a eliminar os operadores de mutação que sejam incluídos empiricamente. Considerando o conjunto CE_{pre} do passo anterior e definindo *ms* = 0,99, obtém-se CE_{pre} = {I-IndVarRepReq, I-IndVarBitNeg, I-CovAllNod, II-ArgAriNeg}.

Passo 4: Estabelecer uma estratégia incremental de aplicação

Dado o custo de aplicação e os requisitos de teste específicos que cada classe de operadores determina, é interessante que seja estabelecida uma estratégia incremental de aplicação entre os operadores do conjunto essencial. A idéia é aplicar, primeiramente, os operadores relevantes a determinados requisitos mínimos de teste (por exemplo, em nível de unidade, cobertura de todos os nós e todos os arcos) e, em seguida, em função do grau de adequação que se quer atingir e da disponibilidade de orçamento e tempo, aplicar os demais operadores relacionados a outros conceitos e requisitos [2]. Desse modo, considerando as cinco classes de operadores, estas foram ordenadas de modo a garantir, inicialmente, que requisitos mínimos do teste de integração fossem satisfeitos e, posteriormente, requisitos mais rigorosos:

1. G-II-comandos – garante que cada chamada de função do programa é necessária;
2. G-II-argumentos – garante que os argumentos utilizados nas chamadas de função estejam corretos;
3. G-I-comandos – garante que comandos de retorno estão nos locais corretos e que todos são necessários. Além disso, nessa classe encontram-se os operadores que

---

empiricamente C_2 com um determinado escore *ms* ($C_1 \Rightarrow^{*} C_2$) se para todo conjunto de casos de teste T_1 C_1-adequado, T_1 é C_2-adequado* para um dado *ms* e existe um T_2 C_2-adequado que não é C_1-adequado*; C_1 e C_2 são equivalentes* se para qualquer T C_1-adequado, T é C_2-adequado* e vice-versa.

garantem a cobertura de todos os comandos e todos os desvios da função chamada a partir de cada ponto de chamada;

4. G-I-operadores – essa classe de operadores é responsável pela inserção de operadores unários lógico, aritmético e de bit em cada uso de variáveis ou constantes; e
5. G-I-variáveis – essa classe de operadores perturba um valor que entra ou sai da função chamada trocando diretamente variáveis de interface e não de interface por outras variáveis e constantes.

Ressalta-se que a ordenação definida neste passo deve ser respeitada no decorrer do procedimento, quando novos operadores tiverem que ser adicionados ao conjunto essencial. Após a aplicação deste passo CE_{pre} = {II-ArgAriNeg, I-CovAllNod, I-IndVarBitNeg, I-IndVarRepReq}.

Passo 5: Selecionar operadores que proporcionem incremento no escore de mutação

De acordo com Barbosa [2], deve-se procurar adicionar um certo número *x* de operadores de mutação (definido pelo testador) de cada uma das classes desde que tais operadores proporcionem um incremento no escore de mutação igual ou superior a um Índice de Incremento Mínimo (*IIM*) que também deve ser definido em função do conjunto de programas sendo utilizado no experimento. Os operadores são inseridos até que *x* operadores de cada classe tenham sido adicionados ou que não existam operadores que proporcionem incremento igual ou superior a *IIM*.

No experimento em questão optou-se por adicionar mais 3 operadores de cada classe (x = 3) desde que tais operadores proporcionassem um incremento no escore de mutação superior a 0,001 (*IIM* = 0,001). Os operadores devem ser adicionados conforme a ordem estabelecida no Passo 4, ou seja: G-II-comandos, G-II-argumentos, G-I-comandos, G-I-operadores e G-I-variáveis. A Tabela 8 apresenta o incremento proporcionado pelos operadores de mutação na primeira iteração deste passo.

Tabela 8 – Incremento Proporcionado pelos Operadores

Operador	Índice de Incremento	Escore CE_{pre} + Incremento
I-DirVarRepCon	0,00667	0,99146
I-DirVarRepReq	0,00611	0,99114
I-DirVarRepPar	0,00571	0,99038
I-DirVarRepGlo	0,00555	0,99004
I-DirVarIncDec	0,00523	0,99028
I-DirVarRepLoc	0,00513	0,99004
I-DirVarBitNeg	0,00489	0,98996
I-IndVarRepExt	0,00449	0,97757
I-IndVarRepLoc	0,00418	0,98890
I-IndVarRepCon	0,00259	0,97563
I-IndVarRepGlo	0,00259	0,98765
I-IndVarRepPar	0,00224	0,98731
I-DirVarRepExt	0,00182	0,98650
I-IndVarIncDec	0,00142	0,97420
II-FunCalDel	0,00068	0,97376
I-CovAllEdg	0,00014	0,98475

Considerando a Tabela 8 e a ordenação proposta no Passo 4, a primeira classe a ser analisada é G-II-comandos; entretanto, o único operador dessa classe não proporciona incremento superior a *IIM*. No caso, o operador II-FunCalDel apresenta um incremento de 0,00068. O mesmo ocorre com os operadores da classe G-II-argumentos e G-I-comandos, de modo que a próxima classe candidata é G-I-operadores. Considerando essa classe tem-se que

o único representante com incremento superior a *IIM* é I-DirVarBitNeg e como tal operador não é incluído empiricamente pelos operadores de CE_{pre}, este é adicionado CE_{pre}.

A cada operador adicionado a CE_{pre}, as informações apresentadas na Tabela 8 devem ser coletadas novamente. Esse processo foi repetido mais três vezes e somente operadores da classe G-I-variáveis foram adicionados. Ao final deste passo, CE_{pre} = {II-ArgAriNeg, I-CovAllNod, I-DirVarBitNeg, I-IndVarBitNeg, I-IndVarRepGlo, I-IndVarRepExt, I-IndVarRepLoc, I-IndVarRepReq}.

Passo 6: Selecionar operadores de alto *strength*

A seleção de operadores de alto *strength* é feita da mesma forma que a seleção dos operadores de alto escore de mutação, ou seja, através de um índice definido pelo testador (Índice Médio de *Strength – IMS*) em função do conjunto de programas utilizados. Existindo operadores de alto *strength*, esses só são adicionados a CE_{pre} se não forem incluídos empiricamente pelos operadores de CE_{pre}.

Para o experimento em questão considerou-se *IMS* = 0,70. De acordo com a Tabela 7(b), que apresenta o *strength* de parte dos operadores de mutação, percebe-se que nenhum deles possui *strength* igual ou superior a *IMS* e, desse modo, nenhum operador de mutação foi adicionado a CE_{pre} neste passo. Ao final deste passo foi obtido o conjunto essencial CE_{IM}, descrito no Tabela 9.

Tabela 9 – Descrição dos Operadores Essenciais de Interface da Linguagem C: CE_{IM}

Operador	Descrição
II-ArgAriNeg	Acrescenta negação aritmética antes de argumento.
I-CovAllNod	Garante cobertura de nós.
I-DirVarBitNeg	Acrescenta negação de bit em variáveis de interface.
I-IndVarBitNeg	Acrescenta negação de bit em variáveis não de interface.
I-IndVarRepGlo	Troca variável não de interface por variáveis globais utilizadas na função chamada.
I-IndVarRepExt	Troca variável não de interface por variáveis globais não utilizadas na função chamada.
I-IndVarRepLoc	Troca variável não de interface por variáveis locais, declaradas na função chamada.
I-IndVarRepReq	Troca variável não de interface por constantes requeridas.

4.2. Análise dos Dados

Na Tabela 10 e no Gráfico 1 são apresentados o escore de mutação e a redução de custo proporcionada pelo conjunto CE_{pre}, em cada passo do procedimento.

Tabela 10 – Resultados Obtidos a cada Passo do Procedimento: CE_{IM}

Passo	Operadores	Escore	Redução de Custo (%)
1	{I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon}	0,97768	62,621
2	{I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon, I-CovAllNod, II-ArgAriNeg}	0,98698	59,313
3	{I-IndVarRepReq, I-IndVarBitNeg, I-CovAllNod, II-ArgAriNeg}	0,98507	84,591
4	{II-ArgAriNeg, I-CovAllNod, I-IndVarBitNeg, I-IndVarRepReq}	0,98507	84,591
5	{II-ArgAriNeg, I-CovAllNod, I-DirVarBitNeg, I-IndVarBitNeg, I-IndVarRepGlo, I-IndVarRepExt, I-IndVarRepLoc, I-IndVarRepReq}	0,99813	73,217
6	{II-ArgAriNeg, I-CovAllNod, I-DirVarBitNeg, I-IndVarBitNeg, I-IndVarRepGlo, I-IndVarRepExt, I-IndVarRepLoc, I-IndVarRepReq}	0,99813	73,217

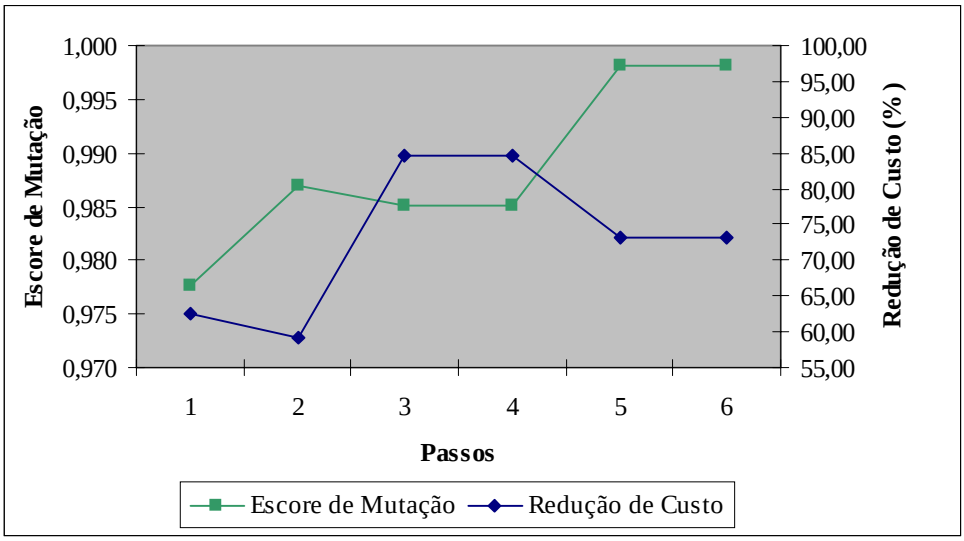


Gráfico 1 – Evolução do Conjunto Essencial Durante os Passos do Procedimento: CE_{IM}

Observa-se que do Passo 1 ao Passo 2 ocorreu um aumento no escore de mutação e um decréscimo na redução de custo devido à inserção de mais dois operadores ao conjunto essencial (I-CovAllNod e II-ArgAriNeg). No Passo 3, no qual são avaliadas as possíveis relações de inclusão entre os operadores de CE_{pre}, três operadores foram removidos ocasionando uma pequena redução no escore de mutação com uma sensível redução no custo de aplicação de CE_{pre}. Em seguida, no Passo 4, nenhuma mudança no escore de mutação e no custo foi observada, pois os operadores de CE_{pre} foram apenas ordenados. No Passo 5, com a inserção de mais quatro operadores (um da classe G-I-operadores e três da classe G-I-variáveis), o escore de mutação obtido foi superior a 0,998, mantendo-se a redução de custo em torno de 73%. No Passo 6, como não foram selecionados operadores de alto *strength*, o escore e a redução de custo permaneceram os mesmos em relação ao passo anterior.

Na próxima seção é apresentada uma estratégia incremental de aplicação do teste de mutação nas fases do teste de unidade e de integração, utilizando os conjuntos essenciais de operadores de mutação de unidade e de interface, respectivamente.

5. Estratégia Essencial de Teste

Explorando o aspecto complementar dos critérios Análise de Mutantes e Mutação de Interface, algumas estratégias de aplicação dos operadores de mutação de unidade e integração foram estabelecidas [24]. Tais estratégias demonstraram que, mesmo com um número reduzido de operadores, é possível determinar conjuntos de casos de teste adequados ou muito próximos da adequação para ambos os critérios, a um menor custo. A Figura 2 ilustra possíveis estratégias incrementais com o objetivo de utilizar os critérios baseados em mutação de forma incremental e complementar dentro das fases do teste de unidade e de integração. Por exemplo, a estratégia (1) indica como, a partir de um conjunto de casos de teste $T_{\text{AM-adequado}}$ evoluir para um conjunto de casos de teste $T_{\text{IM-adequado}}$. A idéia é que, em geral, parte dos requisitos do teste de mutação em nível de integração é coberta por um conjunto adequado à Análise de Mutantes e, desse modo, para tornar tal conjunto adequado ao teste de integração não seria necessário utilizar todos os operadores de mutação de interface, reduzindo o custo de aplicação do critério. Neste artigo somente a estratégia (3), denominada Estratégia Essencial, será investigada. Informações mais detalhadas a respeito das demais estratégias podem ser obtidas em [24].

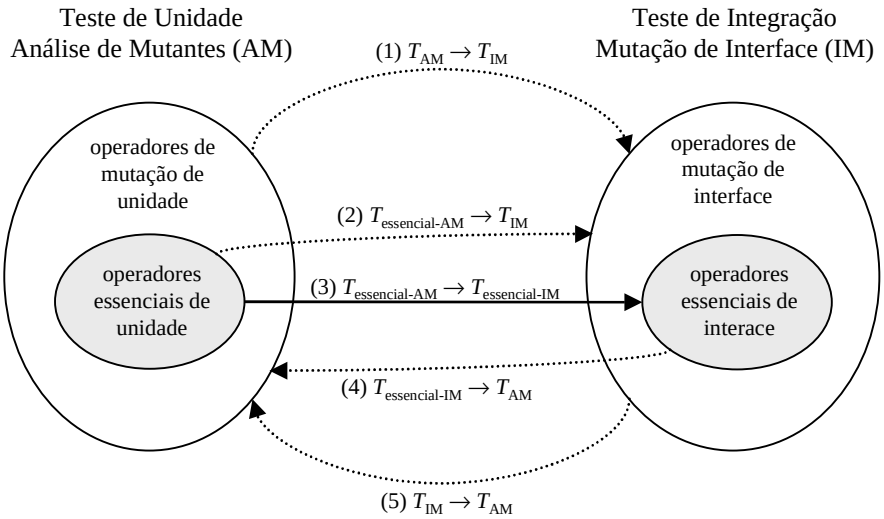


Figura 2 – Estratégias Incrementais de Aplicação do Teste de Mutação

Uma característica importante observada em [24] é que, mesmo obtido um conjunto $T_{\text{AM-adequado}}$, grande parte dos mutantes de interface permanecem vivos. Dentre esses estão os mutantes gerados por 7 dos 8 operadores de CE_{IM} (I-CovAllNod, I-DirVarBitNeg, I-IndVarBitNeg, I-IndVarRepGlo, I-IndVarRepExt, I-IndVarRepLoc e I-IndVarRepReq). Da mesma forma, obtido um conjunto $T_{\text{IM-adequado}}$, 7 dos 9 operadores essenciais de unidade determinados por Barbosa [3] também são necessários para se chegar a conjuntos AM-adequados. Com isso pode-se observar que os operadores do conjunto essencial de unidade e de interface apresentam alto *strength* um em relação ao outro.

Assim sendo, uma alternativa viável, capaz de garantir que grande parte dos requisitos de teste de ambos os critérios sejam satisfeitos a um baixo custo, é a utilização, inicialmente, dos operadores do conjunto essencial tanto no teste de unidade quanto no teste de integração. A título de ilustração, a Tabela 11 apresenta o escore de mutação e a redução de custo (em termos do número de mutantes gerados) proporcionada com a utilização do conjunto essencial de unidade determinado por Barbosa (CE_{AM}) [3] e do conjunto essencial de interface (CE_{IM}) apresentado na Seção 5.1. Ambos os conjuntos são comparados com o número total de mutantes que seriam gerados pelo conjunto total de operadores de mutação de unidade e de interface, respectivamente.

Tabela 11 – Redução de Custo e Escore de Mutação Proporcionados pelo uso dos Conjuntos Essenciais

Programa	**Teste de Unidade – *Proteum***				**Teste de Integração – PROTEUM/IM**			
	Todos Operadores *AM*	Conjunto Essencial CE_{AM}	Redução de Custo (%)	Escore em Relação ao Total	Todos Operadores *IM*	Conjunto Essencial CE_{IM}	Redução de Custo (%)	Escore em Relação ao Total
Cal	4.332	2.127	50,900	0,99998	4.350	810	81,379	0,99808
Checkeq	3.099	1.449	53,243	0,99908	2.954	644	78,199	0,99994
Comm	1.728	576	66,667	0,99769	2.952	942	68,089	0,99747
Look	2.056	589	71,352	0,99126	1.825	732	59,890	0,99771
Uniq	1.619	476	70,599	1,00000	1.943	628	67,679	0,99747
Total	**12.839**	**5.217**	**59,350**	**0,99760**	**14.024**	**3.756**	**73,217**	**0,99813**

Observando a Tabela 11 tem-se que, em nível de unidade, para o grupo de programas utilizado, o conjunto essencial proporcionou uma redução média no custo de aplicação do teste de mutação da ordem de 59,35%, mantendo um escore de mutação em relação ao total de mutantes gerados de aproximadamente 0,99760. Em nível de integração, o escore de

mutação foi praticamente o mesmo (0,99813) e a redução de custo obtida com a utilização do conjunto essencial foi superior a 73%.

A Tabela 12 apresenta a redução média no número de mutantes que seria obtida se somente os conjuntos essenciais fossem utilizados em uma estratégia incremental de aplicação do teste de mutação, cobrindo as fases do teste de unidade e de integração.

Tabela 12 – Custo Total da Utilização do Teste de Mutação nas Fases de Unidade e de Integração

Programa	Todos Operadores *AM + IM*	Conjunto Essencial $CE_{AM} + CE_{IM}$	Redução de Custo (%)
Cal	8.682	2.937	66,171
Checkeq	6.053	2.093	65,422
Comm	4.680	1.518	67,564
Look	3.881	1.321	65,962
Uniq	3.562	1.104	69,006
Total	**26.863**	**8.973**	**66,597**

Outra alternativa seria aplicar os operadores do conjunto essencial de forma incremental, à medida que as restrições de tempo e custo permitirem. Por exemplo, considerando a ordem de aplicação dos operadores do conjunto essencial de unidade estabelecida por Barbosa [3], a Tabela 13 apresenta o escore de mutação obtido após a determinação de um conjunto de casos de teste adequado a determinado operador, o custo do operador em termos do número de mutantes gerados, o custo cumulativo da estratégia incremental e a redução de custo em relação ao total de mutantes gerados.

Tabela 13 – Estratégia Incremental de Aplicação dos Operadores de CE_{AM}

Operador	Escore	Custo do Operador	Custo Cumulativo	Redução de Custo (%)
SWDD	0,65246	18	18	99,85975
SMTC	0,84798	33	51	99,60262
SSDL	0,96695	446	497	96,12747
OLBN	0,97195	81	578	95,49634
ORRN	0,98731	515	1093	91,48356
VTWD	0,99297	422	1515	88,19542
VDTR	0,99572	633	2148	83,26321
Cccr	0,99720	1676	3824	70,20415
Ccsr	0,99762	1393	5217	59,35016

Considerando a primeira linha da Tabela 13 tem-se que um conjunto de casos de teste SWDD-adequado determina um escore de mutação de 0,65246 em relação ao conjunto total de operadores, sendo a redução de custo obtida de 99,85%. Aplicando-se o segundo operador, o escore de mutação aumenta para 0,84798 e a redução de custo mantém-se acima dos 99%. Desse modo o testador pode aplicar incrementalmente os operadores até obter um conjunto de casos de teste essencial-AM-adequado. Para o experimento em questão, obtido tal conjunto, o escore de mutação seria de aproximadamente 0,99762 e a redução de custo de 59,35%.

Partindo-se do conjunto de casos de teste essencial-AM-adequado, os operadores do conjunto essencial de interface também poderiam ser aplicados incrementalmente. Tal estratégia é apresentada na Tabela 14; observando-se tal tabela tem-se que, em média, o conjunto essencial-AM-adequado determina um escore de mutação de 0,91956 em relação ao critério Mutação de Interface. Em seguida, obtendo-se um conjunto de casos de teste adequado ao operador II-ArgAriNeg, o escore de mutação passa a ser de 0,92623 e a redução no custo de aplicação do critério Mutação de Interface é de aproximadamente 99,80%. Como pode ser observado, com a aplicação do quarto operador (I-IndVarBitNeg) já é possível obter um escore de mutação superior a 0,99 com uma redução de custo em torno de 93%. Aplicados

todos os operadores de CE_{IM} o escore e a redução de custo são de 0,998 e 73,21%, respectivamente.

Dessa forma, considerando a Estratégia Essencial, o testador pode, inicialmente, garantir a cobertura dos requisitos mínimos de teste e, de acordo com as restrições de tempo e custo, garantir a cobertura de requisitos mais rigorosos tanto em nível de unidade quanto em nível de integração.

Tabela 14 – Estratégia Incremental de Aplicação dos Operadores do Conjunto Essencial de Interface a partir de um Conjunto Essencial-AM-adequado

Operador	Escore	Custo do Operador	Custo Cumulativo	Redução de Custo (%)
T essencial-AM-adequado	0,91956	-	-	-
II-ArgAriNeg	0,92623	26	26	99,81460
I-CovAllNod	0,97257	438	464	96,69139
I-DirVarBitNeg	0,98229	154	618	95,59327
I-IndVarBitNeg	0,99126	306	924	93,41129
I-IndVarRepGlo	0,99381	312	1236	91,18654
I-IndVarRepExt	0,99718	511	1747	87,54278
I-IndVarRepLoc	0,99858	618	2365	83,13605
I-IndVarRepReq	0,99880	1391	3756	73,21734

6. Conclusão

O teste de mutação apresenta alguns problemas para a sua aplicação em ambientes reais de desenvolvimento de software, entre eles o grande número de mutantes gerados. O desenvolvimento de critérios alternativos que reduzam o número de mutantes gerados sem prejuízo de sua eficácia tem sido bastante explorado. Além disso, procurando aplicar os conceitos do teste de mutação em nível de integração, foi definido o critério Mutação de Interface.

O experimento descrito neste artigo contribuiu nesta perspectiva determinando um conjunto essencial de operadores de mutação de interface, obtido a partir da aplicação do procedimento Essencial, proposto por Barbosa [2]. O conjunto essencial de operadores de mutação de interface possibilitou a seleção de conjuntos de casos de teste com um bom índice de adequação em relação ao critério Mutação de Interface (escores em torno de 0,998) com uma redução de custo, em termos do número de mutantes gerados, superior a 73%.

A utilização dos conjuntos essenciais de unidade e de interface em uma estratégia de teste incremental, denominada Estratégia Essencial, englobando as fases de unidade e de integração, também foi investigada. Os resultados obtidos fornecem evidências de que essas estratégias constituem uma boa alternativa quando o testador necessita garantir que grande parte dos requisitos de teste requeridos pelo teste de mutação sejam satisfeitos a um baixo custo. Com a estratégia apresentada, é possível obter conjuntos de casos de teste que, tanto em nível de unidade quanto em nível de integração, determinam escores de mutação superiores a 0,997 e proporcionam uma redução no custo superior a 66%.

Tais resultados demonstram que o procedimento Essencial permite selecionar subconjuntos de operadores de mutação tanto em nível de unidade quanto em nível de integração, proporcionando uma sensível redução no custo e mantendo um alto grau de cobertura em relação ao conjunto total de operadores. Como continuidade deste trabalho, alguns experimentos utilizando outros programas, em diferentes domínios de aplicação, devem ser conduzidos. Esses experimentos visam a investigar outros aspectos tais como a

eficácia em revelar a presença de erros das estratégias apresentadas, bem como o custo e o *strength* da Estratégia Essencial em relação a outras estratégias de redução de custo (Mutação Aleatória e Mutação Restrita) contribuindo para que os critérios baseados em mutação possam ser empregados em ambientes reais de desenvolvimento, melhorando a qualidade e a confiabilidade dos produtos desenvolvidos.

Agradecimentos

Este trabalho recebeu apoio financeiro do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

Bibliografia

[1] ACREE, A.T.; BUDD, T.A.; DEMILLO, R.A.; LIPTON, R.J.; SAYWARD, F.G. *Mutation Analysis*. Relatório Técnico GIT-ICS-79/08, Georgia Institute of Technology, Atlanta, GA, Setembro, 1979.

[2] BARBOSA, E.F.; VINCENZI, A.M.R.; MALDONADO, J.C. "Uma Contribuição para a Determinação de um Conjunto Essencial de Operadores de Mutação no Teste de Programas C". *Anais do XII Simpósio Brasileiro de Engenharia de Software*, Maringá, PR, Brasil, Outubro, 1998.

[3] BARBOSA, E.F.; Uma Contribuição para a Determinação de um Conjunto Essencial de Operadores de Mutação no Teste de Programas C, Dissertação de Mestrado, ICMC/USP, São Carlos, SP, Novembro,1998.

[4] DELAMARO, M.E.; *Proteum – Um Ambiente de Teste Baseado na Análise de Mutantes*, Dissertação de Mestrado, ICMC/USP, São Carlos, SP, Outubro, 1993.

[5] DELAMARO, M.E. *Mutação de Interface: Um Critério de Adequação Interprocedimental para o Teste de Integração*. Tese de Doutorado, IFSC/USP, São Carlos, SP, Junho, 1997.

[6] DELAMARO, M.E.; MALDONADO, J.C.; MATHUR, A.P. "Interface Mutation: An Approach for Integration Testing", *IEEE Transactions on Software Engeneering*, (sendo submetido), 1998.

[7] DELAMARO, M.E.; MALDONADO, J.C.; PASQUINI, A.; MATHUR, A.P. "Interface Mutation Test Adequacy Criterion: An Empirical Evaluation", Relatório Técnico em preparação, 1998.

[8] DEMILLO, R.A. "Mutation Analysis as a Tool for Software Quality Assurance". *Anais da COMPSAC80*, Chicago, IL, Outubro, 1980.

[9] DEMILLO, R.A.; GWIND, D.S.; KING, K.N.; MCKRAKEN, W.N.; OFFUTT, A.J. "An Extended Overview of the Mothra Testing Environment". *Anais do II Workshop on Software Testing, Verification and Analysis*, Banff, Canadá, Julho, 1988.

[10] FRANKL, P.G.; WEYUKER, E.J. "A Formal Analysis of the Fault-Detecting Ability of Testing Methods". *IEEE Transactions on Software Engineering*, v. 19, n. 3, pp. 202-213, Março, 1993.

[11] HALEY, A.; ZWEBEN, S. "Development and Aplication of a White Box Approach to Integration Testing". *The Journal of Systems and Software*, n. 4, pp. 309-315, Abril, 1984.

[12] HARROLD, M.J.; SOFFA, M.L. “Selecting and Using Data for Integration Test”. *IEEE Software*, v. 8, n. 2, pp. 58-65, Março, 1991.

[13] LINNENKUGEL, U.; MÜLLERBURG, M. “Test Data Selection Criteria for (Software) Integration Testing”. *Anais da I International Conference on Systems Integration*, pp. 709-717, Mornstown, NJ, Abril, 1990.

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

[15] MALDONADO. J.C. *Critérios de Teste de Software: Aspectos Teóricos, Empíricos e de Automatização*. Livre Docência, ICMC-USP, São Carlos, SP, Janeiro, 1997.

[16] MATHUR, A.P.; “Performance, Effectiveness, and Reliability Issues in Software Testing”, *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, Tóquio, Japão, 1991, pp. 604-605.

[17] MATHUR, A.P.; WONG, W.E.; “Evaluation of the Cost of Alternate Mutation Strategies”, *VII SBES – Simpósio Brasileiro de Engenharia de Software*, Rio de Janeiro, RJ, Outubro, 1993.

[18] OFFUTT, A.J. “Coupling Effect: Fact or Fiction”, *Anais do III Symposium on Software Testing, Analysis and Verification*, Key West, FL, pp. 131-140, Dezembro, 1989.

[19] OFFUTT, A.J.; ROTHERMEL, G.; ZAPF, C.; “An Experimental Evaluation of Selective Mutation”, *Proceedings of the 15th International Conference on Software Engineering*, Baltimore, MD, Maio, 1993, pp. 100-107.

[20] OFFUTT, A.J. *et al*.; “An Experimental Determination of Sufficient Mutant Operators”, *ACM Transactions on Software Engineering Methodology*, v. 5, n. 2, Abril, 1996, pp. 99-118.

[21] PRESSMAN, R.S. *Software Engineering – A Practitioner's Approach*. McGraw-Hill, 4ª Edição, 1997.

[22] RAPPS, S.; WEYUKER, E.J; “Selecting Software Test Data Using Data Flow Information”, *IEEE Transactions on Software Engineering*, SE-11(4), Abril, 1985.

[23] SOUZA, S.R.S.; Avaliação do Custo e Eficácia do Critério Análise de Mutantes na Atividade de Teste de Software, Dissertação de Mestrado, ICMC/USP, São Carlos, SP, Junho, 1996.

[24] VINCENZI, A.M.R. *Subsídios para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação*. Dissertação de Mestrado, ICMC/USP, São Carlos, SP, Novembro,1998.

[25] WONG, W.E.; *On Mutation and Data Flow*, Tese de Doutorado, Software Engineering Research Center – Purdue University, West Lafayette, IN, Dezembro, 1993.

[26] WONG, W.E. *et al*.; “Constrained Mutation in C Programs”, *VIII Simpósio Brasileiro de Engenharia de Software*, Curitiba, PR, Outubro, 1994, pp. 439-452.

[27] WONG, W.E. *et al*; “A Comparison of Selective Mutation in C and Fortran”, *Workshop do Projeto Validação e Teste de Sistemas de Operação*, Águas de Lindóia, SP, Janeiro, 1997, pp. 71-84.