Subsídios para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação

Auri Marcelo Rizzo Vincenzi

Orientador: Prof. Dr. José Carlos Maldonado

Dissertação de Mestrado apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC/USP, para a obtenção do título de Mestre na Área de Ciências de Computação e Matemática Computacional.

USP - São Carlos
1998

Aos meus pais

Amauri e Leise

“SE”

“Se puderes guardar o sangue frio diante
de quem fora de si te acusar; e, no instante
em que duvidem de teu ânimo e firmeza
tu puderes ter fé na própria fortaleza,
sem desprezar contudo a confiança alheia...

Se tu puderes não odiar a quem te odeia,
nem pagar com a calúnia a quem te calunia,
sem que tireis daí motivos de ufania;
sonhar, sem permitir que o sonho te domine;
pensar, sem que em pensar tua ambição se confine;
e esperar sempre e sempre, infatigavelmente...

Se com o mesmo sereno olhar indiferente
puderes encarar a derrota e a vitória,
como embustes que são da fortuna ilusória;
e estóico suportar que intrigas e mentiras
deturpem a palavra honesta que profiras...

Se, puderes, ao ver em pedaços destruída
pela sorte maldosa, a obra de tua vida,
tornar de novo, a ferramenta desgastada
e sem queixumes vãos, recomeçar do nada...

Se, tendo loucamente arriscado e perdido
tudo quanto era teu, num só lance atrevido,
tu puderes voltar à faina ingrata e dura,
sem aludir jamais à sinistra aventura...

Se tu puderes coração, músculos, nervos
reduzir da vontade à condição de servos,
que, embora exausto, lhe obedeçam o comando...

Se, andando a par dos reis e com os grandes lidando,
puderes conservar a naturalidade,
e no meio da turba a personalidade;
impávido afrontar adulações, engodos,
opressões, merecer a confiança de todos,
sem que possa contar, todavia, contigo
incondicionalmente o teu melhor amigo...

Se cada minuto os sessenta segundos
tu puderes tornar com o teu suor fecundos...

...a Terra será tua, e os bens que não somem,
e o que é melhor, meu filho, então serás um Homem!!!”

Rudyard Kipling
Tradução: Alcantara Machado

Agradecimentos

A DEUS, pelo dom da vida e por me acompanhar em todos os momentos.

Ao amigo e orientador, Prof. Dr. José Carlos Maldonado, pelo apoio, sugestões e profissionalismo na orientação deste trabalho.

Aos meus familiares, em especial, aos meus pais Amauri e Leise e minhas irmãs Cibele e Andresa, pelo amor, compreensão e incentivo proporcionados.

À Ellen Francine pelo amor, apoio, carinho, revisão, incentivo, esclarecimento de duvidas, etc., etc., etc., os quais foram de fundamental importância para que este trabalho pudesse ser concluído.

Aos demais professores do grupo de Engenharia de Software: Prof.ª Dr.ª Rosely Sanches, Prof. Dr. Paulo Cesar Masiero e Prof.ª Dr.ª Renata P. M. Fortes.

Ao meu irmão, Pe. José Carlos Frederice (Fred), pela amizade, ajuda e conselhos nas horas mais difíceis.

A todos aqueles que, de alguma forma, contribuíram e colaboraram para a realização deste trabalho: Delamaro, Sandra, Simone, Elisa, Tatiana, Luciana, Valéria, Thelma, Trolha, Tchelo, Renata, Andrea, Cris, Plínio e Ana Paula.

Aos companheiros de república: Rud, PC, Dodô, Jalon e Ed.

A todos os professores e funcionários do Instituto que tanto fazem em nosso favor.

Ao CNPq, pelo apoio financeiro.

Índice

Índice de Figuras

Índice de Tabelas

Índice de Quadros

Índice de Gráficos

Resumo

Para sistematizar os testes e contornar as restrições de tempo e custo associadas à atividade de teste, diversas técnicas, critérios e ferramentas têm sido desenvolvidas. Além disso, visando ao estabelecimento de uma estratégia de teste incremental, que apresente baixo custo de aplicação e alta eficácia em revelar a presença de erros, estudos teóricos e empíricos vêm sendo conduzidos pela comunidade de teste. O presente trabalho está inserido nesse contexto e tem como objetivo a realização de estudos empíricos para comparar a adequação entre os critérios baseados em erros – Análise de Mutantes (teste de unidade) e Mutação de Interface (teste de integração) – visando ao estabelecimento de estratégias de teste de baixo custo e eficazes, que englobem todo o ciclo de desenvolvimento de software. Nessa perspectiva, algumas estratégias incrementais de aplicação dos operadores de mutação de unidade e de integração são definidas, explorando o aspecto complementar dos critérios baseados em mutação, reduzindo com isso os custos da atividade de teste durante as fases do teste de unidade e de integração, sem comprometer sua qualidade. Ainda, um conjunto essencial de operadores de mutação para o critério Mutação de Interface é apresentado.

Abstract

Techniques, criteria and tools have been developed and investigated making the testing activity more systematic and aiming at overcoming associated time and cost constraints. Pursuing the establishment of an incremental, low-cost and effective testing strategy, theoretical and empirical studies have been conducted by the testing community. The work proposed here is within this context and aims to conduct empirical studies for evaluating the adequacy between error based criteria – Mutation Analysis (unit testing) and Interface Mutation (integration testing). Therefore, this work intends to establish low-cost and effective testing strategies that would comprise all software development cycle. In this perspective, some incremental testing strategies for mutant operators' application are defined exploring the complementary aspects of unit and integration error based criteria, reducing theirs costs without losses in testing quality. In this scope, an essential mutant operators set for Interface Mutation criterion is characterized.

Capítulo 1

Introdução

1.1 – Contexto

O processo de desenvolvimento de software envolve uma série de atividades sendo que, mesmo com o uso de métodos, técnicas e ferramentas, erros[1] no produto podem ser introduzidos. Com isso, para que o produto de software atinja um grau de qualidade aceitável, são necessárias atividades de garantia de qualidade, dentre elas, de verificação e validação, durante todo o processo de desenvolvimento de software [PRE97]. Dentre as técnicas de verificação e validação, o teste é a atividade mais utilizada sendo de grande importância para a identificação e eliminação de erros que persistem [MAL91]. Entretanto, embora represente a última revisão da especificação, projeto e codificação, a atividade de teste é uma das mais onerosas do desenvolvimento de software [PRE97].

Na tentativa de reduzir esses custos, várias técnicas e critérios que auxiliam na condução e avaliação da atividade de teste têm sido propostas. A diferença entre essas técnicas está, basicamente, na origem da informação que é utilizada para avaliar ou construir conjuntos de

[1] A IEEE, procurando padronizar a terminologia utilizada no contexto de Engenharia de Software, definiu o padrão IEEE 610.12-1990 [I390] diferencia os termos: defeito (*fault*) – passo, processo ou definição de dados incorreto, como por exemplo, uma instrução ou comando incorreto; engano (*mistake*) – ação humana que produz um resultado incorreto, por exemplo, uma ação incorreta tomada pelo programador; erro (*error*) – diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa constitui um erro; e falha (*failure*) – produção de uma saída incorreta com relação à especificação. Neste texto, os termos engano,

casos de teste[2], sendo que cada técnica possui uma variedade de critérios 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 erros comuns que podem ocorrer durante o processo de desenvolvimento de software.

É importante notar que nenhuma das técnicas de teste é completa, no sentido de que nenhuma delas é, em geral, suficiente para garantir a qualidade da atividade de teste. Essas diferentes técnicas se complementam e devem ser aplicadas em conjunto para tentar assegurar um teste de melhor qualidade [PRE97].

Os testes podem ser divididos em 3 fases (ou níveis), usualmente presentes em uma estratégia: **teste de unidade, teste de integração** e **testes de alto nível**. No teste de unidade o objetivo é identificar erros de lógica e de implementação em cada unidade do software. Todas as técnicas de teste citadas acima podem ser aplicadas em nível de unidade. O teste de integração é uma técnica sistemática para integrar as unidades componentes da estrutura do software visando a identificar erros de interface entre os mesmos. Durante essa fase, a técnica mais utilizada é a caixa preta, devido à falta e a dificuldade de aplicação de critérios de teste caixa branca. Os testes de alto nível, divididos em **teste de validação** e **teste de sistema**, são realizados após a integração do sistema e visam a garantir que o software atenda a todas as exigências funcionais, comportamentais e de desempenho descritas na especificação [PRE97].

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, basicamente, utilizam-se os critérios funcionais. Assim sendo, vários estudos empíricos vêm sendo conduzidos objetivando estender alguns critérios estruturais e baseados em erros [HAL84, LIN90, HAR91, JIN95, DEL97c] para o teste de integração permitindo, com isso, que critérios mais sistemáticos e rigorosos possam ser utilizados nessa fase do teste.

Para a aplicação efetiva de um critério é necessária a existência de uma ferramenta de teste automatizada que o apóie. Através do uso de ferramentas é possível reduzir o esforço necessário para a realização do teste, bem como diminuir os erros que são causados pela

defeito e erro serão referenciados como erro (causa) e o termo falha (conseqüência) a um comportamento incorreto do programa. Entretanto, quando tratarem-se de referências a trabalhos de outros autores, os termos originais são mantidos.

[2] Considere uma especificação S e um programa P com domínio de entrada D que implemente tal especificação. Qualquer $x \in D$ constitui um possível **dado de teste** para P. Um **caso de teste** é um par $<x, S(x)>$, onde $S(x)$ representa a saída que x deveria produzir de acordo com a especificação S.

intervenção humana nessa atividade. Além disso, a existência de ferramentas de teste viabiliza a realização de estudos empíricos com o objetivo de avaliar o custo e a eficácia das técnicas e critérios de teste.

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, vários estudos teóricos [RAP85, NTA88, MAL91, FRA93] e empíricos [BUD80a, DEM80, WEY90, WEY91, HOR92, MAT93, OFF93, WON93, DEM94, MAT94, WON94a, WON94b, SOU96, WON97] 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 [SOU96].

Mesmo utilizando as técnicas e critérios existentes, dividindo a atividade de teste em várias fases e utilizando as ferramentas de teste, não se pode garantir um software livre de erros. Os testes somente contribuem para aumentar a confiança de que o software funciona de acordo com o esperado, de modo que grande parte dos erros já foram eliminados [BEI90].

O Grupo de Engenharia de Software do Instituto de Ciências Matemáticas e de Computação ICMC/USP, em colaboração com o Grupo de Engenharia Elétrica da UNICAMP e o Grupo de Engenharia de Software da UEM, tem desenvolvido pesquisas na área de teste, com ênfase em estudos teóricos e empíricos e no desenvolvimento de ferramentas de teste, que resultaram em diversas publicações nacionais e internacionais. As ferramentas desenvolvidas – *Proteum* [DEL93b, DEL96a], *Proteum/IM* [DEL97c] e *Poke-Tool* [CHA91, MAL89] – viabilizaram o início de trabalhos comparativos entre critérios de teste funcional, estrutural e baseados em erros. Esse trabalho é desenvolvido nesse contexto e nessa perspectiva.

1.2 – Motivação

A realização de estudos empíricos intensificou-se nos últimos anos procurando avaliar as diferentes técnicas e critérios de teste existentes, de modo a definir uma estratégia confiável e de baixo custo para a realização da atividade de teste [MAT93, MAT94, WON94b, WON94a, OFF94, MAL91], em que o **custo**, a **eficácia** e a **dificuldade de satisfação** (*strength*) são fatores básicos para comparar a adequação de um critério de teste. O custo refere-se ao esforço necessário para a utilização do critério; a eficácia refere-se à capacidade de determinado critério em revelar a presença de um maior número de erros em relação a outro; e a dificuldade de satisfação refere-se à probabilidade de satisfazer um critério tendo satisfeito outro [MAT94].

O critério **Análise de Mutantes**, um dos critérios da técnica baseada em erros, tem se mostrado, através de estudos teóricos e empíricos [ACR79, DEM80, BUD80a, NTA88, WEY90, WEY91, HOR92, FRA93, MAT93, OFF93, WON93, DEM94, MAT94, WON94b, WON94a, SOU96, WON97], ser altamente eficaz em revelar a presença de erros. Entretanto, existem alguns problemas na sua utilização, devido ao alto custo computacional exigido na execução do grande número de mutantes gerados e ao tempo gasto com a análise dos mutantes vivos. Na tentativa de solucionar esses problemas algumas abordagens vêm sendo propostas, as quais procuram reduzir a quantidade de mutantes gerados através da redução do número operadores de mutação utilizados durante o teste.

Duas dessas abordagens são a **Mutação Aleatória** [ACR79] e a **Mutação Restrita** [MAT91]. Na Mutação Restrita são selecionados operadores de mutação específicos para gerar os mutantes, enquanto que na Mutação Aleatória seleciona-se aleatoriamente uma porcentagem de mutantes a serem gerados. Estudos demonstram que a utilização dessas abordagens reduz significativamente o número de mutantes gerados, sem ocasionar grandes perdas na eficácia em revelar a presença de erros do critério [MAT93, WON94b, WON95a, WON97]. Nesse sentido, a identificação de um subconjunto de operadores de mutação[3] a ser utilizado de modo a garantir um alto grau de adequação em relação à Análise de Mutantes a um baixo custo é uma tarefa relevante. Tais subconjuntos de operadores de mutação são ditos conjuntos essenciais; alguns estudos empíricos têm explorado essa linha de pesquisa no contexto das linguagens Fortran [OFF96b] e C [WON97, BAR98a].

Procurando estender a aplicação do critério Análise de Mutantes para o teste de integração, um novo critério denominado Mutação de Interface (*Interface Mutation – IM*) foi desenvolvido [DEL96b, DEL97c]. Com esse critério é possível testar a interface entre as unidades que compõem o software, ao contrário da Análise de Mutantes, que explora as características das unidades separadamente. Para a aplicação do critério Mutação de Interface foram projetados 33 operadores de mutação responsáveis pela geração dos mutantes de interface. Esses operadores estão implementados na ferramenta *Proteum/IM* [DEL97c], que apóia a aplicação do critério no teste de programas C.

Com o auxílio de ferramentas de teste como a *Proteum* [DEL93b, DEL96a] e a *Mothra* [DEM88], que apóiam o critério Análise de Mutantes, da ferramenta *Proteum/IM* [DEL97c] que apóia o critério Mutação de Interface, e de ferramentas como a *Poke-Tool* [CHA91, MAL89],

Atac [HOR92] e *Asset* [FRA85], que apóiam os critérios Baseados em Fluxo de Dados, vários experimentos têm sido conduzidos [MAT93, MAT94, OFF96b, WON93, WON94b, WON94a, SOU96, BAR98a, BAR98b]. O objetivo desses estudos é medir o desempenho do critério Análise de Mutantes, bem como verificar qual o seu relacionamento com outros critérios visando ao estabelecimento de uma estratégia de teste de baixo custo e alta eficácia em revelar a presença de erros.

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 obter-se a melhor relação custo/eficácia quando são aplicados os critérios Análise de Mutantes e Mutação de Interface no teste de um produto. A proposta de trabalho apresentada neste texto coloca-se nesta perspectiva.

1.3 – Objetivos

O objetivo deste trabalho é verificar, empiricamente, qual o relacionamento entre os critérios Análise de Mutantes e Mutação de Interface e como utilizar tais critérios de forma complementar na atividade de teste. Espera-se com isso contribuir para o estabelecimento de uma estratégia de teste incremental, de baixo custo de aplicação e que garanta um alto grau de adequação em relação a ambos os critérios.

Com os problemas de custo apresentados pelos critérios baseados em mutação, principalmente devido ao grande número de mutantes gerados, o desenvolvimento de estratégias de teste e mecanismos que permitam reduzir o custo de aplicação desses critérios como, por exemplo, a determinação do conjunto essencial de operadores de mutação para o critério Mutação de Interface, é de fundamental importância para viabilizar a aplicação dos mesmos em ambientes reais de desenvolvimento de software.

1.4 – Organização do Trabalho

Este capítulo apresentou o contexto no qual este trabalho está inserido, a motivação para realizá-lo e o objetivo a ser atingido. No Capítulo 2 é feita uma revisão bibliográfica sobre os principais tópicos relacionados a este trabalho sendo dada maior ênfase aos critérios Análise de Mutantes e Mutação de Interface. No Capítulo 3 é apresentado o *framework* utilizado para a condução dos experimentos além de uma estratégia de aplicação incremental dos operadores de

[3] Os operadores de mutação são as regras que definem as alterações que devem ser realizadas no programa original para transformá-lo em um programa mutante.

mutação do critério Mutação de Interface bem como do conjunto essencial de operadores obtido para esse critério. No Capítulo 4 os critérios Análise de Mutantes e Mutação de Interface são comparados com o objetivo de se identificarem alternativas que permitam a aplicação desses critérios de forma complementar durante as fases do teste de unidade e integração. O Capítulo 5 apresenta as contribuições deste trabalho e perspectivas de trabalhos futuros. No Apêndice A são apresentados os operadores de mutação de unidade e de integração que se encontram implementados nas ferramentas *Proteum* e *Proteum/IM*, respectivamente. Finalmente, no Apêndice B são descritas as principais características das ferramentas *Proteum* e *Proteum/IM*.

Capítulo 2

Revisão Bibliográfica

2.1 – Considerações Iniciais

Neste capítulo são apresentados alguns conceitos envolvendo a atividade de teste. Inicialmente, são feitas considerações a respeito da importância do teste durante o processo de desenvolvimento, em seguida são apresentadas as fases do teste e as principais técnicas e critérios que podem ser utilizadas em cada uma delas. Atenção especial é dada aos critérios Análise de Mutantes e Mutação de Interface, dois critérios da técnica Baseada em Erros, por constituírem o alvo do presente trabalho.

A importância da existência de ferramentas automatizadas que auxiliem na condução dos testes também é discutida e são identificados os principais esforços nesta perspectiva. Finalmente é apresentada uma síntese dos principais estudos empíricos conduzidos utilizando-se os critérios Baseados em Mutação, cujos resultados motivam a realização deste trabalho.

2.2 – Teste de Software

Embora durante todo o processo de desenvolvimento de software sejam utilizados métodos, técnicas e ferramentas a fim de evitar que erros sejam introduzidos no produto, a atividade de teste continua sendo de fundamental importância para a eliminação dos erros que persistem [MAL91]. Por isso, o teste de software é um elemento crítico para a garantia de

qualidade do produto e representa a última revisão da especificação, projeto e codificação [PRE97].

Segundo Myers [MYE79], a atividade de teste é o processo de executar um programa com a intenção de encontrar um erro; um bom caso de teste é aquele que tem alta probabilidade de revelar a presença de erros e um teste bem sucedido é aquele que detecta a presença de um erro ainda não descoberto.

Em princípio, o programa deveria ser executado com todas as combinações possíveis do domínio de entrada, entretanto sabe-se que, na maioria das vezes, o teste exaustivo é impraticável devido às restrições de tempo e custo. Desse modo, é necessário determinar quais casos de teste utilizar a fim de que a maioria dos erros existentes possam ser encontrados e que o número de casos de teste utilizados não seja tão grande a ponto de ser impraticável [SOU96].

Nesse sentido, várias técnicas e critérios de teste têm sido elaborados visando a fornecer uma maneira sistemática e rigorosa para selecionar um subconjunto do domínio de entrada e, ainda assim, ser eficaz para revelar a presença dos erros existentes, respeitando as restrições de tempo e custo associados a um projeto de software. Devido ao número de critérios existentes, a Engenharia de Software busca, através da realização de estudos teóricos e empíricos, o desenvolvimento de uma estratégia de teste que seja viável em termos de custo e benefício.

Na abordagem teórica procura-se estabelecer propriedades e características dos critérios de teste, por exemplo, a eficácia de uma estratégia de teste. Na abordagem empírica dados e estatísticas são coletados fornecendo diretrizes para a escolha entre os diversos critérios de teste disponíveis e para o estabelecimento de estratégias de teste.

2.3 – Fases da Atividade de Teste

Além da utilização de técnicas e critérios de teste, quando grandes programas são testados é necessário dividir a atividade de teste em várias fases. Com isso, o testador pode se concentrar em aspectos diferentes do software e utilizar diferentes estratégias de seleção de dados de teste e medidas de cobertura em cada uma delas [LIN90]. De maneira geral, a atividade de teste pode ser considerada como uma atividade incremental realizada em três fases: **teste de unidade**, **teste de integração** e **testes de alto nível** [PRE97].

Inicialmente, os testes de unidade focalizam cada unidade objetivando garantir que os aspectos de implementação de cada um estejam corretos. Durante esta fase utiliza-se muito a técnica de teste estrutural que requer a execução de elementos específicos da estrutura de

controle de cada unidade, com o objetivo de garantir uma completa cobertura e máxima detecção de erros.

Após cada unidade ter sido testada, inicia-se a fase de integração e, conseqüentemente, o teste de integração. Mas por que um programa construído a partir de unidades que individualmente trabalham corretamente – todas foram submetidas ao teste de unidade – não trabalharia corretamente? A resposta é que o teste de unidade apresenta limitações e não pode garantir que cada unidade trabalhe adequadamente em todas as situações: por exemplo, dados podem ser perdidos nas interfaces; uma unidade pode sofrer uma influência adversa, não prevista, de outra unidade; subfunções quando combinadas podem produzir resultados inesperados e estruturas de dados globais podem apresentar problemas [DEL97c].

Além disso, deve-se ressaltar que os tipos de erro geralmente revelados com o teste de integração elevam em muito o custo da atividade de teste se forem detectados somente nos estágios mais avançados, principalmente se a correção do erro forçar modificações em unidades previamente testadas. Desse modo, a realização de testes de integração é de fundamental importância para assegurar uma melhor qualidade do software sendo construído. Segundo Pressman [PRE97], as técnicas de projeto de casos de teste funcional são as mais utilizadas durante esta fase. Procurando estabelecer requisitos de teste mais rigorosos em nível de integração, alguns estudos vêm sendo conduzidos visando a estender os critérios de teste utilizados em nível de unidade (por exemplo, Fluxo de Dados e Análise de Mutantes) para o teste de integração [HAL84, FRA88, LIN90, HAR91, JIN95, DEL97c].

Depois que o software foi integrado, testes de alto nível são realizados: **teste de validação** e **teste de sistema**. O objetivo do teste de validação é verificar se os critérios de validação estabelecidos durante a especificação de requisitos estão corretos e, desse modo, garantir que o software atende a todas as exigências funcionais, comportamentais e de desempenho desejadas. Realizado o teste de validação, o software é combinado com outros elementos (por exemplo, hardware e banco de dados) e através do teste de sistema é possível verificar se todos esses elementos combinam-se adequadamente e se a função/desempenho global do sistema é atingida. A técnica de teste funcional é usada exclusivamente durante os testes de alto nível [PRE97].

2.4 – Técnicas de Teste de Software

Os critérios de teste podem ser classificados em três técnicas: **funcional**, **estrutural** e **baseada em erros**. A diferença entre essas técnicas está na origem da informação utilizada para

avaliar ou construir os conjuntos de casos de teste, sendo que cada uma delas possui um conjunto de critérios para esse fim [MAL91]. Deve-se notar que nenhuma das técnicas de teste é completa, visto que, em geral, nenhuma delas é suficiente para garantir a qualidade da atividade de teste. Na realidade, essas diferentes técnicas se complementam e devem ser aplicadas em conjunto a fim de assegurar um teste de boa qualidade [PRE97].

2.4.1 – Teste Funcional

O teste funcional ou **caixa preta** tem esse nome pelo fato de tratar o software como uma caixa da qual o conteúdo é desconhecido e só é possível visualizar o lado externo. Desse modo, o testador utiliza essencialmente a especificação funcional do programa para derivar os casos de teste que serão empregados sem se importar com os detalhes de implementação [BEI90]. Assim, uma especificação correta e de acordo com os requisitos do usuário é essencial para esse tipo de teste.

Um problema com a técnica funcional é a dificuldade de quantificar a atividade de teste, visto que não se pode garantir que partes essenciais ou críticas do programa sejam executadas. Outro problema é que o teste caixa preta está sujeito às inconsistências decorrentes da especificação, pois é ela a base a partir da qual são derivados os casos de teste.

Exemplos de critérios de teste funcional são [PRE97]:

- **particionamento de equivalência**: divide o domínio de entrada em classes de dados válidas e inválidas que provavelmente exercitarão uma função de software específica;
- **análise do valor limite**: verifica a capacidade de um programa em manipular dados nos limites das condições de entrada;
- **grafo de causa-efeito**: oferece uma representação concisa das condições lógicas e das ações correspondentes possibilitando que o testador valide conjuntos de ações e condições.

2.4.2 – Teste Estrutural

A técnica de teste estrutural, conhecida como **caixa branca** (em oposição ao nome caixa preta), leva em consideração os aspectos de implementação na escolha dos casos de teste. Em geral, a maioria dos critérios dessa técnica utiliza uma representação de programa conhecida como **grafo de fluxo de controle** ou **grafo de programa**.

A representação de um programa *P* como um grafo de fluxo de controle consiste em estabelecer uma correspondência entre nós e blocos e em indicar possíveis fluxos de controle entre blocos através dos arcos. Um grafo de fluxo de controle é portanto um grafo orientado, com um único nó de entrada e um único nó de saída, no qual cada vértice representa um bloco indivisível de comandos e cada aresta representa um possível desvio de um bloco para outro. Cada bloco tem as seguintes características: 1) uma vez que o primeiro comando do bloco é executado, todos os demais são executados seqüencialmente; e 2) não existe desvio de execução para nenhum comando dentro do bloco. Através do grafo de programa podem ser escolhidos os componentes que devem ser executados, caracterizando assim o teste estrutural.

Em geral, como sugerido por alguns autores [MYE79, PRE97], uma possível estratégia de teste é utilizar primeiramente os critérios funcionais e, posteriormente, utilizar os critérios estruturais para avaliar qual a cobertura obtida pelo conjunto de casos de teste funcional. Não se obtendo um conjunto de casos de teste adequado, esse deve ser complementado de modo a tornar-se adequado aos critérios estruturais.

Como exemplo de critérios estruturais pode-se destacar: **Critérios Baseados na Complexidade, Critérios Baseados em Fluxo de Controle** e **Critérios Baseados em Fluxo de Dados**.

Os critérios baseados na complexidade derivam os requisitos de teste a partir da complexidade do programa. Um dos critérios dessa classe, o **critério de McCabe**, requer a execução de um conjunto de caminhos linearmente independentes do grafo de programa; utiliza o conceito de complexidade ciclomática para derivar os requisitos de teste [PRE97].

Critérios baseados em fluxo de controle utilizam somente características de controle da execução do programa, tais como comandos ou desvios, para derivar os requisitos de teste. Os critérios mais conhecidos dessa classe são: **Todos-Nós**, **Todos-Arcos** e **Todos-Caminhos**. O critério Todos-Nós exige que a execução do programa passe pelo menos uma vez em cada vértice do grafo de fluxo de controle, ou seja, que cada comando do programa seja executado pelo menos uma vez; o critério Todos-Arcos requer que cada aresta do grafo, ou seja, cada desvio de fluxo de controle, seja exercitada pelo menos uma vez; e o critério Todos-Caminhos, que em geral é impraticável, requer que todos os caminhos possíveis do programa sejam executados [PRE97].

A classe de critérios baseados em fluxo de dados utiliza, como o próprio nome diz, informações do fluxo de dados do programa para derivar os requisitos de teste. Esses critérios

requerem que sejam testadas as interações que envolvem definições de variáveis e referências a essas definições [RAP85]. Os **critérios de Rapps e Weyuker** [RAP82, RAP85] são exemplos dessa classe de critérios.

Para derivar os requisitos de teste requeridos por esses critérios é necessário adicionar ao grafo de programa informações sobre fluxo de dados, caracterizando o **Grafo Def-Uso** (*Def-Use Graph*) definido por Rapps e Weyuker [RAP82, RAP85]. Neste grafo são exploradas as associações entre a definição e o uso das variáveis determinando os caminhos a serem exercitados. Quando a variável é usada em uma computação, diz-se que seu uso é computacional (**c-uso**); quando usada em uma condição, seu uso é predicativo (**p-uso**). Os critérios desta classe são: **Todas-Definições** (*all-defs*), **Todos-Usos** (*all-uses*), **Todos-Du-Caminhos** (*all-du-paths*); **Todos-P-Usos** (*all-p-uses*), **Todos-P-Usos/Alguns-C-Usos** (*all-p-uses/some-c-uses*) e **Todos-C-Usos/Alguns-P-Usos** (*all-c-uses/some-p-uses*).

A partir dos critérios de Rapps e Weyuker, Maldonado [MAL91] definiu os critérios **Potenciais-Usos**, introduzindo o conceito de potencial-associação. Esses critérios requerem associações independentemente da ocorrência explícita de uma referência a uma determinada variável, ou seja, requerem que caminhos livres de definição[4] para uma variável sejam executados, independentemente de ocorrer uso dessa variável neste caminho. Os critérios básicos que fazem parte dessa família de critérios são: **Todos-Potenciais-Usos**, **Todos-Potenciais-Usos/Du**, **Todos-Potenciais-Du-Caminhos**.

De modo semelhante aos demais critérios de fluxo de dados, os Potenciais-Usos podem utilizar o Grafo-Def-Uso como base para o estabelecimento dos requisitos de teste. Na verdade, basta estender o grafo de programa para que cada nó do grafo passe a conter informações a respeito das definições que ocorrem em cada nó [MAL91].

Segundo Maldonado [MAL91], a **complexidade**[5] e a **relação de inclusão**[6] refletem as propriedades básicas que devem ser consideradas durante a definição de um critério *C*, ou seja, o critério deve:

- incluir o critério Todos-Arcos, ou seja, um conjunto de casos de teste que exercite os elementos requeridos pelo critério *C* deve exercitar todos os arcos do programa;

[4] Um caminho livre de definição ($i, j, ..., k$) é aquele no qual uma variável v definida em i não é redefinida em nenhum dos demais nós ($j, ..., k$) pertencentes ao caminho.

[5] A complexidade é definida como o número máximo de casos de teste requeridos por um critério no pior caso [MAL91].

[6] A relação de inclusão, definida por Rapps e Weyuker [RAP85], estabelece uma ordem parcial entre os critérios, caracterizando uma hierarquia entre eles. Dados dois critérios C_1 e C_2, diz-se que C_1 inclui C_2 se para todo conjunto de casos de teste T_1

- requerer, do ponto de vista de fluxo de dados, ao menos um uso de todo resultado computacional; isto equivale ao critério *C* incluir o critério Todas-Defs; e
- requerer um conjunto de casos de teste finito.

Visto que, na prática, grande parte dos programas possui caminhos não executáveis, o principal problema dos critérios baseados em fluxo de dados, à exceção dos Potenciais-Usos, é o fato de que eles não garantem a inclusão do critério Todos-Arcos na presença de caminhos não executáveis[7]. Os critérios Potenciais-Usos satisfazem os requisitos básicos exigidos de um bom critério de teste, sendo que mesmo na presença de caminhos não executáveis, estes estabelecem uma hierarquia entre os critérios Todos-Arcos e Todos-Caminhos. Além disso, os critérios Potenciais-Usos incluem o critério Todas-Defs e requerem um número finito de casos de teste. Outra característica importante é que nenhum outro critério baseado em fluxo de dados inclui os critérios Potenciais-Usos [MAL91].

Observa-se que não existe uma correspondência direta entre a hierarquia estabelecida pela relação de inclusão e a capacidade de revelar a presença de erros, ou seja, mesmo que um critério C_1 inclua um critério C_2 não implica necessariamente que C_1 tenha uma melhor capacidade de revelar a presença de erros que C_2. Frankl e Weyuker [FRA93] definiram cinco outras relações entre critérios e avaliaram essas relações usando três medidas probabilísticas que procuram agregar à hierarquia estabelecida por essas relações a capacidade de revelar a presença de erros, ou seja, refletir a eficácia do critério.

2.4.3 – Teste Baseado em Erros

A técnica de teste baseada em erros utiliza informações sobre os erros mais freqüentes cometidos no processo de desenvolvimento de software e sobre os tipos específicos de erros que se desejam revelar [DEM87]. Dois critérios típicos que se concentram em erros são os critérios **Semeadura de Erros** (*Error Seeding*) [BUD81] e **Análise de Mutantes** (*Mutation Analysis*) [DEM78].

No critério Semeadura de Erros, uma quantidade conhecida de erros é semeada artificialmente no programa. Após o teste, do total de erros encontrados, verificam-se quais são naturais e quais são artificiais. Usando estimativas de probabilidade, o número de erros naturais ainda existentes no programa pode ser calculado [GOE85].

C_1-adequado, T_1 é C_2-adequado 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.

A Análise de Mutantes é um critério que utiliza um conjunto de programas ligeiramente modificados (**mutantes**) obtidos a partir de determinado programa *P*, para avaliar o quanto um conjunto de casos de teste *T* é adequado para o teste de *P*. O objetivo é encontrar um conjunto de casos de teste *T* capaz de revelar as diferenças de comportamento existentes entre *P* e seus mutantes [DEM87].

Dois outros critérios derivados da Análise de Mutantes são a **Mutação Fraca** (*Weak Mutation*) [HOW82] e a **Mutação Firme** (*Firm Mutation*) [WOO88, WOO93]. A idéia básica da Mutação Fraca, da Mutação Firme e da Análise de Mutantes é a mesma, ou seja, uma pequena mudança no programa é realizada e os resultados da versão original são comparados com os da versão modificada. A diferença está no momento da criação do mutante e no momento em que se comparam os resultados obtidos para decidir se o mutante "morre" ou continua "vivo". Na Mutação Fraca cria-se o mutante imediatamente antes da execução de um componente[8] e os resultados são comparados logo após o término da execução do componente. Na Análise de Mutantes geram-se os mutantes antes do início da execução do programa e comparam-se os resultados após o término de sua execução. Já a Mutação Firme, um critério entre a Mutação Fraca e a Análise de Mutantes, realiza alterações no programa e introduz pontos nos quais os estados do programa original e do programa mutante são comparados antes do final da execução. Outros critérios alternativos, derivados da Análise de Mutantes, são descritos na Seção 2.5.1.1.

Devido à importância do critério Análise de Mutantes no contexto deste trabalho, a seção a seguir apresenta as principais características do critério, seus problemas e alternativas para solucioná-los, uma extensão que permite a sua utilização no teste de integração e as ferramentas de teste que apóiam a sua aplicação.

2.5 – Critério Análise de Mutantes: Aspectos Históricos e Conceitos Básicos

O critério Análise de Mutantes surgiu na década de 70 na *Yale University* e *Georgia Institute of Technology*, possuindo um forte relacionamento com um método clássico para detecção de erros lógicos em circuitos digitais – o modelo de teste de falha única [FRI75].

Um dos primeiros artigos que descrevem o teste de mutantes foi publicado em 1978 [DEM78]. A idéia básica da técnica apresentada por DeMillo [DEM78], conhecida como

[7] Um caminho completo é executável se existe um conjunto de valores que possa ser atribuído às variáveis de entrada do programa causando a execução desse caminho; caso contrário, esse caminho é dito não executável.

[8] Um componente corresponde a estruturas computacionais elementares tais como: referência a uma variável, expressão aritmética, expressão booleana ou relação entre expressões aritméticas.

hipótese do programador competente (*competent programmer hypothesis*), assume que programadores experientes escrevem programas muito próximos do correto. Considerando a validade dessa hipótese, pode-se afirmar que erros são introduzidos nos programas através de pequenos desvios sintáticos que, embora não causem erros sintáticos, alteram a semântica do programa e, conseqüentemente, conduzem o programa a um comportamento incorreto. Para revelar tais erros, a Análise de Mutantes identifica os desvios sintáticos mais comuns e, através da aplicação de pequenas transformações sobre o programa em teste, encoraja o testador a construir casos de testes que mostrem que tais transformações levam a um programa incorreto [AGR89].

Uma outra hipótese explorada na aplicação do critério Análise de Mutantes é o **efeito de acoplamento** (*coupling effect*) [DEM78], o qual assume que erros complexos estão relacionados a erros simples. Assim sendo, espera-se, e alguns estudos empíricos já confirmaram esta hipótese [ACR79, BUD80a, OFF89], que conjuntos de casos de teste capazes de revelar erros simples são também capazes de revelar erros complexos.

A princípio, o testador deve fornecer um programa P a ser testado e um conjunto de casos de teste T cuja adequação deseja-se avaliar. O programa é executado com T e, se apresentar resultados incorretos, então um erro foi revelado e o teste termina. Caso contrário, o programa ainda pode conter erros que o conjunto T não conseguiu revelar. O programa P sofre então pequenas alterações, dando origem aos programas P_1, P_2, ..., P_n, que são mutantes de P, diferindo deste apenas pela ocorrência de erros simples, ou seja, cada mutante contém apenas uma mutação.

A seguir, os mutantes são executados com o mesmo conjunto de casos de teste T. Conseguindo-se obter casos de teste que resultem apenas em mutantes mortos e equivalentes, tem-se um conjunto de casos de teste T adequado ao programa P em teste, no sentido de que, ou P está correto, ou possui erros pouco prováveis de ocorrerem [DEM78].

Através do **escore de mutação** (*mutation score*), que relaciona o número de mutantes mortos com o número de mutantes gerados, a Análise de Mutantes fornece uma medida objetiva do nível de confiança da adequação dos casos de teste analisados. O escore de mutação varia no intervalo [0,1] sendo que, quanto maior o escore, mais adequado é o conjunto de casos de teste para o programa sendo testado. Percebe-se com essa fórmula que apenas $DM(P, T)$ depende do conjunto de casos de teste utilizado e que $EM(P)$ é obtido à medida que o testador decide que determinado mutante vivo é equivalente. Em geral, a equivalência entre programas é uma

questão indecidível; no entanto, alguns métodos e heurísticas têm sido propostos para determinar a equivalência de programas em uma grande porcentagem dos casos de interesse [BUD81].

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)}$$

sendo:

$DM(P, T)$: total de mutantes mortos pelo conjunto de casos de teste T;

$M(P)$: total de mutantes gerados a partir do programa P;

$EM(P)$: total de mutantes equivalentes ao programa P.

Dados um programa P e um conjunto de casos de teste T cuja qualidade deseja-se avaliar, pode-se aplicar o critério Análise de Mutantes através dos seguintes passos:

Primeiro Passo – Geração de Mutantes

Os mutantes são gerados através da aplicação de operadores de mutação no programa P sendo testado. Entende-se por operador de mutação as regras que definem as alterações que devem ser aplicadas no programa original P. A aplicação de um operador de mutação gera, na maioria das vezes, mais que um mutante, visto que P pode conter várias entidades que estão no domínio de um operador e, desse modo, o operador é aplicado a cada uma dessas entidades, uma de cada vez.

A Figura 1(a) apresenta o programa *Fibonacci* e as figuras 1(b) e 1(c) mostram dois de seus possíveis mutantes. O mutante da Figura 1(c) é equivalente ao programa original; já o mutante da Figura 1(b) não é equivalente e, desse modo, deve existir algum caso de teste que seja capaz de mostrar a diferença de comportamento existente entre esse mutante e o programa original. O símbolo ® indica o local onde ocorreu a mutação.

A escolha do conjunto de operadores de mutação depende dos erros que se quer revelar, da cobertura que se quer garantir, da linguagem de programação na qual os programas a serem testados foram escritos e da ferramenta de teste utilizada. Por exemplo, para o teste de programas Fortran, a ferramenta *Mothra* possui um conjunto de 22 operadores de mutação. Para o teste de programas C, em nível de unidade, a ferramenta *Proteum* conta com 71 operadores de mutação e, para o teste de programas C, em nível de integração, a ferramenta *Proteum/IM* possui 33 operadores de mutação.

```
main() /* Programa Fibonacci */
{
   int i, n, fn, fnm1, fnm2;
   fn = 0; fnm1 = 1; fnm2 = 0;
   scanf("%d",&n);
   if (n < 0) then { printf("Erro"); exit(1); }
   if ((n == 0) || (n == 1)) then {
     printf("%d\n", n); exit(0); }
   for (i = 2; i <= n; i++) {
     fn = fnm1 + fnm2;
     fnm2 = fnm1;
     fnm1 = fn;
   }
   printf("%d\n", fn);
}
```

(a)

```
main() /* Programa Mutante */
{
   int i, n, fn, fnm1, fnm2;
   fn = 0; fnm1 = 1; fnm2 = 0;
   scanf("%d",&n);
   if (n < 0) then { printf("Erro"); exit(1); }
   if ((n == 0) || (n == 1)) then {
     printf("%d\n", n); exit(0); }
®  for (i = 2; i < n; i++) {
     fn = fnm1 + fnm2;
     fnm2 = fnm1;
     fnm1 = fn;
   }
   printf("%d", fn);
}
```

(b)

```
main() /* Mutante Equivalente */
{
   int i, n, fn, fnm1, fnm2;
   fn = 0; fnm1 = 1; fnm2 = 0;
   scanf("%d",&n);
   if (n < 0) then { printf("Erro"); exit(1); }
® if ((n == 0) + (n == 1)) then {
     printf("%d\n", n); exit(0); }
   for (i = 2; i <= n; i++) {
     fn = fnm1 + fnm2;
     fnm2 = fnm1;
     fnm1 = fn;
   }
   printf("%d", fn);
}
```

(c)

Figura 1 – Programa Fonte em Teste e Dois de seus Possíveis Mutantes:
(a) Programa Fibonacci, (b) Programa Mutante e (c) Mutante Equivalente

Segundo Passo – Execução do Programa

Deve-se executar o programa original *P* com os casos de teste selecionados e verificar se o resultado é o esperado. Caso o programa apresente um comportamento diferente para algum caso de teste, então um erro foi revelado e o processo termina. Por outro lado, se nenhum dos casos de teste revelar a presença de erro, executa-se o passo seguinte. A tarefa de oráculo, isto é, decidir se o resultado está correto ou não, geralmente é desempenhada pelo testador [VIN97].

Terceiro Passo – Execução dos Mutantes

Esse passo pode ser totalmente automatizado, não requerendo a intervenção do testador. O conjunto de casos de teste *T* é aplicado a cada mutante P_i e os resultados obtidos são comparados com o resultado de *P*. Se os resultados forem diferentes, P_i é dito estar morto e pode ser descartado. Caso contrário, se os resultados forem iguais, P_i continua vivo. Isso pode ocorrer

ou porque P e P_i são equivalentes; ou porque o conjunto de casos de teste utilizado não é adequado o suficiente para distinguir o comportamento de P e P_i.

Terminada a execução dos mutantes pode-se calcular o escore de mutação, o qual servirá como um indicativo da qualidade do conjunto de casos de teste utilizado.

Quarto Passo – Análise dos Mutantes Vivos

Essa é a fase da Análise de Mutantes que requer mais intervenção humana. Primeiramente, é necessário analisar os mutantes que sobreviveram à execução com os casos de teste disponíveis e decidir se tais mutantes são equivalentes ou não ao programa original. Se o mutante é equivalente ao programa original, então ele deve ser descartado. Caso contrário, o conjunto de casos de teste não foi capaz de diferenciá-lo do programa original, fazendo-se necessária a inclusão de novos casos de teste ao conjunto T. Desse modo, o caminho é retornar ao segundo passo do critério, executar P com os novos casos de teste, executar os mutantes, recalcular o escore de mutação e assim por diante, até que se consiga um bom conjunto de casos de teste T.

Deve-se salientar que o procedimento descrito acima exige, além da execução do programa sendo testado, que algumas tarefas sejam realizadas automaticamente. Tarefas como geração e execução de mutantes e comparação dos resultados obtidos demandam grande esforço de processamento, sendo praticamente impossível executá-las de modo eficiente e preciso sem o auxílio de um computador. Assim, faz-se necessário o uso de ferramentas que automatizem a aplicação do critério [DEL93b]. A Seção 2.7 apresenta um histórico das principais ferramentas de teste já desenvolvidas para apoiar a aplicação do critério Análise de Mutantes, dentre elas as ferramentas *Proteum* [DEL93b, DEL96a] e *Proteum/IM* [DEL97a, DEL97c] que serão utilizadas na condução dos experimentos relacionados a este trabalho.

2.5.1 – Alternativas para a Aplicação do Critério

Embora o critério Análise de Mutantes tenha se mostrado, através de estudos teóricos e empíricos, ser um dos mais eficazes para revelar a presença de erros [ACR79, DEM80, BUD80a, BUD81, HOR92, WON93, MAT94, WON94b, SOU96], um dos problemas para a sua utilização é o alto custo computacional exigido na execução do grande número de mutantes gerados, bem como a análise dos mutantes vivos quanto à sua equivalência ou não em relação ao programa original.

Visando à redução desses custos algumas abordagens têm sido propostas:

- utilização de arquiteturas de hardware avançadas para diminuir o tempo de execução dos mutantes [CHO89, KRA88, MAT88];
- desenvolvimento de critérios alternativos derivados da Análise de Mutantes que reduzem o número de mutantes gerados [ACR79, MAT91, OFF93]; e
- minimização de conjuntos de casos de teste que diminuem o número de casos de teste que precisam ser executados com cada mutante durante os testes de regressão [HAR93, SOU96].

Se por um lado a utilização de arquiteturas de hardware de alto desempenho reduzem o custo de aplicação em termos de tempo de execução dos mutantes, por outro lado eleva o custo do teste em termos dos recursos de hardware necessários. Desse modo, tal investimento só seria justificável para o teste de sistemas com requisitos de teste rigorosos [DEL97c].

A abordagem que tem sido mais explorada é o desenvolvimento de critérios alternativos que procuram reduzir o custo de aplicação da Análise de Mutantes diminuindo o número de mutantes que precisam ser executados e analisados. A idéia central é que diversos mutantes podem levar à seleção de um mesmo caso de teste, ou seja, um mesmo caso de teste distingui vários mutantes, caracterizando uma "redundância" de mutantes. Desse modo, pode-se utilizar subconjuntos reduzidos de mutantes que levem à seleção de conjuntos de casos de teste tão efetivos quanto aqueles criados por um conjunto mais completo de mutantes [DEL97c].

A minimização de conjuntos de casos de teste tem também se mostrado como uma boa alternativa para a redução do custo de aplicação do teste de regressão, além possibilitar a obtenção de parâmetros mais reais para quantificar o custo de aplicação de um critério durante o desenvolvimento de estudos empíricos [SOU96].

2.5.1.1 – Critérios Alternativos

As abordagens de **Mutação Aleatória** (*Randomly Selected X% Mutation*) [ACR79], **Mutação Restrita** (*Constrained Mutation*) [MAT91] e **Mutação Seletiva** (*Selective Mutation*) [OFF93] foram definidas a partir do critério Análise de Mutantes e procuram reduzir o custo de sua aplicação através da redução do número de mutantes gerados.

A Mutação Aleatória, embora utilize todos os operadores de mutação, seleciona aleatoriamente uma porcentagem de cada um deles e a análise fica restrita apenas ao subconjunto

de mutantes gerados por esses operadores. Segundo DeMillo *et al*. [DEM88], em geral, mesmo utilizando uma pequena porcentagem do total de mutantes é possível construir bons casos de teste, aspecto este investigado experimentalmente [MAT93, WON97].

A Mutação Seletiva propõe que os operadores de mutação responsáveis por gerar o maior número de mutantes não sejam utilizados [OFF93]. Desse modo, o critério *N*-Seletivo é aquele que deixa de gerar mutantes para *N* operadores de mutação, exatamente aqueles que geram mais mutantes.

Na Mutação Restrita, alguns operadores de mutação específicos são selecionados para a geração dos mutantes [MAT91]. Através da escolha de um bom subconjunto de operadores de mutação é possível obter uma sensível redução no custo de execução dos mutantes sem reduzir de forma significativa a capacidade em revelar a presença de erros [WON95].

O efeito da aplicação dessas abordagens tem sido investigado em vários estudos empíricos [MAT93, WON94b, WON94a, WON95, SOU96, WON97] e os resultados demonstram que é possível reduzir o custo de aplicação do critério Análise de Mutantes 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 de operadores de mutação *SC* 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 [BAR98b].

Nessa perspectiva, alguns trabalhos que vêm sendo desenvolvidos dizem respeito à determinação de conjuntos essenciais de operadores de mutação. Offutt *et al*. [OFF96b], a partir dos operadores de mutação da ferramenta *Mothra*[9] [DEM88], determinaram um conjunto essencial de operadores de mutação para a linguagem Fortran. Wong *et al*. [WON97] compararam a Mutação Restrita no contexto das linguagens C e Fortran, e os resultados obtidos forneceram indícios de que os operadores da ferramenta *Proteum* [DEL93b] utilizados no experimento possuem uma forte relação com os operadores essenciais obtidos por Offutt *et al*. [OFF96b] para a linguagem Fortran.

Outro trabalho relevante nessa direção foi o desenvolvimento de um procedimento genérico para a determinação do conjunto essencial de operadores de mutação, realizado por

[9] A ferramenta *Mothra* [DEM88] apóia a aplicação do critério Análise de Mutantes para programas escritos em Fortran.

Barbosa [BAR98a, BAR98b]: a partir dos operadores de mutação implementados na ferramenta *Proteum*, foi conduzido um experimento e, pela aplicação do procedimento desenvolvido, determinou-se um conjunto essencial de operadores de mutação para a linguagem C em nível de unidade. Os resultados obtidos demonstraram que o conjunto essencial possibilitou uma redução de mais de 65% no custo de aplicação do critério Análise de Mutantes, preservando, na média, um alto escore de mutação (0,996).

Tendo em vista que no contexto deste trabalho, o procedimento definido por Barbosa será utilizado para a determinação do conjunto essencial de operadores de mutação de interface (ferramenta *Proteum/IM*), o experimento conduzido por Barbosa é descrito na Seção 2.8.3 e a aplicação do procedimento no contexto dos operadores de mutação de interface é apresentada na Seção 3.3.2. A seguir, são apresentados os principais trabalhos que vêm sendo conduzidos no contexto do teste de integração.

2.6 – Teste de Integração

Embora critérios de todas as técnicas de teste sejam empregados no teste de unidade, para o teste de integração têm sido utilizados, basicamente, os critérios da técnica funcional.

Alguns trabalhos têm o objetivo de estender os critérios estruturais para o teste de integração. Haley e Zweben [HAL84] definiram um critério para a avaliação da adequação interprocedural baseado na cobertura de conjuntos de caminhos específicos cuja cardinalidade é determinada somente pela complexidade da interface da função, ao invés da complexidade do caminho.

Linnenkugel e Müllerburg [LIN90] estenderam alguns critérios de fluxo de dados e de controle de modo que estes pudessem ser aplicados no teste de integração. Harrold e Soffa [HAR91] apresentaram uma técnica para determinar estruturas definições-usos de interface permitindo a aplicação dos critérios de adequação de fluxo de dados [FRA88] em nível de integração.

Jin e Offutt [JIN95], baseados na classificação de acoplamento de módulos, estabeleceram um critério para o teste de integração. Conforme o tipo de acoplamento existente entre duas unidades, o critério tenta identificar alguns caminhos críticos que deveriam ser cobertos para exercitar adequadamente a conexão.

Além dessas abordagens, Delamaro [DEL97c] propôs o critério Mutação de Interface, que utiliza os mesmos conceitos da Análise de Mutantes para o teste de integração. O critério

Mutação de Interface procura modelar as classes de erros mais comuns ocorridas na conexão entre duas unidades (erros de integração), ao passo que a Análise de Mutantes se preocupa com os erros em nível de unidade. Na seção a seguir são discutidos mais detalhadamente os tipos de erros de integração.

2.6.1 – Erros de Integração

Segundo Haley e Zweben [HAL84], 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[10] 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 existem 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.

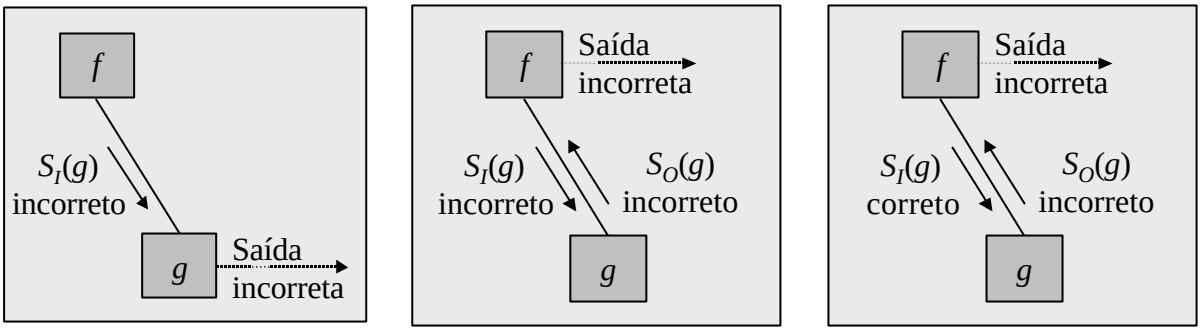
Considere um programa P e um caso de teste t para P. Suponha que em P existam funções f e g tal que f chama g. Considere $S_I(g)$ como o conjunto de valores passados para g e $S_O(g)$ os valores retornados por g. Ao executar P com o caso de teste t, um erro de integração é identificado na chamada de g a partir de f quando [DEL97a]:

- Erro Tipo 1 (Figura 2 (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 2 (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;
- Erro Tipo 3 (Figura 2 (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 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

[10] De acordo com Howden [HOW78] um erro de domínio ocorre quando um caminho incorreto é executado e um erro computacional ocorre quando o caminho correto é executado mas o valor computado é incorreto.

internamente ela realiza um cálculo incorreto produzindo um valor de retorno não esperado que, posteriormente, leva a um resultado incorreto.

A Figura 2 ilustra os três tipos de erro citados acima.



(a) (b) (c)

Figura 2 – Tipos de Erros de Integração: (a) Erro Tipo 1, (b) Erro Tipo 2, (c) Erro Tipo 3 [DEL97c]

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.

Mais de um erro de integração pode estar associado a (ou serem causados por) um defeito simples. Por exemplo, considere um programa com três unidades R, S e T tal que R chama S e, em seguida, R chama T. Suponha que na unidade S um valor incorreto $x \in S_O(S)$ é retornado e esse valor é parte de $S_I(T)$. Suponha que devido a x, T produz uma saída incorreta. Desse modo, um defeito em S produziu um erro Tipo 3 na conexão R-S e um erro Tipo 1 na conexão R-T.

Os conjuntos $S_I(g)$ e $S_O(g)$ dependem, em parte, da linguagem de programação na qual o programa é escrito. Por exemplo, para programas escritos na linguagem C, eles podem ser definidos como:

- $S_I(g)$: parâmetros formais de entrada da função g e variáveis globais usadas em g.
- $S_O(g)$: parâmetros formais de saída da função g, variáveis globais definidas em g e valores retornados por g através de comandos $return$.

Baseado nesses tipos de erros foi definido o critério Mutação de Interface descrito a seguir.

2.6.2 – Mutação de Interface: Critério Baseado em Erros para Teste de Integração

A idéia do teste de mutação é avaliar a qualidade de um conjunto de casos de teste por meio de sua habilidade em revelar a presença de erros simples, que foram introduzidos no programa em teste. Baseado na hipótese do efeito de acoplamento espera-se que um conjunto capaz de revelar erros simples possa também revelar erros mais complexos [DEL97a]. Acree [ACR79], Budd [BUD80a] e Offutt [OFF89] realizaram experimentos que comprovam essa hipótese em nível de unidade.

Experimentos têm mostrado que o critério Análise de Mutantes é bastante efetivo para a avaliação e seleção de casos de teste [ACR79, DEM80, BUD80a, BUD81, HOR92, WON93, MAT94, WON94b, SOU96]. No entanto, o maior problema para a aplicação desse critério é o alto custo computacional exigido na execução e análise do grande número de mutantes gerados. Para o teste de unidade algumas abordagens, como as descritas na Seção 2.5.1.1, têm sido utilizadas buscando a redução desses custos; entretanto, no teste de sistemas maiores, pouco modularizado, mesmo a utilização dessas abordagens podem não ser suficientes, inviabilizando a aplicação do teste de mutação [DEL97c].

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. Por exemplo, se a unidade é chamada em diversos pontos do programa, não é possível garantir que todos esses pontos de chamada sejam testados pois pode existir um conjunto de casos de teste que exercite determinada unidade passando por um único ponto de chamada, distinguindo todos os mutantes não equivalentes gerados para tal unidade. Em resumo, o teste de mutação em nível de unidade tem-se mostrado um critério efetivo para o teste da estrutura interna da unidade, mas não necessariamente para exercitar as interações entre unidades em um programa integrado [DEL97c].

Ao contrário, no teste de integração preocupa-se em assegurar que todas as interações possíveis entre unidades sejam testadas. Assim, o objetivo do critério Mutação de Interface é inserir perturbações nas conexões entre duas unidades, de modo que, para se 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 mesmo raciocínio aplicado à Análise de Mutantes, 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 [DEL97a].

Operadores de mutação destinados ao teste de unidade possuem semelhanças e diferenças em relação aos operadores de mutação de interface. A idéia básica de ambos é a mesma, ou seja, introduzir modificações sintáticas no programa em teste transformando-o em programas mutantes. Por outro lado, os operadores de mutação de interface estão relacionados a uma conexão entre duas unidades. Considere um programa que possua uma unidade *f* que faça duas chamadas a uma unidade *g* (Figura 3). Os operadores de mutação de interface irão gerar dois conjuntos de mutantes, um para cada ponto de chamada de *g* dentro de *f* e os mutantes de determinado conjunto só podem ser distinguidos se forem executados a partir do ponto de chamada para o qual foram criados pois, do contrário, não se estaria testando a conexão desejada [DEL97c].

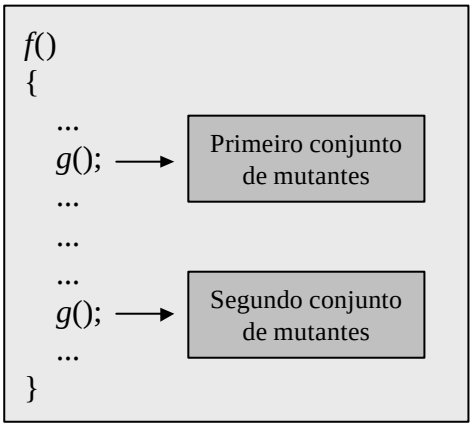


Figura 3 – Representação dos Conjuntos de Mutantes Requeridos no Teste da Conexão *f-g*

O exemplo acima aponta uma diferença sutil entre o teste de unidade, utilizando a Análise de Mutantes, e o teste de integração com a Mutação de Interface. Essa diferença pode ser formalizada através da análise das condições de necessidade e suficiência para um caso de teste distinguir um mutante de unidade [DEL98a]. Segundo DeMillo e Offutt [DEM91], para um caso de teste *t* distinguir um mutante P_i gerado a partir do programa original *P*, três condições são requeridas:

1. Alcançabilidade (*Reachability*): a execução de P_i com o caso de teste *T* deve fazer com que o controle do programa alcance o ponto onde a mutação foi realizada;

2. Necessidade (*Necessity*): o estado de P_i difere do estado de P logo após a execução do comando modificado (mutado); e
3. Suficiência (*Sufficiency*): a diferença nos estados de P e P_i deve propagar-se tal que diferentes saídas sejam produzidas por P e P_i.

Considere a Figura 4 e os três tipos de erros citados anteriormente. O critério Mutação de Interface aplica mutação nos pontos onde f chama g, nos pontos onde as variáveis globais e os parâmetros formais são usados dentro de g e nos pontos onde os valores são retornados de g para f (como no comando *return* em C). Para isso, Delamaro [DEL97c] definiu 33 operadores de mutação de interface para a linguagem C que realizam mutações apenas nesses pontos.

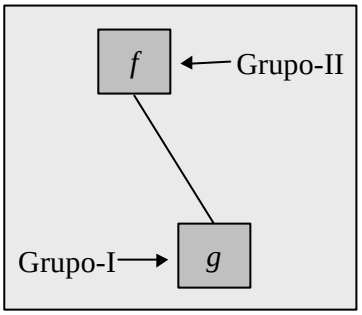


Figura 4 – Conexão entre as Unidades f e g [DEL97c]

Os operadores de mutação de interface estão divididos em dois grupos. O primeiro grupo (Grupo-I) aplica mutações dentro do corpo da unidade g. Esse grupo requer a preparação da mutação no ponto onde f chama g tal que a mutação é aplicada somente se a unidade g é chamada através desse ponto; caso contrário, g se comporta como no programa original. Quando os operadores de mutação do Grupo-I são aplicados, é preciso alterar a condição de alcançabilidade [DEL98a]:

- Condição de Alcançabilidade para Mutação de Interface: a execução do mutante P_i, relacionado a uma conexão f-g, com o caso de teste t, deve fazer com que o controle do programa alcance o ponto onde a mutação foi realizada através da chamada de f para g.

O segundo grupo de operadores (Grupo-II) é aplicado nos pontos onde a unidade f faz chamadas à g.

No teste da conexão f-g acima, considere que determinado operador op fosse aplicado a um único ponto no interior da unidade g produzindo uma função mutante g'. Desse modo, visto que existem duas chamadas à unidade g dentro de f, dois mutantes deveriam ser gerados; o

primeiro é aquele no qual a primeira chamada de *g* é substituída por *g'* (Figura 5(a)) e o segundo é aquele em que a segunda chamada é substituída (Figura 5(b)).

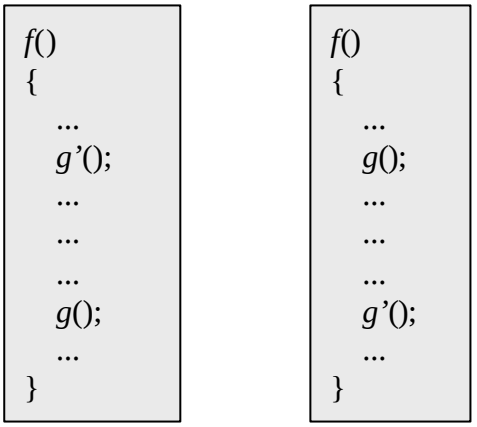


Figura 5 – Mutantes Associados às Chamadas de *g* em *f* [DEL97c]

Por isso, uma mutação aplicada dentro da função chamada só deve ser efetivamente habilitada se a função foi chamada do ponto cuja conexão deseja-se testar. No exemplo da Figura 5, ao se executar o mutante (a) deve-se assegurar que a primeira chamada à função *g* seja substituída pela chamada à função mutante *g'*; entretanto, a segunda chamada à função *g* deve permanecer inalterada de modo que o mutante (a) só poderá ser distinguido através da primeira chamada à *g*. Com isso pode-se exercitar de maneira mais completa a conexão que, na verdade, é composta por mais de uma chamada de função.

Segundo Delamaro [DEL97c], o conjunto de operadores proposto é experimental e foi criado com base, principalmente, em sua experiência no uso do teste de mutação. Além disso, o conjunto proposto busca ser completo, no sentido de exercitar a maioria das interações entre duas unidades mas, por outro lado, tenta também ser restrito, no sentido de que as mutações sejam restritas aos pontos essenciais ao teste de integração, mantendo baixo o custo de aplicação do critério. Entretanto, conforme ressalta o próprio autor, analisando-se a complexidade dos operadores definidos, pode-se esperar que a quantidade de mutantes gerados pelo critério Mutação de Interface seja bastante alta, mesmo para o teste de uma conexão formada por duas unidades simples.

Desse modo, da mesma forma que como foram desenvolvidos critérios de mutação alternativos para reduzir o custo de aplicação da Análise de Mutantes, Delamaro [DEL97c] definiu critérios alternativos para a Mutação de Interface.

O critério *IM*-aleatório seleciona aleatoriamente, para cada operador de mutação de interface, somente uma porcentagem dos mutantes. O critério *IM*-restrito utiliza apenas os

mutantes criados por um subconjunto dos operadores de mutação de interface e o critério *IM-N*-seletivo deixa de utilizar os *N* operadores de mutação de interface responsáveis por gerar o maior número de mutantes. Esses critérios de mutação alternativos podem ser combinados entre si e permitem o estabelecimento de estratégias incrementais de teste utilizando o critério Mutação de Interface. Nessas estratégias, aplica-se uma seqüência de critérios *IM*-alternativos, iniciando-se com o critério de menor custo e supostamente menos efetivo, incrementando-o até que todos os operadores de mutação de interface tenham sido utilizados ou as restrições de custo permitirem [DEL97c].

No Apêndice A são apresentados os 33 operadores de mutação de interface juntamente com uma breve descrição de cada um deles. Todos esses operadores estão implementados na ferramenta *Proteum/IM* (descrita com no Apêndice B), desenvolvida para apoiar o critério Mutação de Interface.

A existência dessa ferramenta possibilitou que Delamaro conduzisse alguns estudos empíricos utilizando o programa *Sort* do UNIX e o programa *Space* da Agência Espacial Européia visando a avaliar o custo e a eficácia em revelar a presença de erros do critério Mutação de Interface [DEL97c, DEL98a, DEL98b]. Além disso, nesses experimentos, alguns estudos iniciais envolvendo o critério Mutação de Interface e critérios de mutação de interface alternativos deram indícios de que é possível reduzir os custos de aplicação do Mutação de Interface sem perda significativa da sua eficácia em revelar a presença de erros. Na Seção 2.8.2 ambos os experimentos são descritos mais detalhadamente.

2.7 – Automatização da Atividade de Teste

A qualidade e produtividade da atividade de teste são dependentes do critério de teste utilizado e da existência de uma ferramenta de teste que o suporte. Sem a existência de uma ferramenta automatizada a aplicação de um critério torna-se uma atividade propensa a erros e limitada a programas muito simples.

A disponibilidade de ferramentas de teste permite a transferência de tecnologia para as indústrias e contribui para uma contínua evolução de tais ambientes, fatores indispensáveis para a produção de software de alta qualidade. Além disso, tais ferramentas auxiliam pesquisadores e alunos de Engenharia de Software a adquirir os conceitos básicos e experiência na comparação, seleção e estabelecimento de estratégias de teste.

Outro fator importante é o suporte oferecido pelas ferramentas aos testes de regressão. Os casos de teste utilizados durante a atividade de teste podem ser facilmente obtidos para revalidação do software após uma modificação. Com isso, é possível checar se a funcionalidade do software foi alterada, reduzir o custo para gerar os testes de regressão e comparar os resultados obtidos nos testes de regressão com os resultados do teste original [SOU96].

O primeiro sistema construído para apoiar o critério Análise de Mutantes foi o ***FMS.1*** (*Fortran Mutation System* – Versão 1) desenvolvida na *Yale University* em 1979. FMS.1 tratava somente de um subconjunto do Fortran, ou seja, programas compostos apenas de uma única subrotina com aritmética de inteiros e sem comandos de entrada e saída. O sucesso desse projeto foi suficiente para motivar a construção de outros sistemas mais elaborados. Com o FMS.2, também desenvolvido na *Yale University* e posteriormente no *Georgia Institute of Technology*, já era possível testar programas com várias subrotinas em ANSI Fortran, exceto comandos de entrada e saída.

O *Cobol Mutation System* (***CMS.1***) foi desenvolvido no *Georgia Institute of Technology* por Allen Acree, Richard DeMillo, Jeanne Hanks e Fred Sayward, baseado no *Pilot Mutation System* (PIMS, nome anterior do FMS.1) [ACR80].

Em 1981, Budd [BUD81] desenvolveu na *Yale University* um sistema de mutação para programas Fortran – ***EXPER*** (*Experimental Mutation System*) – baseado na mesma idéia dos sistemas acima.

Atualmente, novas ferramentas vêm sendo propostas a fim de suprir as deficiências encontradas nas anteriores. Entre elas destacam-se a ***Mothra*** [DEM88], a ***Proteum*** [DEL93b] e a ***Proteum/IM*** [DEL97c].

Mothra é uma ferramenta de apoio ao critério Análise de Mutantes para o teste de programas na linguagem Fortran-77 desenvolvida na *Purdue University* e *Georgia Institute of Technology* e possui 22 operadores de mutação [CHO89, DEM88]. Além disso, a ferramenta apresenta interface baseada em janelas, facilitando a visualização das informações, e permite a incorporação de outras ferramentas (gerador de casos de teste, verificador de equivalência e oráculo). Atualmente, a *Mothra* possui acoplada a si uma ferramenta para geração automática de dados de teste chamada ***Godzilla*** [DEM91].

As ferramentas *Proteum* [DEL93b] e *Proteum/IM* [DEL97c] foram desenvolvidas no Instituto de Ciências Matemáticas e de Computação – ICMC/USP e apóiam os critérios Análise de Mutantes e Mutação de Interface respectivamente. Ambas são ferramentas multi-linguagem,

ou seja, permitem testar programas escritos em diferentes linguagens de programação; atualmente estão configuradas para o teste de programas escritos na linguagem C. Considerando que essas ferramentas serão utilizadas na condução dos experimentos referentes a este trabalho, no Apêndice B estão descritas com detalhes as principais características de cada uma delas.

Como dito anteriormente vários estudos vêm sendo conduzidos na tentativa de se desenvolver estratégias de teste de baixo custo e que permitam revelar grande parte dos erros existentes em produtos de software. A seguir são apresentados alguns desses estudos empíricos, principalmente os relacionados ao critério Análise de Mutantes, a descrição desses estudos foi extraída de trabalhos anteriores [SOU96, VIN97].

2.8 – Estudos Empíricos

Segundo Wong [WON94b], o **custo**, a **eficácia** e a **dificuldade de satisfação** (*strength*) são os fatores básicos para comparar a adequação de um critério de teste, sendo que o custo refere-se ao esforço necessário para que o critério seja usado; a eficácia refere-se à capacidade que um critério possui em revelar a presença de um maior número de erros em relação a outro, e a dificuldade de satisfação refere-se à probabilidade de satisfazer um critério C_1 tendo satisfeito um critério C_2, ou seja, qual a probabilidade de conjuntos de casos de teste que satisfaçam a todos os requisitos de teste exigidos pelo critério C_1 também satisfazerem a todos os requisitos de teste exigidos por C_2.

Utilizando-se esses fatores comparativos, duas abordagens são utilizadas com o objetivo de encontrar formas econômicas e produtivas para se desenvolver os testes: a **abordagem teórica** e a **abordagem empírica**. Na abordagem teórica procura-se estabelecer propriedades e características dos critérios de teste, por exemplo, a eficácia de uma estratégia de teste ou uma relação de inclusão entre os critérios. Na abordagem empírica dados e estatísticas são coletados os quais registram, por exemplo, a freqüência na qual diferentes estratégias de teste revelam a presença de erros em uma determinada coleção de programas [HOW78], fornecendo diretrizes para a escolha entre os diversos critérios disponíveis.

Com o objetivo de avaliar a grande diversidade de critérios de teste existentes, os estudos empíricos intensificaram-se nos últimos anos [NTA88, WEY90, WEY91, MAT93, DEM94, MAT94, WON94b, WON94a, OFF94, OFF96a, OFF96b, SOU96, DEL97b, DEL97c, WON97, BAR98a, BAR98b, DEL98a, DEL98b]. O objetivo principal desses estudos é desenvolver uma estratégia de teste que apresente um baixo custo de aplicação, uma alta eficácia em revelar a presença de erros e que possa ser aplicada de modo incremental, ou seja, dependendo da

qualidade e da confiabilidade que se quer garantir, do tempo e recursos disponíveis, o testador decide quais critérios aplicar e quando aplicá-los [MAL97].

A seguir, são descritas uma série de experimentos realizados com o objetivo de comparar o critério Análise de Mutantes com outras abordagens de mutação alternativas, bem como com critérios Baseados em Análise de Fluxo de Dados.

2.8.1 – Experimentos Envolvendo o Critério Análise de Mutantes

Mathur e Wong [MAT93] compararam Mutação Aleatória (*Randomly Selected X% Mutation*) e Mutação Restrita (*Constrained Mutation*). Esse experimento foi conduzido para comparar qual dessas abordagens apresenta melhor relação custo/eficácia. Segundo os autores, Mutação Restrita e Mutação Aleatória mostram-se igualmente eficazes, obtendo-se sensível redução no número de mutantes a serem analisados sem perda significativa na eficácia dos critérios em revelar a presença de erros.

Em outro trabalho realizado por Mathur e Wong [MAT94], os critérios Análise de Mutantes e Todos-Usos foram comparados com o objetivo de verificar a dificuldade de satisfação entre os dois critérios, bem como seus custos. Neste estudo, os conjuntos de casos de teste adequados ao critério Análise de Mutantes também se mostraram adequados ao critério Todos-Usos. No entanto, os conjuntos de casos de teste adequados ao critério Todos-Usos não se mostraram, em muitos dos casos, adequados para o critério Análise de Mutantes. Esses resultados demonstram que o critério Análise de Mutantes é mais difícil de satisfazer que o critério Todos-Usos, podendo-se dizer, na prática, que Análise de Mutantes inclui Todos-Usos [MAT94].

Wong *et al*. [WON94b] utilizaram a Mutação Aleatória (10%) e a Mutação Restrita para comparar o critério Análise de Mutantes com o critério Todos-Usos; o objetivo era verificar o custo, eficácia e dificuldade de satisfação desses critérios. Como os próprios autores concluíram, Mutação Restrita, Mutação Aleatória (10%) e Todos-Usos representam, nesta ordem, o decréscimo do custo necessário para a aplicação do critério, ou seja, o critério Todos-Usos requer menos casos de teste para ser satisfeito do que a Mutação Restrita. Em relação à eficácia, a ordem (do mais eficaz para o menos) é Mutação Restrita, Todos-Usos e Mutação Aleatória. Observou-se, com isso, que examinar somente uma pequena porcentagem de mutantes pode ser uma heurística útil na avaliação e construção de conjuntos de casos de teste na prática. Desse modo, quando o testador possui pouco tempo para efetuar os testes (devido ao prazo de entrega do produto) pode-se usar o critério Análise de Mutantes para testar partes críticas do software e

alternativas mais econômicas, tal como a Mutação Restrita ou o critério Todos-Usos, para o teste das demais partes do software.

Offutt *et al.* [OFF96a] também realizaram um experimento comparando o critério Análise de Mutantes com o critério Todos-Usos. Os resultados foram semelhantes àqueles obtidos por Wong *et al.* [WON94b], ou seja, o critério Análise de Mutantes revelou a presença de um maior número de erros do que o critério Todos-Usos e mais casos de testes foram necessários para satisfazer o critério Análise de Mutantes. Além disso, os conjuntos de casos de teste AM-adequados foram adequados ao critério Todos-Usos, não sendo o inverso verdadeiro, resultado semelhante ao de Mathur e Wong [MAT94].

Nos trabalhos de Wong *et al.* [WON94a] e Souza [SOU96] seis diferentes critérios de mutação restrita foram comparados quanto ao custo e eficácia. Com isso, foi possível o estabelecimento de uma ordem incremental para a utilização desses critérios com base na eficácia e custo de cada um. Dessa maneira, os conjuntos de casos de testes podem ser construídos inicialmente de forma a serem adequados ao critério com menor relação custo/eficácia. Em seguida, quando as restrições de custo permitirem, esses conjuntos podem ser melhorados de modo a satisfazer os critérios com maior relação custo/eficácia.

Souza [SOU96] realizou um estudo empírico com a finalidade de avaliar o *strength* e o custo do critério Análise de Mutantes empregando, para efeito comparativo, os critérios Potenciais-Usos [MAL91], os quais incluem o critério Todos-Usos. Os resultados demonstram que o custo de aplicação do critério Análise de Mutantes, estimado pelo número de casos de teste necessário para satisfazer o critério, apresentou-se maior do que o custo dos critérios Potenciais-Usos. Em relação à dificuldade de satisfação (*strength*) observa-se que, de uma maneira geral, os critérios Análise de Mutantes e Todos-Potenciais-Usos não possuem grandes diferenças entre si e podem ser vistos como equivalentes, do ponto de vista da relação de inclusão. Já os critérios Todos-Potenciais-Usos/Du e Todos-Potenciais-Du-Caminhos apresentam maior *strength* que o critério Todos-Potenciais-Usos em relação à Análise de Mutantes, o que motiva a investigar-se o aspecto complementar desses critérios quanto à eficácia.

Ainda nesse trabalho, Souza avaliou o efeito da minimização de conjuntos de casos de teste na eficácia em revelar a presença de erros do critério Análise de Mutantes. A estratégia de minimização visa a obter uma melhor estimativa e uma redução do custo associado à utilização do critério, visto que, na maioria das vezes, o custo é estimado pelo tamanho do conjunto de casos de teste necessário para satisfazer o critério. Além disso, os testes de regressão também

têm seus custos diminuídos, uma vez que utilizam o conjunto de casos de teste obtido durante a fase de desenvolvimento como base para a revalidação do software. Os resultados obtidos demonstram que a minimização proporciona uma redução significativa no tamanho dos conjuntos de casos de teste, principalmente para conjuntos com maior cardinalidade. Em relação à eficácia dos conjuntos mínimos houve, em alguns casos, uma pequena redução comparada à eficácia dos conjuntos não mínimos [SOU96].

A seguir, dois experimentos que comparam o critério Mutação de Interface com critérios de mutação de interface alternativos são descritos. Como será visto, o critério Mutação de Interface apresenta uma alta eficácia em revelar a presença de erros mas, da mesma forma como a Análise de Mutantes, seu alto custo de aplicação exige a utilização de abordagens alternativas que viabilizem sua utilização de maneira mais econômica, sem degradar sua eficácia em revelar a presença de erros.

2.8.2 – Experimentos Envolvendo o Critério Mutação de Interface

Para verificar o custo e a eficácia do critério Mutação de Interface em revelar a presença de erros de integração, bem como compará-lo com alguns critérios de Mutação de Interface alternativos, Delamaro [DEL97b, DEL98a] realizou um experimento utilizando o programa *Sort* do UNIX e a ferramenta *Proteum/IM*. Como a *Proteum/IM* permite a seleção de subconjuntos de operadores de mutação, os seguintes critérios alternativos foram selecionados:

- critério *IM*-aleatório utilizando 10% dos mutantes selecionados aleatoriamente;
- critério *IM*-restrito com operadores do Grupo-I;
- critério *IM*-restrito com operadores do Grupo-II;
- critério *IM*-restrito com operadores aplicados em variáveis de interface (operadores 'DirVar');
- critério *IM*-restrito com operadores aplicados em variáveis que não são de interface (operadores 'IndVar');
- critério *IM*-restrito com operadores aplicados aos comandos de retorno (operadores 'RetSta'); e
- critério *IM*-*N*-seletivo, com *N* variando de 1 a 30.

Nesse estudo, o critério Mutação de Interface apresentou uma eficácia de quase 100% para todos os experimentos. Dos 330 conjuntos de casos de teste utilizados, apenas 1 não foi

capaz de revelar a presença de erro. De forma similar, os critérios de mutação alternativos, também apresentaram alta eficácia.

Em termos de custo, observou-se que a utilização de todos os operadores de mutação tende a gerar um grande número de mutantes, além de requerer mais casos de teste, se comparado com os critério alternativos. Por exemplo, para o experimento realizado o critério Mutação de Interface gerou 2.309 mutantes e selecionou, em média, 41,34 casos de teste. Utilizando-se o critério *IM*-10%-aleatório obteve-se uma redução de custo de aproximadamente 90% em termos do número de mutantes gerados e 50% no número de casos de teste requeridos e mesmo assim, a eficácia média do critério IM-10%-aleatório foi de 99,27%, 0,37% inferior a obtida com o critério Mutação de Interface.

Utilizando-se o critério *IM*-19-seletivo, a redução de custo no número de mutantes foi de aproximadamente 95% e no número de casos de teste ficou em torno de 62% preservando uma efetividade de 97,91%.

Outro experimento realizado por Delamaro [DEL97c, DEL98b] foi conduzido utilizando o programa *Space*. Trata-se de um programa que vem sendo utilizado pela Agência Espacial Européia e que possui um histórico do seu desenvolvimento no qual encontram-se documentados os 33 defeitos encontrados durante sua evolução. O objetivo do experimento é avaliar qual o custo de aplicação e a eficácia em revelar a presença de erros dos critérios Mutação de Interface alternativos descritos abaixo. Para isso, foram geradas 10 versões do programa *Space*, cada uma contento de 14 a 17 defeitos, totalizando 159 defeitos.

A seqüência de critérios *IM*-alternativos empregadas no experimento utilizou uma combinação de três abordagens. A primeira foi a seleção de apenas um pequeno número de operadores de mutação. Os operadores de mutação utilizados para caracterizar a seqüência de critério são:

1. critério *IM*-restrito com operadores do Grupo-II;
2. critério *IM*-restrito com operadores de troca de variáveis de interface (operadores “DirVarRep”);
3. critério *IM*-restrito com operadores de incremento/decremento de variáveis de interface (operadores “DirVarIncDec”);
4. critério *IM*-restrito com operadores de inclusão de operadores unários em ocorrências de variáveis de interface (operadores “DirVarxxxNeg”);

5. critério *IM*-restrito com operadores aplicados aos comandos de retorno (operadores 'RetSta');
6. critério *IM*-restrito com operadores de troca de variáveis não de interface (operadores "IndVarRep");
7. critério *IM*-restrito com operadores de incremento/decremento de variáveis não de interface (operadores "IndVarIncDec"); e
8. critério *IM*-restrito com operadores de inclusão de operadores unários em ocorrências de variáveis não de interface (operadores "IndVarxxxNeg").

A segunda foi limitar o número máximo de mutantes que determinado operador poderia gerar por ponto de mutação. Para os critérios de 1 a 5 foi estabelecido como 2 o número máximo de mutantes por ponto de mutação. Para os demais, o número máximo foi limitado a 1 mutante. A terceira forma de reduzir o número de mutantes foi a seleção aleatória de 10% dos mutantes. A combinação dessas duas abordagens foi feita primeiro limitando-se o número máximo por ponto de aplicação e dentre os mutantes gerados, selecionou-se 10% aleatoriamente.

Além disso, em cinco conexões do programa *Space* que sempre são executadas, nenhum operador de mutação foi aplicado supondo que defeitos nelas existentes pudessem ser revelados através do teste das outras conexões. Considerando as conexões restantes, se todos os operadores de mutação que compõem as abordagens de mutação restrita fossem utilizados, seriam gerados 91.354 mutantes. Com as parametrizações impostas esse número caiu para 5.129, uma redução de aproximadamente 94%.

Os resultados demonstraram que dos 159 defeitos implantados no programa *Space*, apenas 4 não puderam ser revelados, indicando uma eficácia de 97.48%. Analisando individualmente cada experimento, observou-se que somente em 3 não foram revelados todos os defeitos implantados. Em termos de custo, observou-se que com um número reduzido de mutantes – 5.129 para um programa com 4.500 linhas de código – os resultados em termos de eficácia foram bastante satisfatórios. Da mesma forma que o experimento descrito anteriormente, muitos dos defeitos foram revelados com um número reduzido de mutantes. No caso do *Space*, utilizando somente o critério restrito formado pelos operadores do Grupo-II, 89,31% (142 de 159) dos defeitos implantados foram revelados e o número de mutantes gerados por esses operadores foi de no máximo 447 mutantes. Em seguida, aplicando-se os demais critérios *IM*-alternativos de forma incremental, observou-se que com 1.515 mutantes gerados, mais 5 defeitos foram revelados; com 2.215 mutantes, mais 8 defeitos foram revelados e com 3.946 mutantes, 1

defeito foi revelado. Entre 3.946 e 5.129 mutantes nenhum defeito foi revelado e restaram ainda 3 defeitos.

Segundo Delamaro, isso demonstra certa eficácia da estratégia incremental utilizada pois permite, que muitos dos defeitos – principalmente os mais fáceis – sejam removidos rapidamente a um baixo custo. Porém, até esse ponto, não é clara qual a melhor seqüência de operadores de mutação que deve ser empregada, pelo fato de não se ter um conhecimento preciso sobre as características de cada operador de mutação quando utilizados na prática [DEL97c].

Além das abordagens alternativas para a redução de custo dos critérios Análise de Mutantes e Mutação de Interface avaliadas nas seções acima, outros trabalhos que vêm sendo realizados dizem respeito à determinação de conjuntos essenciais de operadores de mutação. A seção a seguir apresenta a descrição dos principais experimentos nessa linha.

2.8.3 – Determinação de Conjuntos Essenciais de Operadores de Mutação

Offutt *et al.* [OFF96b] realizaram um estudo utilizando o conjunto de operadores de mutação implementados na ferramenta *Mothra*. Os 22 operadores de mutação dessa ferramenta estão divididos em 3 categorias: operadores de troca de operandos (*replacement-of-operand*), operadores de modificação de expressões (*expression modification*) e operadores de modificação de comandos (*statement modification*). Com base nessas categorias, Offutt *et al.* definiram quatro classes de mutação seletiva: *ES-selective*, *RS-selective*, *RE-selective* e *E-selective*, que utilizam operadores *expression/statement*, *replacement/statement*, *replacement/expression* e *expression*, respectivamente, na criação dos mutantes. Os resultados obtidos indicaram que a utilização de apenas 5 dos 22 operadores de mutação da ferramenta *Mothra* (exatamente os pertencentes à classe *E-selective*) proporcionou uma redução significativa de custo em termos do número de mutantes gerados (77,56%), preservando um alto índice de cobertura (escores de mutação acima de 98%) em relação à Análise de Mutantes.

Um estudo preliminar realizado por Wong *et al.* [WON97], comparando a Mutação Restrita no contexto das linguagens C e Fortran, forneceu indícios de que os operadores da ferramenta *Proteum* [DEL93b] utilizados no experimento possuem uma forte relação com os operadores essenciais obtidos por Offutt *et al.* [OFF96b] para a linguagem Fortran. Os resultados obtidos demonstram que a maioria dos critérios restritos utilizados apresentaram praticamente a mesma eficácia em revelar a presença de erros do critério Análise de Mutantes a um custo de aplicação inferior. Além disso, os operadores selecionados por Wong *et al.* constituem um

conjunto preliminar para a determinação de um conjunto essencial de operadores de mutação para C.

Utilizando um conjunto de 27 programas que compõem um editor de texto simplificado e também um conjunto de 5 programas utilitários do UNIX, Barbosa [BAR98a] conduziu um experimento para a determinação de um conjunto essencial de operadores de mutação para a linguagem C. Inicialmente, a autora reproduziu o experimento de Offutt *et al*. [OFF96b] no contexto dessa linguagem e também avaliou o conjunto sugerido por Wong *et al*. [WON97] nesses conjuntos de programas. Visto que os resultados obtidos não foram tão satisfatórios, Barbosa procurou identificar quais as principais características que deveriam estar presentes em um conjunto essencial de operadores de mutação e desenvolveu um procedimento genérico, denominado **Essencial**, para a determinação do conjunto essencial de operadores de mutação. A aplicação do procedimento desenvolvido utilizando os operadores de mutação implementados na ferramenta *Proteum* [DEL93b] resultou na determinação de um conjunto essencial de operadores de mutação para a linguagem C.

Os resultados obtidos (Tabela 1) demonstraram que o conjunto essencial obtido com a estratégia de Barbosa, embora apresente maior custo, é o que proporciona o maior escore de mutação em relação ao critério Análise de Mutantes se comparado com o conjunto essencial de Wong *et al*. [WON97] e o obtido com a estratégia de Offutt *et al*. [OFF96b]. Além disso, se for utilizado apenas parte do conjunto essencial de Barbosa (apenas o melhor operador de cada classe de mutação) o custo de aplicação passa a ser inferior ao obtido pelas duas outras estratégias e, ainda assim, o escore de mutação obtido em relação à Análise de Mutantes é superior.

Tabela 1 – Resultados Obtidos por Barbosa [BAR98a]

Critério	Escore	Custo
Conjunto Essencial (Barbosa)[11]	0,99616	0,34139
Conjunto Essencial Restrito (Barbosa)[12]	0,98505	0,20118
Conjunto Essencial (Offutt *et al*.)[13]	0,97143	0,21061
Conjunto Essencial (Wong *et al*.)[14]	0,98107	0,27152

Visto que o critério Mutação de Interface apresenta o mesmo problema de custo do critério Análise de Mutantes, faz-se necessário investigar se a aplicação do procedimento proposto por Barbosa [BAR98a] resulta em um conjunto essencial de operadores de mutação de

[11] Conjunto Essencial obtido por Barbosa = {SWDD, SMTC, SSDL, OLBN, ORRN, VTWD, VDTR, Cccr, Ccsr}
[12] Conjunto Essencial Restrito obtido por Barbosa = {SSDL, ORRN, VTWD, Ccsr}
[13] Conjunto Essencial obtido reproduzindo-se o experimento de Offutt *et al.* no contexto da linguagem C = {Cccr, Ccsr, CRCR}

interface que levam a resultados semelhantes aos obtidos com o conjunto essencial em nível de unidade. Para isso, na Seção 3.3.2, o procedimento desenvolvido por Barbosa é descrito e aplicado a um conjunto de cinco programas utilizando os operadores de mutação de interface implementados na ferramenta *Proteum/IM*.

2.9 – Considerações Finais

Neste capítulo foram apresentadas as principais características da atividade de teste, as fases necessárias para sua condução e as técnicas e critérios de teste mais utilizados. É importante destacar o aspecto complementar destas técnicas, visto que cada uma delas se baseia em um conjunto diferente de informações para derivar os requisitos de teste. Ênfase foi dada ao critério Análise de Mutantes, sendo apresentadas suas principais características e algumas das ferramentas que apóiam sua aplicação.

Embora os estudos empíricos tenham demonstrado ser o critério Análise de Mutantes altamente eficaz em revelar a presença de erros [ACR79, DEM80, BUD80a, NTA88, WEY90, WEY91, HOR92, FRA93, MAT93, OFF93, WON93, DEM94, MAT94, WON94b, WON94a, SOU96, WON97], problemas relacionados ao grande número de mutantes gerados tornam necessário o desenvolvimento de abordagens que permitam a sua utilização de forma economicamente viável. As abordagens mais exploradas nesse sentido são as que procuram reduzir o número de mutantes gerados, dentre elas Mutação Aleatória e Mutação Restrita. A eficácia dessas abordagens foi investigada em alguns estudos empíricos e os resultados obtidos demonstram que é possível reduzir sensivelmente o custo de aplicação do critério Análise de Mutantes sem que haja uma perda significativa em sua eficácia em revelar a presença de erros.

Ainda, tendo em vista a eficácia do critério Análise de Mutantes em revelar a presença de erros de unidade e a necessidade de desenvolver critérios mais rigorosos para o teste de integração, foi apresentado o critério Mutação de Interface [DEL97c] que estende os conceito da Análise de Mutantes para teste de integração. Alguns estudos empíricos relacionados ao critério Mutação de Interface [DEL97b, DEL97c, DEL98a, DEL98b] demonstram que, assim como a Análise de Mutantes, ele apresenta uma alta eficácia e também um alto custo de aplicação em termos do número de mutantes gerados.

[14] Conjunto Essencial obtido por Wong *et al.* = {OLLN, OLNG, ORRN, VDTR, VTWD, STRP}

Essas conclusões porém não significam que o critério Mutação de Interface não deva ser utilizado, mas sugerem que uma estratégia incremental para a sua aplicação seja definida, aspecto este explorado no próximo capítulo.

Além disso, com a existência dos critérios Análise de Mutantes e Mutação de Interface, que utilizam o mesmo conceito de mutação nas diferentes fases do teste (unidade e integração), é natural questionar qual o relacionamento existente entre eles e como utilizá-los de forma complementar na atividade de teste, reduzindo os custos de aplicação associados a tais critérios e enfatizando-se o teste de unidade e o teste de integração. Desse modo, um estudo empírico envolvendo ambos os critérios também foi realizado e será descrito no Capítulo 4.

Capítulo 3

Operadores de Mutação de Interface: Uma Avaliação Empírica

3.1 – Considerações Iniciais

De um modo geral, os estudos empíricos descritos anteriormente dão evidências da alta eficácia apresentada pelos critérios Análise de Mutantes e Mutação de Interface quando utilizados nas fases do teste de unidade e integração. Além disso, também observou-se que ambos os critérios apresentam problemas de custo que, de certa forma, dificultam a utilização prática dos mesmos. Entretanto, a utilização de alternativas, tais como a determinação de conjuntos essenciais de operadores de mutação mostra-se como uma boa estratégia para reduzir os custos de aplicação desses critérios.

Assim sendo, o objetivo deste trabalho é analisar qual a relação existente entre os critérios Análise de Mutantes e Mutação de Interface, buscando subsídios para o estabelecimento de uma estratégia de teste incremental que permita reduzir os custos de aplicação desses critérios quando aplicados nas fases de unidade e integração, respectivamente (Figura 6). Como visto anteriormente, duas são as abordagens que vêm sendo utilizadas para a comparação entre os critérios de teste: abordagem teórica e empírica. Neste trabalho, é utilizada a abordagem empírica.

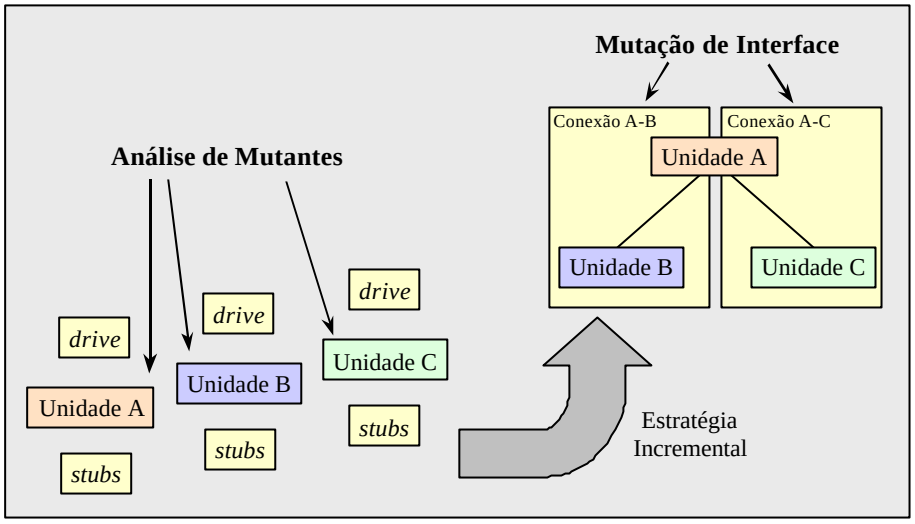


Figura 6 – Aplicação dos Critérios Análise de Mutantes e Mutação de Interface

A condução de estudos empíricos envolve o levantamento de uma série de requisitos que possibilitam avaliar a viabilidade e a disponibilidade de recursos para a realização dos mesmos. Com base nesses requisitos é elaborado um *framework* que organiza a condução dos experimentos a serem realizados. Além disso, quando a quantidade de informações a ser coletada é relativamente grande, organizar a condução dos experimentos em etapas facilita o entendimento dos mesmos e contribui para a coleta e análise dos dados. Os experimentos realizados neste trabalho foram divididos em três etapas.

Neste capítulo são realizados alguns estudos empíricos envolvendo o critério Mutação de Interface os quais possibilitam analisar a relação existente entre os operadores de mutação de interface, estabelecer uma estratégia incremental para a aplicação dos mesmos e, com a aplicação do procedimento **Essencial** definido por Barbosa [BAR98a], determinar o conjunto essencial de operadores de mutação de interface.

Em uma segunda etapa, descrita no Capítulo 4, comparou-se o custo de aplicação dos critérios Análise de Mutantes e Mutação de Interface e o *strength* de um em relação ao outro buscando-se subsídios para o estabelecimento de uma estratégia incremental de aplicação de seus operadores de mutação de modo que a maioria dos requisitos de teste de ambos seja satisfeita a um menor custo, ou seja, utilizando-se menos operadores de mutação e, conseqüentemente, gerando menos mutantes. Finalmente, essa mesma comparação é realizada considerando-se os conjuntos essenciais de operadores de mutação dos critérios Análise de Mutantes [BAR98a] e Mutação de Interface, determinado neste capítulo.

3.2 – Descrição do *Framework* dos Experimentos

Em geral, um *framework* de estudo empírico é composto das seguintes atividades: seleção de programas (*benchmark*), geração de conjuntos de casos de teste, seleção de ferramentas de teste, aplicação do *benchmark* e a coleta e análise dos resultados. A seguir, cada uma dessas atividades é detalhada considerando as informações sobre os experimentos referentes a este trabalho.

Seleção dos Programas

Todo resultado obtido com estudos empíricos é dependente do conjunto de programas utilizado. Desse modo, a escolha dos programas é uma das atividades mais importantes na preparação de um experimento e deve ser realizada com cuidado, tendo-se em mente os objetivos que se deseja atingir.

Este experimento foi conduzido a partir de um grupo formado por cinco programas utilitários do UNIX escritos na linguagem C. Tais programas além de terem sido utilizados em estudos anteriores [WON93, WON97], devido ao seu uso intenso, possuem uma baixa probabilidade de apresentarem erros naturais em seu código. Tal característica torna tais programas bastante apropriados à injeção de defeitos visando, em trabalhos futuros, à análise de eficácia das estratégias aqui propostas. A Tabela 2 apresenta a descrição e total de linhas de código (LOC) dos programas utilizados. Na Tabela 3 são apresentados o total de mutantes gerados e equivalentes considerando todos os operadores do critério Análise de Mutantes. Na Tabela 4, são apresentados o total de número de mutantes gerados e equivalentes obtidos pela aplicação dos operadores de mutação do critério Mutação de Interface.

Tabela 2 – Conjunto de Programas do Experimento

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

Tabela 3 – Complexidade dos Programas: Ferramenta *Proteum*

Programa	Mutantes Gerados	Mutantes Equivalentes	Porcentagem de Equivalentes
Cal	4.332	332	7,70%
Checkeq	3.099	223	7,20%
Comm	1.728	195	11,30%
Look	2.056	257	12,50%
Uniq	1.619	171	10,60%
Total	12.834	1.178	9,20%

Tabela 4 – Complexidade dos Programas: Ferramenta *Proteum/IM*

Programa	Mutantes Gerados	Mutantes Equivalentes	Porcentagem de Equivalentes
Cal	4.350	568	13,10%
Checkeq	2.954	18	0,60%
Comm	2.952	517	17,50%
Look	1.825	353	19,30%
Uniq	1.943	220	11,30%
Total	14.024	1.676	12,00%

Um ponto importante a ser destacado é que a determinação dos mutantes equivalentes é uma questão indecidível, exigindo do testador um completo conhecimento da funcionalidade dos programas. Essa atividade foi feita manualmente e as informações coletadas são de fundamental importância para o desenvolvimento de heurísticas que permitam sua automatização, reduzindo o custo e o esforço de aplicação dos critérios.

Seleção de Ferramentas de Teste

A seleção das ferramentas de teste deve levar em consideração quais critérios serão analisados, a linguagem nas quais os programas estão escritos e, principalmente, o suporte oferecido para a realização de experimentos, ou seja, deve-se observar se a funcionalidade e o conjunto de informações disponibilizados pela ferramenta são suficientes para atingir os objetivos propostos.

As atividades típicas de uma ferramenta de teste são: analisar o código fonte, auxiliar na instrumentação do código, medir a cobertura fornecida por um conjunto de casos de teste e fornecer relatórios [DEU92]. Especificamente para a aplicação dos critérios baseados em mutação, essas atividades se traduzem em um conjunto essencial de tarefas que devem ser executadas:

- definição de casos de teste;
- execução do programa em teste;
- geração de mutantes;
- execução dos mutantes;
- análise dos mutantes vivos;
- cálculo do escore de mutação; e
- geração de relatórios.

Para os experimentos em questão, as únicas ferramentas disponíveis atualmente que apóiam a aplicação dos critérios Análise de Mutantes e Mutação de Interface para o teste de programas na linguagem C são a *Proteum* [DEL93b] e a *Proteum/IM* [DEL97c], respectivamente. Tais ferramentas possuem uma série de características que as tornam adequadas para a realização deste trabalho.

O teste utilizando-se a *Proteum* ou a *Proteum/IM* é dirigido por **sessões de teste**. O estado de uma sessão de teste é caracterizado por uma base de dados identificada por um nome e composta, basicamente, por dados sobre os casos de teste e sobre os mutantes utilizados e alguns arquivos intermediários que descrevem o programa sendo testado. Uma sessão de teste é constituída de uma seqüência de operações realizadas sobre essa base de dados através de chamadas aos programas que compõem as ferramentas [DEL97c].

A vantagem de se utilizar as sessões de teste é que o testador pode, através das operações disponíveis nas ferramentas, obter diferentes resultados a partir de uma única sessão de teste, realizando operações sobre a mesma. Por exemplo, ambas as ferramentas oferecem as seguintes funcionalidades:

- habilitar e desabilitar diferentes grupos de mutantes;
- habilitar e desabilitar diferentes operadores de mutação;
- habilitar e desabilitar diferentes conjuntos de casos de teste.

Desse modo, diferentes combinações de conjuntos de casos de teste podem ser avaliadas com diferentes conjuntos de mutantes em uma mesma sessão de teste. Pode-se, por exemplo, avaliar qual a capacidade de um conjunto de casos de teste adequado a um único operador em distinguir os mutantes gerados pelos demais, característica essa fundamental para a condução dos experimentos realizados neste trabalho. No Apêndice B são apresentadas informações adicionais sobre as ferramentas *Proteum* e *Proteum/IM*.

Geração dos Conjuntos de Casos de Teste

Embora o teste exaustivo – testar um programa para todos os possíveis valores de entrada – possa ser desejável, na prática isto se torna inviável por razões de custo e tempo superiores do que o disponível. A maioria das técnicas e critérios apresentados anteriormente auxiliam a atividade de teste dividindo o domínio de entrada do programa em subdomínios e fazendo com que pelo menos um ponto de cada subdomínio seja executado.

A tarefa de geração de dados de teste visa a selecionar pontos de cada subdomínio a fim de satisfazer um determinado critério, revelando um maior número de erros possível. Embora a automatização dessa tarefa seja desejável, não existe um algoritmo de propósito geral para determinar um conjunto de dados de teste que satisfaça um dado critério. Não é possível nem mesmo determinar se esse conjunto existe, o que torna o problema de geração de dados de teste indecidível.

As principais limitações inerentes à geração de dados de teste são:

- **correção coincidente**: ocorre quando um programa possui um defeito que é executado por um dado de teste; um estado de erro é produzido, mas coincidentemente um resultado correto é obtido;
- **caminhos ausentes**: quando uma certa funcionalidade deixa de ser implementada no programa, não existe um caminho que corresponda àquela funcionalidade e, como conseqüência, nenhum dado de teste será requerido para exercitá-lo;
- **caminhos não executáveis**: um caminho é não executável se não existir um conjunto de valores para as variáveis de entrada, parâmetros e variáveis globais que causem a sua execução;
- **mutantes equivalentes**: a modificação feita no programa original para criar um mutante não altera a função implementada, ou seja, o programa mutante implementa a mesma função que o programa original.

Visto que os melhores pontos do domínio para a seleção de dados de teste são aqueles com maior probabilidade de revelar erros, a estratégia empregada para selecionar esses pontos é de fundamental importância, pois dela dependerá a efetividade do critério.

As principais estratégias para gerar dados são: geração aleatória [ACR79], geração com execução simbólica [BOY75, CLA76, RAM76, HOW77], geração dinâmica [KOR90] e técnicas de geração de dados sensíveis a erros (*ad hoc*) [WHI80, DEM91, TAI93]. Dentre elas, a mais utilizada para a condução de estudos empíricos tem sido a geração aleatória, na qual pontos do domínio são selecionados aleatoriamente. Esta estratégia não garante a seleção dos melhores pontos, nem mesmo a satisfação do critério. Entretanto, apesar dessas limitações, ela é defendida por vários autores [ACR80, BUD80b, DUR84, HAM88] por sua facilidade de automatização e capacidade de gerar grandes conjuntos de dados de teste a baixo custo. Além disso, a técnica

aleatória elimina qualquer possível influência do testador em conduzir a geração dos dados de teste conforme o conhecimento prévio dos programas utilizados.

Para a geração dos dados de teste utilizados nos experimentos descritos neste trabalho, utilizou-se, principalmente, a estratégia de geração aleatória e os mecanismos desenvolvidos por Wong [WON93]. Para cada programa foi criado um p*ool* de dados de teste o qual possui a seguinte composição:

- dados de teste funcionais *ad hoc* (DT_{func}): gerados a partir da especificação funcional de cada programa; e
- dados de teste aleatórios (DT_{aleat}): gerados por scripts desenvolvidos e utilizados por Wong [WON93].

A Tabela 5 apresenta, para cada programa, a cardinalidade do *pool* de dados de teste.

Tabela 5 – Cardinalidade de Cada *Pool* de Dados de Teste

Programa	DT_{func}	DT_{aleat}	Total
Cal	1	162	163
Checkeq	12	166	178
Comm	39	754	793
Look	52	193	245
Uniq	42	431	473

Aplicação do *Benchmark*

Nesta fase, a base de dados a partir da qual as informações serão coletadas é gerada. A partir do *pool* de dados de teste, para cada programa e para cada critério (Análise de Mutantes e Mutação de Interface), 11 conjuntos de casos de teste foram gerados. Quando os dados de teste do *pool* não eram suficientes para garantir um conjunto adequado a determinado critério, dados de teste manuais (DT_{man}) foram adicionados até que um conjunto de casos de teste adequado fosse obtido. Assim sendo, 11 diferentes sessões de teste foram construídas para cada programa, cada uma contendo 1 conjunto de casos de teste adequado, totalizando 55 sessões de teste para cada critério.

É importante que diferentes conjuntos de casos de teste sejam avaliados pois, para um dado critério, pode existir um número infinito de conjuntos de casos de teste que o satisfaça e, selecionar apenas um desses conjuntos pode levar a falsas conclusões [WON97].

Nas Tabelas 6 e 7 são apresentadas as cardinalidades dos conjuntos de casos de teste adequados para os critérios Análise de Mutantes e Mutação de Interface, respectivamente. Nos conjuntos de casos de teste adequados só existem casos de teste efetivos, ou seja, casos de teste

que foram capazes de distinguir ao menos um mutante. O Gráfico 1 ilustra a média dos conjuntos de casos de teste para cada programa e critério.

Tabela 6 – Cardinalidade dos Conjuntos AM-adequados

Programas	CT_1	CT_2	CT_3	CT_4	CT_5	CT_6	CT_7	CT_8	CT_9	CT_{10}	CT_{11}	***Média***
Cal	36	26	26	25	27	27	26	26	27	27	24	27,00
Checkeq	46	39	38	40	40	39	38	40	42	40	40	40,18
Comm	41	31	30	30	30	32	32	35	29	33	34	32,45
Look	43	36	40	36	38	34	38	39	33	41	37	37,73
Uniq	28	29	27	28	27	31	28	24	28	30	26	27,82

Tabela 7 – Cardinalidade dos Conjuntos *IM*-adequados

Programas	CT_1	CT_2	CT_3	CT_4	CT_5	CT_6	CT_7	CT_8	CT_9	CT_{10}	CT_{11}	***Média***
Cal	19	19	22	22	19	17	18	17	19	18	19	19,00
Checkeq	26	23	22	20	19	21	24	22	18	21	22	21,64
Comm	66	55	53	53	59	55	55	59	57	53	60	56,82
Look	48	27	34	35	36	31	31	31	31	27	28	32,64
Uniq	44	36	39	42	35	37	32	35	33	39	32	36,73

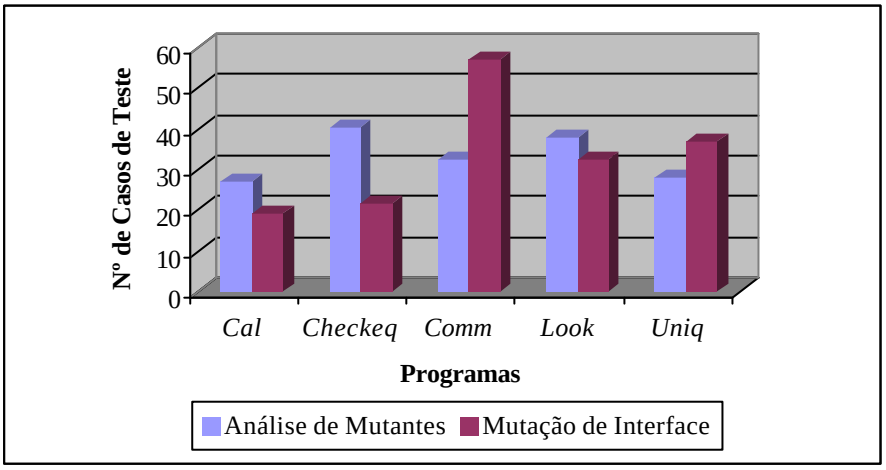


Gráfico 1 – Cardinalidade Média dos Conjuntos de Casos de Teste AM e *IM*-adequados

Observa-se na Tabela 7 que o programa *Comm* é o que necessita de mais casos de teste para se obter conjuntos *IM*-adequados. Isso se deve ao fato de que esse programa, se comparado aos demais, é o que apresenta o maior número de conexões e chamadas de função. Assim sendo, devido à condição de alcançabilidade do critério Mutação de Interface, discutida na Seção 2.6.2, é natural que o programa *Comm* necessite de mais casos de teste pois, para cada ponto de chamada de função, um grupo de mutantes de interface é gerado e casos de teste distintos que exercitem cada grupo de mutantes passando pelo ponto de chamada para o qual foram criados são requeridos.

Coleta e Análise dos Dados

Terminada a fase anterior, as informações relevantes para se atingir o objetivo proposto são coletadas a partir da base de dados construída durante a aplicação do *benchmark*.

Para cada um dos experimentos realizados, diferentes informações foram coletadas de modo a permitir, por exemplo, determinar o conjunto essencial de operadores de mutação de interface, verificar a adequação de conjuntos de casos de teste AM-adequados em relação ao critério Mutação de Interface, dentre outras. Após a coleta das informações, essas são analisadas e os resultados obtidos apresentados.

3.3 – Avaliação do Critério Mutação de Interface

A primeira etapa do experimento visa a avaliar o custo de aplicação do critério Mutação de Interface e a estabelecer uma estratégia incremental para a aplicação de seus operadores de mutação. Inicialmente, procurou-se estabelecer uma ordenação entre os 33 operadores de mutação de interface de modo a facilitar a aplicação dos mesmos incrementalmente, de acordo com a disponibilidade de tempo e recursos destinados a esta fase do teste.

Em seguida, o procedimento **Essencial** [BAR98a] foi utilizado para se chegar ao conjunto essencial de operadores de mutação de interface. Obtido tal conjunto considerou-se como ficaria a estratégia incremental de aplicação dos operadores dando-se prioridade aos operadores do conjunto essencial em relação aos demais operadores de mutação.

3.3.1 - Estratégia Incremental de Aplicação dos Operadores de Interface

Para o estabelecimento de uma estratégia incremental de aplicação de operadores de mutação é necessário, primeiramente, identificar quais aspectos devem ser priorizados, por exemplo: custo, cobertura (em termos do escore de mutação ou associações já executadas) e eficácia. No caso deste experimento serão considerados os aspectos de cobertura (escore de mutação) e custo dos operadores de mutação, ou seja, quais os *n* melhores operadores que devem ser aplicados de modo que se obtenham conjuntos de casos de teste adequados ao critério Mutação de Interface com menor custo.

Para isso, uma informação fundamental é a capacidade de conjuntos de casos de teste adequados a cada operador em distinguir os mutantes gerados pelos demais. A Tabela 8 apresenta parte desta informação coletada para um dos conjuntos de casos de teste *IM*-adequados do programa *Comm*. 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 primeira linha da Tabela 8, tem-se que, um conjunto de casos de teste I-CovAllEdg-adequado determina escores de mutação de 1,00, 0,87, 1,00, 1,00, 1,00 e 1,00 em relação aos operadores I-CovAllNod, I-DirVarAriNeg, I-DirVarBitNeg, II-ArgStcAli, II-ArgStcDif e II-FunCalDel, respectivamente.

Tabela 8 – Escore de Mutação que cada Operador Determina em Relação aos Demais: Programa *Comm*

Linha	Operador	1	2	3	4	...	31	32	33
1	I-CovAllEdg	1,00	1,00	0,87	1,00	...	1,00	1,00	1,00
2	I-CovAllNod	0,99	1,00	0,87	1,00	...	1,00	1,00	1,00
3	I-DirVarAriNeg	0,95	0,97	1,00	1,00	...	1,00	1,00	0,94
4	I-DirVarBitNeg	0,95	0,97	0,87	1,00	...	1,00	1,00	1,00
...	...	...	...	...	...	...	...	...	...
31	II-ArgStcAli	0,29	0,33	0,00	0,09	...	1,00	0,21	0,40
32	II-ArgStcDif	0,93	0,95	0,73	0,91	...	1,00	1,00	1,00
33	II-FunCalDel	0,93	0,95	0,73	0,91	...	1,00	1,00	1,00

Essa tabela foi gerada 11 vezes para cada programa, correspondendo aos 11 conjuntos de casos de teste *IM*-adequados. Em seguida, para cada programa, foi gerada a Tabela 9 contendo a média obtida para os 11 conjuntos de casos de teste.

Tabela 9 – Média para os 11 conjuntos *IM*-adequados: Programa *Comm*

Linha	Operador	1	2	3	4	...	31	32	33
1	I-CovAllEdg	1,000	0,995	0,850	0,956	...	1,000	0,981	0,948
2	I-CovAllNod	0,996	1,000	0,856	0,956	...	1,000	0,981	0,944
3	I-DirVarAriNeg	0,947	0,960	1,000	0,991	...	1,000	0,975	0,907
4	I-DirVarBitNeg	0,948	0,961	0,880	1,000	...	1,000	0,981	0,927
...	...	...	...	...	...	...	...	...	...
31	II-ArgStcAli	0,526	0,566	0,300	0,381	...	1,000	0,392	0,445
32	II-ArgStcDif	0,939	0,950	0,786	0,896	...	1,000	1,000	0,967
33	II-FunCalDel	0,954	0,964	0,811	0,921	...	1,000	1,000	1,000

Desse modo, considerando-se que o *benchmark* é composto de cinco programas, cinco tabelas sintetizando os resultados de cada programa foram geradas e a Tabela 10 apresenta a média obtida para os cinco programas. Observe que os operadores de mutação aparecem ordenados de forma decrescente pela média, ou seja, do operador que mais inclui os demais para o que menos inclui.

Seja op_1, op_2, ..., op_n o conjunto total de operadores de mutação de interface ordenados pela média de inclusão que cada um determina em relação ao outro, ou seja, para os operadores da Tabela 10, op_1=I-IndVarRepReq, op_2=I-IndVarIncDec, e op_n=II-ArgStcAli. Considere Top_i como sendo o conjunto de casos de teste adequado ao operador op_i. Um conjunto de casos de

teste $T = Top_1 \cup Top_2 \cup ... \cup Top_n$ é *IM*-adequado. Quando esses operadores de mutação são aplicados de forma incremental, pode ocorrer que determinado operador op_i seja incluído pelos demais operadores aplicados antes dele, ou seja, o conjunto de casos de teste adequado ao operador op_i (Top_i) é um subconjunto do conjunto de casos de teste formado por $Top_1 \cup ... \cup Top_{i-1}$ e, desse modo, pode-se construir um conjunto de caso de teste $T = Top_1 \cup ... \cup Top_{i-1} \cup Top_{i+1} \cup ... \cup Top_n$, tal que T continuará sendo *IM*-adequado. O ideal é construir um conjunto de casos de teste T_j *IM*-adequado formado por conjuntos de casos de teste adequados aos operadores op_1, op_2, ..., op_n ($T_j = Top_1 \cup Top_2 \cup ... \cup Top_n$), tal que qualquer que seja $op_i \in \{op_1, op_2, ..., op_n\}$ se $T_j - Top_i \neq T_j$, T_j deixa de ser *IM*-adequado.

Tabela 10 – Média dos Escores de Mutação

Linha	Operador	1	2	3	4	...	31	32	33	Média[15]	Desvio Padrão
1	I-IndVarRepReq	1,000	0,998	0,989	1,000	...	0,959	1,000	1,000	0,973	0,030
2	I-IndVarIncDec	0,993	1,000	0,982	0,991	...	0,959	1,000	1,000	0,970	0,030
3	I-IndVarBitNeg	0,986	0,990	1,000	0,969	...	0,945	1,000	1,000	0,968	0,029
4	I-IndVarLogNeg	0,990	0,993	0,971	1,000	...	0,957	1,000	1,000	0,964	0,033
...	...	...	...	...	...	...	...	...	...	...	...
31	I-RetStaDel	0,573	0,582	0,580	0,568	...	1,000	0,783	0,676	0,667	0,131
32	II-ArgDel	0,498	0,513	0,515	0,486	...	0,625	1,000	1,000	0,657	0,212
33	II-ArgStcAli	0,265	0,287	0,274	0,231	...	0,361	0,431	1,000	0,393	0,202

Com o objetivo de estabelecer uma estratégia de aplicação dos operadores de mutação que se aproxime do resultado ideal, para cada programa deve ser gerada uma tabela contendo as seguintes informações:

- **Operador (Op)**: nome do operador de mutação op_i;
- **Escore de mutação anterior (EAn)**: escore de mutação obtido pelo conjunto de casos de teste $T = Top_1 \cup Top_2 \cup ... \cup Top_{i-1}$;
- **Escore de mutação atual (EAt)**: escore de mutação obtido pelo conjunto de caso de teste $T = Top_1 \cup Top_2 \cup ... \cup Top_{i-1} \cup Top_i$;
- **Incremento (Inc)**: diferença entre o escore atual e o anterior Inc = EAt - EAn;
- **Custo do Operador (CO)**: número de mutantes gerados pelo operador op_i;
- **Custo Cumulativo (CC)**: número de mutantes gerados por todos os operadores já aplicados até o momento ($CC = \sum_{k=1}^{i} CO_k$);

[15] Os valores da média e desvio padrão referem-se aos 33 operadores de mutação de interface e não somente aos que estão representados na tabela.

- **Redução de Custo (RC)**: porcentagem de mutantes que deixaram de ser gerados até a aplicação do operador op_i (RC = (CC / $\sum_{k=1}^{n} CO_k$) * 100).

Por exemplo, na Tabela 11 são apresentadas as informações acima coletadas para um conjunto de casos de teste *IM*-adequado do programa *Comm*.

Tabela 11 – Dados Coletados para um Conjunto de Casos de Teste *IM*-adequado: Programa *Comm*

Op	EAn	EAt	Inc	CO	CC	RC
I-IndVarRepReq	0,00000	0,97829	0,97829	320	320	89,15989
I-IndVarIncDec	0,97829	0,97870	0,00041	170	490	83,40108
I-IndVarBitNeg	0,97870	0,98279	0,00409	77	567	80,79268
I-IndVarLogNeg	0,98279	0,98279	0,00000	77	644	78,18428
I-IndVarRepCon	0,98279	0,98279	0,00000	308	952	67,75068
I-DirVarIncDec	0,98279	0,98443	0,00164	210	1162	60,63686
I-DirVarRepReq	0,98443	0,98730	0,00287	150	1312	55,55556
I-DirVarRepCon	0,98730	0,98894	0,00164	54	1366	53,72629
I-IndVarRepPar	0,98894	0,99263	0,00369	109	1475	50,03388
I-IndVarAriNeg	0,99263	0,99263	0,00000	77	1552	47,42547
I-IndVarRepLoc	0,99263	0,99386	0,00123	110	1662	43,69919
I-DirVarRepPar	0,99386	0,99386	0,00000	57	1719	41,76829
I-DirVarRepExt	0,99386	0,99467	0,00081	168	1887	36,07724
I-IndVarRepExt	0,99467	1,00000	0,00533	254	2141	27,47290
I-CovAllNod	1,00000	1,00000	0,00000	129	2270	23,10298
...	...	...	...	...	...	...
II-ArgStcAli	1,00000	1,00000	0,00000	1	2952	0,00000

Esta tabela foi gerada 11 vezes para se obter a média referente a um único programa. O resultado final para os 5 programas é apresentado na Tabela 12.

Observe que o incremento proporcionado com a seleção dos casos de teste de cada operador varia. Incremento zero indica que o operador em questão não está agregando nenhum caso de teste novo ao conjunto *T*, ou seja, o operador em questão está sendo incluído pelos operadores aplicados antes dele. Desse modo, ordenar a Tabela 12 pelo incremento faz com que somente operadores que realmente contribuam para a melhoria no escore de mutação sejam considerados. Sempre que os operadores de mutação são reordenados, essa avaliação deve ser repetida. O processo termina quando todos os operadores de mutação estiverem ordenados de forma decrescente a partir de seu incremento.

3.3.1.1 – Coleta e Análise de Dados

Pela Tabela 12 observa-se que, em média, 30 operadores de mutação deveriam ser utilizados para se obter um conjunto *IM*-adequado; entretanto, desses 30 operadores, vários estão sendo incluídos pelos demais de modo que "eliminar" tais operadores não interfere na cobertura obtida (o conjunto de casos de teste selecionado pelos demais operadores continuaria sendo *IM*-

adequado) e possibilitaria uma maior redução no custo de aplicação do critério. Neste experimento, a eliminação do operador é feita através da ordenação dos mesmos a partir do incremento proporcionado.

Tabela 12 – Estratégia Incremental de Aplicação dos Operadores de Interface: Primeira Iteração

Op	EAn	EAt	Inc	CO	CC	RC
I-IndVarRepReq	0,00000	0,97239	0,97239	1391	1391	90,08129
I-IndVarIncDec	0,97239	0,97377	0,00138	684	2075	85,20394
I-IndVarBitNeg	0,97377	0,97564	0,00187	306	2381	83,02196
I-IndVarLogNeg	0,97564	0,97564	0,00000	306	2687	80,83999
I-IndVarRepCon	0,97564	0,97768	0,00204	2555	5242	62,62122
I-DirVarIncDec	0,97768	0,98762	0,00994	536	5778	58,79920
I-DirVarRepReq	0,98762	0,99111	0,00349	770	6548	53,30861
I-DirVarRepCon	0,99111	0,99176	0,00065	1380	7928	43,46834
I-IndVarRepPar	0,99176	0,99276	0,00100	672	8600	38,67655
I-IndVarAriNeg	0,99276	0,99276	0,00000	306	8906	36,49458
I-IndVarRepLoc	0,99276	0,99436	0,00160	618	9524	32,08785
I-DirVarRepPar	0,99436	0,99463	0,00027	385	9909	29,34256
I-DirVarRepExt	0,99463	0,99547	0,00084	494	10403	25,82002
I-IndVarRepExt	0,99547	0,99749	0,00202	511	10914	22,17627
I-CovAllNod	0,99749	0,99944	0,00195	438	11352	19,05305
I-DirVarLogNeg	0,99944	0,99944	0,00000	154	11506	17,95493
I-CovAllEdg	0,99944	0,99945	0,00001	390	11896	15,17399
I-DirVarBitNeg	0,99945	0,99945	0,00000	154	12050	14,07587
I-DirVarRepLoc	0,99945	0,99946	0,00001	488	12538	10,59612
II-ArgStcDif	0,99946	0,99946	0,00000	43	12581	10,28950
I-IndVarRepGlo	0,99946	0,99991	0,00045	312	12893	8,06475
I-DirVarAriNeg	0,99991	0,99991	0,00000	154	13047	6,96663
II-ArgBitNeg	0,99991	0,99991	0,00000	26	13073	6,78123
I-DirVarRepGlo	0,99991	0,99994	0,00003	170	13243	5,56902
II-ArgLogNeg	0,99994	0,99994	0,00000	26	13269	5,38363
II-ArgAriNeg	0,99994	0,99994	0,00000	26	13295	5,19823
I-RetStaRep	0,99994	0,99994	0,00000	42	13337	4,89875
II-ArgIncDec	0,99994	0,99994	0,00000	200	13537	3,47262
II-ArgRepReq	0,99994	0,99994	0,00000	55	13592	3,08043
II-FunCalDel	0,99994	1,00000	0,00006	254	13846	1,26925
...	...	...	...	...	...	...
II-ArgStcAli	1,00000	1,00000	0,00000	21	14024	0,00000

A Tabela 13 apresenta parte dos dados coletados na segunda iteração do processo. Observe que a partir do operador I-DirVarRepLoc, todos os operadores foram incluídos pelos demais (apresentam incremento zero). Além disso, o número de operadores a serem utilizados para obterem-se conjuntos de casos de teste *IM*-adequados baixou de 30 para 18 melhorando em torno de 10% a redução de custo proporcionada pela estratégia incremental. Se somente os 5 primeiros operadores de mutação forem considerados, a redução de custo em relação aos 33 operadores fica em torno de 58,90% e mesmo assim consegue-se conjuntos de casos de teste que proporcionam escore de mutação superior a 0,99 em relação ao critério Mutação de Interface.

Para o conjunto de programas em questão esse processo foi repetido mais três vezes e na Tabela 14 estão sintetizados os resultados obtidos na última iteração. Ao final do processo observa-se que todos os operadores de mutação apresentam-se ordenados de forma decrescente pelo incremento. Com apenas 4 operadores de mutação (I-IndVarRepReq, I-DirVarIncDec, I-IndVarRepLoc e I-DirVarRepReq) é possível gerar conjuntos de casos de teste que, em média, proporcionam um escore de mutação superior a 0,99 em relação ao critério Mutação de Interface com um número de mutantes 76% inferior ao gerado pelo conjunto total de operadores de mutação. Desejando-se obter conjuntos *IM*-adequados, devem ser utilizados, em média, 17 operadores de mutação, o que representa uma redução de custo de mais de 21% em relação ao conjunto total de operadores.

Tabela 13 – Estratégia Incremental de Aplicação dos Operadores de Interface: Segunda Iteração

Op	EAn	EAt	Inc	CO	CC	RC
I-IndVarRepReq	0,00000	0,97239	0,97239	1391	1391	90,08129
I-DirVarIncDec	0,97239	0,98268	0,01029	536	1927	86,25927
I-DirVarRepReq	0,98268	0,98730	0,00462	770	2697	80,76868
I-IndVarRepCon	0,98730	0,98982	0,00252	2555	5252	62,54991
I-IndVarRepExt	0,98982	0,99312	0,00330	511	5763	58,90616
I-CovAllNod	0,99312	0,99533	0,00221	438	6201	55,78294
I-IndVarBitNeg	0,99533	0,99649	0,00116	306	6507	53,60097
I-IndVarRepLoc	0,99649	0,99803	0,00154	618	7125	49,19424
I-IndVarIncDec	0,99803	0,99810	0,00007	684	7809	44,31689
I-IndVarRepPar	0,99810	0,99877	0,00067	672	8481	39,52510
I-DirVarRepExt	0,99877	0,99919	0,00042	494	8975	36,00257
I-DirVarRepCon	0,99919	0,99920	0,00001	1380	10355	26,16229
I-IndVarRepGlo	0,99920	0,99966	0,00046	312	10667	23,93754
I-DirVarRepPar	0,99966	0,99989	0,00023	385	11052	21,19224
II-FunCalDel	0,99989	0,99996	0,00007	254	11306	19,38106
I-DirVarRepGlo	0,99996	0,99998	0,00002	170	11476	18,16885
I-CovAllEdg	0,99998	0,99999	0,00001	390	11866	15,38791
I-DirVarRepLoc	0,99999	1,00000	0,00001	488	12354	11,90816
...	...	...	...	...	...	...
II-ArgStcDif	1,00000	1,00000	0,00000	43	14024	0,00000

Observe que, entre os 17 primeiros operadores de mutação da Tabela 14, existe apenas 1 operador do Grupo-II. Isso é justificável pois, como mostra a Tabela 15, analisando-se a adequação de conjuntos de casos de teste Grupo-I-adequados em relação ao Grupo-II e ao critério Mutação de Interface, observa-se que, para o conjunto de programas utilizado neste experimento, os operadores do Grupo-I praticamente incluem os operadores do Grupo-II. Somente para o programa *Look*, conjuntos de casos de teste adequados ao Grupo-I não foram adequados aos operadores do Grupo-II.

Tabela 14 – Estratégia Incremental de Aplicação dos Operadores de Interface: Última Iteração

Op	EAn	EAt	Inc	CO	CC	RC
I-IndVarRepReq	0,00000	0,97239	0,97239	1391	1391	90,08129
I-DirVarIncDec	0,97239	0,98268	0,01029	536	1927	86,25927
I-IndVarRepLoc	0,98268	0,98936	0,00668	618	2545	81,85254
I-DirVarRepReq	0,98936	0,99304	0,00368	770	3315	76,36195
I-IndVarRepExt	0,99304	0,99572	0,00268	511	3826	72,71820
I-CovAllNod	0,99572	0,99768	0,00196	438	4264	69,59498
I-IndVarRepPar	0,99768	0,99848	0,00080	672	4936	64,80319
I-IndVarRepGlo	0,99848	0,99898	0,00050	312	5248	62,57844
I-DirVarRepExt	0,99898	0,99938	0,00040	494	5742	59,05590
I-DirVarRepPar	0,99938	0,99968	0,00030	385	6127	56,31061
I-IndVarRepCon	0,99968	0,99981	0,00013	2555	8682	38,09184
I-IndVarBitNeg	0,99981	0,99987	0,00006	306	8988	35,90987
II-FunCalDel	0,99987	0,99993	0,00006	254	9242	34,09869
I-DirVarRepGlo	0,99993	0,99996	0,00003	170	9412	32,88648
I-IndVarIncDec	0,99996	0,99998	0,00002	684	10096	28,00913
I-CovAllEdg	0,99998	0,99999	0,00001	390	10486	25,22818
I-DirVarRepLoc	0,99999	1,00000	0,00001	488	10974	21,74843
...	...	...	...	...	...	...
II-ArgStcDif	1,00000	1,00000	0,00000	43	14024	0,00000

Tabela 15 – Média Geral do Escore de Mutação e do Custo de Cada Grupo de Operadores de Mutação de Interface

Programas	Critérios	Grupo-I	Grupo-II	Mutação de Interface
Cal	**Grupo-I**	1,0000	1,0000	1,0000
	Grupo-II	0,7589	1,0000	0,7734
Checkeq	**Grupo-I**	1,0000	1,0000	1,0000
	Grupo-II	0,1695	1,0000	0,1740
Comm	**Grupo-I**	1,0000	1,0000	1,0000
	Grupo-II	0,9366	1,0000	0,9445
Look	**Grupo-I**	1,0000	0,9874	0,9997
	Grupo-II	0,4928	1,0000	0,5052
Uniq	**Grupo-I**	1,0000	1,0000	1,0000
	Grupo-II	0,5890	1,0000	0,6147
Média	**Grupo-I**	1,0000	0,9975	0,9999
	Grupo-II	0,5893	1,0000	0,6024

Entretanto, os operadores do Grupo-II são responsáveis por apenas 5,00% do total de mutantes gerados (Tabela 16) e, desse modo, iniciar os teste a partir desse operadores pode ser uma boa estratégia quando, inicialmente, deseja-se garantir que requisitos mínimos do teste de integração sejam satisfeitos: todas as chamadas de função existentes no programa foram exercitadas e os parâmetros sendo passados para as funções estão corretos.

Além disso, experimentos realizados por Delamaro [DEL97c] forneceram indícios de que os operadores do Grupo-II também apresentam uma boa eficácia em revelar a presença de erros. Desse modo, uma nova estratégia incremental de aplicação dos operadores de mutação de interface foi estabelecida considerando que os operadores do Grupo-II tivessem prioridade em relação aos operadores do Grupo-I. Dentro de cada grupo manteve-se a ordem estabelecida na

Tabela 10. O resultado obtido com a aplicação desta estratégia na primeira iteração do processo está sintetizado na Tabela 17.

Tabela 16 – Total de Mutantes Gerados pelos Grupos de Operadores de Mutação de Interface

Programas	Grupo-I	% Grupo-I	Grupo-II	% Grupo-II	Mutação de Interface
Cal	4.120	95,00	230	5,00	4.350
Checkeq	2.938	99,00	16	1,00	2.954
Comm	2.647	90,00	305	10,00	2.952
Look	1.743	96,00	82	4,00	1.825
Uniq	1.833	94,00	110	6,00	1.943
Total	13.281	95,00	743	5,00	14.024

Tabela 17 – Estratégia Incremental a partir dos Operadores do Grupo-II: Primeira Iteração

Op	EAn	EAt	Inc	CO	CC	RC
II-ArgStcDif	0,00000	0,31413	0,31413	43	43	99,69338
II-ArgBitNeg	0,31413	0,31482	0,00069	26	69	99,50799
II-ArgLogNeg	0,31482	0,31482	0,00000	26	95	99,32259
II-ArgAriNeg	0,31482	0,31822	0,00340	26	121	99,13719
II-ArgIncDec	0,31822	0,58449	0,26627	200	321	97,71107
II-ArgRepReq	0,58449	0,58572	0,00123	55	376	97,31888
II-FunCalDel	0,58572	0,60236	0,01664	254	630	95,50770
II-ArgDel	0,60236	0,60236	0,00000	92	722	94,85168
II-ArgStcAli	0,60236	0,60236	0,00000	21	743	94,70194
I-IndVarRepReq	0,60236	0,97528	0,37292	1391	2134	84,78323
I-IndVarIncDec	0,97528	0,97666	0,00138	684	2818	79,90588
I-IndVarBitNeg	0,97666	0,97850	0,00184	306	3124	77,72390
I-IndVarLogNeg	0,97850	0,97850	0,00000	306	3430	75,54193
I-IndVarRepCon	0,97850	0,98054	0,00204	2555	5985	57,32316
I-DirVarIncDec	0,98054	0,98960	0,00906	536	6521	53,50114
I-DirVarRepReq	0,98960	0,99276	0,00316	770	7291	48,01055
I-DirVarRepCon	0,99276	0,99339	0,00063	1380	8671	38,17028
I-IndVarRepPar	0,99339	0,99440	0,00101	672	9343	33,37849
I-IndVarAriNeg	0,99440	0,99440	0,00000	306	9649	31,19652
I-IndVarRepLoc	0,99440	0,99599	0,00159	618	10267	26,78979
I-DirVarRepPar	0,99599	0,99626	0,00027	385	10652	24,04450
I-DirVarRepExt	0,99626	0,99711	0,00085	494	11146	20,52196
I-IndVarRepExt	0,99711	0,99913	0,00202	511	11657	16,87821
I-CovAllNod	0,99913	0,99950	0,00037	438	12095	13,75499
I-DirVarLogNeg	0,99950	0,99950	0,00000	154	12249	12,65687
I-CovAllEdg	0,99950	0,99952	0,00002	390	12639	9,87593
I-DirVarBitNeg	0,99952	0,99952	0,00000	154	12793	8,77781
I-DirVarRepLoc	0,99952	0,99952	0,00000	488	13281	5,29806
I-IndVarRepGlo	0,99952	0,99998	0,00046	312	13593	3,07330
I-DirVarAriNeg	0,99998	0,99998	0,00000	154	13747	1,97519
I-DirVarRepGlo	0,99998	1,00000	0,00002	170	13917	0,76298
I-RetStaRep	1,00000	1,00000	0,00000	42	13959	0,46349
I-RetStaDel	1,00000	1,00000	0,00000	65	14024	0,00000

Observa-se que, a ordem de aplicação definida na Tabela 17 requer que 32 operadores de mutação sejam aplicados até que se obtenha um conjunto *IM*-adequado o que significa uma redução de custo de apenas 0,46%, entretanto, dentre esses 32 operadores de mutação, vários estão sendo incluídos empiricamente pelos demais. Desse modo, da mesma forma como

realizado anteriormente, os operadores são ordenados pelo incremento mantendo a prioridade dos operadores do Grupo-II que proporcionaram algum incremento. O processo termina quando todos os operadores de ambos os grupos estiverem ordenados de forma decrescente. Para se chegar ao resultado final foram necessárias mais quatro iterações e os resultados obtidos estão apresentados na Tabela 18.

Tabela 18 – Estratégia Incremental a partir dos Operadores do Grupo-II: Última Iteração

Op	EAn	EAt	Inc	CO	CC	RC
II-ArgStcDif	0,00000	0,31413	0,31413	43	43	99,69338
II-ArgIncDec	0,31413	0,58057	0,26644	200	243	98,26726
II-FunCalDel	0,58057	0,59945	0,01888	254	497	96,45608
II-ArgAriNeg	0,59945	0,60129	0,00184	26	523	96,27068
II-ArgRepReq	0,60129	0,60212	0,00083	55	578	95,87849
II-ArgBitNeg	0,60212	0,60236	0,00024	26	604	95,69310
I-IndVarRepReq	0,60236	0,97528	0,37292	1391	1995	85,77439
I-DirVarIncDec	0,97528	0,98469	0,00941	536	2531	81,95237
I-IndVarRepLoc	0,98469	0,99123	0,00654	618	3149	77,54564
I-DirVarRepReq	0,99123	0,99469	0,00346	770	3919	72,05505
I-IndVarRepExt	0,99469	0,99735	0,00266	511	4430	68,41129
I-IndVarRepPar	0,99735	0,99817	0,00082	672	5102	63,61951
I-IndVarRepGlo	0,99817	0,99867	0,00050	312	5414	61,39475
I-DirVarRepExt	0,99867	0,99907	0,00040	494	5908	57,87222
I-CovAllNod	0,99907	0,99944	0,00037	438	6346	54,74900
I-DirVarRepPar	0,99944	0,99974	0,00030	385	6731	52,00371
I-IndVarRepCon	0,99974	0,99987	0,00013	2555	9286	33,78494
I-IndVarBitNeg	0,99987	0,99993	0,00006	306	9592	31,60297
I-IndVarIncDec	0,99993	0,99996	0,00003	684	10276	26,72561
I-DirVarRepGlo	0,99996	0,99998	0,00002	170	10446	25,51341
I-CovAllEdg	0,99998	0,99999	0,00001	390	10836	22,73246
I-DirVarRepLoc	0,99999	1,00000	0,00001	488	11324	19,25271
...	...	...	...	...	...	...
I-RetStaRep	1,00000	1,00000	0,00000	42	14024	0,00000

A partir da Tabela 18, observa-se que a aplicação de seis operadores de mutação do Grupo-II determina um escore de mutação em relação ao critério Mutação de Interface de aproximadamente 0,602 com uma redução no custo superior a 95%.

No geral, para se obter um conjunto de casos de teste *IM*-adequado a partir dessa estratégia, é necessária a utilização de 22 operadores de mutação sendo que a redução de custo obtida é da ordem de 19,25%. Se comparada com a estratégia apresentada na Tabela 14, observa-se que são necessários 5 operadores a mais para obterem-se conjuntos *IM*-adequados a um custo aproximadamente 2,5% superior, entretanto, como dito anteriormente, só com a aplicação dos operadores do Grupo-II, os requisitos mínimos do teste de integração (garantia de que todas as chamadas de funções foram testadas e que todos os parâmetros sendo passados estão corretos) estariam sendo cobertos primeiro com um custo muito baixo.

Em geral, o que se observa é que devido à relação de inclusão existente entre os operadores de interface, a utilização de apenas um subconjunto desses operadores permite que sejam obtidos conjuntos de casos de teste *IM*-adequados. A questão é quais operadores selecionar de modo a obter a maior cobertura com um menor custo. Um trabalho relevante nesse sentido foi a proposição de um procedimento para a determinação de um conjunto essencial de operadores de mutação [BAR98a]. Como descrito na Seção 2.8.3, a aplicação do procedimento em nível de unidade, considerando os operadores de mutação implementados na ferramenta *Proteum* [DEL93b], resultou em um conjunto essencial de operadores, o qual determinou um alto grau de adequação em relação ao critério Análise de Mutantes (0,996) com uma redução de custo da ordem de 65%.

Com o objetivo de verificar se os mesmos resultados se confirmam para os operadores de mutação de interface, na próxima seção o procedimento **Essencial** definido por Barbosa [BAR98a] é descrito e aplicado no contexto do teste de integração, utilizando os operadores de mutação implementados na ferramenta *Proteum/IM* [DEL97c].

3.3.2 – Determinação do Conjunto Essencial de Operadores de Mutação de Interface

O procedimento **Essencial** [BAR98a] requer que, inicialmente, algumas informações estejam disponíveis. Da mesma forma que o experimento anterior (vide Tabela 10), é necessária a existência de uma tabela que contenha dados a respeito do escore de mutação que conjuntos de casos de teste adequados a cada operador determinam em relação ao demais. Na Tabela 19 são apresentadas essas informações. Uma informação adicional que pode ser extraída a partir de cada coluna dessa tabela diz respeito ao *strength* de cada operador de mutação, ou seja, qual a dificuldade dos demais operadores de mutação em incluir um operador específico.

Observe que o processo para gerar a Tabela 19 é o mesmo utilizado na geração da Tabela 10, ou seja, inicialmente essa tabela é gerada 11 vezes para cada programa (correspondendo aos 11 conjuntos de casos de teste *IM*-adequados); a média dessas 11 tabelas resulta na média de cobertura obtida para um determinado programa. Obtidos os resultados médios para cada programa é gerada uma tabela (Tabela 19) que sintetiza os resultados em relação a todos os programas.

As linhas da Tabela 19 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 0,982, indicando que o conjunto I-IndVarIncDec-adequado é capaz de distinguir, em média, 98,20% dos mutantes de I-IndVarBitNeg. O escore médio de um operador em relação ao total é obtido através da média geral de 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, ou seja, conjuntos I-IndVarIncDec-adequados distinguem, em média, 97% dos mutantes gerados pelos demais operadores de mutação.

Tabela 19 – Média dos Escores e *Strength* dos Operadores para o Conjunto Total de Programas do Experimento

Operador	1	2	3	4	...	31	32	33	Média Geral[16]	Desvio Padrão
I-IndVarRepReq	1,000	0,998	0,989	1,000	...	0,959	1,000	1,000	0,973	0,030
I-IndVarIncDec	0,993	1,000	0,982	0,991	...	0,959	1,000	1,000	0,970	0,030
I-IndVarBitNeg	0,986	0,990	1,000	0,969	...	0,945	1,000	1,000	0,968	0,029
I-IndVarLogNeg	0,990	0,993	0,971	1,000	...	0,957	1,000	1,000	0,964	0,033
...	...	...	...	...	...	...	...	...	...	...
I-RetStaDel	0,573	0,582	0,580	0,568	...	1,000	0,783	0,676	0,667	0,131
II-ArgDel	0,498	0,513	0,515	0,486	...	0,625	1,000	1,000	0,657	0,212
II-ArgStcAli	0,265	0,287	0,274	0,231	...	0,361	0,431	1,000	0,393	0,202
Média Geral	0,804	0,819	0,806	0,784	...	0,862	0,941	0,903	-	-
Desvio Padrão	0,179	0,172	0,173	0,188	...	0,126	0,109	0,129	-	-

Conforme mencionado anteriormente, analisando-se uma determinada coluna 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-IndVarLogNeg (quarta coluna) tem-se que o *strength* deste operador em relação a I-IndVarRepReq (primeira linha) é zero (1 - 1.000), indicando que o conjunto I-IndVarRepReq-adequado é também I-IndVarLogNeg-adequado; em relação ao operador II-ArgDel (linha), o *strength* de I-IndVarLogNeg (quarta coluna) é de 0,514 (1 - 0,486), ou seja, o conjunto II-ArgDel-adequado não é capaz de distinguir 51,4% dos mutantes gerados por I-IndVarLogNeg. Da mesma forma que o escore, tem-se que o *strength* 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 - 0,784).

Os operadores da Tabela 19 foram ordenados segundo a média geral de escore e *strength*, dando origem às tabelas 20(a) e 20(b), respectivamente. Além do escore de mutação e do

strength, outra informação requerida para a aplicação do procedimento **Essencial** [BAR98a] é o custo associado a cada operador; essa informação também foi coletada e é apresentada Tabela 20(c).

Tabela 20 – 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-IndVarAriNeg	0,941
I-IndVarRepLoc	0,932
I-DirVarRepPar	0,918
I-DirVarRepExt	0,913
I-IndVarRepExt	0,907
I-CovAllNod	0,897
I-DirVarLogNeg	0,896
I-CovAllEdg	0,894
I-DirVarBitNeg	0,890
I-DirVarRepLoc	0,882
II-ArgStcDif	0,857
I-IndVarRepGlo	0,846
I-DirVarAriNeg	0,838
II-ArgBitNeg	0,838
I-DirVarRepGlo	0,838
II-ArgLogNeg	0,837
II-ArgAriNeg	0,833
I-RetStaRep	0,734
II-ArgIncDec	0,717
II-ArgRepReq	0,711
II-FunCalDel	0,676
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
I-IndVarRepExt	0,189
I-DirVarBitNeg	0,188
I-DirVarRepReq	0,186
I-IndVarIncDec	0,181
I-DirVarRepLoc	0,176
I-IndVarRepGlo	0,170
II-FunCalDel	0,159
I-DirVarRepExt	0,154
I-DirVarIncDec	0,151
I-DirVarAriNeg	0,150
I-DirVarRepGlo	0,145
I-RetStaRep	0,141
I-RetStaDel	0,138
I-DirVarLogNeg	0,137
I-CovAllEdg	0,129
I-CovAllNod	0,102
II-ArgIncDec	0,099
II-ArgStcAli	0,097
II-ArgAriNeg	0,075
II-ArgDel	0,059
II-ArgBitNeg	0,044
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-DirVarAriNeg	154
I-DirVarBitNeg	154
I-DirVarLogNeg	154
I-DirVarRepGlo	170
II-ArgIncDec	200
II-FunCalDel	254
I-IndVarAriNeg	306
I-IndVarBitNeg	306
I-IndVarLogNeg	306
I-IndVarRepGlo	312
I-DirVarRepPar	385
I-CovAllEdg	390
I-CovAllNod	438
I-DirVarRepLoc	488
I-DirVarRepExt	494
I-IndVarRepExt	511
I-DirVarIncDec	536
I-IndVarRepLoc	618
I-IndVarRepPar	672
I-IndVarIncDec	684
I-DirVarRepReq	770
I-DirVarRepCon	1380
I-IndVarRepReq	1391
I-IndVarRepCon	2555

Obtidas as informações acima pode-se iniciar a aplicação do procedimento definido por Barbosa [BAR98a]. O procedimento **Essencial** exige ainda que, a cada passo, algumas variáveis, dependentes do conjunto de programas utilizados, sejam instanciadas. Além disso, o procedimento requer a divisão dos operadores em determinadas classes. Em uma primeira análise, descrita na próxima seção, considerou-se a divisão original dos operadores de mutação da *Proteum/IM*, ou seja, Grupo-I e Grupo-II. Na Seção 3.3.2.2 é feita uma nova divisão dos

[16] 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.

operadores de mutação de interface e o procedimento é aplicado novamente e os resultados obtidos são comparados.

3.3.2.1 – Operadores Essenciais de Interface: Grupo-I e Grupo-II

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*. Mais detalhes sobre o procedimento **Essencial** podem ser encontrados em [BAR98a, BAR98b].

Passo 1: Selecionar operadores que determinem alto escore de mutação

A fim de garantir que o conjunto essencial de operadores determine um alto escore de mutação em relação ao critério Mutação de Interface, devem ser selecionados, em princípio, os operadores *op* tal que conjuntos de casos de teste *op*-adequados sejam capazes de distinguir a maioria dos mutantes gerados pelo conjunto total de operadores [BAR98a].

Esse passo consiste em selecionar os operadores de mutação que proporcionam os melhores escores de mutação médios. Tal informação pode ser obtida a partir da Tabela 20(a).

Como dito anteriormente, a seleção desses operadores 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, utilizou-se *IMES* = 0,960. Considerando esse índice, observa-se que os cinco primeiros operadores da Tabela 20(a) são selecionados para compor CE_{pre} (Conjunto Essencial Preliminar). Ao final deste passo tem-se CE_{pre} = {I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon}.

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

Em nível de unidade os operadores de mutação estão divididos em quatro classes – comandos, operadores, variáveis e constantes – que modelam tipos de erros de unidade em certos elementos do programa. Em nível de integração os operadores de mutação estão divididos em dois grupos (classes) – Grupo-I e Grupo-II – sendo que cada uma delas modela erros típicos de integração entre duas unidades que compõem uma conexão. A divisão dos operadores em classes facilita ao testador escolher quais operadores de mutação utilizar quando se deseja identificar tipos de erros específicos nos programas sendo testados. Assim sendo, uma boa divisão dos

operadores de mutação permite identificar melhor quais classes são mais relevantes em cada domínio de programas.

Como os operadores de mutação de interface estão divididos em apenas duas classes, isso pode dificultar a seleção dos operadores que melhor representem tipos específicos de erros de integração e, desse modo, o resultado obtido com a aplicação do procedimento pode não ser satisfatório. Assim sendo, na Seção 3.3.2.2, uma nova divisão dos operadores de mutação de interface é sugerida e o procedimento **Essencial** é reaplicado para comparar os resultados obtidos.

Uma das diretrizes consideradas por Barbosa para se obter um bom conjunto essencial é procurar garantir que este contenha ao menos um operador de cada classe de mutação, pois assim, tal conjunto estaria considerando uma maior diversidade de tipos de erros e, conseqüentemente, poderia ser aplicável a uma maior variedade de programas. Assim sendo, 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[17] por um subconjunto de operadores, um operador op pertencente a 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. Para o experimento em questão considerou-se $ms = 0{,}99$.

A seleção do operador de mutação de determinada classe é feita respeitando a ordem estabelecida na Tabela 20(a). Por exemplo, observe que o conjunto CE_{pre}, obtido com a aplicação do Passo 1, não apresenta operador do Grupo-II. De acordo com a Tabela 20(a), o primeiro operador desta classe é II-ArgStcDif, entretanto, o conjunto de casos de teste selecionado pelos operadores de CE_{pre} é capaz de distinguir 100% dos mutantes de II-ArgStcDif de modo que esse operador não é selecionado para fazer parte de CE_{pre}, pois está sendo incluído empiricamente pelos operadores já selecionados. O próximo candidato do Grupo-II é o operador II-ArgBitNeg que também é incluído empiricamente pelos operadores de CE_{pre}. Na Tabela 21 é apresentado o escore de mutação que o conjunto de casos de teste CE_{pre}-adequado determina em relação aos quatro primeiros operadores do Grupo-II.

[17] Como define Barbosa [BAR98a], 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. O símbolo $\Rightarrow^*$ denota a relação de inclusão empírica.

Tabela 21 – Escore de Mutação Determinado por CE_{pre} em Relação aos Operadores do Grupo-II

Operador	Escore
II-ArgStcDif	1,000
II-ArgBitNeg	0,991
II-ArgLogNeg	1,000
II-ArgAriNeg	0,973

Observe que II-ArgAriNeg é o primeiro operador que não é incluído empiricamente pelos operadores de CE_{pre}, ou seja, o escore de mutação de T CE_{pre}-adequado em relação a II-ArgAriNeg (0,973) é inferior a ms, e, desse modo, tal operador passa a fazer parte de CE_{pre}.

Como o conjunto CE_{pre} já contém ao menos um representante de cada classe de mutação pode-se passar para o próximo passo. Se todos os operadores do Grupo-II fossem incluídos empiricamente pelos operadores de CE_{pre} nenhum seria selecionado e o passo terminaria. Ao final deste passo CE_{pre} = {I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon, II-ArgAriNeg}.

Passo 3: Avaliar inclusão empírica entre operadores

Deve-se remover os operadores que são incluídos empiricamente por outros operadores do conjunto essencial, visto que tais operadores elevam o custo de aplicação do conjunto, em termos do número de mutantes gerados, e não contribuem efetivamente para a melhoria da atividade de teste [BAR98a].

No Passo 1 foram selecionados todos os operadores com escore de mutação igual ou superior a *IMES* mas não foi feita nenhuma análise com respeito à relação de inclusão entre esses operadores de mutação. 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, as seguintes relações de inclusão empírica foram observadas:

$$\{\text{I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon, II-ArgAriNeg}\} \overset{0,9995}{\Rightarrow}{}^{*} \{\text{I-IndVarRepReq}\}$$

$$\{\text{I-IndVarRepReq, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon, II-ArgAriNeg}\} \overset{0,9996}{\Rightarrow}{}^{*} \{\text{I-IndVarIncDec}\}$$

$$\{\text{I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarRepCon, II-ArgAriNeg}\} \overset{1,0000}{\Rightarrow}{}^{*} \{\text{I-IndVarLogNeg}\}$$

$$\{\text{I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, II-ArgAriNeg}\} \overset{0,9951}{\Rightarrow}{}^{*} \{\text{I-IndVarRepCon}\}$$

De acordo com o procedimento, deve-se eliminar um operador de cada vez, sempre o mais incluído. Toda vez que um operador é eliminado as relações de inclusão devem ser

analisadas novamente. Para as relações de inclusão identificadas acima, o operador eliminado é I-IndVarLogNeg. Após a sua eliminação, permaneceram as seguintes relações de inclusão:

$$\{\text{I-IndVarIncDec, I-IndVarBitNeg, I-IndVarRepCon, II-ArgAriNeg}\} \overset{0,9995}{\Rightarrow}^{*} \{\text{I-IndVarRepReq}\}$$

$$\{\text{I-IndVarRepReq, I-IndVarBitNeg, I-IndVarRepCon, II-ArgAriNeg}\} \overset{0,9996}{\Rightarrow}^{*} \{\text{I-IndVarIncDec}\}$$

$$\{\text{I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, II-ArgAriNeg}\} \overset{0,9951}{\Rightarrow}^{*} \{\text{I-IndVarRepCon}\}$$

Novamente, elimina-se o operador mais incluído I-IndVarIncDec e avalia-se novamente se existem relações de inclusão. Esse processo é repetido até que não exista operador em CE_{pre} incluído pelos demais.

Para este experimento, este passo foi encerrado quando CE_{pre} continha os seguintes operadores: {I-IndVarRepReq, I-IndVarBitNeg, II-ArgAriNeg}

Passo 4: Estabelecer uma estratégia incremental de aplicação

Dado o custo de aplicação e os requisitos de teste específicos que cada classe 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 [BAR98a]. No teste de integração, considera-se como requisito mínimo de teste a garantia de que ao menos todas as chamadas de função existentes no programa sejam necessárias e tenham sido exercitadas ao menos uma vez; em seguida, procura-se garantir que os parâmetros formais estejam corretos; os comandos de retorno e os valores sendo retornados estejam corretos e todas as variáveis envolvidas em uma dada conexão estejam sendo utilizadas de modo adequado. Desse modo, considerando as duas classes de operadores, formadas pelos grupos I e II, a estratégia de aplicação escolhida foi a seguinte:

- operadores do Grupo-II, que geram poucos mutantes e garantem ao menos que cada chamada de função existente no programa bem como os parâmetros formais sendo passados estejam corretos; e
- operadores do Grupo-I, que tendem a gerar um número maior de mutantes e exercitam de forma mais completa a conexão entre duas unidades.

Dentro de cada grupo, os operadores são ordenados pelo custo (número de mutantes gerados – Tabela 20(c)).

Nos próximos passos, todos os operadores que venham a ser adicionados a CE_{pre} devem respeitar a ordenação estabelecida.

Após a aplicação deste passo tem-se CE_{pre} = {II-ArgAriNeg, I-IndVarBitNeg, I-IndVarRepReq}.

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

Sabe-se que, em geral, independentemente da qualidade dos conjuntos de casos de teste utilizados, 80% dos mutantes gerados morrem na primeira execução [BUD80a]. Considerando então que somente em torno de 20% dos mutantes gerados contribuem efetivamente para a melhoria do conjunto de casos de teste, um incremento de 1% no escore de mutação representa 5% do número de mutantes significativos. Além disso, são esses mutantes que contribuem efetivamente para a eficácia do critério.

Desse modo, deve-se investigar entre os operadores não selecionados se existe um ou mais operadores que, se adicionados ao conjunto essencial, proporcionem incremento no escore de mutação [BAR98a]. Para isso, o conjunto de casos de teste adequado a determinado operador de mutação é acrescentado ao conjunto de casos de teste CE_{pre}-adequado verificando-se qual o escore de mutação obtido em relação ao critério Mutação de Interface. Na Tabela 22 é apresentada tal informação. Por exemplo, o operador I-DirVarRepCon, se adicionado a CE_{pre}, elevará o escore de mutação de CE_{pre} em relação ao critério Mutação de Interface em 0,01332.

Tabela 22 – Incremento no Escore Proporcionado pelos Operadores: Primeira Iteração

Operador	Índice de Incremento	Escore CE_{pre} + Incremento
I-DirVarRepCon	0,01332	0,98877
I-DirVarRepReq	0,01275	0,98839
I-DirVarRepPar	0,01076	0,98609
I-DirVarRepGlo	0,01069	0,98518
I-DirVarIncDec	0,00998	0,98573
I-DirVarBitNeg	0,00989	0,98553
I-CovAllNod	0,00982	0,98517
I-CovAllEdg	0,00827	0,98361
I-DirVarRepExt	0,00824	0,98338
I-IndVarRepExt	0,00610	0,97036
I-IndVarRepLoc	0,00547	0,98161
I-RetStaDel	0,00514	0,98077
I-DirVarRepLoc	0,00513	0,98063
I-DirVarLogNeg	0,00459	0,98023
I-IndVarRepPar	0,00427	0,97990
I-IndVarRepGlo	0,00352	0,97916
I-IndVarRepCon	0,00306	0,96726
II-FunCalDel	0,00163	0,97727
I-IndVarIncDec	0,00142	0,96540

De acordo com a autora, 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. No caso deste experimento procurou-se adicionar mais 4 operadores de cada classe (x = 4) desde que tais operadores proporcionassem um incremento no escore de mutação superior a 0,001 (*IIM* = 0,001).

Os operadores devem ser adicionados respeitando-se a ordem estabelecida no Passo 4. Para este experimento significa que inicialmente deve-se procurar adicionar um operador do Grupo-II seguido de um operador do Grupo-I até que x operadores de cada grupo tenham sido adicionados ou que não existam operadores que proporcionem incremento igual ou superior a *IIM*.

Os operadores só devem ser adicionados ao conjunto essencial preliminar se estes não forem incluídos empiricamente pelos operadores que já fazem parte de CE_{pre}. Caso existam dois ou mais operadores de determinado grupo que proporcionem incremento dentro da mesma faixa, deve-se adicionar a CE_{pre} o que apresentar o maior $strength$ em relação a CE_{pre}.

Considerando a Tabela 22 e a ordenação proposta no Passo 4, tem-se que o único operador do Grupo-II com índice de incremento superior a *IIM* é II-FunCalDel. Como tal operador não é incluído empiricamente por CE_{pre}, este é adicionado a CE_{pre}. Toda vez que um novo operador é adicionado, deve-se obter novamente as informações apresentadas na Tabela 22 (vide Tabela 23).

Inserido um operador do Grupo-II procura-se inserir um operador do Grupo-I. A partir da Tabela 23, tem-se que os quatro primeiros operadores (I-DirVarRepCon, I-DirVarRepReq, I-DirVarRepPar, I-DirVarRepGlo) apresentam incremento dentro da mesma faixa (0,01). Na Tabela 24 é apresentado o escore de mutação determinado pelo conjunto de casos de teste CE_{pre}-adequado em relação a tais operadores.

Observe que o operador I-IndVarRepPar é o que apresenta maior $strength$ (menor escore) em relação aos operadores de CE_{pre} e, desse modo, tal operador é adicionado CE_{pre}.

Esse processo é repetido até que o requisito de parada seja satisfeito, ou seja, que mais três operadores de cada classe sejam adicionados a CE_{pre} ou que não existam mais operadores que proporcionem um incremento superior a *IIM*.

No caso, esse processo foi repetido mais três vezes e só operadores do Grupo-I foram adicionados visto que nenhum operador do Grupo-II apresentou incremento superior a *IIM*. Ao final deste passo, CE_{pre} = {II-ArgAriNeg, II-FunCalDel, I-IndVarBitNeg, I-DirVarRepPar, I-IndVarRepExt, I-IndVarRepLoc, I-DirVarRepCon, I-IndVarRepReq}.

Tabela 23 – Incremento no Escore Proporcionado pelos Operadores: Segunda Iteração

Operador	Índice de Incremento	Escore CE_{pre} + Incremento
I-DirVarRepCon	0,01332	0,99040
I-DirVarRepReq	0,01275	0,99002
I-DirVarRepPar	0,01076	0,98772
I-DirVarRepGlo	0,01069	0,98681
I-DirVarIncDec	0,00998	0,98736
I-DirVarBitNeg	0,00989	0,98716
I-CovAllEdg	0,00827	0,98524
I-CovAllNod	0,00825	0,98523
I-DirVarRepExt	0,00824	0,98501
I-IndVarRepExt	0,00610	0,97199
I-IndVarRepLoc	0,00547	0,98324
I-RetStaDel	0,00514	0,98241
I-DirVarRepLoc	0,00513	0,98226
I-DirVarLogNeg	0,00459	0,98186
I-IndVarRepPar	0,00427	0,98153
I-IndVarRepGlo	0,00352	0,98079
I-IndVarRepCon	0,00306	0,96889
I-IndVarIncDec	0,00142	0,96703

Tabela 24 – Escore de Mutação Proporcionado por CE_{pre} em Relação aos Operadores de Maior Incremento

Operador	Escore de Mutação
I-DirVarRepCon	0,940
I-DirVarRepReq	0,947
I-DirVarRepPar	0,934
I-DirVarRepGlo	0,951

Passo 6: Selecionar operadores de alto *strength*

Outros operadores a serem considerados durante a determinação do conjunto essencial são os que apresentam alto *strength* em relação ao conjunto total de operadores. A principal característica dos operadores de alto *strength* é que, em média, eles são pouco incluídos pelos demais, fator que pode ser importante no que diz respeito à eficácia do conjunto essencial [BAR98a].

A seleção de tais operadores 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*). Existindo operadores de alto *strength* estes 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, ou seja, considera-se que um determinado operador de mutação op apresenta um alto $strength$ se conjuntos de casos de teste adequados aos demais operadores de mutação deixam de distinguir, em média, 70% dos mutantes gerados por op. Observando-se a Tabela 20(b), que apresenta o $strength$ de todos os operadores de mutação de interface, percebe-se que nenhum deles apresenta um $strength$ igual ou superior a IMS e, desse modo, nenhum operador de mutação foi adicionado a CE_{pre} neste passo. Ao final deste passo, CE_{pre} = {II-ArgAriNeg, II-FunCalDel, I-IndVarBitNeg, I-DirVarRepPar, I-IndVarRepExt, I-IndVarRepLoc, I-DirVarRepCon, I-IndVarRepReq}.

Aplicado o procedimento tem-se que o conjunto essencial de operadores de mutação de interface obtido é: CE_{IM1} = {II-ArgAriNeg, II-FunCalDel, I-IndVarBitNeg, I-DirVarRepPar, I-IndVarRepExt, I-IndVarRepLoc, I-DirVarRepCon, I-IndVarRepReq}. A Tabela 25 e Gráfico 2 apresentam o escore de mutação e a redução de custo proporcionada pelo conjunto essencial de operadores de mutação no decorrer do procedimento.

Tabela 25 – Resultados Obtidos a cada Passo do Procedimento: CE_{IM1}

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, II-ArgAriNeg}	0,97800	62,436
3	{I-IndVarRepReq, I-IndVarBitNeg, II-ArgAriNeg}	0,97564	87,714
4	{II-ArgAriNeg, I-IndVarBitNeg, I-IndVarRepReq}	0,97564	87,714
5	{II-ArgAriNeg, II-FunCalDel, I-IndVarBitNeg, I-DirVarRepPar, I-IndVarRepExt, I-IndVarRepLoc, I-DirVarRepCon, I-IndVarRepReq}	0,99795	65,267
6	{II-ArgAriNeg, II-FunCalDel, I-IndVarBitNeg, I-DirVarRepPar, I-IndVarRepExt, I-IndVarRepLoc, I-DirVarRepCon, I-IndVarRepReq}	0,99795	65,267

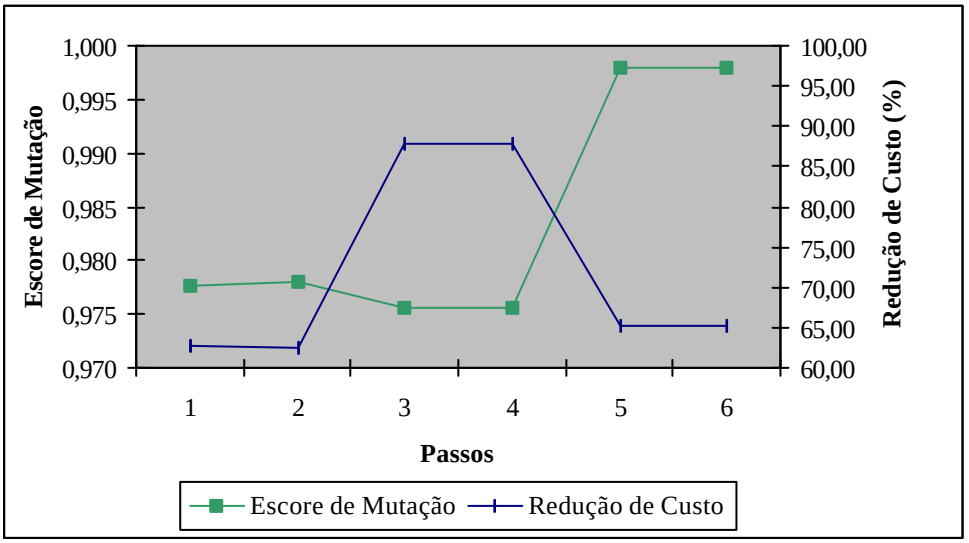


Gráfico 2 – Evolução do Conjunto Essencial Durante os Passos do Procedimento: CE_{IM1}

Como pode-se observar, no Passo 2, a inserção do operador de mutação II-ArgAriNeg proporcionou um pequeno acréscimo no escore de mutação e no custo de aplicação do conjunto essencial. No Passo 3, eliminados os operadores do conjunto essencial que eram incluídos empiricamente pelos demais, observa-se que ocorreu um significativo aumento na redução de custo do conjunto com um pequeno decréscimo no escore de mutação. No Passo 4, como os operadores de mutação só foram ordenados (nenhum operador foi removido ou inserido no conjunto essencial), não ocorreu mudanças no custo e escore de mutação determinado pelo conjunto essencial. Com a inserção de mais 5 operadores de mutação no Passo 5, o escore de mutação obtido pelo conjunto essencial passou a ser superior a 0,997 e mesmo assim, a redução de custo proporcionada com a aplicação desse subconjunto de operadores de mutação é da ordem de 65%. Como não foi encontrado nenhum operador de mutação com alto *strength*, no Passo 6, o escore de mutação e a redução de custo permaneceram os mesmos.

Considerando o conjunto CE_{IM1} obtido, tem-se que o escore de mutação que conjuntos de casos de teste CE_{IM1}-adequados determinam, em média, um escore de mutação de 0,998 em relação ao critério Mutação de Interface e proporcionam uma redução no custo de aplicação do critério de aproximadamente 65%. Se comparado com a estratégia incremental definida na Tabela 14 da Seção 3.3.1.1, observa-se que a aplicação dos sete primeiros operadores da estratégia incremental (I-IndVarRepReq, I-DirVarIncDec, I-IndVarRepLoc, I-DirVarRepReq, I-IndVarRepExt, I-CovAllNod, I-IndVarRepPar) obtém um escore de mutação de 0,998 em relação ao critério Mutação de Interface e permitem uma redução de custo de aproximadamente 64,80%, praticamente a mesma proporcionada pelos conjunto de operadores essenciais. Observa-se ainda que, três operadores (I-IndVarRepReq, I-IndVarRepLoc e I-IndVarRepExt) estão presentes em ambos os conjuntos. Assim sendo, seria necessário um estudo sobre a eficácia de tais conjuntos para se decidir qual o melhor a ser utilizado de modo a proporcionar a maior cobertura e máxima detecção da presença de erros.

Os resultados obtidos com a aplicação do procedimento **Essencial** [BAR98a] no contexto dos operadores de mutação de interface demonstraram que, também para outros conjuntos de operadores de mutação (não somente operadores de mutação de unidade), o procedimento resultou em um conjunto de operadores que determina uma alta cobertura em relação ao critério Mutação de Interface e ainda assim, proporciona uma significativa redução de custo em termos do número de mutantes gerados.

Buscando a divisão dos operadores de mutação de interface em classes mais significativas, a seguir, os operadores dos grupos I e II são subdivididos em três e duas classes, respectivamente, e o procedimento para se determinar o conjunto essencial de operadores de mutação é reaplicado considerando essas cinco classes de operadores.

3.3.2.2 – Operadores Essenciais de Interface: Grupo-I (Comandos, Operadores e Variáveis) e Grupo-II (Comandos e Argumentos)

Os operadores de mutação para a linguagem C, em nível de unidade, estão divididos em quatro classes: comandos, operadores, variáveis e constantes (Apêndice A). Procurando-se estabelecer uma maior relação entre os operadores de mutação de interface e as classes de operadores de mutação de unidade, bem como procurando melhorar a divisão dos operadores de mutação de interface de modo a refletir de forma mais específica os tipos de erros sendo modelados, os operadores dos grupos I e II foram subdivididos em cinco classes: G-I-comandos (Quadro 1), G-I-operadores (Quadro 2), G-I-variáveis (Quadro 3), G-II-comandos (Quadro 4) e G-II-argumentos (Quadro 5). A divisão foi baseada na entidade na qual o operador de mutação é aplicado. Por exemplo, os operadores do Grupo-I que manipulam operadores foram agrupados na classe G-I-operadores e os operadores que alteram os valores das variáveis foram agrupados na classe G-I-variáveis. Na descrição dos operadores de mutação de interface aparecem alguns conjuntos que são definidos a seguir:

- *P* – conjunto de parâmetros formais utilizados na chamada da função;
- *G* – conjunto de variáveis globais utilizadas na função chamada;
- *L* – conjunto de variáveis declaradas na função chamada (variáveis locais);
- *E* – conjunto de variáveis globais não utilizadas na função chamada;
- *C* – conjunto de constantes utilizadas na função chamada; e
- *R* – conjunto de constantes requeridas.

Quadro 1 – Operadores do Grupo I: Comandos (G-I-comandos)

Operador	Descrição
I-CovAllEdg	Garante cobertura de desvios.
I-CovAllNod	Garante cobertura de nós.
I-RetStaDel	Elimina comando *return*.
I-RetStaRep	Troca comando *return*.

Quadro 2 – Operadores do Grupo I: Operadores (G-I-operadores)

Operador	Descrição
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.

Quadro 3 – Operadores do Grupo I: Variáveis (G-I-variáveis)

Operador	Descrição
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.

Quadro 4 – Operadores do Grupo II: Comandos (G-II-comandos)

Operador	Descrição
II-FunCalDel	Elimina chamada de função.

Quadro 5 – Operadores do Grupo II – Argumentos (G-II-argumentos)

Operador	Descrição
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.

Com base nessas cinco classes o procedimento para a determinação do conjunto essencial de operadores de mutação é aplicado novamente. Observe que os dados coletados anteriormente a respeito da cobertura que cada operador determina em relação aos demais (Tabela 20(a)), custo (Tabela 20(c)) e *strength* (Tabela 20(b)) de cada operador são os mesmos. Os resultados obtidos a cada passo são apresentados a seguir.

Passo 1: Selecionar operadores que determinem alto escore de mutação

O resultado obtido neste passo é o mesmo obtido com o experimento anterior, ou seja, considerando-se *IMES* = 0,960, os cinco primeiros operadores da Tabela 20(a) são selecionados

para compor CE_{pre}. Ao final deste passo tem-se, CE_{pre} = {I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon}.

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

Visto que agora, ao invés de duas, existem cinco classes de operadores de mutação, procura-se adicionar a CE_{pre} um operador de cada classe de mutação que ainda não possua representantes em CE_{pre}. No caso, 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 20(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}.

Em seguida, procura-se pelo próximo operador que não pertença às classes G-I-operadores, G-I-variáveis e G-I-comandos. O primeiro operador encontrado é II-ArgStcDif da classe G-II-argumentos; entretanto, tal operador é incluído empiricamente pelos operadores de CE_{pre}. Quando isso ocorre, procura-se selecionar o próximo operador desta classe até que se encontre um que não seja incluído por CE_{pre} ou até que todos os operadores da classe sejam avaliados e considerados incluídos empiricamente. No caso deste experimento, para a classe G-II-argumentos, mais três operadores foram avaliados (II-ArgBitNeg, II-ArgLogNeg, II-ArgAriNeg). Destes, o único não incluído foi II-ArgAriNeg, passando a integrar CE_{pre}.

Da classe G-II-comandos, o único operador existente (II-FunCalDel) foi incluído empiricamente pelos operadores de CE_{pre} de modo que o passo deve ser encerrado sem que CE_{pre} possua representantes dessa classe de operadores de mutação.

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

Considerando o conjunto CE_{pre} do passo anterior e definindo ms = 0,99, as seguintes relações de inclusão empírica foram observadas:

$$\{\text{I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon, I-CovAllNod, II-ArgAriNeg}\} \overset{0,9995}{\Longrightarrow}^{*} \{\text{I-IndVarRepReq}\}$$

$$\{\text{I-IndVarRepReq, I-IndVarBitNeg, I-IndVarLogNeg, I-IndVarRepCon, I-CovAllNod, II-ArgAriNeg}\} \overset{0,9996}{\Longrightarrow}^{*} \{\text{I-IndVarIncDec}\}$$

$$\{\text{I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarRepCon, I-CovAllNod, II-ArgAriNeg}\} \overset{1,0000}{\Longrightarrow}^{*} \{\text{I-IndVarLogNeg}\}$$

$$\{\text{I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-IndVarLogNeg, I-CovAllNod, II-ArgAriNeg}\} \overset{0,9962}{\Longrightarrow}^{*} \{\text{I-IndVarRepCon}\}$$

Para as relações de inclusão identificadas acima, o operador eliminado é I-IndVarLogNeg. Após a sua eliminação, permaneceram as seguintes relações de inclusão:

$$\{\text{I-IndVarIncDec, I-IndVarBitNeg, I-IndVarRepCon, I-CovAllNod, II-ArgAriNeg}\} \overset{0,9995}{\Longrightarrow}^{*} \{\text{I-IndVarRepReq}\}$$

$$\{\text{I-IndVarRepReq, I-IndVarBitNeg, I-IndVarRepCon, I-CovAllNod, II-ArgAriNeg}\} \overset{0,9996}{\Longrightarrow}^{*} \{\text{I-IndVarIncDec}\}$$

$$\{\text{I-IndVarRepReq, I-IndVarIncDec, I-IndVarBitNeg, I-CovAllNod, II-ArgAriNeg}\} \overset{0,9962}{\Longrightarrow}^{*} \{\text{I-IndVarRepCon}\}$$

Novamente, elimina-se o operador mais incluído I-IndVarIncDec e avalia-se novamente se existem relações de inclusão. Esse processo foi repetido até que não houvesse operador em CE_{pre} incluído pelos demais.

Para este experimento, este passo foi encerrado quando CE_{pre} continha os seguintes operadores: {I-IndVarRepReq, I-IndVarBitNeg, I-CovAllNod, II-ArgAriNeg}.

Passo 4: Estabelecer uma estratégia incremental de aplicação

Considerando as cinco classes de operadores, estes foram ordenados respeitando a ordenação anterior, ou seja, Grupo-II depois Grupo-I. Dentro do Grupo-II, inicialmente optou-se por aplicar os operadores da classe G-II-comandos seguido dos operadores da classe G-II-argumentos. Dentro do Grupo-I a ordem estabelecida foi: G-I-comandos, G-I-operadores e G-I-variáveis. Os operadores de mesma classe são ordenados pelo custo (número de mutantes gerados – Tabela 20(c)). É importante observar que os grupos foram ordenados de modo a garantir, inicialmente, que requisitos mínimos do teste de integração sejam satisfeitos e, posteriormente, os 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 garantes 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.

Deve-se lembrar 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 tem-se CE_{pre} = {II-ArgAriNeg, I-CovAllNod, I-IndVarBitNeg, I-IndVarRepReq}.

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

Desta vez, 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 respeitando-se a ordem estabelecida no Passo 4:

1. G-II-comandos;
2. G-II-argumentos;
3. G-I-comandos;
4. G-I-operadores; e
5. G-I-variáveis.

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*. Na primeira iteração deste passo, a Tabela 26 apresenta o incremento proporcionado pelos operadores de mutação.

Considerando a Tabela 26 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}.

Como um novo operador foi adicionado a CE_{pre}, as informações apresentadas na Tabela 26 devem ser coletadas novamente; os resultados obtidos são apresentados na Tabela 27.

Tabela 26 – Incremento Proporcionado pelos Operadores: Primeira Iteração

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

Tabela 27 – Incremento Proporcionado pelos Operadores: Segunda Iteração

Operador	Índice de Incremento	Escore CE_{pre} + Incremento
I-IndVarRepExt	0,00421	0,99145
I-IndVarRepLoc	0,00418	0,99382
I-DirVarRepCon	0,00277	0,98972
I-IndVarRepCon	0,00259	0,98979
I-IndVarRepGlo	0,00203	0,99202
I-DirVarRepGlo	0,00189	0,98852
I-DirVarRepExt	0,00182	0,99143
I-IndVarRepPar	0,00174	0,99174
I-IndVarIncDec	0,00142	0,98833
I-DirVarRepReq	0,00109	0,99105
I-DirVarRepPar	0,00099	0,99060
I-DirVarIncDec	0,00086	0,99083
II-FunCalDel	0,00068	0,98792
I-DirVarRepLoc	0,00019	0,99002
I-CovAllEdg	0,00014	0,98967

Considerando a classe G-I-variáveis, dois operadores apresentam incremento dentro da mesma faixa: I-IndVarRepExt e I-IndVarRepLoc. Analisando qual a adequação do conjunto de casos de teste CE_{pre}-adequado em relação a esses operadores obtêm-se os escores de 0,996 e 0,964, respectivamente, de modo que I-IndVarRepLoc é adicionado a CE_{pre}.

Esse processo foi repetido mais duas 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*

Novamente, considerando *IMS* = 0,70 nenhum operador de mutação de interface apresenta alto *strength* (Tabela 20 (b)) e, desse modo, CE_{pre} continua com os mesmos operadores do Passo 5: CE_{pre} = {II-ArgAriNeg, I-CovAllNod, I-DirVarBitNeg, I-IndVarBitNeg, I-IndVarRepGlo, I-IndVarRepExt, I-IndVarRepLoc, I-IndVarRepReq}.

Ao final deste experimento, o segundo conjunto essencial obtido foi CE_{IM2} = {II-ArgAriNeg, I-CovAllNod, I-DirVarBitNeg, I-IndVarBitNeg, I-IndVarRepGlo, I-IndVarRepExt, I-IndVarRepLoc, I-IndVarRepReq}. No Quadro 6 é apresentada a descrição de tais operadores e na Tabela 28 e no Gráfico 3 são apresentados o escore de mutação e a redução de custo proporcionada pelo conjunto essencial CE_{pre} em cada passo do procedimento.

Quadro 6 – Descrição dos Operadores Essenciais de Interface da Linguagem C: CE_{IM2}

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.

Tabela 28 – Resultados Obtidos a cada Passo do Procedimento: CE_{IM2}

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

Novamente observa-se que do Passo 1 para o Passo2 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 avaliam-se possíveis relações de inclusão entre os operadores de CE_{pre}, três operadores foram removidos ocasionando uma pequena redução no escore de mutação mas, em contrapartida, reduziram sensivelmente o custo de aplicação de CE_{pre}. Em seguida, no Passo 4, nenhuma mudança no escore de mutação e na redução de custo ocorreu pois os operadores de CE_{pre} só foram 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 a redução 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 do passo anterior.

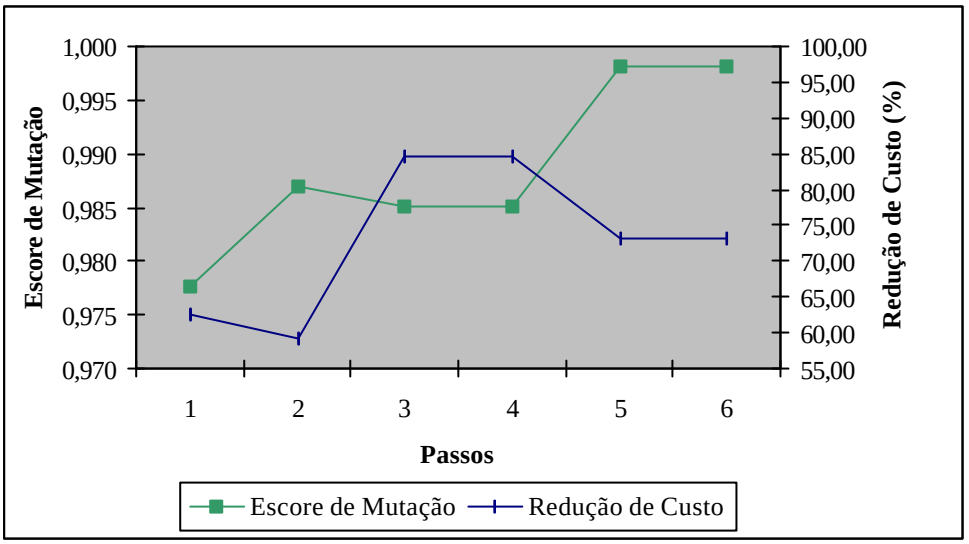


Gráfico 3 – Evolução do Conjunto Essencial Durante os Passos do Procedimento: CE_{IM2}

Procurando comparar os conjuntos CE_{IM1} e CE_{IM2}, as informações referentes ao escore de mutação e a redução de custo que tais conjuntos determinam em relação ao critério Mutação de Interface, estão sintetizadas nos gráficos 4 e 5 respectivamente.

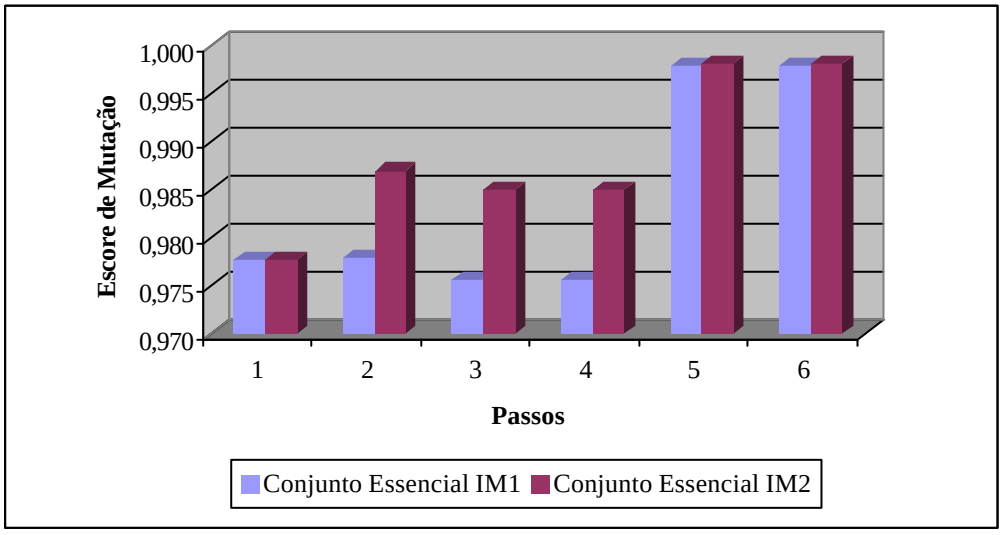


Gráfico 4 – Comparação dos Escores de Mutação Obtidos por CE_{IM1} e CE_{IM2}

Em termos de escore de mutação (Gráfico 4), somente no Passo 1 CE_{IM1} e CE_{IM2} apresentaram o mesmo grau de cobertura em relação à Mutação de Interface; nos demais passos, CE_{IM2} foi sempre melhor do que CE_{IM1} e o grau de adequação obtido em relação ao critério Mutação de Interface foi superior a 0,998. Em termos de redução de custo (Gráfico 5), até o

Passo 4 CE_{IM1} apresentava uma melhor redução de custo que CE_{IM2}; entretanto, no final, CE_{IM2} também proporcionou uma maior redução de custo do que CE_{IM1} e, desse modo, pode-se dizer que o melhor conjunto essencial obtido é aquele formado pelos operadores de mutação de CE_{IM2}. Isso demonstra que a divisão dos operadores de mutação de interface em classes mais significativas fez com que o procedimento **Essencial** [BAR98a] pudesse selecionar operadores com melhores características para o teste dos programas utilizados no experimento. Espera-se que devido a tal classificação o conjunto essencial obtido também apresente resultados satisfatórios quando aplicados a outros domínios de aplicação.

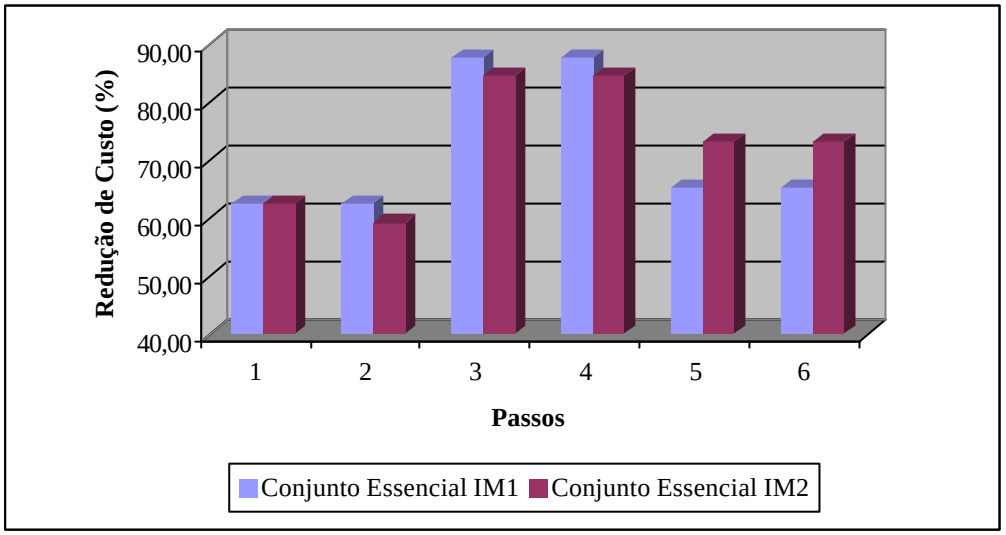


Gráfico 5 – Comparação das Reduções de Custo Proporcionadas por CE_{IM1} e CE_{IM2}

Em relação à estratégia incremental definida na Seção 3.3.1.1 (Tabela 14), observa-se que, para se obter um escore de mutação próximo a 0,998, deveriam ser utilizados os 7 primeiros operadores e a redução de custo proporcionada seria em torno de 64,80%, aproximadamente 8,42% a menos do que seria obtido com a utilização de CE_{IM2}. Desse modo, uma estratégia incremental que inicie com a aplicação dos operadores de CE_{IM2} mostra-se melhor em termos de escore de mutação e redução de custo do que a estratégia incremental definida na Tabela 14. Assim sendo, na próxima seção, a estratégia incremental de aplicação dos operadores de mutação de interface é reavaliada considerando que os operadores de CE_{IM2} tenham prioridade em relação aos demais operadores.

No restante deste trabalho, a menos que se diga o contrário, deve-se entender por conjunto essencial de operadores de mutação de interface (CE_{IM}) o conjunto formado pelos operadores de CE_{IM2}.

3.3.3 – Estratégia Incremental a Partir do Conjunto Essencial

Para estabelecer uma ordem inicial de aplicação dos operadores de mutação de interface, priorizando os operadores de CE_{IM}, os operadores foram classificados da seguinte forma:

1. operadores de CE_{IM} respeitando a ordem de aplicação incremental estabelecida pelo próprio procedimento **Essencial** [BAR98a]; e
2. demais operadores respeitando a ordem estabelecida na Tabela 20(a), ou seja, operadores ordenados pelo escore de mutação.

A partir dessa nova classificação, o mesmo processo utilizado na Seção 3.3.1 foi repetido e o resultado obtido na primeira iteração é ilustrado na Tabela 29.

Tabela 29 – Estratégia Incremental a Partir de CE_{IM}: Primeira Iteração

Op	EAn	EAt	Inc	CO	CC	RC
II-ArgAriNeg	0,00000	0,29979	0,29979	26	26	99,81460
I-CovAllNod	0,29979	0,84883	0,54904	438	464	96,69139
I-DirVarBitNeg	0,84883	0,90869	0,05986	154	618	95,59327
I-IndVarBitNeg	0,90869	0,98178	0,07309	306	924	93,41129
I-IndVarRepGlo	0,98178	0,98581	0,00403	312	1236	91,18654
I-IndVarRepExt	0,98581	0,99440	0,00859	511	1747	87,54278
I-IndVarRepLoc	0,99440	0,99756	0,00316	618	2365	83,13605
I-IndVarRepReq	0,99756	0,99813	0,00057	1391	3756	73,21734
I-IndVarIncDec	0,99813	0,99824	0,00011	684	4440	68,33999
I-IndVarLogNeg	0,99824	0,99824	0,00000	306	4746	66,15801
I-IndVarRepCon	0,99824	0,99849	0,00025	2555	7301	47,93925
I-DirVarIncDec	0,99849	0,99857	0,00008	536	7837	44,11723
I-DirVarRepReq	0,99857	0,99857	0,00000	770	8607	38,62664
I-DirVarRepCon	0,99857	0,99889	0,00032	1380	9987	28,78637
I-IndVarRepPar	0,99889	0,99926	0,00037	672	10659	23,99458
I-IndVarAriNeg	0,99926	0,99926	0,00000	306	10965	21,81261
I-DirVarRepPar	0,99926	0,99949	0,00023	385	11350	19,06731
I-DirVarRepExt	0,99949	0,99989	0,00040	494	11844	15,54478
I-DirVarLogNeg	0,99989	0,99989	0,00000	154	11998	14,44666
I-CovAllEdg	0,99989	0,99991	0,00002	390	12388	11,66572
I-DirVarRepLoc	0,99991	0,99991	0,00000	488	12876	8,18597
II-ArgStcDif	0,99991	0,99991	0,00000	43	12919	7,87935
I-DirVarAriNeg	0,99991	0,99991	0,00000	154	13073	6,78123
II-ArgBitNeg	0,99991	0,99991	0,00000	26	13099	6,59584
I-DirVarRepGlo	0,99991	0,99994	0,00003	170	13269	5,38363
II-ArgLogNeg	0,99994	0,99994	0,00000	26	13295	5,19823
I-RetStaRep	0,99994	0,99994	0,00000	42	13337	4,89875
II-ArgIncDec	0,99994	0,99994	0,00000	200	13537	3,47262
II-ArgRepReq	0,99994	0,99994	0,00000	55	13592	3,08043
II-FunCalDel	0,99994	1,00000	0,00006	254	13846	1,26925
...	...	...	...	...	...	...
II-ArgStcAli	1,00000	1,00000	0,00000	21	14024	0,00000

Para garantir que os operadores do conjunto essencial sejam aplicados primeiro, somente os operadores não pertencentes a CE_{IM} são reordenados de acordo com o incremento no escore de mutação que cada um proporciona.

Novamente, o processo termina quando todos os operadores não pertencentes a CE_{IM} estiverem ordenados de forma decrescente pelo incremento. Para isso, foram necessárias mais três iterações e o resultado final obtido é apresentado na Tabela 30.

Como dito anteriormente, somente com a aplicação dos 8 primeiros operadores de mutação da Tabela 30 (conjunto CE_{IM}) é possível obter um escore de mutação de 0,998 e uma redução de custo de 73,22%. Desejando-se obter conjuntos de casos de teste *IM*-adequados, outros 10 operadores de mutação devem ser aplicados e a redução de custo obtida é de aproximadamente 25,96%, 4,21% superior à obtida com a estratégia incremental definida na Tabela 14, na qual para se obter conjuntos *IM*-adequados eram necessários utilizar 17 operadores de mutação e a redução de custo proporcionada era da ordem de 21,75%.

Tabela 30 – Estratégia Incremental a Partir de CE_{IM}: Última Iteração

Op	EAn	EAt	Inc	CO	CC	RC
II-ArgAriNeg	0,00000	0,29979	0,29979	26	26	99,81460
I-CovAllNod	0,29979	0,84883	0,54904	438	464	96,69139
I-DirVarBitNeg	0,84883	0,90869	0,05986	154	618	95,59327
I-IndVarBitNeg	0,90869	0,98178	0,07309	306	924	93,41129
I-IndVarRepGlo	0,98178	0,98581	0,00403	312	1236	91,18654
I-IndVarRepExt	0,98581	0,99440	0,00859	511	1747	87,54278
I-IndVarRepLoc	0,99440	0,99756	0,00316	618	2365	83,13605
I-IndVarRepReq	0,99756	0,99813	0,00057	1391	3756	73,21734
I-IndVarRepPar	0,99813	0,99895	0,00082	672	4428	68,42556
I-DirVarRepExt	0,99895	0,99935	0,00040	494	4922	64,90302
I-DirVarRepPar	0,99935	0,99965	0,00030	385	5307	62,15773
I-IndVarRepCon	0,99965	0,99978	0,00013	2555	7862	43,93896
I-DirVarIncDec	0,99978	0,99987	0,00009	536	8398	40,11694
II-FunCalDel	0,99987	0,99993	0,00006	254	8652	38,30576
I-IndVarIncDec	0,99993	0,99996	0,00003	684	9336	33,42841
I-DirVarRepGlo	0,99996	0,99998	0,00002	170	9506	32,21620
I-CovAllEdg	0,99998	0,99999	0,00001	390	9896	29,43525
I-DirVarRepLoc	0,99999	1,00000	0,00001	488	10384	25,95550
...	...	...	...	...	...	...
II-ArgStcDif	1,00000	1,00000	0,00000	43	14024	0,00000

A Figura 7 ilustra três subconjuntos de operadores de mutação de interface (A, B e C) correspondentes às três estratégias incrementais definidas no decorrer das seções 3.3.1 e 3.3.3:

- Subconjunto A: ordem incremental estabelecida a partir dos operadores que proporcionam os maiores escores de mutação (Tabela 14);
- Subconjunto B: ordem incremental estabelecida priorizando os operadores do Grupo-II (Tabela 18); e
- Subconjunto C: ordem incremental estabelecida priorizando os operadores de CE_{IM} (Tabela 30).

Observa-se que 16 operadores de mutação são comuns aos subconjuntos A, B e C o que demonstra que esses operadores, em geral, sempre serão requeridos para garantir conjuntos de casos de teste *IM*-adequados. Na verdade, para o conjunto de programas utilizados, somente a utilização desses operadores de mutação garante um escore de mutação igual a 1,00 em relação ao critério Mutação de Interface e uma redução de custo de aproximadamente 72,76%. Isso ocorre pois as estratégias incrementais estabelecidas são baseadas em uma determinada ordem de aplicação dos operadores e, desse modo, se a ordem de aplicação estabelecida for alterada pode ocorrer que alguns operadores sejam incluídos. No caso da estratégia A, o operador I-DirVarRepReq é o primeiro a ser aplicado; entretanto, se fosse o último o conjunto formado pelos 16 operadores seria capaz de distinguir todos os mutantes gerados por I-DirVarRepReq. Assim sendo, o que deve ser observado é que as estratégias propostas foram baseadas em determinada ordem de aplicação e, para torná-las independentes dessa ordem seria necessário avaliar a relação de inclusão entre todos os operadores eliminando os que fossem incluídos.

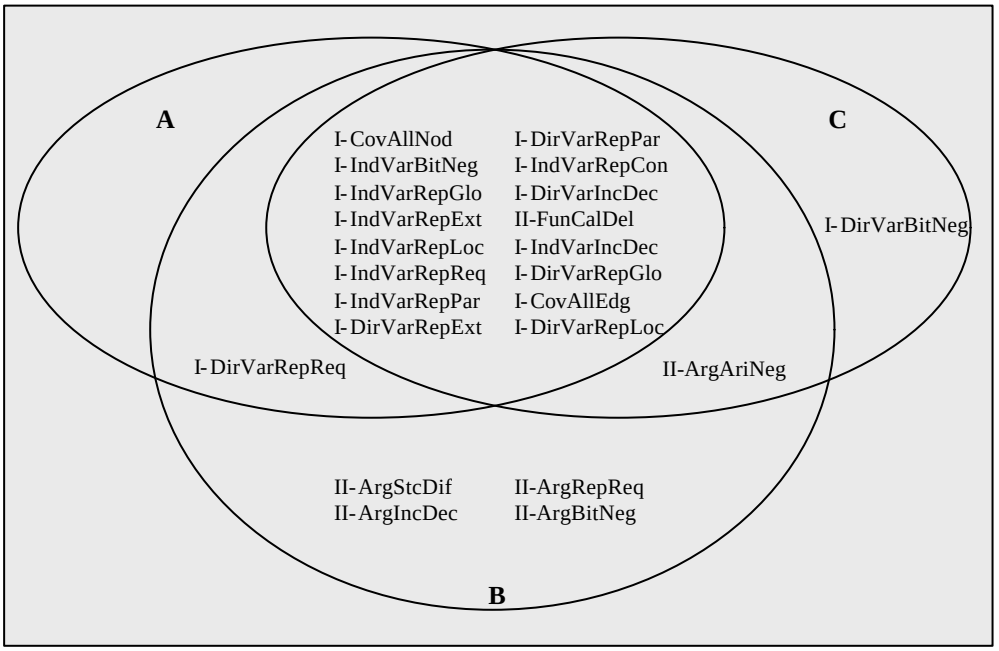


Figura 7 – Representação de Subconjuntos de Operadores de Interface que Selecionam Conjuntos de Casos de Teste *IM*-adequados

Considerando os subconjuntos A e C observa-se que a substituição do operador I-DirVarRepReq pelos operadores II-ArgAriNeg e I-DirVarBitNeg fez com que a redução de custo obtida passasse de 21,75% para 25,96% e mesmo assim garantisse a seleção de conjuntos de casos de teste *IM*-adequados. Desse modo, a estratégia C apresenta-se, no geral, como sendo a de menor custo para se obterem conjuntos de casos de teste *IM*-adequados.

O subconjunto B, embora seja o que proporciona a menor redução de custo (19,25%), prioriza a aplicação dos operadores do Grupo-II que, dependendo das restrições de tempo e custo disponíveis, pode ser uma boa alternativa para, a um baixo custo (com apenas 4,31% do total de mutantes gerados), pelo menos garantir que os requisitos mínimos do teste de integração – testar todas as chamadas de função e garantir que todos os argumentos estejam corretos – sejam cobertos.

3.4 – Considerações Finais

Neste capítulo, foi apresentado o *framework* utilizado para a condução dos experimentos, bem como uma descrição das etapas a serem cumpridas na realização dos experimentos. Em seguida, foram apresentadas algumas abordagens para a determinação de uma estratégia incremental de aplicação dos operadores de mutação de interface. Utilizando tais abordagens, inicialmente, duas seqüências de aplicação dos operadores de mutação de interface forma estabelecidas: uma a partir dos operadores de mutação de interface que determinavam os maiores escores de mutação em relação aos demais e outra a partir dos operadores do Grupo-II. Ambas as seqüências de operadores produziam conjuntos *IM*-adequados e possibilitavam redução de custo entre 21,76% e 19,25%.

Além disso, o procedimento **Essencial** [BAR98a] foi aplicado utilizando-se os operadores de mutação da ferramenta *Proteum/IM* e um conjunto essencial de operadores de mutação de interface para a linguagem C foi obtido. A partir do conjunto essencial, uma nova estratégia incremental de aplicação dos operadores foi estabelecida.

Uma comparação das três estratégias de aplicação dos operadores foi realizada e, a formada a partir dos operadores essenciais apresentou-se como a que seleciona conjuntos *IM*-adequados com o menor custo.

No geral, resultados demonstram que podem existir inúmeros subconjuntos de operadores de mutação capazes de selecionar conjuntos de casos de teste *IM*-adequados e, para se obter o melhor subconjunto, o ideal seria que todas as combinações possíveis de operadores de mutação fossem consideradas. Entretanto, visto que o critério Mutação de Interface possui 33 operadores de mutação, o número de combinações que deveriam ser investigadas é de 2^{33} o que, de certa forma, dificulta tal análise devido ao alto custo computacional envolvido para se calcular o escore de mutação e a redução de custo obtidos a partir de cada combinação.

As abordagens apresentadas procuram fornecer alternativas pragmáticas para selecionar subconjuntos de operadores de mutação que possibilitem uma sensível redução no custo de aplicação do critério e ainda assim garantam a seleção de conjuntos de casos de teste *IM*-adequados.

A seguir, serão feitas algumas análises entre os critérios baseados em mutação (Análise de Mutantes e Mutação de Interface), com o objetivo de estabelecer formas alternativas – de menor custo – de aplicação dos mesmos, em uma estratégia que aborde as fases do teste de unidade e integração.

Capítulo 4

Subsídios para a Determinação de Estratégias de Teste Baseadas na Técnica de Mutação: Um Estudo de Caso

4.1 – Considerações Iniciais

O estudo empírico conduzido neste trabalho visa a fornecer subsídios para o estabelecimento de estratégias de teste baseadas na técnica de mutação que sejam incrementais, de baixo custo e eficazes, e apóiem todo o ciclo de desenvolvimento de software.

No capítulo anterior mostrou-se que, devido às relações de inclusão existentes entre os operadores de mutação de interface, é possível identificar subconjuntos de operadores que selecionavam os mesmos conjuntos de casos de teste que o conjunto total de operadores selecionaria e, desse modo, a utilização de apenas um subconjunto permitia uma maior redução no custo de aplicação, em termos do número de mutantes gerados, mantendo a mesma adequação em relação ao critério Mutação de Interface.

Inicialmente, neste capítulo são comparados os critérios Análise de Mutantes e Mutação de Interface a fim de verificar a real necessidade de aplicação de ambos e avaliar qual o *strength*

de um em relação ao outro. Posteriormente, o mesmo raciocínio empregado no Capítulo 3 é utilizado para identificar formas alternativas de se aplicar os critérios de teste baseados em mutação. O que se deseja é avaliar quais as relações de inclusão existentes entre os operadores de mutação utilizados em nível de unidade e os operadores de mutação utilizados em nível de integração, ou seja, avaliar a adequação de conjuntos de casos de teste AM-adequados em relação aos operadores de interface e vice-versa. Identificadas tais relações, é possível estabelecer novas estratégias incrementais de aplicação desses operadores, garantindo a mesma adequação que o conjunto total obteria com um menor custo. Finalmente, a partir do conjunto essencial de operadores de mutação de unidade definidos por Barbosa [BAR98a] e do conjunto essencial de operadores de mutação de interface (Capítulo 3), as análises de cobertura que um critério determina em relação ao outro são reavaliadas e novas estratégias de aplicação dos operadores são estabelecidas a partir de conjuntos de casos de teste adequados aos conjuntos essenciais.

4.2 – Análise de Mutantes ´ Mutação de Interface

Nesta seção é avaliado o *strength* dos critérios Análise de Mutantes e Mutação de Interface. Com base nessa avaliação, duas estratégias incrementais de aplicação de operadores de mutação são definidas: a primeira indica quais operadores de mutação de interface devem ser aplicados se o teste de unidade utilizando-se o critério Análise de Mutantes já foi realizado; e a segunda indica quais operadores de mutação de unidade devem ser aplicados se, no teste de integração, for planejada a utilização do critério Mutação de Interface.

4.2.1 – Avaliação do *Strength* dos Critérios

A primeira avaliação realizada entre os critérios Análise de Mutantes e Mutação de Interface diz respeito ao *strength* de um em relação ao outro. Através dessa análise pode-se avaliar a necessidade da utilização do critério Mutação de Interface se, no teste de unidade, já foi utilizado o critério Análise de Mutantes. Por outro lado, também é possível avaliar qual a necessidade de se aplicar o critério Análise de Mutantes se, no teste de integração, planeja-se utilizar o critério Mutação de Interface; obviamente, faz-se aqui uma abstração da relevância em se antecipar à identificação da presença de erros.

A avaliação de *strength* consiste em verificar qual a adequação de conjuntos de casos de teste AM-adequados em relação ao critério Mutação de Interface e vice-versa. Assim, utilizando

o mesmo conjunto de programas UNIX do Capítulo 3, os passos abaixo descrevem a forma como foram conduzidos os experimentos:

Critério Análise de Mutantes

- geração de todos os mutantes de unidade;
- determinação de todos os mutantes equivalentes;
- geração de 11 conjuntos de casos de teste AM-adequados para cada programa em teste;
- avaliação da adequação de cada conjunto AM-adequado em relação ao critério Mutação de Interface.

Critério Mutação de Interface

- geração de todos os mutantes de interface;
- determinação de todos os mutantes equivalentes;
- geração de 11 conjuntos de casos de teste *IM*-adequados para cada programa em teste;
- avaliação da adequação de cada conjunto *IM*-adequado em relação ao critério Análise de Mutantes.

A média de cobertura obtida para cada programa (calculada a partir da média de cobertura determinada pelos 11 conjuntos de casos de teste adequados) e a média geral são apresentadas na Tabela 31.

Tabela 31 – *Strength* Entre os Critérios Análise de Mutantes e Mutação de Interface

Programas	$T_{\text{AM-adequado}}$ **Mutação de Interface**	$T_{\textit{IM}\text{-adequado}}$ **Análise de Mutantes**
Cal	0,92743	0,87969
Checkeq	0,95808	0,92650
Comm	0,93070	0,94064
Look	0,93122	0,95528
Uniq	0,92919	0,98337
Média	0,93532	0,93710
Desvio Padrão	0,01281	0,03838

Considerando os dados da Tabela 31, observa-se que os critérios Análise de Mutantes e Mutação de Interface são incomparáveis do ponto de vista da relação de inclusão; ou seja, conjuntos da casos de teste AM-adequados não são *IM*-adequados e conjuntos de casos de teste *IM*-adequados não são AM-adequados. A cobertura média que um critério determina em relação

ao outro é de 93%; entretanto, sabe-se que, em geral, independentemente da qualidade dos conjuntos de casos de teste utilizados, 80% dos mutantes gerados morrem na primeira execução [BUD80a]. Considerando então que somente em torno de 20% dos mutantes gerados contribuem efetivamente para a melhoria do conjunto de casos de teste, o que se deseja é, a partir da análise de *strength* dos critérios, identificar quais operadores são incluídos por outros, ou que geram os mutantes que morrem facilmente e desconsiderar esses operadores, reduzindo o custo de aplicação dos critérios. Procurando identificar quais são esses operadores, a avaliação de *strength* foi detalhada considerando características mais específicas relacionadas a cada critério.

Embora os critérios Análise de Mutantes e Mutação de Interface utilizem o conceito de mutação, eles são destinados a fases distintas da atividade de teste e, desse modo, para comparar esses critérios é necessário que se identifiquem quais entidades serão utilizadas na comparação. Nesse ponto, uma informação importante a ser considerada é a forma como tais critérios realizam as mutações nos programas em teste e como os operadores de mutação estão divididos.

Para o critério Análise de Mutantes, os operadores de mutação estão divididos em quatro classes e os mutantes gerados por tais operadores não pressupõem interação entre as unidades; desse modo, cada unidade pode ser testada individualmente. Já os operadores de mutação de interface estão divididos em dois grupos sendo que, os mutantes gerados pelos operadores do Grupo-I, responsáveis pelas mutações realizadas na função chamada, só serão mortos se o caso de teste exercitar a mutação passando por determinado ponto de chamada. Desse modo, os operadores de mutação de interface exercitam interações entre duas unidades, ou seja, uma conexão.

Assim sendo, os critérios Análise de Mutantes e Mutação de Interface foram comparados considerando, em nível de integração, conexões e, em nível de unidade, os pares de unidades que formam uma determinada conexão. Por exemplo, na Figura 8 estão representados os grafos de chamadas dos cinco programas utilizados no experimento. A partir desses gráficos é possível identificar todas as unidades que compõem os programas bem como as conexões existentes entre elas.

Obtidos conjuntos de casos de teste adequados para ambos os critérios de teste, comparou-se o quanto conjuntos de casos de teste AM-adequados a duas unidades eram capazes de distinguir os mutantes gerados para uma dada conexão (referente a tais unidades) e vice-versa, ou seja:

- dadas duas unidades *f* e *g*, verificou-se o quanto o conjunto de casos de teste *f*-AM-adequado ∪ *g*-AM-adequado (Análise de Mutantes) era adequado a conexão *f-g* (Mutação de Interface); e
- dada a conexão *f-g*, verificou-se o quanto o conjunto de casos de teste *f-g-IM*-adequado (Mutação de Interface) era adequado às unidades *f* e *g* (Análise de Mutantes).

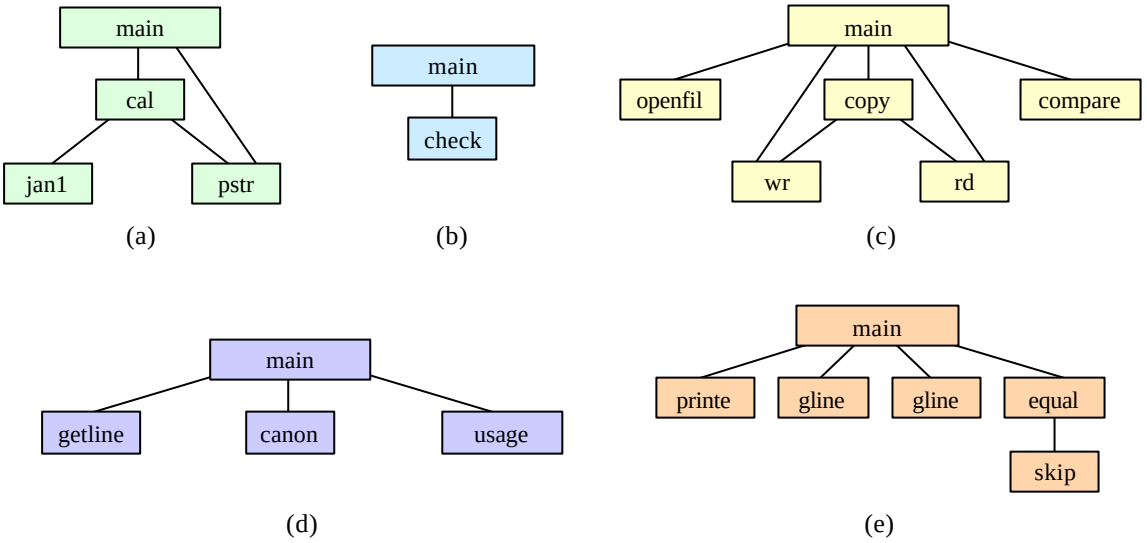


Figura 8 – Grafo de Chamada dos Programas: (a) *Cal*, (b) *Checkeq*, (c) *Comm*, (d) *Look* e (e) *Uniq*

Considerando o programa *Comm* e suas respectivas conexões entre unidades, a Tabela 32 apresenta a capacidade de um conjunto de casos de teste *f*-AM-adequado ∪ *g*-AM-adequado distinguir os mutantes gerados pelo Grupo-I, Grupo-II e Mutação de Interface (*IM*) quando aplicados à conexão *f-g*. Na Tabela 33 é apresentado o inverso, ou seja, qual a capacidade de um conjunto de casos de teste *f-g-IM*-adequado em distinguir os mutantes gerados pelas classes de operadores de mutação de Comandos, Operadores, Variáveis, Constantes e Análise de Mutantes (AM) quando aplicados às unidades *f* e *g*.

Tabela 32 – Avaliação de um Conjunto AM-adequado em Relação à Mutação de Interface: Programa *Comm*

Unidades	Conexão	Grupo-I	Grupo-II	*IM*
main + openfil	main-openfil	0,885	1,000	0,887
main + copy	main-copy	0,867	0,933	0,890
main + compare	main-compare	1,000	1,000	1,000
main + wr	main-wr	1,000	1,000	1,000
main + rd	main-rd	0,954	1,000	0,958
copy + wr	copy-wr	0,416	1,000	0,500
copy + rd	copy-rd	0,948	1,000	0,951

Observando-se as conexões *main-wr* e *copy-wr* da Tabela 32, pode-se concluir que, como o conjunto de casos de teste *main*-AM-adequado ∪ *wr*-AM-adequado permite distinguir todos os mutantes de interface e o mesmo não ocorre com o conjunto *copy*-AM-adequado ∪ *wr*-AM-adequado, os conjuntos de casos de teste *wr*-AM-adequado foram obtidos executando muito mais a conexão *main-wr* do que a conexão *copy-wr*. Assim sendo, observa-se a relevância do teste de integração ao requerer a execução das conexões.

Analisando a Tabela 33 tem-se o inverso, ou seja, o conjunto de casos de teste *main-wr-IM*-adequado não é capaz de incluir todas as classes de operadores de mutação de unidade aplicadas às unidades *main* e *wr*. Já o conjunto *copy-wr-IM*-adequado inclui todas as classes de mutação aplicadas às unidades *copy* e *wr*. Isso indica que, no teste de unidade e de integração, os requisitos de teste exigidos pelos critérios Análise de Mutantes e Mutação de Interface captam aspectos distintos e satisfazer 100% dos requisitos de teste exigidos por um critério pode não ser suficiente para garantir 100% de cobertura do outro, evidenciando o aspecto complementar desses critérios.

Tabela 33 – Avaliação de um Conjunto *IM*-adequado em Relação à Análise de Mutantes: Programa *Comm*

Conexão	Unidades	Comandos	Operadores	Variáveis	Constantes	AM
main-openfil	main + openfil	0,762	0,742	0,865	0,756	0,773
main-copy	main + copy	0,895	0,895	0,904	0,838	0,884
main-compare	main + compare	0,760	0,704	0,880	0,759	0,769
main-wr	main + wr	0,815	0,829	0,878	0,770	0,822
main-rd	main + rd	0,916	0,917	0,926	0,871	0,909
copy-wr	copy + wr	1,000	1,000	1,000	1,000	1,000
copy-rd	copy + rd	0,986	0,888	0,904	0,800	0,899

Essas tabelas foram geradas 11 vezes para cada programa, correspondendo aos 11 conjuntos de casos de teste adequados a cada critério. Os resultados médios obtido para cada programa do experimento e cada critério são apresentados nas tabelas 34 e 35.

Analisando a Tabela 34 observa-se que, para os programas *Cal*, *Checkeq* e *Look*, os conjuntos de casos de teste AM-adequados foram também Grupo-II-adequados para todas as conexões. Para os programas *Comm* e *Uniq* observa-se que em 3 e 2 conexões, respectivamente, conjuntos AM-adequados também se mostraram Grupo-II-adequados. Na média, observa-se que 96,2% dos mutantes gerados pelos operadores do Grupo-II são distinguidos por conjuntos de casos de teste AM-adequados. Em relação ao Grupo-I, a média de cobertura obtida é da ordem de 0,88, indicando que os mutantes gerados por esses operadores são mais difíceis de serem mortos por conjuntos de casos de teste AM-adequados. Isso é justificável tendo em vista que

esses mutantes são dependentes do ponto de chamada para o qual foram gerados, exigindo casos de teste mais específicos, que exercitem melhor a conexão entre as unidades.

Tabela 34 – Média dos 11 Conjuntos AM-adequados em Relação à Mutação de Interface

Programas	Unidades	Conexão	Grupo-I	Grupo-II	*IM*
Cal	main + cal	main-cal	0,914	1,000	0,918
	main + pstr	main-pstr	1,000	1,000	1,000
	cal + jan1	cal-jan1	1,000	1,000	1,000
Checkeq	main + check	main-check	0,958	1,000	0,958
Comm	main + openfil	main-openfil	0,730	1,000	0,736
	main + copy	main-copy	0,865	0,914	0,882
	main + compare	main-compare	1,000	1,000	1,000
	main + wr	main-wr	0,940	1,000	0,950
	main + rd	main-rd	0,932	0,993	0,937
	copy + wr	copy-wr	0,492	0,979	0,562
	copy + rd	copy-rd	0,885	0,949	0,889
Look	main + getline	main-getline	0,977	1,000	0,978
	main + canon	main-canon	0,917	1,000	0,919
	main + usage	main-usage	1,000	1,000	1,000
Uniq	main + printe	main-printe	0,500	0,500	0,500
	main + gline	main-gline	0,957	1,000	0,962
	main + pline	main-pline	0,768	0,955	0,779
	main + equal	main-equal	0,966	0,992	0,969
	equal + skip	equal-skip	0,867	1,000	0,868
Média			**0,877**	**0,962**	**0,885**

Tabela 35 – Média dos 11 Conjuntos *IM*-adequados em Relação à Análise de Mutantes

Programas	Conexão	Unidades	Comandos	Operadores	Variáveis	Constantes	AM
Cal	main-cal	main + cal	0,841	0,907	0,857	0,850	0,867
	main-pstr	main + pstr	0,790	0,845	0,754	0,750	0,782
	cal-jan1	cal + jan1	0,862	0,861	0,812	0,812	0,834
Checkeq	main-check	main + check	0,962	0,813	0,968	0,978	0,926
Comm	main-openfil	main + openfil	0,806	0,821	0,887	0,792	0,823
	main-copy	main + copy	0,890	0,880	0,909	0,849	0,881
	main-compare	main + compare	0,762	0,711	0,878	0,756	0,771
	main-wr	main + wr	0,817	0,828	0,878	0,771	0,823
	main-rd	main + rd	0,917	0,908	0,920	0,862	0,903
	copy-wr	copy + wr	0,992	1,000	0,998	0,996	0,996
	copy-rd	copy + rd	0,986	0,888	0,903	0,800	0,899
Look	main-getline	main + getline	0,865	0,921	0,888	0,903	0,897
	main-canon	main + canon	0,901	0,941	0,938	0,948	0,935
	main-usage	main + usage	0,153	0,039	0,066	0,065	0,074
Uniq	main-printe	main + printe	0,706	0,606	0,620	0,612	0,638
	main-gline	main + gline	0,947	0,916	0,942	0,807	0,911
	main-pline	main + pline	0,933	0,906	0,950	0,823	0,908
	main-equal	main + equal	0,930	0,896	0,941	0,795	0,898
	equal-skip	equal + skip	0,990	0,991	1,000	1,000	0,994
Média			**0,845**	**0,825**	**0,848**	**0,798**	**0,830**

Na Tabela 35 observa-se que conjuntos *IM*-adequados, em geral, não são capazes de determinar uma alta cobertura em relação às classes de operadores de mutação de unidade. No entanto, para uma análise mais detalhada, é necessário identificar qual a cobertura dos conjuntos de casos de teste em relação a cada operador de mutação de unidade. Assim sendo, mais uma vez foi feito um refinamento nos dados coletados, permitindo uma melhor análise dos critérios Análise de Mutantes e Mutação de Interface. Para isso, o experimento foi repetido da seguinte forma:

- dadas duas unidades f e g, verificou-se o quanto o conjunto f-AM-adequado $\cup$ g-AM-adequado (Análise de Mutantes) era adequado a cada um dos 33 operadores de mutação de interface aplicados à conexão f-g (Mutação de Interface); e
- dada a conexão f-g, verificou-se o quanto o conjunto f-g-*IM*-adequado (Mutação de Interface) era adequado a cada um dos 71 operadores de mutação de unidade aplicados às unidades f e g (Análise de Mutantes).

O que se deseja com essas avaliações é identificar operadores que estejam sendo incluídos, deixando de aplicá-los; reduzindo assim o custo dos critérios de teste baseados em mutação.

Para exemplificar as informações coletadas considere novamente o programa *Comm*. Na Tabela 36 é apresentada parcialmente a avaliação de um conjunto de casos de teste AM-adequado em relação aos operadores de interface aplicados a cada conexão do programa. O sinal (-) é utilizado para indicar que determinado operador não gerou mutantes para uma dada conexão.

Tabela 36 – Avaliação de um Conjunto AM-adequado em relação aos Operadores de Interface: Programa *Comm*

Operador	main-openfil	main-copy	main-compare	main-wr	main-rd	copy-wr	copy-rd
I-CovAllEdg	0,818	0,924	1,000	1,000	0,980	0,761	0,932
I-CovAllNod	0,855	0,939	1,000	0,981	0,987	0,682	0,955
I-DirVarAriNeg	0,682	0,727	-	0,945	-	0,364	-
I-DirVarBitNeg	0,682	0,818	-	1,000	-	0,591	-
I-DirVarIncDec	0,932	0,912	-	0,943	0,941	0,600	0,909
I-DirVarLogNeg	0,841	0,939	-	0,970	-	0,682	-
...	...	...	...	...	...	...	...
II-ArgRepReq	-	-	-	-	-	0,945	-
II-ArgStcAli	-	-	1,000	-	-	-	-
II-ArgStcDif	-	0,939	-	1,000	0,989	-	-
II-FunCalDel	1,000	0,939	1,000	1,000	0,998	1,000	0,982
Total	**0,736**	**0,882**	**1,000**	**0,950**	**0,937**	**0,562**	**0,889**

De acordo com a Tabela 36, considerando um conjunto de casos de teste *T* AM-adequado para as unidades *main* e *openfil*, *T* determina um escore de mutação de 0,818 em relação ao operador de mutação de interface I-CovAllEdg aplicado à conexão *main-openfil*, ou seja, *T* é capaz de distinguir 81,80% dos mutantes gerados por I-CovAllEdg na conexão *main-openfil*. Considerando o operador II-FunCalDel, observa-se que *T* é capaz de distinguir todos os mutantes gerados por este operador.

Na Tabela 37 tem-se o inverso, ou seja, qual a capacidade de um conjunto de casos de teste *IM*-adequado para uma dada conexão em distinguir os mutantes gerados pelos operadores de mutação de unidade quando aplicados ao par de unidades que formam tal conexão. Por exemplo, a partir de um conjunto de caso de teste *T IM*-adequado para a conexão *main-openfil*, tem-se que *T* determina um escore de mutação de 0,591 em relação ao operador SMTC para as funções *main* e *openfil*.

Tabela 37 – Avaliação de um Conjunto *IM*-adequado em relação aos Operadores de Unidade: Programa *Comm*

Operador	**main + openfil**	**main + copy**	**main + compare**	**main + wr**	**main + rd**	**copy + wr**	**copy + rd**
SMTC	0,591	0,727	0,667	0,500	0,727	1,000	1,000
SMTT	1,000	1,000	1,000	1,000	1,000	1,000	1,000
SRSR	0,835	0,948	0,833	0,831	0,961	1,000	1,000
SSDL	0,787	0,856	0,731	0,807	0,884	0,983	0,941
...	...	...	...	...	...	...	...
OLBN	0,886	0,955	0,545	1,000	0,750		0,500
OLLN	0,773	0,909	0,091	1,000	1,000		1,000
ORLN	0,615	0,759	0,489	0,655	0,913	1,000	1,000
ORRN	0,805	0,880	0,672	0,764	0,922	1,000	0,909
...	...	...	...	...	...	...	...
VDTR	0,835	0,799	0,804	0,857	0,891	0,990	1,000
Vprr	0,936	0,898	0,951	0,857	0,980	1,000	1,000
VTWD	0,786	0,742	0,784	0,808	0,777	1,000	0,800
...	...	...	...	...	...	...	...
Cccr	0,752	0,852	0,689	0,708	0,867	-	0,778
Ccsr	0,775	0,795	0,753	0,821	0,806	1,000	0,800
CRCR	0,914	0,874	0,893	0,891	0,895	0,995	0,824
Total	**0,823**	**0,881**	**0,771**	**0,823**	**0,903**	**0,996**	**0,899**

Novamente, essas tabelas foram geradas 11 vezes para cada programa, correspondendo aos 11 conjuntos de casos de teste adequados a cada critério. A média de cobertura que os conjuntos de casos de teste AM-adequados determinam em relação a cada operador de mutação de interface, consideradas todas as conexões de todos os programas, é apresentada na Tabela 38. A cobertura média dos conjuntos *IM*-adequados em relação a cada operador de mutação de unidade é apresentada na Tabela 39.

Na Seção 4.2.2, as informações apresentadas na Tabela 38 são utilizadas para o estabelecimento de uma estratégia incremental de aplicação dos operadores de mutação de

interface a partir de um conjunto de casos de teste AM-adequado e, na Seção 4.2.3, utilizando as informações da Tabela 39, é apresentada um estratégia incremental de aplicação dos operadores de unidade a partir de um conjunto *IM*-adequado.

4.2.2 – Estratégia Incremental de Aplicação dos Operadores de Interface a Partir de Conjuntos AM-adequados

A partir de cada linha da Tabela 38 pode-se avaliar a cobertura média que conjuntos de casos de testes AM-adequados (considerando um par de unidades) determinam em relação a cada operador de mutação de interface aplicado a uma dada conexão. Por exemplo, conjuntos de casos de testes AM-adequados para as unidades *main* e *cal* determinam, em média, um escore de mutação de 0,940 em relação ao operador I-CovAllEdg, aplicado à conexão *main-cal*.

Fazendo essa análise para todas as conexões e todos os programas, pode-se obter a cobertura média que conjuntos de casos de teste AM-adequados determinam em relação a cada operador de mutação de interface. No caso, conjuntos AM-adequados determinam uma cobertura média de 0,939 para o operador I-CovAllEdg.

Com base nesses valores de cobertura, uma questão que surge é: "considerando que o critério Análise de Mutantes já foi aplicado, sendo obtido um conjunto de caso de teste AM-adequado, qual a melhor forma de se complementar esse conjunto de casos de teste de modo a torná-lo *IM*-adequado?", ou seja, possuindo um conjunto AM-adequado, qual a melhor seqüência de aplicação dos operadores de mutação de interface de maneira a convergir o mais rápido possível para um conjunto *IM*-adequado.

Nesse sentido, uma estratégia para a aplicação dos operadores de mutação de interface seria iniciar o teste de integração a partir dos operadores que apresentam maior *strength* em relação ao conjunto AM-adequado, ou seja, aqueles para os quais o conjunto AM-adequado determina os menores escores de mutação. Começar o teste de integração a partir de tais operadores implica, inicialmente, selecionar novos casos de teste que sejam capazes de distinguir os mutantes, que até então, mostraram-se os mais difíceis de serem mortos por um conjunto de casos de testes AM-adequado, aprimorando o grau de adequação do conjunto em relação ao critério Mutação de Interface.

Tabela 38 – Média Geral de Cobertura de Conjuntos AM-adequados em Relação ao Critério Mutação de Interface

	Cal			Checkeq	Comm							Look			Uniq					
Operador	**main-cal**	**main-pstr**	**cal-jan1**	**main-check**	**main-openfil**	**main-copy**	**main-compare**	**main-wr**	**main-rd**	**copy-wr**	**copy-rd**	**main-getline**	**main-canon**	**main-usage**	**main-printe**	**main-gline**	**main-pline**	**main-equal**	**equal-skip**	**Média**
I-CovAllEdg	0,940	1,000	1,000	0,932	0,818	0,924	1,000	1,000	0,980	0,761	0,932	1,000	1,000	-	-	0,985	0,718	1,000	0,972	**0,939**
I-CovAllNod	0,959	1,000	1,000	0,958	0,855	0,939	1,000	0,981	0,987	0,682	0,955	1,000	1,000	1,000	0,500	0,988	0,742	1,000	0,975	**0,922**
I-DirVarAriNeg	0,912	1,000	1,000	1,000	0,682	0,727	-	0,945	-	0,364	-	-	1,000	-	-	-	0,920	1,000	0,773	**0,860**
I-DirVarBitNeg	0,912	1,000	1,000	0,969	0,682	0,818	-	1,000	-	0,591	-	-	0,556	-	-	-	0,784	1,000	0,818	**0,844**
I-DirVarIncDec	0,978	1,000	1,000	1,000	0,932	0,912	-	0,943	0,941	0,600	0,909	1,000	0,959	-	0,500	-	0,784	1,000	0,952	**0,901**
I-DirVarLogNeg	0,912	1,000	1,000	1,000	0,841	0,939	-	0,970	-	0,682	-	-	1,000	-	-	-	0,824	1,000	0,852	**0,918**
I-DirVarRepCon	0,918	1,000	1,000	0,970	0,727	0,939	-	0,937	-	0,424	-	-	0,674	-	-	-	0,762	1,000	0,879	**0,853**
I-DirVarRepExt	0,916	1,000	-	-	0,750	0,900	1,000	0,909	0,989	0,182	0,909	1,000	0,773	-	0,500	0,990	0,705	0,993	0,910	**0,839**
I-DirVarRepGlo	1,000	-	-	-	-	-	-	0,941	-	0,509	-	-	0,742	-	-	-	0,782	1,000	0,847	**0,832**
I-DirVarRepLoc	0,914	1,000	1,000	0,961	-	-	1,000	-	0,980	-	0,909	1,000	0,649	-	-	1,000	-	0,955	0,866	**0,936**
I-DirVarRepPar	0,904	1,000	-	-	0,682	0,773	1,000	0,945	0,989	0,571	0,909	-	0,782	-	0,500	-	0,793	0,977	1,000	**0,845**
I-DirVarRepReq	0,912	1,000	1,000	0,975	0,699	0,838	-	0,967	-	0,550	-	-	0,697	-	-	-	0,765	1,000	0,834	**0,853**
I-IndVarAriNeg	0,900	1,000	1,000	1,000	0,788	1,000	1,000	0,818	0,966	0,091	0,970	1,000	0,995	-	0,500	1,000	0,864	1,000	0,727	**0,868**
I-IndVarBitNeg	0,909	1,000	1,000	0,950	0,664	0,939	1,000	0,864	0,937	0,000	0,945	1,000	0,956	-	0,500	1,000	0,818	0,985	0,803	**0,848**
I-IndVarIncDec	0,932	1,000	1,000	0,964	0,782	0,902	1,000	0,886	0,921	0,159	0,939	0,996	0,972	-	0,500	0,982	0,845	0,987	0,803	**0,865**
I-IndVarLogNeg	0,897	1,000	1,000	0,950	0,664	0,864	1,000	0,909	0,902	0,182	0,945	0,992	0,979	-	0,500	0,970	0,818	0,989	0,864	**0,857**
I-IndVarRepCon	0,904	1,000	1,000	0,945	0,664	-	1,000	-	0,922	-	0,941	0,994	0,976	-	0,500	0,982	0,818	0,989	0,833	**0,898**
I-IndVarRepExt	1,000	1,000	-	1,000	0,545	0,455	1,000	0,909	0,851	0,182	0,616	0,865	-	-	0,500	0,893	0,688	0,867	0,896	**0,767**
I-IndVarRepGlo	1,000	-	-	0,975	-	-	-	0,859	-	0,152	-	-	0,949	-	-	-	0,753	0,818	0,835	**0,792**
I-IndVarRepLoc	0,908	1,000	1,000	0,928	-	-	1,000	-	0,865	-	0,845	0,985	0,890	-	-	1,000	-	1,000	0,790	**0,934**
I-IndVarRepPar	0,915	1,000	1,000	-	0,697	0,864	1,000	0,898	0,947	0,121	0,909	1,000	0,980	-	0,500	0,709	0,836	0,955	0,966	**0,841**
I-IndVarRepReq	0,902	1,000	1,000	0,957	0,734	0,898	1,000	0,864	0,953	0,136	0,941	0,998	0,986	-	0,500	0,989	0,818	0,986	0,805	**0,859**
I-RetStaDel	-	-	1,000	-	1,000	-	1,000	1,000	0,972	0,455	0,955	1,000	-	-	-	0,970	0,614	1,000	1,000	**0,914**
I-RetStaRep	-	-	-	-	-	-	1,000	-	0,972	-	0,955	1,000	-	-	-	0,955	-	0,985	-	**0,978**
II-ArgAriNeg	1,000	1,000	1,000	-	-	0,727	-	1,000	-	0,909	-	-	-	-	-	-	-	-	-	**0,939**
II-ArgBitNeg	1,000	1,000	1,000	-	-	0,818	-	1,000	-	1,000	-	-	-	-	-	-	-	-	-	**0,970**
II-ArgDel	1,000	1,000	1,000	1,000	-	0,939	1,000	1,000	0,989	-	-	1,000	1,000	-	0,500	1,000	0,955	1,000	-	**0,956**
II-ArgIncDec	1,000	1,000	1,000	1,000	1,000	0,912	1,000	1,000	0,991	1,000	0,909	1,000	1,000	-	0,500	1,000	0,955	0,977	1,000	**0,958**
II-ArgLogNeg	1,000	1,000	1,000	-	-	0,939	-	1,000	-	1,000	-	-	-	-	-	-	-	-	-	**0,990**
II-ArgRepReq	1,000	-	1,000	-	-	-	-	-	-	0,945	-	-	-	-	-	-	-	-	-	**0,982**
II-ArgStcAli	1,000	-	-	-	-	-	1,000	-	-	-	-	-	1,000	-	0,500	-	-	1,000	-	**0,900**
II-ArgStcDif	1,000	1,000	-	-	-	0,939	-	1,000	0,989	-	-	-	-	-	-	-	-	-	-	**0,986**
II-FunCalDel	1,000	1,000	1,000	1,000	1,000	0,939	1,000	1,000	0,998	1,000	0,982	1,000	1,000	1,000	0,500	1,000	0,955	1,000	1,000	**0,967**
Total	**0,918**	**1,000**	**1,000**	**0,958**	**0,736**	**0,882**	**1,000**	**0,950**	**0,937**	**0,562**	**0,889**	**0,978**	**0,919**	**1,000**	**0,500**	**0,962**	**0,779**	**0,969**	**0,868**	**0,885**

Tabela 39 – Média Geral de Cobertura de Conjuntos *IM*-adequados em Relação ao Critério Análise de Mutantes

	Cal			Checkeq	Comm							Look			Uniq					
Operador	**main + cal**	**main + pstr**	**cal + jan1**	**main + check**	**main + openfil**	**main + copy**	**main + compare**	**main + wr**	**main + rd**	**copy + wr**	**copy + rd**	**main + getline**	**main + canon**	**main + usage**	**main + printe**	**main + gline**	**main + pline**	**main + equal**	**equal + skip**	**Média**
SBRC	-	1,000	-	1,000	-	-	-	-	-	-	-	0,886	1,000	0,000	1,000	1,000	1,000	1,000	-	**0,876**
SBRn	-	-	-	1,000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**1,000**
SCRB	-	-	-	1,000	1,000	1,000	1,000	1,000	1,000	-	-	-	-	-	1,000	1,000	1,000	1,000	-	**1,000**
SCRn	-	-	-	1,000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**1,000**
SDWD	-	-	-	-	-	1,000	-	-	-	1,000	1,000	-	-	-	0,545	1,000	1,000	1,000	-	**0,935**
SGLR	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SMTC	1,000	1,000	1,000	0,977	0,591	0,727	0,667	0,500	0,727	1,000	1,000	0,800	0,800	0,000	0,636	0,977	1,000	1,000	0,818	**0,801**
SMTT	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,000	0,667	1,000	1,000	1,000	1,000	**0,930**
SMVB	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,000	0,545	0,833	0,750	0,750	1,000	**0,888**
SRSR	0,859	0,804	0,865	0,971	0,835	0,948	0,833	0,831	0,961	1,000	1,000	0,918	0,934	0,186	0,731	0,964	0,962	0,962	1,000	**0,872**
SSDL	0,812	0,754	0,850	0,941	0,787	0,856	0,731	0,807	0,884	0,983	0,941	0,786	0,863	0,164	0,642	0,937	0,931	0,914	1,000	**0,820**
SSWM	1,000	-	0,455	-	0,857	0,857	0,857	0,900	0,857	1,000	-	0,667	0,667	0,333	-	-	1,000	-	-	**0,787**
STRI	0,756	0,714	0,909	0,975	0,688	0,797	0,573	0,722	0,857	0,985	1,000	0,874	0,905	0,000	0,737	0,900	0,864	0,864	1,000	**0,796**
STRP	0,831	0,770	0,871	0,957	0,837	0,902	0,779	0,843	0,929	0,994	1,000	0,876	0,910	0,169	0,747	0,962	0,948	0,944	1,000	**0,856**
SWDD	-	1,000	-	1,000	1,000	1,000	0,500	1,000	1,000	-	1,000	1,000	1,000	0,500	0,364	0,909	0,000	0,682	1,000	**0,810**
OAAA	1,000	1,000	0,708	1,000	-	-	-	-	-	-	-	0,945	1,000	0,600	-	-	-	-	-	**0,893**
OAAN	0,914	0,846	0,913	0,906	-	-	-	1,000	0,250	1,000	0,250	0,750	0,833	0,000	0,182	0,636	0,727	0,909	-	**0,674**
OABA	1,000	1,000	0,623	1,000	-	-	-	-	-	-	-	1,000	1,000	0,667	-	-	-	-	-	**0,899**
OABN	0,933	0,864	0,929	1,000	-	-	-	0,970	0,333	1,000	0,333	0,778	0,833	0,000	-	-	-	-	-	**0,725**
OAEA	1,000	1,000	0,727	1,000	-	-	-	-	-	-	-	1,000	1,000	1,000	-	-	-	-	-	**0,961**
OALN	0,924	0,864	0,866	0,800	-	-	-	1,000	1,000	1,000	1,000	1,000	1,000	0,000	-	-	-	-	-	**0,859**
OARN	0,924	0,869	0,880	0,900	-	-	-	1,000	1,000	1,000	1,000	1,000	1,000	0,000	-	-	-	-	-	**0,870**
OASA	1,000	1,000	0,571	1,000	-	-	-	-	-	-	-	1,000	1,000	0,500	-	-	-	-	-	**0,867**
OASN	0,909	0,826	0,962	0,900	-	-	-	1,000	0,000	1,000	0,000	0,576	0,779	0,000	-	-	-	-	-	**0,632**
OBAA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OBAN	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,455	1,000	1,000	1,000	-	**0,891**
OBBA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			-	-
OBBN	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,455	1,000	1,000	1,000	-	**0,891**
OBEA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OBLN	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,455	1,000	1,000	1,000	-	**0,891**
OBNG	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,455	1,000	1,000	1,000	-	**0,891**
OBRN	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,364	1,000	1,000	1,000	-	**0,873**
OBSA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OBSN	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,455	1,000	1,000	1,000	-	**0,891**
OCNG	1,000	1,000	1,000	1,000	0,941	1,000	0,838	0,949	1,000	1,000	1,000	1,000	1,000	0,071	0,686	1,000	1,000	1,000	1,000	**0,920**
OCOR	-	-	-	-	-	-	-	-	1,000	-	1,000	1,000	1,000	1,000	-	1,000	1,000	-	-	**1,000**

Continua na Próxima Página

Continuação da Tabela 39

	Cal			Checkeq	Comm							Look			Uniq					
Operador	**main + cal**	**main + pstr**	**cal + jan1**	**main + check**	**main + openfil**	**main + copy**	**main + compare**	**main + wr**	**main + rd**	**copy + wr**	**copy + rd**	**main + getline**	**main + canon**	**main + usage**	**main + printe**	**main + gline**	**main + pline**	**main + equal**	**equal + skip**	**Média**
OEAA	0,866	0,797	0,850	0,907	1,000	1,000	1,000	1,000	1,000	-	1,000	0,902	0,936	0,000	0,292	0,898	0,919	0,862	1,000	**0,846**
OEBA	0,883	0,818	0,862	0,889	1,000	1,000	1,000	1,000	1,000	-	1,000	0,885	0,932	0,000	0,210	0,662	0,687	0,559	1,000	**0,799**
OESA	0,870	0,800	0,871	0,923	1,000	1,000	1,000	1,000	1,000	-	1,000	0,884	0,933	0,038	0,255	0,974	1,000	0,964	1,000	**0,862**
Oido	1,000	1,000	1,000	1,000	0,636	0,705	0,813	0,625	0,786	-	0,833	1,000	1,000	0,000	0,576	1,000	1,000	0,991	1,000	**0,831**
OIPM	1,000	1,000	1,000	-	-	-	-	-	1,000	-	1,000	1,000	1,000	1,000	-	1,000	1,000	-	-	**1,000**
OLAN	0,769	0,667	1,000	0,952	0,864	0,773	0,523	0,750	0,852	-	0,750	1,000	0,977	0,000	0,727	1,000	1,000	1,000	1,000	**0,811**
OLBN	0,400	0,000	1,000	0,909	0,886	0,955	0,545	1,000	0,750	-	0,500	1,000	0,960	0,000	0,758	1,000	1,000	1,000	1,000	**0,759**
OLLN	0,250	0,000	1,000	0,810	0,773	0,909	0,091	1,000	1,000	-	1,000	1,000	1,000	0,000	0,909	1,000	1,000	1,000	1,000	**0,763**
OLNG	1,000	1,000	0,697	1,000	1,000	1,000	1,000	1,000	1,000	-	1,000	1,000	1,000	0,000	1,000	0,667	0,667	0,667	1,000	**0,872**
OLRN	0,714	0,600	0,848	0,983	0,901	0,818	0,618	0,800	0,802	-	0,667	1,000	0,948	0,000	0,773	1,000	1,000	1,000	1,000	**0,804**
OLSN	0,250	0,000	1,000	0,636	0,841	0,591	0,500	0,500	0,455	-	0,000	1,000	1,000	0,000	0,682	1,000	1,000	1,000	1,000	**0,636**
ORAN	0,939	0,905	0,870	0,693	0,770	0,852	0,653	0,756	0,915	1,000	0,958	0,945	0,890	0,026	0,770	1,000	1,000	1,000	1,000	**0,839**
ORBN	1,000	0,968	0,877	0,641	0,692	0,805	0,549	0,695	0,945	1,000	1,000	0,966	0,877	0,000	0,835	1,000	1,000	1,000	1,000	**0,834**
ORLN	1,000	1,000	0,886	0,521	0,615	0,759	0,489	0,655	0,913	1,000	1,000	0,934	0,847	0,000	0,788	0,833	0,750	0,769	1,000	**0,777**
ORRN	0,866	0,774	0,844	0,797	0,805	0,880	0,672	0,764	0,922	1,000	0,909	0,916	0,904	0,079	0,818	0,854	0,778	0,806	0,950	**0,807**
ORSN	0,891	0,854	0,767	0,625	0,667	0,701	0,471	0,600	0,769	1,000	0,900	0,957	1,000	0,056	0,870	0,917	0,857	0,857	1,000	**0,777**
OSAA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OSAN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OSBA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OSBN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OSEA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OSLN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OSRN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OSSN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OSSA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Varr	0,944	0,906	0,628	1,000	0,777	0,996	0,701	0,758	1,000	-	-	1,000	1,000	0,000	0,446	1,000	1,000	0,995	1,000	**0,832**
VDTR	0,890	0,862	0,966	0,894	0,835	0,799	0,804	0,857	0,891	0,990	1,000	0,883	0,928	0,135	0,859	0,812	0,858	0,835	1,000	**0,847**
Vprr	1,000	1,000	1,000	-	0,936	0,898	0,951	0,857	0,980	1,000	1,000	0,875	1,000	0,000	0,740	0,700	0,700	0,700	1,000	**0,852**
VSCR	0,091	0,000	-	-	-	-	-	-	1,000	-	1,000	1,000	1,000	0,000	-	1,000	1,000	-	-	**0,677**
Vsrr	0,871	0,756	0,819	0,979	0,945	0,940	0,935	0,935	0,931	1,000	0,892	0,868	0,932	0,070	0,588	0,979	0,985	0,976	1,000	**0,863**
Vtrr	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
VTWD	0,837	0,723	0,746	0,978	0,786	0,742	0,784	0,808	0,777	1,000	0,800	0,886	0,915	0,081	0,720	0,875	0,900	0,857	1,000	**0,801**
Cccr	0,855	0,750	0,763	0,989	0,752	0,852	0,689	0,708	0,867	-	0,778	0,862	0,894	0,017	0,558	0,736	0,739	0,727	1,000	**0,752**
Ccsr	0,834	0,726	0,860	0,967	0,775	0,795	0,753	0,821	0,806	1,000	0,800	0,901	0,948	0,082	0,692	0,857	0,880	0,833	1,000	**0,807**
CRCR	0,888	0,829	0,854	0,987	0,914	0,874	0,893	0,891	0,895	0,995	0,824	0,940	0,983	0,083	0,689	0,923	0,933	0,917	1,000	**0,858**
Total	**0,867**	**0,782**	**0,834**	**0,926**	**0,823**	**0,881**	**0,771**	**0,823**	**0,903**	**0,996**	**0,899**	**0,897**	**0,935**	**0,075**	**0,638**	**0,911**	**0,908**	**0,898**	**0,994**	**0,830**

Com base nessas considerações, uma possível seqüência de aplicação dos operadores de mutação de interface, procurando complementar o conjunto de casos de teste AM-adequado, é obtida através da ordenação dos operadores da Tabela 38 de forma decrescente a partir da média de cobertura, ou seja, dos operadores que apresentam maior *strength* em relação a conjuntos AM-adequados para os de menor *strength*. Tal seqüência é apresentada na Tabela 40.

Tabela 40 - Ordenação dos Operadores de Mutação de Interface

Operador	Média	Operador	Média	Operador	Média
I-IndVarRepExt	0,767	I-IndVarRepReq	0,859	I-DirVarRepLoc	0,936
I-IndVarRepGlo	0,792	I-DirVarAriNeg	0,860	I-CovAllEdg	0,939
I-DirVarRepGlo	0,832	I-IndVarIncDec	0,865	II-ArgAriNeg	0,939
I-DirVarRepExt	0,839	I-IndVarAriNeg	0,868	II-ArgDel	0,956
I-IndVarRepPar	0,841	I-IndVarRepCon	0,898	II-ArgIncDec	0,958
I-DirVarBitNeg	0,844	II-ArgStcAli	0,900	II-FunCalDel	0,967
I-DirVarRepPar	0,845	I-DirVarIncDec	0,901	II-ArgBitNeg	0,970
I-IndVarBitNeg	0,848	I-RetStaDel	0,914	I-RetStaRep	0,978
I-DirVarRepCon	0,853	I-DirVarLogNeg	0,918	II-ArgRepReq	0,982
I-DirVarRepReq	0,853	I-CovAllNod	0,922	II-ArgStcDif	0,986
I-IndVarLogNeg	0,857	I-IndVarRepLoc	0,934	II-ArgLogNeg	0,990

A partir dessa ordenação, o mesmo procedimento utilizado na Seção 3.2.1 para determinar uma estratégia incremental dos operadores de mutação de interface é repetido. A Tabela 41 apresenta o resultado da aplicação dos operadores da Tabela 40 de forma incremental, a partir de um conjunto AM-adequado.

Analisando-se a Tabela 41 observa-se que, em média, um conjunto de casos de teste AM-adequado determina um escore de mutação de 0,935 em relação ao critério Mutação de Interface. Em seguida, obtendo-se um conjunto de casos de teste adequado ao operador I-IndVarRepExt ocorre um incremento no escore de mutação da ordem de 0,032, garantindo um grau de adequação em relação ao critério Mutação de Interface de 0,967 e uma redução de custo de 96,36%. Com a aplicação do segundo operador (I-IndVarRepGlo) o escore obtido é da ordem de 0,977 e a redução de custo de 94,13%. Desse modo, aplicando-se 24 operadores de mutação de interface é possível complementar o conjunto AM-adequado e torná-lo também *IM*-adequado.

Observa-se que entre os 24 operadores de mutação, 7 apresentaram incremento zero, indicando que estão sendo incluídos pelos demais operadores de mutação de interface. Desse modo, da mesma forma como apresentado na Seção 3.2.1, os operadores são reordenados a partir do incremento proporcionado, fazendo com que os operadores que proporcionaram incremento zero deixem de ser considerados. A Tabela 42 apresenta os resultados obtidos após a primeira ordenação desses operadores.

Tabela 41 – Estratégia de Aplicação dos Operadores de Mutação de Interface a Partir de um Conjunto AM-adequado: Primeira Iteração

Op[18]	**EAn**	**EAt**	**Inc**	**CO**	**CC**	**RC**
T AM-adequado	0,00000	0,93532	0,93532	-	-	-
I-IndVarRepExt	0,93532	0,96686	0,03154	511	511	96,35625
I-IndVarRepGlo	0,96686	0,97660	0,00974	312	823	94,13149
I-DirVarRepGlo	0,97660	0,98004	0,00344	170	993	92,91928
I-DirVarRepExt	0,98004	0,99523	0,01519	494	1487	89,39675
I-IndVarRepPar	0,99523	0,99635	0,00112	672	2159	84,60496
I-DirVarBitNeg	0,99635	0,99637	0,00002	154	2313	83,50685
I-DirVarRepPar	0,99637	0,99667	0,00030	385	2698	80,76155
I-IndVarBitNeg	0,99667	0,99798	0,00131	306	3004	78,57958
I-DirVarRepCon	0,99798	0,99804	0,00006	1380	4384	68,73930
I-DirVarRepReq	0,99804	0,99804	0,00000	770	5154	63,24872
I-IndVarLogNeg	0,99804	0,99804	0,00000	306	5460	61,06674
I-IndVarRepReq	0,99804	0,99825	0,00021	1391	6851	51,14803
I-DirVarAriNeg	0,99825	0,99825	0,00000	154	7005	50,04991
I-IndVarIncDec	0,99825	0,99829	0,00004	684	7689	45,17256
I-IndVarAriNeg	0,99829	0,99829	0,00000	306	7995	42,99059
I-IndVarRepCon	0,99829	0,99927	0,00098	2555	10550	24,77182
II-ArgStcAli	0,99927	0,99927	0,00000	21	10571	24,62208
I-DirVarIncDec	0,99927	0,99928	0,00001	536	11107	20,80006
I-RetStaDel	0,99928	0,99928	0,00000	65	11172	20,33657
I-DirVarLogNeg	0,99928	0,99928	0,00000	154	11326	19,23845
I-CovAllNod	0,99928	0,99947	0,00019	438	11764	16,11523
I-IndVarRepLoc	0,99947	0,99998	0,00051	618	12382	11,70850
I-DirVarRepLoc	0,99998	0,99999	0,00001	488	12870	8,22875
I-CovAllEdg	0,99999	1,00000	0,00001	390	13260	5,44780
...	...	...	...	...	...	...
II-ArgLogNeg	1,00000	1,00000	0,00000	26	14024	0,00000

Na segunda iteração do processo, observa-se que o número de operadores de mutação de interface necessários para se obter um conjunto *IM*-adequado caiu de 24 para 17 e a redução de custo obtida é de aproximadamente 18,11%. Visto que os operadores ainda não se encontram ordenados pelo incremento, o processo deve ser repetido. No caso, para que todos os operadores permanecessem ordenados de forma decrescente pelo incremento foram necessárias mais duas iterações; o resultado final é apresentado na Tabela 43.

A estratégia incremental definida na Tabela 43 demonstra que com a utilização de 16 operadores de mutação de interface é possível complementar o conjunto AM-adequado e torná-lo também *IM*-adequado com uma redução de custo, em termos do número de mutantes gerados, superior a 27%. Além disso, com a utilização de apenas 3 operadores de mutação obtém-se um conjunto de casos de teste que apresenta um grau de adequação em relação ao critério Mutação de Interface superior a 99% e uma redução de custo de 90,61%.

Tabela 42 – Estratégia de Aplicação dos Operadores de Mutação de Interface a Partir de um Conjunto AM-adequado: Segunda Iteração

Op	EAn	EAt	Inc	CO	CC	RC
T AM-adequado	0,00000	0,93532	0,93532	-	-	-
I-IndVarRepExt	0,93532	0,96686	0,03154	511	511	96,35625
I-DirVarRepExt	0,96686	0,98579	0,01893	494	1005	92,83371
I-IndVarRepGlo	0,98579	0,99285	0,00706	312	1317	90,60896
I-DirVarRepGlo	0,99285	0,99523	0,00238	170	1487	89,39675
I-IndVarBitNeg	0,99523	0,99722	0,00199	306	1793	87,21477
I-IndVarRepPar	0,99722	0,99766	0,00044	672	2465	82,42299
I-IndVarRepCon	0,99766	0,99878	0,00112	2555	5020	64,20422
I-IndVarRepLoc	0,99878	0,99935	0,00057	618	5638	59,79749
I-DirVarRepPar	0,99935	0,99965	0,00030	385	6023	57,05220
I-IndVarRepReq	0,99965	0,99972	0,00007	1391	7414	47,13349
I-CovAllNod	0,99972	0,99992	0,00020	438	7852	44,01027
I-DirVarRepCon	0,99992	0,99992	0,00000	1380	9232	34,16999
I-IndVarIncDec	0,99992	0,99996	0,00004	684	9916	29,29264
I-DirVarBitNeg	0,99996	0,99997	0,00001	154	10070	28,19452
I-CovAllEdg	0,99997	0,99998	0,00001	390	10460	25,41358
I-DirVarIncDec	0,99998	0,99999	0,00001	536	10996	21,59156
I-DirVarRepLoc	0,99999	1,00000	0,00001	488	11484	18,11181
...	...	...	...	...	...	...
II-FunCalDel	1,00000	1,00000	0,00000	254	14024	0,00000

Tabela 43 – Estratégia de Aplicação dos Operadores de Mutação de Interface a Partir de um Conjunto AM-adequado: Última Iteração

Op	EAn	EAt	Inc	CO	CC	RC
T AM-adequado	0,00000	0,93532	0,93532	-	-	-
I-IndVarRepExt	0,93532	0,96686	0,03154	511	511	96,35625
I-DirVarRepExt	0,96686	0,98579	0,01893	494	1005	92,83371
I-IndVarRepGlo	0,98579	0,99285	0,00706	312	1317	90,60896
I-DirVarRepGlo	0,99285	0,99523	0,00238	170	1487	89,39675
I-IndVarBitNeg	0,99523	0,99722	0,00199	306	1793	87,21477
I-IndVarRepCon	0,99722	0,99849	0,00127	2555	4348	68,99601
I-IndVarRepLoc	0,99849	0,99906	0,00057	618	4966	64,58928
I-DirVarRepPar	0,99906	0,99939	0,00033	385	5351	61,84398
I-IndVarRepPar	0,99939	0,99965	0,00026	672	6023	57,05220
I-CovAllNod	0,99965	0,99984	0,00019	438	6461	53,92898
I-IndVarRepReq	0,99984	0,99992	0,00008	1391	7852	44,01027
I-IndVarIncDec	0,99992	0,99996	0,00004	684	8536	39,13292
I-DirVarBitNeg	0,99996	0,99997	0,00001	154	8690	38,03480
I-DirVarIncDec	0,99997	0,99998	0,00001	536	9226	34,21278
I-DirVarRepLoc	0,99998	0,99999	0,00001	488	9714	30,73303
I-CovAllEdg	0,99999	1,00000	0,00001	390	10104	27,95208
...	...	...	...	...	...	...
II-FunCalDel	1,00000	1,00000	0,00000	254	14024	0,00000

Outro ponto importante a ser observado é que comparando esses 16 operadores com os 16 operadores que se mostraram comuns às três estratégias incrementais definidas no Capítulo 3 (Figura 7), tem-se que os conjuntos de operadores são praticamente os mesmos. A diferença é

[18] As abreviações usadas nesta tabela foram definidas na Seção 3.2.1 e representam: Op – **Op**erador; EAn – **E**score de mutação **An**terior; EAt – **E**score de mutação **At**ual; Inc – **Inc**remento; CO – **C**usto do **O**perador; CC – **C**usto **C**umulativo e RC – **R**edução de **C**usto.

que, naquele conjunto havia o operador II-FunCalDel e neste o operador selecionado foi I-DirVarBitNeg; os outros 15 operadores são os mesmos. Tais resultados demonstram que, independentemente da qualidade do conjunto de casos de teste *T* em nível de unidade, a aplicação do critério Mutação de Interface faz-se necessária para garantir uma completa cobertura dos requisitos de teste exigidos no teste de integração, refletidos nos operadores de mutação. O ponto que se coloca é como aplicá-lo de modo a obter a melhor relação custo/benefício.

Além disso, observa-se que 7 dos 8 operadores essenciais de mutação de interface, definidos no Capítulo 3 (Quadro 6), aparecem entre os 16 operadores, demonstrando que os operadores de interface do conjunto essencial apresentam um alto *strength* em relação a conjuntos de casos de teste *AM*-adequados. O único operador do conjunto essencial que não está presente entre os 16 é II-ArgAriNeg, incluído pelos demais devido à ordem de aplicação estabelecida.

4.2.3 – Estratégia Incremental de Aplicação dos Operadores de Unidade a Partir de Conjuntos *IM*-adequados

Da mesma forma que na seção anterior, a média geral de cobertura dos conjuntos *IM*-adequados em relação aos operadores de mutação de unidade, apresentada na Tabela 39, é utilizada para se estabelecer uma estratégia incremental de aplicação desses operadores, ou seja, deseja-se saber como melhorar o conjunto *IM*-adequado de modo a torná-lo AM-adequado.

Embora possa parecer estranho querer avaliar a adequação de um critério destinado ao teste de unidade após já ter sido aplicado o teste de integração, o que se deseja é verificar quais operadores de mutação de unidade são incluídos por conjuntos *IM*-adequados. Desse modo, considerando uma estratégia incremental de teste envolvendo as fases de unidade e integração e, sabendo-se que no teste de integração será utilizado o critério Mutação de Interface, os operadores de unidade que são incluídos pelos conjuntos *IM*-adequados não precisariam necessariamente serem aplicados, visto que os requisitos de teste requeridos por tais operadores seriam cobertos posteriormente no teste de integração.

Ordenando os operadores da Tabela 39 de forma decrescente, de acordo com a média de inclusão, pode-se estabelecer uma estratégia inicial de aplicação dos operadores de mutação de unidade, começando pelos operadores de maior *strength* em relação a conjuntos *IM*-adequados; tal ordenação é apresentada na Tabela 44. Os operadores SGLR, OBAA, OBBA, OBEA, OBSA,

OSAA, OSAN, OSBA, OSBN, OSEA, OSLN, OSRN, OSSN, OSSA e Vtrr foram desconsiderados pois, em nenhum dos programas utilizados esses operadores geraram mutantes.

A partir de um conjunto de casos de teste *IM*-adequado, e considerando a ordem dos operadores estabelecida na Tabela 44, na Tabela 45 são apresentados os incrementos no escore de mutação proporcionados após a aplicação de cada operador. Na primeira iteração, 33 operadores foram necessários até que o conjunto *IM*-adequado fosse complementado para se tornar AM-adequado, resultando em uma redução de custo de aproximadamente 10%.

Tabela 44 – Ordenação dos Operadores de Mutação de Unidade

Operador	Média	Operador	Média	Operador	Média	Operador	Média
OASN	0,632	SMTC	0,801	STRP	0,856	OBLN	0,891
OLSN	0,636	OLRN	0,804	CRCR	0,858	OBNG	0,891
OAAN	0,674	Ccsr	0,807	OALN	0,859	OBSN	0,891
VSCR	0,677	ORRN	0,807	OESA	0,862	OAAA	0,893
OABN	0,725	SWDD	0,810	Vsrr	0,863	OABA	0,899
Cccr	0,752	OLAN	0,811	OASA	0,867	OCNG	0,920
OLBN	0,759	SSDL	0,820	OARN	0,870	SMTT	0,930
OLLN	0,763	Oido	0,831	SRSR	0,872	SDWD	0,935
ORSN	0,777	Varr	0,832	OLNG	0,872	OAEA	0,961
ORLN	0,777	ORBN	0,834	OBRN	0,873	SBRn	1,000
SSWM	0,787	ORAN	0,839	SBRC	0,876	SCRB	1,000
STRI	0,796	OEAA	0,846	SMVB	0,888	SCRn	1,000
OEBA	0,799	VDTR	0,847	OBAN	0,891	OCOR	1,000
VTWD	0,801	Vprr	0,852	OBBN	0,891	OIPM	1,000

Para se chegar à estratégia final de aplicação dos operadores de mutação de unidade foi necessário repetir o processo de ordenação mais 4 vezes; o resultado final é apresentado na Tabela 46. Considerando tal tabela, observa-se que o número de operadores necessários para se obter o conjunto AM-adequado passou de 33 para 13 e, em média, a utilização desses 13 operadores proporciona uma redução de custo de 50% no número de mutantes gerados. Com isso, pode-se concluir que mesmo que no teste de integração seja utilizado o critério Mutação de Interface, é interessante utilizar tais operadores no teste de unidade visto que, mesmo em posse de um conjunto *IM*-adequado, mutantes desses operadores permanecem vivos, evidenciando o aspecto complementar desses dois critérios de teste, como era de se esperar.

4.3 – Essencial Análise de Mutantes ´ Essencial Mutação de Interface

Como descrito na Seção 3.3.2, Barbosa [BAR98a] desenvolveu o procedimento **Essencial** para a determinação de um conjunto essencial de operadores de mutação. A partir da aplicação do procedimento **Essencial**, um conjunto essencial de operadores de mutação de unidade para a linguagem C foi obtido [BAR98a, BAR98b] (Quadro 7).

Tabela 45 – Estratégia de Aplicação dos Operadores de Mutação de Unidade a Partir de um Conjunto *IM*-adequado: Primeira Iteração

Op	EAn	EAt	Inc	CO	CC	RC
T IM-adequado	0,00000	0,93710	0,93710	-	-	-
OASN	0,93710	0,95733	0,02023	82	82	99,36107
OLSN	0,95733	0,97327	0,01594	54	136	98,94031
OAAN	0,97327	0,97433	0,00106	182	318	97,52221
VSCR	0,97433	0,97433	0,00000	43	361	97,18716
OABN	0,97433	0,97468	0,00035	123	484	96,22877
Cccr	0,97468	0,99089	0,01621	1676	2160	83,16971
OLBN	0,99089	0,99200	0,00111	81	2241	82,53857
OLLN	0,99200	0,99200	0,00000	27	2268	82,32819
ORSN	0,99200	0,99463	0,00263	196	2464	80,80100
ORLN	0,99463	0,99488	0,00025	206	2670	79,19589
SSWM	0,99488	0,99488	0,00000	23	2693	79,01667
STRI	0,99488	0,99549	0,00061	160	2853	77,76999
OEBA	0,99549	0,99726	0,00177	255	3108	75,78308
VTWD	0,99726	0,99759	0,00033	422	3530	72,49494
SMTC	0,99759	0,99779	0,00020	33	3563	72,23781
OLRN	0,99779	0,99795	0,00016	162	3725	70,97553
Ccsr	0,99795	0,99798	0,00003	1393	5118	60,12155
ORRN	0,99798	0,99829	0,00031	515	5633	56,10877
SWDD	0,99829	0,99829	0,00000	18	5651	55,96852
OLAN	0,99829	0,99829	0,00000	135	5786	54,91663
SSDL	0,99829	0,99858	0,00029	446	6232	51,44148
Oido	0,99858	0,99859	0,00001	91	6323	50,73243
Varr	0,99859	0,99859	0,00000	169	6492	49,41561
ORBN	0,99859	0,99873	0,00014	294	6786	47,12482
ORAN	0,99873	0,99891	0,00018	490	7276	43,30684
OEAA	0,99891	0,99902	0,00011	425	7701	39,99532
VDTR	0,99902	0,99987	0,00085	633	8334	35,06311
Vprr	0,99987	0,99987	0,00000	105	8439	34,24497
STRP	0,99987	0,99987	0,00000	448	8887	30,75425
CRCR	0,99987	0,99987	0,00000	751	9638	24,90260
OALN	0,99987	0,99987	0,00000	90	9728	24,20134
OESA	0,99987	0,99987	0,00000	170	9898	22,87673
Vsrr	0,99987	1,00000	0,00013	1621	11519	10,24622
...	...	...	...	...	...	...
OIPM	1,00000	1,00000	0,00000	10	12834	0,00000

Na Seção 3.3.2.2, o procedimento **Essencial** foi aplicado no contexto do critério Mutação de Interface resultando no conjunto essencial de operadores de mutação desse critério (Quadro 8). Ressalta-se que a informação presente no Quadro 8 é a mesma apresentada no Quadro 6 da Seção 3.3.2.2, sendo duplicada para facilitar a apresentação dos dados.

Considerando tais conjuntos de operadores de mutação, nesta seção são repetidas as mesmas comparações realizadas na Seção 4.2. Entretanto, ao invés de se utilizarem conjuntos de casos de teste AM-adequados e *IM*-adequados, serão utilizados os conjuntos de casos de teste adequados aos conjuntos essenciais de operadores de mutação: essencial-AM-adequados e essencial-*IM*-adequados.

Tabela 46 – Estratégia de Aplicação dos Operadores de Mutação de Unidade a Partir de um Conjunto *IM*-adequado: Última Iteração

Op	EAn	EAt	Inc	CO	CC	RC
T IM-adequado	0,00000	0,93710	0,93710	-	-	-
Cccr	0,93710	0,98858	0,05148	1676	1676	86,94094
ORSN	0,98858	0,99338	0,00480	196	1872	85,41374
OEBA	0,99338	0,99554	0,00216	255	2127	83,42683
VDTR	0,99554	0,99729	0,00175	633	2760	78,49462
VTWD	0,99729	0,99787	0,00058	422	3182	75,20648
OLBN	0,99787	0,99839	0,00052	81	3263	74,57535
OASN	0,99839	0,99882	0,00043	82	3345	73,93642
ORRN	0,99882	0,99912	0,00030	515	3860	69,92364
SSDL	0,99912	0,99941	0,00029	446	4306	66,44850
Vsrr	0,99941	0,99965	0,00024	1621	5927	53,81798
SMTC	0,99965	0,99985	0,00020	33	5960	53,56085
ORBN	0,99985	0,99999	0,00014	294	6254	51,27006
OLRN	0,99999	1,00000	0,00001	162	6416	50,00779
...	...	...	...	...	...	...
OIPM	1,00000	1,00000	0,00000	10	12834	0,00000

Quadro 7 – Operadores Essenciais de Unidade da Linguagem C [BAR98a]

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.

Quadro 8 – Operadores Essenciais de Interface da Linguagem C

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.

4.3.1 – Avaliação do *Strength* dos Critérios em Relação a Conjuntos Essencial-Adequados

Primeiramente, são avaliados o *strength* do critério Mutação de Interface em relação a conjuntos de casos de teste essencial-AM-adequados e o *strength* do critério Análise de Mutantes em relação a conjuntos essencial-*IM*-adequados. A Tabela 47 sintetiza os resultados obtidos.

Tabela 47 – *Strength* dos Critérios Análise de Mutantes e Mutação de Interface em Relação a Conjuntos Essencial-Adequados

Programas	$T_{\text{essencial-AM-adequado}}$ **Mutação de Interface**	$T_{\text{essencial-}IM\text{-adequado}}$ **Análise de Mutantes**
Cal	0,92743	0,87736
Checkeq	0,95796	0,92580
Comm	0,90994	0,93501
Look	0,89295	0,95316
Uniq	0,90951	0,98261
Média	0,91956	0,93479
Desvio Padrão	0,02469	0,03874

Em geral, comparando-se com os resultados apresentados na Tabela 31, observa-se que conjuntos essencial-*IM*-adequados continuam mantendo, em média, um grau de adequação em relação ao critério Análise de Mutantes da ordem de 93,48%. Já os conjuntos essencial-AM-adequados estão cobrindo, em média, 91,96% dos requisitos de teste exigidos pelo critério Mutação de Interface, 1,86% a menos do que o obtido por conjuntos AM-adequados. Tais resultados dão evidências de que os conjuntos essenciais de operadores de mutação obtidos permitem selecionar praticamente os mesmos conjuntos de casos de teste que seriam selecionados com a utilização de todos os operadores, proporcionando significativa redução no custo de aplicação dos critérios. Com isso, o que se espera é avaliar se essas pequenas diferenças no grau de cobertura terão algum reflexo nas estratégias incrementais de aplicação dos operadores definidas nas seções anteriores. Assim sendo, da mesma forma como apresentado na Seção 4.2.1, a coleta de dados foi detalhada de modo a permitir análise de cobertura de cada operador, ou seja:

- dadas duas unidades f e g, verificou-se o quanto o conjunto essencial-f-AM-adequado $\cup$ essencial-g-AM-adequado (Análise de Mutantes) era adequado a cada um dos 33 operadores de mutação de interface aplicados na conexão f-g (Mutação de Interface); e
- dada a conexão f-g, verificou-se o quanto o conjunto essencial-f-g-*IM*-adequado (Mutação de Interface) era adequado a cada um dos 71 operadores de mutação de unidade aplicados às unidades f e g (Análise de Mutantes).

Os dados a respeito da cobertura que os conjuntos essencial-AM-adequados determinam em relação aos operadores de mutação de interface são apresentados na Tabela 48. Na Tabela 49 tem-se a cobertura que os conjuntos essencial-*IM*-adequados determinam em relação aos operadores de mutação de unidade.

Tabela 48 – Média Geral de Cobertura de Conjuntos Essencial-AM-adequados em Relação ao Critério Mutação de Interface

	Cal			Checkeq	Comm							Look			Uniq					
Operador	**main-cal**	**main-pstr**	**cal-jan1**	**main-check**	**main-openfil**	**main-copy**	**main-compare**	**main-wr**	**main-rd**	**copy-wr**	**copy-rd**	**main-getline**	**main-canon**	**main-usage**	**main-printe**	**main-gline**	**main-pline**	**main-equal**	**equal-skip**	**Média**
I-CovAllEdg	0,937	1,000	1,000	0,931	0,818	0,879	1,000	1,000	0,972	0,761	0,932	0,992	1,000	-	-	0,977	0,684	0,992	0,949	**0,931**
I-CovAllNod	0,958	1,000	1,000	0,958	0,855	0,924	1,000	0,968	0,981	0,682	0,955	0,995	1,000	1,000	0,500	0,982	0,710	0,994	0,955	**0,916**
I-DirVarAriNeg	0,912	1,000	1,000	1,000	0,682	0,682	-	0,891	-	0,364	-	-	1,000	-	-	-	0,920	1,000	0,614	**0,839**
I-DirVarBitNeg	0,912	1,000	1,000	0,969	0,682	0,788	-	1,000	-	0,591	-	-	0,424	-	-	-	0,778	1,000	0,688	**0,819**
I-DirVarIncDec	0,978	1,000	1,000	1,000	0,932	0,806	-	0,902	0,922	0,600	0,909	1,000	0,859	-	0,500	-	0,778	1,000	0,814	**0,875**
I-DirVarLogNeg	0,912	1,000	1,000	1,000	0,841	0,924	-	0,949	-	0,682	-	-	0,970	-	-	-	0,813	1,000	0,722	**0,901**
I-DirVarRepCon	0,918	1,000	1,000	0,970	0,727	0,924	-	0,888	-	0,424	-	-	0,532	-	-	-	0,738	0,977	0,777	**0,823**
I-DirVarRepExt	0,916	1,000	-	-	0,741	0,809	1,000	0,848	0,969	0,182	0,909	1,000	0,750	-	0,500	0,986	0,657	0,976	0,799	**0,815**
I-DirVarRepGlo	1,000	-	-	-	-	-	-	0,901	-	0,509	-	-	0,621	-	-	-	0,758	1,000	0,718	**0,787**
I-DirVarRepLoc	0,914	1,000	1,000	0,961	-	-	1,000	-	0,953	-	0,909	1,000	0,549	-	-	1,000	-	0,909	0,739	**0,911**
I-DirVarRepPar	0,904	1,000	-	-	0,682	0,742	1,000	0,909	0,966	0,571	0,909	-	0,739	-	0,500	-	0,773	0,966	0,909	**0,826**
I-DirVarRepReq	0,912	1,000	1,000	0,975	0,699	0,805	-	0,938	-	0,550	-	-	0,574	-	-	-	0,747	0,991	0,702	**0,824**
I-IndVarAriNeg	0,900	1,000	1,000	1,000	0,788	1,000	1,000	0,667	0,958	0,091	0,970	1,000	0,944	-	0,500	1,000	0,864	1,000	0,511	**0,844**
I-IndVarBitNeg	0,909	1,000	1,000	0,950	0,664	0,924	1,000	0,727	0,923	0,000	0,945	0,990	0,907	-	0,500	1,000	0,758	0,985	0,644	**0,824**
I-IndVarIncDec	0,932	1,000	1,000	0,964	0,782	0,818	1,000	0,803	0,908	0,159	0,939	0,987	0,949	-	0,500	0,976	0,809	0,987	0,644	**0,842**
I-IndVarLogNeg	0,897	1,000	1,000	0,950	0,664	0,712	1,000	0,848	0,893	0,182	0,945	0,985	0,954	-	0,500	0,960	0,758	0,989	0,742	**0,832**
I-IndVarRepCon	0,904	1,000	1,000	0,945	0,664	-	1,000	-	0,911	-	0,941	0,988	0,957	-	0,500	0,976	0,758	0,989	0,693	**0,882**
I-IndVarRepExt	1,000	1,000	-	1,000	0,534	0,409	1,000	0,848	0,828	0,182	0,616	0,852	-	-	0,500	0,865	0,688	0,830	0,792	**0,746**
I-IndVarRepGlo	1,000	-	-	0,975	-	-	-	0,758	-	0,152	-	-	0,914	-	-	-	0,682	0,773	0,705	**0,745**
I-IndVarRepLoc	0,908	1,000	1,000	0,928	-	-	1,000	-	0,854	-	0,845	0,970	0,861	-	-	1,000	-	1,000	0,654	**0,918**
I-IndVarRepPar	0,915	1,000	1,000	-	0,697	0,712	1,000	0,807	0,927	0,121	0,909	0,992	0,946	-	0,500	0,691	0,800	0,949	0,955	**0,819**
I-IndVarRepReq	0,902	1,000	1,000	0,957	0,734	0,792	1,000	0,758	0,939	0,136	0,941	0,989	0,958	-	0,500	0,985	0,758	0,986	0,646	**0,832**
I-RetStaDel	-	-	1,000	-	1,000	-	1,000	1,000	0,960	0,455	0,955	0,970	-	-	-	0,955	0,534	0,977	1,000	**0,900**
I-RetStaRep	-	-	-	-	-	-	1,000	-	0,960	-	0,955	0,985	-	-	-	0,939	-	0,985	-	**0,971**
II-ArgAriNeg	1,000	1,000	1,000	-	-	0,682	-	1,000	-	0,909	-	-	-	-	-	-	-	-	-	**0,932**
II-ArgBitNeg	1,000	1,000	1,000	-	-	0,788	-	1,000	-	1,000	-	-	-	-	-	-	-	-	-	**0,965**
II-ArgDel	1,000	1,000	1,000	1,000	-	0,924	1,000	1,000	0,977	-	-	1,000	1,000	-	0,500	1,000	0,955	1,000	-	**0,954**
II-ArgIncDec	1,000	1,000	1,000	1,000	1,000	0,801	1,000	1,000	0,980	1,000	0,909	1,000	1,000	-	0,500	1,000	0,955	0,977	0,909	**0,946**
II-ArgLogNeg	1,000	1,000	1,000	-	-	0,924	-	1,000	-	1,000	-	-	-	-	-	-	-	-	-	**0,987**
II-ArgRepReq	1,000	-	1,000	-	-	-	-	-	-	0,945	-	-	-	-	-	-	-	-	-	**0,982**
II-ArgStcAli	1,000	-	-	-	-	-	1,000	-	-	-	-	-	1,000	-	0,500	-	-	1,000	-	**0,900**
II-ArgStcDif	1,000	1,000	-	-	-	0,924	-	1,000	0,989	-	-	-	-	-	-	-	-	-	-	**0,983**
II-FunCalDel	1,000	1,000	1,000	1,000	1,000	0,909	1,000	1,000	0,993	1,000	0,982	1,000	1,000	1,000	0,500	1,000	0,955	1,000	1,000	**0,965**
Total	**0,918**	**1,000**	**1,000**	**0,958**	**0,734**	**0,818**	**1,000**	**0,912**	**0,922**	**0,562**	**0,889**	**0,970**	**0,874**	**1,000**	**0,500**	**0,952**	**0,751**	**0,958**	**0,747**	**0,867**

Tabela 49 – Média Geral de Cobertura de Conjuntos Essencial-*IM*-adequados em Relação ao Critério Análise de Mutantes

	Cal			Checkeq	Comm							Look			Uniq					
Operador	**main + cal**	**main + pstr**	**cal + jan1**	**main + check**	**main + openfil**	**main + copy**	**main + compare**	**main + wr**	**main + rd**	**copy + wr**	**copy + rd**	**main + getline**	**main + canon**	**main + usage**	**main + printe**	**main + gline**	**main + pline**	**main + equal**	**equal + skip**	**Média**
SBRC	-	1,000	-	1,000	-	-	-	-	-	-	-	0,773	1,000	0,000	1,000	1,000	1,000	1,000	-	**0,864**
SBRn	-	-	-	1,000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**1,000**
SCRB	-	-	-	1,000	1,000	1,000	1,000	1,000	1,000	-	-	-	-	-	1,000	1,000	1,000	1,000	-	**1,000**
SCRn	-	-	-	1,000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**1,000**
SDWD	-	-	-	-	-	1,000	-	-	-	1,000	1,000	-	-	-	0,545	1,000	1,000	1,000	-	**0,935**
SGLR	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
SMTC	1,000	1,000	1,000	0,977	0,591	0,727	0,667	0,500	0,697	1,000	1,000	0,800	0,800	0,000	0,636	0,955	0,970	1,000	0,818	**0,797**
SMTT	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,000	0,667	1,000	1,000	1,000	0,982	**0,929**
SMVB	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,000	0,545	0,697	0,750	0,750	1,000	**0,881**
SRSR	0,859	0,804	0,865	0,971	0,833	0,943	0,833	0,827	0,961	1,000	1,000	0,918	0,934	0,186	0,731	0,965	0,964	0,964	1,000	**0,872**
SSDL	0,812	0,754	0,850	0,941	0,786	0,855	0,731	0,767	0,884	0,983	0,941	0,765	0,861	0,161	0,642	0,933	0,933	0,909	1,000	**0,816**
SSWM	1,000	-	0,455	-	0,857	0,857	0,857	0,900	0,857	1,000	-	0,667	0,667	0,333	-	-	1,000	-	-	**0,787**
STRI	0,756	0,714	0,909	0,975	0,682	0,797	0,573	0,659	0,857	0,985	1,000	0,870	0,905	0,000	0,737	0,904	0,867	0,863	1,000	**0,792**
STRP	0,831	0,770	0,871	0,957	0,836	0,902	0,779	0,804	0,929	0,994	1,000	0,875	0,910	0,169	0,747	0,963	0,950	0,941	1,000	**0,854**
SWDD	-	1,000	-	1,000	1,000	1,000	0,500	1,000	1,000	-	1,000	1,000	1,000	0,500	0,364	0,909	0,000	0,682	1,000	**0,810**
OAAA	1,000	1,000	0,708	1,000	-	-	-	-	-	-	-	0,855	1,000	0,600	-	-	-	-	-	**0,880**
OAAN	0,914	0,846	0,913	0,909	-	-	-	1,000	0,250	1,000	0,250	0,750	0,833	0,000	0,182	0,545	0,636	0,909	-	**0,663**
OABA	1,000	1,000	0,623	1,000	-	-	-	-	-	-	-	1,000	1,000	0,667	-	-	-	-	-	**0,899**
OABN	0,933	0,864	0,929	1,000	-	-	-	0,970	0,333	1,000	0,333	0,778	0,833	0,000	-	-	-	-	-	**0,725**
OAEA	1,000	1,000	0,727	1,000	-	-	-	-	-	-	-	1,000	1,000	1,000	-	-	-	-	-	**0,961**
OALN	0,924	0,864	0,866	0,800	-	-	-	1,000	1,000	1,000	1,000	1,000	1,000	0,000	-	-	-	-	-	**0,859**
OARN	0,924	0,869	0,880	0,900	-	-	-	1,000	1,000	1,000	1,000	1,000	1,000	0,000	-	-	-	-	-	**0,870**
OASA	1,000	1,000	0,571	1,000	-	-	-	-	-	-	-	1,000	1,000	0,500	-	-	-	-	-	**0,867**
OASN	0,909	0,826	0,962	0,900	-	-	-	1,000	0,000	1,000	0,000	0,576	0,779	0,000	-	-	-	-	-	**0,632**
OBAA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OBAN	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,455	1,000	1,000	1,000	-	**0,891**
OBBA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OBBN	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,455	1,000	1,000	1,000	-	**0,891**
OBEA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OBLN	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,455	1,000	1,000	1,000	-	**0,891**
OBNG	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,455	1,000	1,000	1,000	-	**0,891**
OBRN	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,364	1,000	1,000	1,000	-	**0,873**
OBSA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OBSN	-	-	-	-	-	-	-	-	-	-	-	-	1,000	-	0,455	1,000	1,000	1,000	-	**0,891**
OCNG	1,000	1,000	1,000	1,000	0,941	1,000	0,838	0,909	1,000	1,000	1,000	1,000	1,000	0,065	0,686	1,000	1,000	1,000	1,000	**0,918**
OCOR	-	-	-	-	-	-	-	-	1,000	-	1,000	1,000	1,000	1,000	-	1,000	1,000	-	-	**1,000**

Continua na Próxima Página

Continuação da Tabela 49

	Cal			Checkeq	Comm							Look			Uniq					
Operador	**main + cal**	**main + pstr**	**cal + jan1**	**main + check**	**main + openfil**	**main + copy**	**main + compare**	**main + wr**	**main + rd**	**copy + wr**	**copy + rd**	**main + getline**	**main + canon**	**main + usage**	**main + printe**	**main + gline**	**main + pline**	**main + equal**	**equal + skip**	**Média**
OEAA	0,866	0,797	0,850	0,907	1,000	1,000	1,000	0,997	1,000	-	1,000	0,874	0,933	0,000	0,292	0,864	0,919	0,842	1,000	**0,841**
OEBA	0,883	0,818	0,862	0,889	1,000	1,000	1,000	0,992	1,000	-	1,000	0,864	0,928	0,000	0,210	0,646	0,687	0,524	1,000	**0,795**
OESA	0,870	0,800	0,871	0,923	1,000	1,000	1,000	1,000	1,000	-	1,000	0,844	0,928	0,035	0,255	0,935	1,000	0,927	1,000	**0,855**
Oido	1,000	1,000	1,000	1,000	0,636	0,693	0,813	0,625	0,792	-	0,833	1,000	1,000	0,000	0,576	1,000	1,000	0,973	1,000	**0,830**
OIPM	1,000	1,000	1,000	-	-	-	-	-	1,000	-	1,000	1,000	1,000	1,000	-	1,000	1,000	-	-	**1,000**
OLAN	0,769	0,667	1,000	0,952	0,830	0,773	0,523	0,750	0,818	-	0,750	1,000	0,966	0,000	0,727	1,000	1,000	1,000	1,000	**0,807**
OLBN	0,400	0,000	1,000	0,909	0,864	0,955	0,545	1,000	0,750	-	0,500	1,000	0,939	0,000	0,758	1,000	1,000	1,000	1,000	**0,757**
OLLN	0,250	0,000	1,000	0,810	0,727	0,909	0,091	1,000	1,000	-	1,000	1,000	1,000	0,000	0,909	1,000	1,000	1,000	1,000	**0,761**
OLNG	1,000	1,000	0,697	1,000	1,000	1,000	1,000	1,000	1,000	-	1,000	1,000	1,000	0,000	1,000	0,667	0,667	0,667	1,000	**0,872**
OLRN	0,714	0,600	0,848	0,983	0,876	0,818	0,618	0,800	0,777	-	0,667	1,000	0,926	0,000	0,773	1,000	1,000	1,000	1,000	**0,800**
OLSN	0,250	0,000	1,000	0,636	0,795	0,591	0,500	0,500	0,386	-	0,000	1,000	1,000	0,000	0,682	1,000	1,000	1,000	1,000	**0,630**
ORAN	0,929	0,906	0,870	0,693	0,742	0,852	0,653	0,674	0,912	1,000	0,958	0,928	0,861	0,024	0,770	1,000	1,000	1,000	1,000	**0,830**
ORBN	1,000	0,968	0,846	0,642	0,648	0,805	0,549	0,583	0,942	1,000	1,000	0,966	0,845	0,000	0,835	1,000	1,000	1,000	1,000	**0,822**
ORLN	1,000	1,000	0,886	0,521	0,567	0,759	0,489	0,527	0,913	1,000	1,000	0,913	0,807	0,000	0,785	0,838	0,756	0,775	1,000	**0,765**
ORRN	0,840	0,775	0,844	0,797	0,794	0,879	0,672	0,696	0,916	1,000	0,909	0,906	0,895	0,072	0,817	0,851	0,780	0,809	0,998	**0,803**
ORSN	0,891	0,854	0,767	0,625	0,616	0,701	0,471	0,600	0,769	1,000	0,900	0,957	1,000	0,051	0,870	0,917	0,857	0,857	1,000	**0,774**
OSAA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OSAN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OSBA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OSBN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OSEA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OSLN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OSRN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OSSN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
OSSA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
Varr	0,944	0,906	0,628	1,000	0,773	0,989	0,701	0,746	1,000	-	-	0,957	1,000	0,000	0,446	1,000	1,000	0,995	1,000	**0,829**
VDTR	0,890	0,862	0,966	0,893	0,824	0,799	0,804	0,857	0,877	0,990	1,000	0,881	0,907	0,135	0,859	0,870	0,841	0,835	1,000	**0,847**
Vprr	1,000	1,000	1,000	-	0,918	0,898	0,951	0,857	0,949	1,000	1,000	0,875	1,000	0,000	0,740	0,680	0,707	0,707	1,000	**0,849**
VSCR	0,091	0,000	-	-	-	-	-	-	1,000	-	1,000	1,000	1,000	0,000	-	1,000	1,000	-	-	**0,677**
Vsrr	0,871	0,756	0,819	0,978	0,941	0,940	0,935	0,935	0,928	1,000	0,892	0,845	0,930	0,070	0,588	0,960	0,984	0,971	1,000	**0,860**
Vtrr	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**-**
VTWD	0,829	0,728	0,746	0,978	0,768	0,742	0,784	0,808	0,754	1,000	0,800	0,872	0,904	0,081	0,720	0,875	0,900	0,857	0,995	**0,797**
Cccr	0,848	0,750	0,763	0,990	0,742	0,849	0,689	0,694	0,856	-	0,778	0,853	0,882	0,017	0,554	0,728	0,736	0,726	1,000	**0,747**
Ccsr	0,835	0,727	0,860	0,965	0,749	0,795	0,753	0,821	0,799	1,000	0,800	0,875	0,945	0,082	0,692	0,857	0,880	0,833	1,000	**0,804**
CRCR	0,888	0,829	0,854	0,985	0,896	0,874	0,893	0,891	0,871	0,995	0,824	0,940	0,982	0,083	0,689	0,923	0,933	0,917	1,000	**0,856**
Total	**0,783**	**0,762**	**0,816**	**0,864**	**0,794**	**0,872**	**0,730**	**0,767**	**0,904**	**1,000**	**1,000**	**0,863**	**0,909**	**0,067**	**0,631**	**0,878**	**0,887**	**0,879**	**1,000**	**0,811**

As tabelas 48 e 49 são utilizadas nas seções 4.3.2 e 4.3.3, respectivamente, para o estabelecimento da estratégia incremental de aplicação dos operadores de mutação de unidade e de interface a partir de conjuntos de casos de teste essencial-adequados.

4.3.2 – Estratégia Incremental de Aplicação dos Operadores de Interface a Partir de Conjuntos Essencial-AM-adequados

Através da ordenação dos operadores de mutação da Tabela 48 a partir da média, tem-se os operadores que apresentam os maiores *strength* em relação aos conjuntos essencial-AM-adequados. Na Tabela 50 é apresentada a ordenação obtida.

Tabela 50 – Ordenação dos Operadores de Mutação de Interface de Acordo com o *Strength* em Relação a Conjuntos Essencial-AM-adequados

Operador	Média	Operador	Média	Operador	Média
I-IndVarRepGlo	0,745	I-IndVarRepReq	0,832	I-IndVarRepLoc	0,918
I-IndVarRepExt	0,746	I-DirVarAriNeg	0,839	I-CovAllEdg	0,931
I-DirVarRepGlo	0,787	I-IndVarIncDec	0,842	II-ArgAriNeg	0,932
I-DirVarRepExt	0,815	I-IndVarAriNeg	0,844	II-ArgIncDec	0,946
I-IndVarRepPar	0,819	I-DirVarIncDec	0,875	II-ArgDel	0,954
I-DirVarBitNeg	0,819	I-IndVarRepCon	0,882	II-ArgBitNeg	0,965
I-DirVarRepCon	0,823	II-ArgStcAli	0,900	II-FunCalDel	0,965
I-IndVarBitNeg	0,824	I-RetStaDel	0,900	I-RetStaRep	0,971
I-DirVarRepReq	0,824	I-DirVarLogNeg	0,901	II-ArgRepReq	0,982
I-DirVarRepPar	0,826	I-DirVarRepLoc	0,911	II-ArgStcDif	0,983
I-IndVarLogNeg	0,832	I-CovAllNod	0,916	II-ArgLogNeg	0,987

Comparando tal ordenação com a que foi apresentada na Tabela 40, observa-se que, embora o grau de cobertura médio dos conjuntos essencial-AM-adequados seja praticamente o mesmo dos conjuntos AM-adequados, 15 operadores de mutação encontram-se em posições diferentes. Considerando a aplicação incremental dos operadores da Tabela 50, o incremento no escore de mutação que cada um proporciona é apresentado na Tabela 51.

Observa-se que na primeira iteração, 24 operadores de mutação são necessários para que, a partir de um conjunto de casos de teste essencial-AM-adequado, se obtenha um conjunto *IM*-adequado. A redução de custo proporcionada com a aplicação desses 24 operadores é da ordem de 5,45%.

Foram necessárias mais cinco iterações até que todos os operadores estivessem ordenados de forma decrescente a partir do incremento no escore de mutação. O resultado final obtido é apresentado na Tabela 52. Observa-se ao final que 16 operadores são necessários até que se obtenha um conjunto de casos de teste *IM*-adequado e a redução de custo proporcionada é de aproximadamente 27,95%. Além disso, com a aplicação dos três primeiros operadores (I-IndVarRepGlo, I-DirVarRepExt e I-IndVarRepExt) obtém-se um escore de mutação de

aproximadamente 0,991, com uma redução no custo de aplicação do critério Mutação de Interface de 90,61%.

Tabela 51 – Estratégia de Aplicação dos Operadores de Mutação de Interface a Partir de um Conjunto Essencial-AM-adequado: Primeira Iteração

Op	EAn	EAt	Inc	CO	CC	RC
T essencial-AM-adequado	0,00000	0,91956	0,91956	-	-	-
I-IndVarRepGlo	0,91956	0,96406	0,04450	312	312	97,77524
I-IndVarRepExt	0,96406	0,97540	0,01134	511	823	94,13149
I-DirVarRepGlo	0,97540	0,97979	0,00439	170	993	92,91928
I-DirVarRepExt	0,97979	0,99498	0,01519	494	1487	89,39675
I-IndVarRepPar	0,99498	0,99625	0,00127	672	2159	84,60496
I-DirVarBitNeg	0,99625	0,99626	0,00001	154	2313	83,50685
I-DirVarRepCon	0,99626	0,99636	0,00010	1380	3693	73,66657
I-IndVarBitNeg	0,99636	0,99771	0,00135	306	3999	71,48460
I-DirVarRepReq	0,99771	0,99775	0,00004	770	4769	65,99401
I-DirVarRepPar	0,99775	0,99800	0,00025	385	5154	63,24872
I-IndVarLogNeg	0,99800	0,99800	0,00000	306	5460	61,06674
I-IndVarRepReq	0,99800	0,99825	0,00025	1391	6851	51,14803
I-DirVarAriNeg	0,99825	0,99825	0,00000	154	7005	50,04991
I-IndVarIncDec	0,99825	0,99829	0,00004	684	7689	45,17256
I-IndVarAriNeg	0,99829	0,99829	0,00000	306	7995	42,99059
I-DirVarIncDec	0,99829	0,99829	0,00000	536	8531	39,16857
I-IndVarRepCon	0,99829	0,99928	0,00099	2555	11086	20,94980
II-ArgStcAli	0,99928	0,99928	0,00000	21	11107	20,80006
I-RetStaDel	0,99928	0,99928	0,00000	65	11172	20,33657
I-DirVarLogNeg	0,99928	0,99928	0,00000	154	11326	19,23845
I-DirVarRepLoc	0,99928	0,99954	0,00026	488	11814	15,75870
I-CovAllNod	0,99954	0,99974	0,00020	438	12252	12,63548
I-IndVarRepLoc	0,99974	0,99999	0,00025	618	12870	8,22875
I-CovAllEdg	0,99999	1,00000	0,00001	390	13260	5,44780
...	...	...	...	...	...	...
II-ArgLogNeg	1,00000	1,00000	0,00000	26	14024	0,00000

O conjunto de 16 operadores resultante é o mesmo que foi obtido na Seção 4.2.2 e, desse modo, a mesma consideração a respeito do conjunto essencial de operadores de mutação é válida, ou seja, 7 dos 8 operadores que compõem o conjunto essencial de interface são requeridos para obter um conjunto de caso de teste *IM*-adequado a partir de um conjunto essencial-AM-adequado.

4.3.3 – Estratégia Incremental de Aplicação dos Operadores de Unidade a Partir de Conjuntos Essencial-*IM*-adequados

Fazendo a análise inversa, considere os operadores de mutação de unidade da Tabela 49 e a média de cobertura que conjuntos de casos de teste essencial-*IM*-adequados determinam em relação a cada um desses operadores. A Tabela 53 apresenta tais operadores ordenados de forma decrescente pela média de cobertura, ou seja, dos operadores de unidade com maior $strength$ em relação a conjuntos de casos de teste essencial-*IM*-adequados para os de menor $strength$.

Tabela 52 – Estratégia de Aplicação dos Operadores de Mutação de Interface a Partir de um Conjunto Essencial-AM-adequado: Última Iteração

Op	EAn	EAt	Inc	CO	CC	RC
T essencial-AM-adequado	0,00000	0,91956	0,91956	-	-	-
I-IndVarRepGlo	0,91956	0,96406	0,04450	312	312	97,77524
I-DirVarRepExt	0,96406	0,98795	0,02389	494	806	94,25271
I-IndVarRepExt	0,98795	0,99190	0,00395	511	1317	90,60896
I-DirVarRepGlo	0,99190	0,99498	0,00308	170	1487	89,39675
I-IndVarBitNeg	0,99498	0,99713	0,00215	306	1793	87,21477
I-IndVarRepCon	0,99713	0,99839	0,00126	2555	4348	68,99601
I-IndVarRepLoc	0,99839	0,99900	0,00061	618	4966	64,58928
I-IndVarRepPar	0,99900	0,99931	0,00031	672	5638	59,79749
I-DirVarRepPar	0,99931	0,99961	0,00030	385	6023	57,05220
I-CovAllNod	0,99961	0,99981	0,00020	438	6461	53,92898
I-IndVarRepReq	0,99981	0,99992	0,00011	1391	7852	44,01027
I-IndVarIncDec	0,99992	0,99996	0,00004	684	8536	39,13292
I-DirVarBitNeg	0,99996	0,99997	0,00001	154	8690	38,03480
I-DirVarIncDec	0,99997	0,99998	0,00001	536	9226	34,21278
I-DirVarRepLoc	0,99998	0,99999	0,00001	488	9714	30,73303
I-CovAllEdg	0,99999	1,00000	0,00001	390	10104	27,95208
...	...	...	...	...	...	...
II-FunCalDel	1,00000	1,00000	0,00000	254	14024	0,00000

Tabela 53 – Ordenação dos Operadores de Mutação de Unidade de Acordo com o *Strength* em Relação a Conjuntos Essencial-*IM*-adequados

Operador	Média	Operador	Média	Operador	Média	Operador	Média
OLSN	0,630	VTWD	0,797	STRP	0,854	OBBN	0,891
OASN	0,632	OLRN	0,800	OESA	0,855	OBLN	0,891
OAAN	0,663	ORRN	0,803	CRCR	0,856	OBNG	0,891
VSCR	0,677	Ccsr	0,804	OALN	0,859	OBSN	0,891
OABN	0,725	OLAN	0,807	Vsrr	0,860	OABA	0,899
Cccr	0,747	SWDD	0,810	SBRC	0,864	OCNG	0,918
OLBN	0,757	SSDL	0,816	OASA	0,867	SMTT	0,929
OLLN	0,761	ORBN	0,822	OARN	0,870	SDWD	0,935
ORLN	0,765	Varr	0,829	SRSR	0,872	OAEA	0,961
ORSN	0,774	Oido	0,830	OLNG	0,872	SBRn	1,000
SSWM	0,787	ORAN	0,830	OBRN	0,873	SCRB	1,000
STRI	0,792	OEAA	0,841	OAAA	0,880	SCRn	1,000
OEBA	0,795	VDTR	0,847	SMVB	0,881	OCOR	1,000
SMTC	0,797	Vprr	0,849	OBAN	0,891	OIPM	1,000

Novamente, se a classificação apresentada na Tabela 53 for comparada com a apresentada na Tabela 44, observa-se que vários operadores de mutação aparecem em ordem diferente, indicando que o conjunto essencial-*IM*-adequado não determina a mesma cobertura que o conjunto *IM*-adequado.

Na Tabela 54 é apresentada a primeira iteração do processo para o estabelecimento de uma estratégia de aplicação dos operadores de mutação de unidade a partir de um conjunto essencial-*IM*-adequado.

Na primeira iteração, complementar um conjunto essencial-*IM*-adequado de modo a torná-lo AM-adequado, implica em aplicar mais 33 operadores de mutação de unidade com uma redução de custo de aproximadamente 10,25% em relação ao critério Análise de Mutantes. Após cinco iterações, reordenando os operadores a partir do incremento no escore proporcionado, obtém-se a ordem de aplicação dos operadores ilustrada na Tabela 55.

Tabela 54 – Estratégia de Aplicação dos Operadores de Mutação de Unidade a Partir de um Conjunto Essencial-*IM*-adequado: Primeira Iteração

Op	EAn	EAt	Inc	CO	CC	RC
T essencial-*IM*-adequado	0,00000	0,93479	0,93479	-	-	-
OLSN	0,93479	0,95897	0,02418	54	54	99,57924
OASN	0,95897	0,97170	0,01273	82	136	98,94031
OAAN	0,97170	0,97275	0,00105	182	318	97,52221
VSCR	0,97275	0,97275	0,00000	43	361	97,18716
OABN	0,97275	0,97311	0,00036	123	484	96,22877
Cccr	0,97311	0,99006	0,01695	1676	2160	83,16971
OLBN	0,99006	0,99142	0,00136	81	2241	82,53857
OLLN	0,99142	0,99142	0,00000	27	2268	82,32819
ORLN	0,99142	0,99430	0,00288	206	2474	80,72308
ORSN	0,99430	0,99431	0,00001	196	2670	79,19589
SSWM	0,99431	0,99431	0,00000	23	2693	79,01667
STRI	0,99431	0,99495	0,00064	160	2853	77,76999
OEBA	0,99495	0,99676	0,00181	255	3108	75,78308
SMTC	0,99676	0,99698	0,00022	33	3141	75,52595
VTWD	0,99698	0,99767	0,00069	422	3563	72,23781
OLRN	0,99767	0,99783	0,00016	162	3725	70,97553
ORRN	0,99783	0,99820	0,00037	515	4240	66,96276
Ccsr	0,99820	0,99823	0,00003	1393	5633	56,10877
OLAN	0,99823	0,99823	0,00000	135	5768	55,05688
SWDD	0,99823	0,99823	0,00000	18	5786	54,91663
SSDL	0,99823	0,99852	0,00029	446	6232	51,44148
ORBN	0,99852	0,99866	0,00014	294	6526	49,15069
Varr	0,99866	0,99866	0,00000	169	6695	47,83388
Oido	0,99866	0,99867	0,00001	91	6786	47,12482
ORAN	0,99867	0,99891	0,00024	490	7276	43,30684
OEAA	0,99891	0,99902	0,00011	425	7701	39,99532
VDTR	0,99902	0,99987	0,00085	633	8334	35,06311
Vprr	0,99987	0,99987	0,00000	105	8439	34,24497
STRP	0,99987	0,99987	0,00000	448	8887	30,75425
OESA	0,99987	0,99987	0,00000	170	9057	29,42964
CRCR	0,99987	0,99987	0,00000	751	9808	23,57800
OALN	0,99987	0,99987	0,00000	90	9898	22,87673
Vsrr	0,99987	1,00000	0,00013	1621	11519	10,24622
...	...	...	...	...	...	...
OIPM	1,00000	1,00000	0,00000	10	12834	0,00000

Analisando-se a Tabela 55, tem-se que com apenas 13 operadores de mutação é possível, a partir de um conjunto essencial-*IM*-adequado, obter um conjunto AM-adequado com uma redução de custo de 49,93% no total de mutantes gerados. Utilizando-se apenas dois operadores de mutação (Cccr e ORLN) é possível obter um escore de mutação superior a 0,99 em relação à Análise de Mutantes, com uma redução no número de mutantes gerados da ordem de 85%.

Observa-se que, também para os operadores de mutação de unidade, os resultados obtidos são semelhantes aos apresentados na Seção 4.2.3. Na verdade, os conjuntos obtidos diferem em apenas um operador. O conjunto de operadores da Seção 4.2.3 possui o operador ORSN e o conjunto da Tabela 55 possui o operador ORLN. Assim sendo, o custo de aplicação dos operadores da Tabela 55 apresenta-se 0,08% superior visto que o operador ORLN gera 20 mutantes a mais que ORSN. Além disso, o conjunto da Tabela 55 e o conjunto de operadores essenciais de unidade apresentado no Quadro 7, observa-se que 7 dos 9 operadores essenciais de unidade estão presentes entre os operadores necessários para se obterem conjuntos AM-adequados.

Tabela 55 – Estratégia de Aplicação dos Operadores de Mutação de Unidade a Partir de um Conjunto Essencial-*IM*-adequado: Última Iteração

Op	EAn	EAt	Inc	CO	CC	RC
T essencial-*IM*-adequado	0,00000	0,93479	0,93479	-	-	-
Cccr	0,93479	0,98775	0,05296	1676	1676	86,94094
ORLN	0,98775	0,99278	0,00503	206	1882	85,33583
OEBA	0,99278	0,99497	0,00219	255	2137	83,34892
VDTR	0,99497	0,99699	0,00202	633	2770	78,41671
VTWD	0,99699	0,99759	0,00060	422	3192	75,12856
ORRN	0,99759	0,99818	0,00059	515	3707	71,11579
OLBN	0,99818	0,99870	0,00052	81	3788	70,48465
OASN	0,99870	0,99912	0,00042	82	3870	69,84572
SSDL	0,99912	0,99941	0,00029	446	4316	66,37058
Vsrr	0,99941	0,99965	0,00024	1621	5937	53,74007
SMTC	0,99965	0,99985	0,00020	33	5970	53,48294
ORBN	0,99985	0,99999	0,00014	294	6264	51,19215
OLRN	0,99999	1,00000	0,00001	162	6426	49,92987
...	...	...	...	...	...	...
OIPM	1,00000	1,00000	0,00000	10	12834	0,00000

4.4 – Avaliação dos Resultados Obtidos

A seguir, os resultados apresentados nas seções 4.2.2, 4.2.3, 4.3.2 e 4.3.3 são sintetizados para uma avaliação final das estratégias definidas.

A Tabela 56(a) apresenta a ordem de aplicação dos operadores de mutação de interface a partir de um conjunto AM-adequado, e a Tabela 56(b) apresenta a ordem de aplicação dos operadores a partir de um conjunto essencial-AM-adequado.

Observa-se que ambos os conjuntos são compostos pelos mesmos operadores, sendo que a diferença está na ordem de aplicação estabelecida. Com isso, pode-se concluir que os operadores de mutação de interface apresentam praticamente o mesmo $strength$, tanto em relação aos conjuntos de casos de teste AM-adequados quanto em relação aos conjuntos essencial-AM-adequados. Na verdade, tais resultados eram esperados tendo em vista que o conjunto essencial

de operadores de mutação de unidade, obtido com a aplicação do procedimento **Essencial** [BAR98a], determina um alto escore de mutação em relação ao critério Análise de Mutantes, dando indícios de que os operadores essenciais selecionam praticamente o mesmo conjunto de casos de teste que o conjunto total de operadores selecionaria.

Tabela 56 – Estratégia de Aplicação dos Operadores de Interface: (a) a Partir de um Conjunto AM-adequado e (b) a Partir de um Conjunto Essencial-AM-adequado

(a)

***T* AM-adequado**
I-IndVarRepExt
I-DirVarRepExt
I-IndVarRepGlo
I-DirVarRepGlo
I-IndVarBitNeg
I-IndVarRepCon
I-IndVarRepLoc
I-DirVarRepPar
I-IndVarRepPar
I-CovAllNod
I-IndVarRepReq
I-IndVarIncDec
I-DirVarBitNeg
I-DirVarIncDec
I-DirVarRepLoc
I-CovAllEdg

(b)

***T* essencial-AM-adequado**
I-IndVarRepGlo
I-DirVarRepExt
I-IndVarRepExt
I-DirVarRepGlo
I-IndVarBitNeg
I-IndVarRepCon
I-IndVarRepLoc
I-IndVarRepPar
I-DirVarRepPar
I-CovAllNod
I-IndVarRepReq
I-IndVarIncDec
I-DirVarBitNeg
I-DirVarIncDec
I-DirVarRepLoc
I-CovAllEdg

Em relação aos operadores de mutação de unidade, os conjuntos de operadores também são praticamente os mesmos. As tabelas 57(a) e 57(b) ilustram qual a ordem de aplicação dos operadores de unidade a partir de um conjunto de casos de teste *IM*-adequado e essencial-*IM*-adequado, respectivamente. Ressalta-se que ambos os conjuntos de operadores de unidade apresentam ao menos um representante de cada classe de mutação, aspecto este destacado por Barbosa [BAR98b] como sendo de fundamental importância para que o conjunto de operadores apresente alto grau de adequação, além de modelar diferentes tipos de erros de unidade.

Observa-se ainda que os conjuntos possuem 12 operadores de mutação em comum. A diferença entre eles é que o conjunto da Tabela 57(a) possui o operador ORSN e o conjunto da Tabela 57(b) o operador ORLN. Tal resultado confirma que, também em nível de unidade, os operadores de mutação apresentam praticamente o mesmo *strength* tanto em relação a conjuntos *IM*-adequados quanto a conjuntos essencial-*IM*-adequados. Além disso, como tais operadores de mutação não são incluídos por conjuntos de casos de teste *IM*-adequados, é importante que eles sejam utilizados no teste de unidade, uma vez que, os requisitos de teste por eles exigidos não estarão sendo cobertos na fase de integração, mesmo se todos os operadores de mutação de interface forem utilizados.

Tabela 57 – Estratégia de Aplicação dos Operadores de Unidade: (a) a Partir de um Conjunto *IM*-adequado e (b) a Partir de um Conjunto Essencial-*IM*-adequado

(a)

***T IM*-adequado**
Cccr
ORSN
OEBA
VDTR
VTWD
OLBN
OASN
ORRN
SSDL
Vsrr
SMTC
ORBN
OLRN

(b)

***T* essencial-*IM*-adequado**
Cccr
ORLN
OEBA
VDTR
VTWD
ORRN
OLBN
OASN
SSDL
Vsrr
SMTC
ORBN
OLRN

Com respeito ao custo de aplicação das estratégias, considerando que, ao invés de utilizar as estratégias apresentadas neste capítulo, fossem utilizados os critérios Análise de Mutantes e Mutação de Interface, o custo total, para os cinco programas do experimento, seria de 26.858 mutantes gerados, 12.834 referentes a Análise de Mutantes e 14.024 a Mutação de Interface. Na Tabela 58 são apresentados os custos envolvidos na utilização das estratégias que foram definidas.

Tabela 58 – Análise de Custo das Estratégias Incrementais

Estratégia	Custo	Redução de Custo
AM → *IM* (Tabela 56(a))	12.384 + 10.104 = 22.488	4.370 (16,27%)
Essencial-AM → *IM* (Tabela 56(b))	5.217 + 10.104 = 15.321	11.537 (42,96%)
Essencial-*IM* → AM (Tabela 57(b))	3.756 + 6.426 = 10.182	16.676 (62,10%)
IM → AM (Tabela 57(a))	14.024 + 6.416 = 20.440	6.418 (23,90%)

Considerando a primeira linha da Tabela 58, observa-se uma estratégia incremental na qual utilizar um conjunto AM-adequado para se obter um conjunto *IM*-adequado (AM → *IM*) apresenta um custo de 22.488 mutantes (12.384 mutantes referentes à obtenção do conjunto AM-adequado Tabela 56(a)), 4.370 mutantes a menos do que seria gerado com a utilização de todos os operadores de mutação de ambos os critérios.

Partindo-se de um conjunto essencial-AM-adequado para se obter um conjunto *IM*-adequado (Essencial-AM → *IM*) , obtém-se uma redução de custo de aproximadamente 42,96%. Na verdade, como era de se esperar, as estratégias de menor custo são as que envolvem os conjuntos essenciais de operadores de mutação (linhas 2 e 3). Na linha 4 é representada a estratégia inversa à da linha 1, ou seja, a partir de um conjunto *IM*-adequado, como obter um conjunto AM-adequado.

Considerando o conjunto CE_{IM} (Quadro 8) e os conjuntos (a) e (b) da Tabela 56, tem-se que, mesmo obtido um conjunto de caso de teste AM-adequado ou essencial-AM-adequado, 7 dos 8 operadores de CE_{IM} (I-CovAllNod, I-DirVarBitNeg, I-IndVarBitNeg, I-IndVarRepGlo, I-IndVarRepExt, I-IndVarRepLoc e I-IndVarRepReq) foram requeridos para garantir um conjunto de casos de teste *IM*-adequado. Com isso, pode-se concluir que tais operadores apresentam um alto *strength* em relação ao critério Análise de Mutantes, constituindo uma boa alternativa para se iniciar o teste de integração.

Da mesma forma, considerando o conjunto CE_{AM} (Quadro 7) e os conjuntos (a) e (b) da Tabela 57, observa-se que, mesmo obtido um conjunto *IM*-adequado ou essencial-*IM*-adequado, 7 dos 9 operadores essenciais de unidade são selecionados por apresentarem um alto *strength* em relação ao critério Mutação de Interface, sendo necessários para garantir conjuntos AM-adequados.

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 utilizarem-se, inicialmente, os operadores do conjunto essencial tanto no teste de unidade quanto no teste de integração, de forma incremental.

Na Figura 9 estão representadas algumas das estratégias de aplicação incremental dos critérios baseados em mutação.

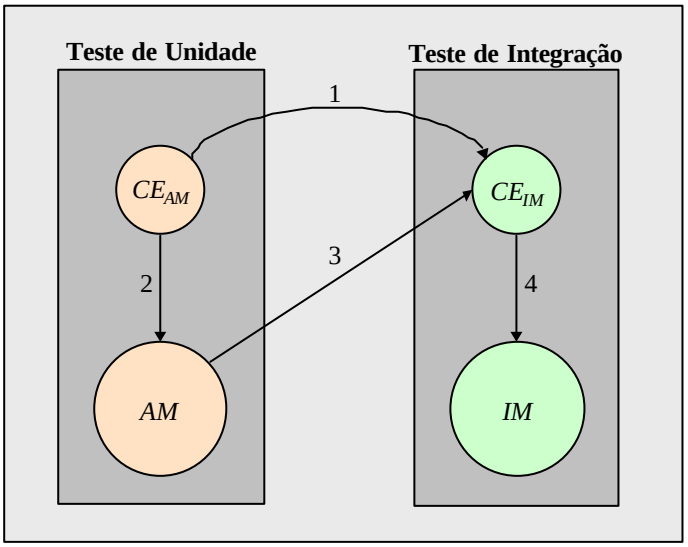


Figura 9 – Estratégias Incrementais

Por exemplo, uma possível estratégia incremental seria seguir os passos 2, 3 e 4, ou seja, iniciar os testes de unidade utilizando os operadores essenciais (CE_{AM}) e, em seguida, obter um conjunto AM-adequado (2), seguindo a mesma estratégia apresentada na Seção 3.3.3. Obtido o

conjunto AM-adequado, aplicam-se os operadores essenciais de interface procurando complementar o conjunto AM-adequado para torná-lo CE_{IM}-adequado (3). Finalmente, os demais operadores de interface podem ser utilizados incrementalmente de modo a satisfazer todos os requisitos do critério Mutação de Interface (4).

Uma estratégia mais pragmática seria percorrer o caminho 1 e 4, ou seja, no teste de unidade utilizarem-se os operadores do conjunto essencial (CE_{AM}) e, em seguida, no teste de integração serem aplicados os operadores de CE_{IM} de forma incremental. Se o grau de cobertura obtido em relação ao critério Mutação de Interface ainda não for satisfatório, outros operadores de interface podem ser utilizados de modo a garantir um conjunto de casos de teste *IM*-adequado. Tal estratégia é ilustrada na Tabela 59.

Tabela 59 – Essencial-AM e Essencial-IM

Op	EAn	EAt	Inc	CO	CC	RC
T essencial-AM-adequado	0,00000	0,91956	0,91956	-	-	-
II-ArgAriNeg	0,91956	0,92623	0,00667	26	26	99,81460
I-CovAllNod	0,92623	0,97257	0,04634	438	464	96,69139
I-DirVarBitNeg	0,97257	0,98229	0,00972	154	618	95,59327
I-IndVarBitNeg	0,98229	0,99126	0,00897	306	924	93,41129
I-IndVarRepGlo	0,99126	0,99381	0,00255	312	1236	91,18654
I-IndVarRepExt	0,99381	0,99718	0,00337	511	1747	87,54278
I-IndVarRepLoc	0,99718	0,99858	0,00140	618	2365	83,13605
I-IndVarRepReq	0,99858	0,99880	0,00022	1391	3756	73,21734
I-IndVarRepPar	0,99880	0,99922	0,00042	672	4428	68,42556
I-DirVarRepExt	0,99922	0,99953	0,00031	494	4922	64,90302
I-DirVarRepPar	0,99953	0,99981	0,00028	385	5307	62,15773
I-IndVarRepCon	0,99981	0,99991	0,00010	2555	7862	43,93896
I-IndVarIncDec	0,99991	0,99995	0,00004	684	8546	39,06161
I-DirVarRepGlo	0,99995	0,99997	0,00002	170	8716	37,84940
I-CovAllEdg	0,99997	0,99998	0,00001	390	9106	35,06845
I-DirVarIncDec	0,99998	0,99999	0,00001	536	9642	31,24643
I-DirVarRepLoc	0,99999	1,00000	0,00001	488	10130	27,76669
...	...	...	...	...	...	...
II-FunCalDel	1,00000	1,00000	0,00000	254	14024	0,00000

Com isso, considerando apenas a utilização dos conjuntos essenciais, o total de mutantes gerados é de 8.973; destes, 5.217 são gerados por CE_{AM} e 3.756 por CE_{IM}. Comparado ao custo total de utilização dos critérios Análise de Mutantes e Mutação de Interface, que é de 26.858 mutantes (12.834 mutantes de unidade e 14.024 mutantes de interface), a redução de custo proporcionada pela utilização dos conjuntos essenciais é de 17.855 mutantes. Com isso, observa-se que, mesmo com uma redução no custo de aplicação dos critérios da ordem de 66,59% é possível obterem-se conjuntos de casos de teste que determinam um alto grau de cobertura dos critérios baseados em mutação. Restaria, desse modo, investigar o aspecto de eficácia dessas estratégias em revelar a presença de erros.

4.5 – Considerações Finais

Durante este capítulo foram apresentadas algumas estratégias alternativas para a aplicação dos conjuntos de operadores de mutação de unidade e de integração. Na verdade, o que se procurou demonstrar foi que, embora os critérios Análise de Mutantes e Mutação de Interface sejam destinados a fases distintas da atividade de teste, esses apresentam algumas características em comum, representadas através de seus operadores de mutação. Tendo em vista que ambos os critérios apresentam problemas de custo devido ao grande número de mutantes gerados, identificar as partes comuns a ambos os critérios permite reduzir os custos de aplicação dos mesmos, uma vez que evita "redundâncias" em ambas as fases do teste.

De um modo geral, observou-se que tanto a partir de um conjunto de casos de teste AM-adequado quanto a partir de um conjunto essencial-AM-adequado, a utilização de mais 16 operadores de mutação de interface resulta em um conjunto de casos de teste *IM*-adequado, com uma redução no custo de aplicação dos critérios Análise de Mutantes e Mutação de Interface entre 16,27% e 42,96%, dependendo se o conjunto *IM*-adequado foi obtido a partir de um conjunto AM-adequado ou essencial-AM-adequado, respectivamente.

Da mesma forma, um conjunto de casos de teste *IM*-adequado ou um conjunto de casos de teste essencial-*IM*-adequado requer que mais 13 operadores de mutação de unidade sejam utilizados até que se obtenha um conjunto de casos de teste AM-adequado. Isso representa uma redução no custo de aplicação do critério Análise de Mutantes de aproximadamente 50% e uma redução no custo total dos critérios Análise de Mutantes e Mutação de Interface entre 23,90% e 62,10% se o conjunto AM-adequado for obtido a partir de um conjunto *IM*-adequado ou essencial-*IM*-adequado, respectivamente.

Além disso, demostrou-se que somente a utilização dos conjuntos essenciais de operadores de mutação é suficiente para se construir conjuntos de casos de teste que determinem escores de mutação superiores a 0,99, com uma redução no custo total de aplicação dos critérios Análise de Mutantes e Mutação de Interface de aproximadamente 66,59%.

Assim sendo, as alternativas investigadas demonstram que é possível reduzir os custos de aplicação dos critérios baseados em mutação e, ainda assim, permitir que sejam selecionados conjuntos de casos de teste que determinem um alto grau de adequação em relação a esses critérios. Além disso, observa-se que os critérios Análise de Mutantes e Mutação de Interface podem e devem ser utilizados de forma complementar na atividade de teste, englobando as fases do teste de unidade e de integração.

Capítulo 5

Conclusões e Trabalhos Futuros

A atividade de teste consiste em uma análise dinâmica do produto, sendo relevante para a identificação e eliminação de erros que persistem, constituindo um dos elementos para fornecer evidências da confiabilidade do software, em complemento a outras atividades de VV&T. O principal objetivo do teste de software é revelar a presença de erros no produto. Portanto, o teste bem sucedido é aquele que consegue determinar casos de teste para os quais o programa em teste falhe.

Um problema, entretanto, é que a atividade de teste tem sido apontada entre as mais onerosas no desenvolvimento de software e, na tentativa de reduzir os custos associados ao teste, faz-se necessária a aplicação de técnicas e critérios que dêem indicações de como testar o software, quando parar o teste e que, se possível, forneçam uma medida objetiva do nível de confiança e de qualidade alcançados com os testes realizados.

Devido ao aspecto complementar das técnicas de teste, sempre que possível, o testador deve aplicá-las em conjunto para obter um teste de boa qualidade. Assim sendo, surge a questão de qual estratégia de teste utilizar de modo a obter uma maior eficácia em revelar a presença de erros a um menor custo.

O objetivo principal desses estudos é estabelecer uma estratégia de teste que apresente um baixo custo de aplicação, uma alta eficácia em revelar a presença de erros e que possa ser aplicada de modo incremental, ou seja, dependendo da qualidade e da confiabilidade que se quer garantir, do tempo e recursos disponíveis, o testador decide quais critérios aplicar e quando aplicá-los.

Um dos critérios que mais tem se destacado devido à sua eficácia em revelar a presença de erros é o critério Análise de Mutantes; entretanto, problemas como o grande número de mutantes gerados tornam necessário o desenvolvimento de estratégias que permitam a sua utilização de forma economicamente viável. Nesse sentido, o desenvolvimento de critérios alternativos, que reduzam o número de mutantes gerados, mantendo a alta eficácia do critério em revelar a presença de erros, têm sido bastante explorados. Além disso, procurando aplicar os conceitos do teste de mutação em nível de integração, foi definido o critério Mutação de Interface.

Nesse contexto, é natural questionar qual seria o relacionamento entre os critérios Análise de Mutantes e Mutação de Interface, e como utilizá-los de forma incremental e complementar na atividade de teste. Tais critérios foram avaliados empiricamente, do ponto de vista de custo e *strength*, contribuindo para o estabelecimento de estratégias de teste que permitam reduzir os custos de aplicação dos mesmos.

Inicialmente, procurando reduzir o custo de aplicação do critério Mutação de Interface, uma estratégia incremental de aplicação de seus operadores foi estabelecida. A utilização dessa estratégia possibilitou uma redução no custo de aplicação do critério em torno de 25% e, ainda assim, conjuntos de casos de teste *IM*-adequados foram obtidos. Além disso, foi determinado um conjunto essencial de operadores de mutação de interface, obtido a partir da aplicação do procedimento **Essencial** proposto por Barbosa [BAR98a]. O conjunto essencial de operadores de mutação de interface possibilitou a seleção de conjuntos de casos de teste altamente adequados 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%.

O *strength* entre os critérios Análise de Mutantes e Mutação de Interface também foi avaliado, observando-se que tais critérios são incomparáveis do ponto de vista da relação de inclusão, devendo ser utilizados de forma complementar para assegurar um teste de melhor qualidade. Explorando o aspecto complementar desses critérios, algumas estratégias de aplicação dos operadores de mutação de unidade e integração foram estabelecidas. 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.

Os resultados obtidos motivam que outros aspectos relacionados à aplicação dessas estratégias (tais como a eficácia) sejam investigados, contribuindo para que os critérios baseados em mutação possam ser empregados em ambientes reais de desenvolvimento, melhorando a qualidade e a confiabilidade dos produtos de software desenvolvidos.

Ressalta-se o esforço necessário para a determinação dos mutantes equivalentes que, é uma questão indecidível e exige do testador completo conhecimento dos programas. Além disso, para a condução dos experimentos foi necessária a construção de mais de 70 *scripts* de teste resultando em um total de, aproximadamente, 8.500 linhas de código. Parte dos *scripts* utilizados, bem como os resultados obtidos após a execução dos mesmos e os aspectos envolvidos em sua elaboração estão sendo sintetizados para compor uma nota didática [VIN98].

5.1 – Contribuições deste Trabalho

Pode-se destacar como contribuições deste trabalho:

- Análise empírica dos critérios Análise de Mutantes e Mutação de Interface avaliando custo e *strength* dos mesmos;
- Desenvolvimento de estratégias de teste utilizando os critérios de teste baseados em mutação;
- Avaliação do procedimento **Essencial** [BAR98a] para outro grupo de operadores de mutação;
- Determinação do conjunto essencial de operadores de mutação de interface;
- Análise empírica dos operadores de mutação de interface e estabelecimento de uma estratégia incremental de aplicação dos mesmos; e
- Produção de dados históricos a respeito dos critérios Análise de Mutantes e Mutação de Interface.

5.2 – Trabalhos Futuros

Algumas das atividades que podem ser realizadas como continuidade deste trabalho são:

- Reproduzir os experimentos apresentados neste trabalho em outros domínios de aplicação com o objetivo de obter informações mais abrangentes a respeito dos operadores de mutação de unidade e de interface;
- Avaliar a eficácia em revelar a presença de erros das estratégias propostas e do conjunto essencial de operadores de mutação de interface; e
- Com base nas informações históricas acumuladas durante a realização deste trabalho e também com base nas informações de outros estudos empíricos que já foram realizados pelo grupo, desenvolver heurísticas que auxiliem na determinação de mutantes equivalentes.

Nesse sentido, alguns experimentos envolvendo o programa *Sort* do UNIX e o programa *Space* da Agência Espacial Européia já estão sendo conduzidos com o objetivo de investigar a eficácia em revelar a presença de erros das estratégias propostas neste trabalho. Posteriormente, obtidas as informações a respeito da eficácia dos operadores de mutação de unidade e integração, novas comparações entre os mesmos podem ser realizadas, contribuindo para o desenvolvimento de estratégias que, além de garantirem uma alta cobertura em relação aos critérios Análise de Mutantes e Mutação de Interface, também garantam uma maior eficácia em revelar a presença de erros tanto em nível de unidade quando em nível de integração. Essa perspectiva pode levar a um refinamento do procedimento **Essencial** [BAR98a].

Referências Bibliográficas

[ACR79] 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.

[ACR80] ACREE, A. *On Mutation*. Tese de Doutorado, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA, Agosto, 1980.

[AGR89] AGRAWAL, H.; DEMILLO, R.A.; HATHAWAY, R.; HSU, W.; HSU, Wy.; KRAUSER, E.W.; MARTIN, R.J.; MATHUR, A.; SPAFFORD, E.H. *Design of Mutant Operators for the C Programming Language*. Relatório Técnico SERC-TR-41-P, Software Engineering Research Center, Purdue University, West Lafayette, IN, Março, 1989.

[BAR98a] BARBOSA, E. F. *Uma Contribuição para a Determinação de um Conjunto Essencial de Operadores de Mutação no Teste de Programas C*. Mestrado em andamento, ICMC/USP, São Carlos, SP, 1998.

[BAR98b] 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.

[BEI90] BEIZER, B. *Software Testing Techniques*. Van Nostrand Reinhold, 2ª Edição, New York, NY, 1990.

[BOY75] BOYER, R.S.; ELSPAS, B.; KARL, N.L. "A Formal System for Testing na Debugging Programs by Simbolic Execution". *ACM SigPlan Notices*, v. 10, n. 6, pp. 213-222, Junho, 1975.

[BUD80a] BUDD, T.A.; DEMILLO, R.A.; LIPTON, R.J.; Sayward, F.G. "Theoretical and Empirical Studies on Using Program Mutation to Test the Functional Correctness of Programs.". *Anais* do *VII ACM Symposium on Principles of Programming Languages*, pp. 220-233, New York, NY, Janeiro, 1980.

[BUD80b] BUDD, T.A. *Mutation Analysis of Program Test Data*. Tese de Doutorado, Yale University, New Haven, CT, 1980.

[BUD81] BUDD, T.A. *Mutation Analysis: Ideas, Examples, Problems and Prospects*. North-Holand Publishing Company, 1981.

[CHA91] CHAIM, M.L. *POKE-TOOL – Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseados em Análise de Fluxo de Dados*. Tese de Mestrado, DCA/FEE/UNICAMP, Campinas, SP, Abril, 1991.

[CHO89] CHOI, B.J.; DEMILLO, R.A.; KRAUSER, E.W.; MARTIN, R.J.; MATHUR, A.P.; OFFUTT, A.J.; PAN, H.; SPAFFORD, E.H. "The Mothra Tool Set". *Anais da XXII Annual Hawaii International Conference on Systems and Software*, Kona, Hawaii, Janeiro, 1989.

[DEL93a] DELAMARO, M.E.; MALDONADO, J.C. *Uma Visão sobre a Aplicação da Análise de Mutantes*. Notas do ICMC, n. 133, São Carlos, SP, Março, 1993.

[DEL93b] DELAMARO, M.E. *Proteum – Um Ambiente de Teste Baseado na Análise de Mutantes*. Dissertação de Mestrado, ICMC/USP, São Carlos, SP, Outubro, 1993.

[DEL96a] DELAMARO, M.E.; MALDONADO, J.C. "Proteum: A Tool for the Assessment of Test Adequacy for C Programs". *Anais da Conference on Performability in Computing Systems (PCS 96)*, Brunswick, NJ, Julho, 1996.

[DEL96b] DELAMARO, M.E.; MALDONADO, J.C.; MATHUR, A.P. "Integration Testing Using Interface Mutation". *Anais do VII International Symposium on Software Reliability Engineering (ISSRE'96)*, pp. 112-121, New York, NY, Novembro, 1996.

[DEL97a] DELAMARO, M.E.; MALDONADO, J.C. "Interface Mutation: An Approach for Integration Testing". *Anais do Workshop do Projeto Validação e Teste de Sistemas de Operação*, pp. 177-189, Águas de Lindóia, SP, Janeiro, 1997.

[DEL97b] DELAMARO, M.E.; MALDONADO, J.C. "Interface Mutation: A Case Study". *Anais do Workshop do Projeto Validação e Teste de Sistemas de Operação*, pp. 191-202, Águas de Lindóia, SP, Janeiro, 1997.

[DEL97c] 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.

[DEL98a] DELAMARO, M.E.; MALDONADO, J.C.; MATHUR, A.P. "Interface Mutation: An Approach for Integration Testing", *IEEE Transactions on Software Engeneering*, (sendo submetido), 1998.

[DEL98b] DELAMARO, M.E.; MALDONADO, J.C.; PASQUINI, A.; MATHUR, A.P. "Interface Mutation Test Adequacy Criterion: An Empirical Evaluation", em preparação, 1998.

[DEM78] DEMILLO, R.A.; LIPTON, R.J.; SAYWARD, F.G. "Hints on Test Data Selection: Help for the Practicing Programmer". *IEEE Computer*, v. 11, n. 4, pp. 34-41, Abril, 1978.

[DEM80] DEMILLO, R.A. "Mutation Analysis as a Tool for Software Quality Assurance". *Anais da COMPSAC80*, Chicago, IL, Outubro, 1980.

[DEM87] DEMILLO, R.A. *Software Testing and Evaluation*. The Benjamin/Cummings Publishing Company, 1987.

[DEM88] 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.

[DEM91] DEMILLO, R.A.; OFFUTT, A.J. "Constrained-Based Automatic Test Data Generation". *IEEE Transactions on Software Engineering*, v. 17, n. 9, pp. 900-910, Setembro, 1991.

[DEM94] DEMILLO, R.A.; MATHUR, A.P.; WONG, W.E. "Some Critical Remarks on a Hierarchy of the Fault-Detecting Ability of Test Methods". *IEEE Transactions on Software Engineering*, v. 21, n. 10, pp. 858-863, Outubro, 1993.

[DEU92] DEUTSCH, M.S. *Software Verification and Validation*. Englewwod Cliffs, Prentice-Hall, 1982.

[DUR84] DURAN, J.W.; NTAFOS, S.C. "An Evaluation of Randon Testing". *IEEE Transactions on Software Engineering*, v. 10, n. 4, Julho, 1984.

[FRA85] FRANKL, P.G.; WEYUKER, E.J. "Data Flow Testing Tools". *Anais da IEEE Softfair II*, pp. 46-53, São Francisco, CA, Dezembro, 1985.

[FRA88] FRANKL, P.G.; WEYUKER, E.J. "An Applicable Family of Dataflow Criteria". *IEEE Transactions on Software Engineering*, v. 14, n.10, pp. 1483-1498 Outubro, 1988.

[FRA93] 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.

[FRI75] FRIEDMAN, A.D. *Logical Design of Digital Systems*. Computer Science Press, 1975.

[GOE85] GOEL, A.L. "Software Reability Models: Assumptions, Limitations, and Applicability". *IEEE Transactions on Software Engineering*, v. 11, n. 12, Dezembro, 1985.

[HAL84] 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.

[HAM88] HAMLET, D.; TAYLOR, R. "Partition Testing does not Inspire Confidence". *Anais* do *II Workshop on Software Testing, Verification and Analysis*, Computer Science Press, pp.206-215, Banff, Canadá, Julho, 1988.

[HAR91] HARROLD, M.J.; SOFFA, M.L. "Selecting and Using Data for Integration Test". *IEEE Software*, v. 8, n. 2, pp. 58-65, Março, 1991.

[HAR93] HARROLD, M.J.; SOFFA, M.L.; GUPTA, R. "A Methodology for Controlling the Size of a Test Suite". *ACM Transactions on Software Engineering and Methodology*, v. 2, n. 3, pp. 270-285, Julho, 1993.

[HOR92] HORGAN, J.R.; MATHUR, A.P. "Assessing Testing Tools in Research and Education". *IEEE Software*, v. 8, n.3, pp. 61-69, Maio, 1992.

[HOW77] HOWDEN, W.E. "Symbolic Testing and Dissect Symbolic Evaluation System". *IEEE Transactions on Software Engineering*, v. 3, n. 4, pp. 266-278, Julho, 1977.

[HOW78] HOWDEN, W.E. "Theoretical and Empirical Studies of Program Testing". *IEEE Transactions on Software Engineering*, v. 4, n. 4, pp. 293-298, Julho, 1978.

[HOW82] HOWDEN, W.E. "Weak Mutation Testing and Completness of Test Sets". *IEEE Transactions on Software Engineering*, v. 8, n. 4, pp. 371-379, Julho, 1982.

[I390] *IEEE Standard Glossary of Software Engineering Terminology*. Padrão 610.12, IEEE, 1990.

[JIN95] JIN, Z.; OFFUTT, A.J. *Integration Testing Based on Software Couplings*, Relatório Técnico ISSE-TR-95-100, George Mason University, disponível *on-line* – URL: http://isse.gmu.edu/techrep/1995/, Janeiro, 1995.

[KOR90] KOREL, B. "Automated Software Test Data Generation". *IEEE Transactions on Software Engineering*, v. 16, n. 8, pp. 870-879, Agosto, 1990.

[KRA88] KRAUSER, E.W.; MATHUR, A.P.; REGO, V. "High Performance Testing on SIMD Machines". *Anais do II Workshop on Software Testing, Verification and Analysis*, Banff, Canadá, 1988.

[LIN90] 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.

[MAL89] MALDONADO, J.C.; CHAIM, M.L.; JINO, M. "Arquitetura de uma Ferramenta de Teste de Apoio aos Critérios Potenciais Usos". *Anais do XXII Congresso Nacional de Informática*, São Paulo, SP, Setembro, 1989.

[MAL91] MALDONADO, J.C. *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*. Tese de Doutorado, DCA/FEE/UNICAMP, Campinas, SP, Julho, 1991.

[MAL95] MALDONADO, J.C.; DELAMARO, M.E.; SOUZA, S.R.S; PAILO, L.F. "Análise de Mutantes: Uma Avaliação Empírica do Axioma de Antiextensionalidade e da Propriedade de Equivalência". *Anais do II Workshop de Qualidade de Software - IX Simpósio Brasileiro de Engenharia de Software*, pp.136-140, Recife, PE, Outubro, 1995.

[MAL97] MALDONADO. J.C. *Critérios de Teste de Software: Aspectos Teóricos, Empíricos e de Automatização*. Livre Docência, ICMC-USP, São Carlos, SP, Janeiro, 1997.

[MAT88] MATHUR, A.P.; KRAUSER, E.W. "Modeling Mutation on Vector Processor". *Anais do II Workshop on Software Testing, Verification and Analysis*, Banff, Canadá, 1988.

[MAT91] MATHUR, A.P. "Performance, Effectiveness, and Reliability Issues in Software Testing". *Anais da XV Annual International Computer Software and Applications Conference*, pp. 604-605, Tokio, Japão, Setembro, 1991.

[MAT93] MATHUR, A.P.; WONG, W.E. "Evaluation of the Cost of Alternate Mutation Strategies". *Anais do VII Simpósio Brasileiro de Engenharia de Software*, pp.320-335, Rio de Janeiro, RJ, 1993.

[MAT94] MATHUR, A.P.; WONG, W.E. "An Empirical Comparison of Data Flow and Mutation-Based Test Adequacy Criteria". *The Journal of Software Testing, Verification and Reliability*, v. 4, n. 1, pp. 9-31, Março, 1994.

[MYE79] MYERS, G.J. *The Art of Software Testing*. John Wiley & Sons, New York, NY, 1979.

[NTA88] NTAFOS, S.C. "A Comparison of Some Structural Testing Strategies". *IEEE Transactions on Software Engineering*, v. 14, n. 6, pp. 868-874, Junho, 1988.

[OFF89] 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.

[OFF93] OFFUTT, A.J.; ROTHERMEL, G.; ZAPF, C. "An Experimental Evaluation of Selective Mutation". *Anais da XV International Conference on Software Engineering*, pp. 100-107, Baltimore, MD, Maio, 1993.

[OFF94] OFFUTT, A.J.; LEE, S. "An Empirical Evaluation of Weak Mutation". *IEEE Transactions on Software Engineering*, v. 20, n. 5, pp. 337-344, Maio, 1994.

[OFF96a] OFFUTT, A.J.; PAN, J.; TEWARY, K.; ZHANG T. "An Experimental Evaluation of Data Flow and Mutation Testing". *Software-Practice and Experience*, v. 26, n. 2, pp. 165-176, Fevereiro, 1996.

[OFF96b] OFFUTT, A.J.; LEE, A.; ROTHERMEL, G.; UNTCH, R.H.; ZAPF, C. "An Experimental Determination of Sufficient Mutant Operators". *ACM Transactions on Software Engineering Methodology*, v. 5, n. 2, pp. 99-118, Abril, 1996.

[PRE97] PRESSMAN, R.S. *Software Engineering – A Practitioner's Approach*. McGraw-Hill, 4ª Edição, 1997.

[RAP82] RAPPS, S.; WEYUKER, E.J. "Data Flow Analysis Techniques for Test Data Selection". *Anais da International Conference on Software Engineering*, pp. 272-278, Tóquio, Japão, Setembro, 1982.

[RAP85] RAPPS, S.; WEYUKER, E.J. "Selecting Software Test Data Using Data Flow Information". *IEEE Transactions on Software Engineering*, v. 11, n. 4, pp. 367-375, Abril, 1985.

[SOU96] 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.

[VIN97] VINCENZI, A.M.R.; BARBOSA, E.F.; DELAMARO, M.E.; SOUZA, S.R.S.; MALDONADO, J.C. "Critério Análise de Mutantes: Estado Atual e Perspectivas". *Anais do Workshop do Projeto Validação e Teste de Sistemas de Operação*, pp. 15-26, Águas de Lindóia, SP, Janeiro, 1997.

[VIN98] VINCENZI, A.M.R.; BARBOSA, E.F.; DELAMARO, M.E.; SOUZA, S.R.S.; MALDONADO, J.C. *Proteum & Proteum/IM*: *Elaboração de Scripts de Teste*, Notas Didáticas do ICMC em preparação, São Carlos, SP, 1998.

[WEY90] WEYUKER, E.J. "The Cost of Data Flow Testing: An Empirical Study". *IEEE Transactions on Software Engineering*, v. 16, n. 2, pp. 121-128, Fevereiro, 1990.

[WEY91] WEYUKER, E.J.; WEISS, S.N.; HAMLET, R.G. "Comparison of Program Testing Strategies". *Anais do IV Symposium on Software Testing, Analysis and Verification*, pp. 154-164, Victoria, British Columbia, Canadá, New York, NY, Outubro, 1991.

[WHI80] WHITE, L.J.; COHEN, E.I. "A Domain Strategy for Computer Program Testing". *IEEE Transactions on Software Engineering*, v. 6, n. 3, pp. 247-257, Maio, 1980.

[WON93] WONG, W.E. *On Mutation and Data Flow*. Tese de Doutorado, Software Engineering Research Center – Purdue University, West Lafayette, IN, Dezembro, 1993.

[WON94a] WONG, W.E.; MALDONADO, J.C.; DELAMARO, M.E.; MATHUR, A.P. "Constrained Mutation in C programs". *Anais do VIII Simpósio Brasileiro de Engenharia de Software*, pp. 439-452, Curitiba, PR, Outubro, 1994.

[WON94b] WONG, W.E.; MALDONADO, J.C.; MATHUR, A.P. "Mutation versus All-Uses: An Empirical Evaluation of Cost, Strength, and Effectiveness". *Software Quality and Productivity – Theory, Practice, Education and Training*, Hong Kong, Dezembro, 1994.

[WON95] WONG, W.E.; MATHUR, A.P. "Reducing the Cost of Mutation Testing: An Empirical Study". *The Journal of Systems and Software*, v. 31, n. 3, pp. 185-196, Dezembro, 1995.

[WON97] WONG, W.E.; MALDONADO, J.C.; DELAMARO, M.E.; SOUZA, S.R.S. "A Comparison of Selective Mutation in C and Fortran". *Anais do Workshop do Projeto Validação e Teste de Sistemas de Operação*, pp. 71-84, Águas de Lindóia, SP, Janeiro, 1997.

[WOO88] WOODWARD, M.R., e Halewood, K., "From Weak to Strong, Dead or Alive? An Analysis of Some Mutation Testing Issues", *Anais do II Workshop on Software Testing, Verification and Analysis*, pp. 152-158, Banff, Canadá, Julho, 1988.

[WOO93] WOODWARD, M.R. "Mutation Testing – Its Origin and Evolution". *Information and Software Technology*, v. 35, n.3, pp. 163-169, Março, 1993.

Apêndice A

Operadores de Mutação das Ferramentas *Proteum* e *Proteum/IM*

As transformações sintáticas que devem ser realizadas em determinado programa para transformá-lo em programas mutantes são definidas através de operadores de mutação (*mutant operators*). Os operadores de mutação são construídos para satisfazer a um entre dois propósitos: 1) induzir mudanças sintáticas simples com base nos erros típicos cometidos durante o processo de desenvolvimento de software (como trocar o nome de uma variável); ou 2) forçar determinados objetivos de teste (como executar cada arco do programa) [OFF96b].

A seguir, são apresentados os operadores de mutação de unidade, implementados na ferramenta *Proteum*, e os operadores de mutação de interface, implementados na ferramenta *Proteum/IM*.

Conforme mencionado anteriormente, os operadores de mutação destinados ao teste de unidade possuem semelhanças e diferenças em relação aos operadores de mutação de interface. A idéia básica de ambos é a mesma, ou seja, introduzir modificações sintáticas no programa em teste transformando-o em programas mutantes. A diferença é que os operadores de mutação de interface estão relacionados a uma chamada de função (conexão) ao passo que os operadores de mutação unidade estão restritos a uma única unidade.

A.1 – Operadores de Mutação de Unidade: Ferramenta *Proteum*

Conforme dito anteriormente, esses operadores estão divididos em quatro classes: mutação de comandos (Quadro 9), mutação de variáveis (Quadro 10), mutação de constantes (Quadro 11) e mutação de operadores (Quadro 12). Ao todo são 71 operadores de mutação implementados na ferramenta *Proteum*. Informações mais detalhadas a respeito de tais operadores podem ser encontradas em [DEL93b, DEL96a].

Quadro 9 – Operadores de Mutação de Comandos

Operador	Descrição
SBRC	Substitui break por continue quando possível.
SBRn	Troca o comando continue ou break por uma função break_out_to_level_n(J) sendo que J varia de acordo com o número de laços aninhados. Essa função força a interrupção dos J laços externos.
SCRB	Troca do comando continue por break.
SCRn	Troca o comando continue ou break por uma função continue_out_to_level_n(J) sendo que J varia de acordo com o número de laços aninhados. Essa função força a transferência do programa para o final de J laços acima.
SDWD	Troca o comando do-while por while.
SGLR	Troca os comandos goto l por todos os rótulos existentes na função.
SMTC	Interrompe a execução do laço após 2 execuções.
SMTT	Força a execução dos laços mais de uma vez.
SMVB	Move "}" para cima e para baixo, quando possível.
SRSR	Troca cada comando por todos os returns que existem na função em teste.
SSDL	Retira um comando de cada vez do programa.
SSWM	Força a execução de todos os cases do comando switch.
STRI	Força a execução para true e false em cada if.
STRP	Força a execução de todos os comandos do programa.
SWDD	Troca o comando while por do-while.

Quadro 10 – Operadores de Mutação de Variáveis

Operador	Descrição
Varr	Substitui as referências a vetores por variáveis escalares, globais e locais do programa.
VDTR	Força cada referência escalar possuir cada um dos valores: negativo, positivo e zero.
Vprr	Substitui as referências a apontadores por variáveis escalares, globais e locais do programa.
VSCR	Troca as referências a componentes de uma estrutura por demais componentes da mesma estrutura.
Vsrr	Substitui as referências escalares por variáveis escalares, globais e locais do programa.
Vtrr	Substitui as referências a estruturas e uniões por variáveis escalares, globais e locais do programa.
VTWD	Substitui referência escalar pelo seu valor sucessor e antecessor.

Quadro 11 – Operadores de Mutação de Constantes

Operador	Descrição
Cccr	Troca constantes por todas constantes do programa.
Ccsr	Troca referências escalares por constantes.
CRCR	Troca constantes por: 0,1, -1, dependendo do tipo de referência

Quadro 12 – Operadores de mutação de Operadores

Operador	Descrição
OAAA	Troca atribuição aritmética por outra atribuição aritmética.
OAAN	Troca operador aritmético por outro aritmético.
OABA	Troca atribuição aritmética por operador de atribuição *bitwise*.
OABN	Troca operador aritmético por operador *bitwise*.
OAEA	Troca atribuição aritmética por operador de atribuição plana.
OALN	Troca operador aritmético por operador lógico.
OARN	Troca operador aritmético por operador relacional.
OASA	Troca atribuição aritmética por operador atribuição de deslocamento.
OASN	Troca operador aritmético por operador de deslocamento.
OBAA	Troca atribuição *bitwise* por atribuição aritmética.
OBAN	Troca operador *bitwise* por operador aritmético.
OBBA	Troca atribuição *bitwise* por atribuição *bitwise*.
OBBN	Troca operador *bitwise* por operador *bitwise*.
OBEA	Troca atribuição *bitwise* por atribuição plana.
OBLN	Troca operador *bitwise* por operador lógico.
OBNG	Insere negação no operador *bitwise*.
OBRN	Troca operador *bitwise* por operador relacional.
OBSA	Troca atribuição *bitwise* por operador de atribuição de deslocamento.
OBSN	Troca operador *bitwise* por operador de deslocamento.
OCNG	Insere negação lógica.
OCOR	Troca o tipo primitivo do operador *cast*.
OEAA	Troca atribuição plana por atribuição aritmética.
OEBA	Troca atribuição plana por atribuição *bitwise*.
OESA	Troca atribuição plana por operador de atribuição de deslocamento.
Oido	Troca operador de incremento/decremento.
OIPM	Substitui os operadores de incremento/decremento que não possuem direção.
OLAN	Troca operador lógico por operador aritmético.
OLBN	Troca operador lógico por operador *bitwise*.
OLLN	Troca operador lógico por outro operador lógico.
OLNG	Insere negação lógica em condições compostas.
OLRN	Troca operador lógico por operador relacional.
OLSN	Troca operador lógico por operador de deslocamento.
ORAN	Troca operador relacional por operador aritmético.
ORBN	Troca operador relacional por operador *bitwise*.
ORLN	Troca operador relacional por operador lógico.
ORRN	Troca operador relacional por outro relacional.
ORSN	Troca operador relacional por operador de deslocamento.
OSAA	Troca atribuição de deslocamento por atribuição aritmética.
OSAN	Troca operador de deslocamento por operador aritmético.
OSBA	Troca atribuição de deslocamento por atribuição *bitwise*.
OSBN	Troca operador de deslocamento por operador *bitwise*.
OSEA	Troca atribuição de deslocamento por atribuição plana.
OSLN	Troca operador de deslocamento por operador lógico.
OSRN	Troca operador de deslocamento por operador relacional.
OSSA	Troca atribuição de deslocamento por outra de deslocam.
OSSN	Troca operador de deslocamento por outro de deslocam.

A.2 – Operadores de Mutação de Interface: Ferramenta *Proteum/IM*

Os operadores de mutação de interface estão divididos em dois grupos: Grupo-I - realiza a mutação dentro da função que está sendo chamada (Quadro 13) e Grupo-II - realiza a mutação no ponto de chamada à outra função (Quadro 14). Em [DEL97c] podem ser encontradas mais informações a respeito desses operadores.

Na descrição dos operadores de mutação de interface aparecem alguns conjuntos que são definidos a seguir:

- *P* – conjunto de parâmetros formais utilizados na chamada da função;
- *G* – conjunto de variáveis globais utilizadas na função chamada;
- *L* – conjunto de variáveis declaradas na função chamada (variáveis locais);
- *E* – conjunto de variáveis globais não utilizadas na função chamada;
- *C* – conjunto de constantes utilizadas na função chamada; e
- *R* – conjunto de constantes requeridas.

Quadro 13 – Operadores do Grupo I

Operador	Descrição
I-CovAllEdg	Garante cobertura de desvios.
I-CovAllNod	Garante cobertura de nós.
I-DirVarAriNeg	Acrescenta negação aritmética em variáveis de interface.
I-DirVarBitNeg	Acrescenta negação de bit em variáveis de interface.
I-DirVarIncDec	Acrescenta incremento (++) e decremento (--) em variável de interface.
I-DirVarLogNeg	Acrescenta negação lógica em variáveis 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-IndVarAriNeg	Acrescenta negação aritmética em variáveis não de interface.
I-IndVarBitNeg	Acrescenta negação de bit em variáveis não de interface.
I-IndVarIncDec	Acrescenta incremento (++) e decremento (--) em variável não de interface.
I-IndVarLogNeg	Acrescenta negação lógica em variáveis 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.
I-RetStaDel	Elimina comando *return*.
I-RetStaRep	Troca comando *return*.

Quadro 14 – Operadores do Grupo II

Operador	Descrição
II-ArgAriNeg	Acrescenta negação aritmética antes de argumento.
II-ArgBitNeg	Acrescenta negação de bit antes de argumento.
II-ArgDel	Elimina argumento.
II-ArgIncDec	Incrementa e decrementa argumento.
II-ArgLogNeg	Acrescenta negação lógico antes de 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.
II-FunCalDel	Elimina chamada de função.

Apêndice B

As Ferramentas de Teste *Proteum* e *Proteum/IM*

As ferramentas *Proteum* e *Proteum/IM* estão disponíveis para os sistemas operacionais SunOS e Linux e configuradas para o teste de programas escritos na linguagem C. Entretanto, devido ao aspecto multi-linguagem, essas ferramentas também podem ser configuradas para o teste de programas escritos em outras linguagens. A arquitetura e implementação dessas ferramentas são similares (em [DEL96a, DEL97c] podem ser encontradas informações detalhadas a respeito da arquitetura dessas ferramentas). A diferença existente entre elas é, basicamente, o conjunto de operadores de mutação que cada uma utiliza e o fato de que a *Proteum* destina-se ao teste de unidade enquanto que a *Proteum/IM* oferece características para testar a conexão entre as unidades, ou seja, teste de integração [VIN97].

A *Proteum* apresenta 71 operadores de mutação divididos em quatro classes [DEL93a]: mutação de comandos (*statement mutations*), mutação de operadores (*operator mutations*), mutação de variáveis (*variable mutations*) e mutação de constantes (*constant mutations*). Essa divisão permite que se escolham os operadores de acordo com a classe de erros que se deseja revelar, de modo que a geração dos mutantes possa ser feita em etapas ou que se possa dividir a atividade de teste entre vários testadores trabalhando independentemente.

A *Proteum/IM* possui 33 operadores de mutação divididos em dois grupos. Como descrito anteriormente, considerando a conexão *f-g*, o primeiro grupo aplica mutações dentro do corpo da função *g* e o segundo aplica mutações no ponto onde a função *g* é chamada dentro de *f* [DEL97a].

Ambas as ferramentas oferecem ao testador recursos para avaliar a adequação de um conjunto de casos de teste *T* para um determinado programa *P* e com o resultado dessa avaliação o testador pode melhorar o conjunto *T* visando a satisfazer o critério Análise de Mutantes ou Mutação de Interface. Com isso, essas ferramentas podem ser utilizadas não só como instrumento de avaliação de casos de teste, mas também para a seleção dos mesmos.

As operações mínimas suportadas pelas ferramentas são [DEL96a]:

- manipulação de casos de teste: execução, inclusão e exclusão;
- manipulação de mutantes: criação, execução e análise;
- análise de adequação: escore de mutação e relatórios estatísticos.

As ferramentas *Proteum* e *Proteum/IM* realizam algumas dessas operações de maneira completamente automatizada – como a execução dos mutantes – e outras através da intervenção do testador – como a análise de equivalência de mutantes. Além disso, visando a facilitar a condução de experimentos, diversas características foram adicionadas às ferramentas. É o caso, por exemplo, de permitir que cada mutante seja executado com todos os casos de teste, independentemente de estar vivo ou morto (teste tipo *research*). Tal característica, embora não seja o comportamento normal, permite a coleta de dados sobre a eficiência dos operadores de mutação, além de auxiliar na determinação de estratégias de minimização de conjunto de casos de teste [DEL97c].

A condução dos testes utilizando a *Proteum* ou a *Proteum/IM* é feita através das sessões de teste. A base de dados referente a uma sessão de teste é composta basicamente de: uma base de dados com informações sobre os casos de teste utilizados, uma base de dados com informações sobre os mutantes e alguns arquivos intermediários que descrevem o programa em teste. Uma sessão de teste constitui-se de uma seqüência de operações realizadas sobre essa base de dados através da chamada aos programas que compõem as ferramentas. Desse modo, pode-se iniciar uma sessão de teste (através da criação da base de dados), executar diversas operações, interromper a sessão e retomá-la posteriormente a partir do ponto em que foi interrompida [DEL97c].

Os programas que permitem realizar as operações citadas acima estão divididos em dois grupos (Quadro 15). O primeiro grupo é composto por programas básicos que agem diretamente na base de teste que caracteriza uma sessão. O segundo é composto por programas utilitários que usam os programas básicos para realizar algumas operações específicas durante uma sessão de teste [DEL97b].

Quadro 15 – Programas que Compõem as Ferramentas *Proteum* e *Proteum/IM* [DEL97c]

Programas Básicos	
li	transforma um programa C em uma representação intermediária chamada LI
li2nli	cria o grafo de programa e adiciona informações sobre os nós do grafo na representação intermediária (LI)
ptest	cria e manipula arquivos de teste de programas, os quais descrevem as características gerais da sessão de teste
tcase	cria e manipula a base de dados de casos de teste
muta	cria e manipula a base de dados dos mutantes
exemuta	constrói o código fonte dos mutantes e executa mutantes; é também utilizado para ativar/desativar mutantes
opmuta	aplica operadores de mutação no programa original, criando descritores de mutação
report	cria um relatório sobre a efetividade dos casos de teste
Programas Utilitários	
test-new	cria uma nova sessão de teste
tcase-add	insere um caso de teste interativamente
muta-gen	gera descritores de mutação e os insere na base de dados dos mutantes
muta-view	permite visualizar e analisar os mutantes

A chamada a esses programas pode ser feita diretamente na linha de comando, através da *shell* do sistema, utilizando *scripts*, ou de forma transparente para o testador através da interface gráfica que cuida de invocar os programas necessários.

Utilizando a interface gráfica, o usuário iniciante pode conduzir uma sessão de teste explorando e aprendendo os conceitos do teste de mutação e da própria ferramenta de uma maneira fácil e controlada. Os recursos de visualização dos casos de teste e dos mutantes são melhores nesse tipo de interface o que facilita, por exemplo, a identificação dos mutantes equivalentes [DEL97c]. A maneira mais fácil para se conduzir uma sessão de teste é, sem dúvida, através da interface gráfica. Entretanto, esse tipo de interface, além de depender de constante interação do testador, é menos flexível do que chamar diretamente os programas que compõem as ferramentas.

Na Figura 10 é apresentada a tela principal da ferramenta *Proteum* ilustrando o conjunto de opções disponíveis em seus menus. A tela principal da ferramenta *Proteum/IM* é similar, apresentando as mesmas funcionalidades.

Scripts de teste reduzem o número de interações com as ferramentas e possibilitam a execução de longas sessões de teste em *batch*: o usuário pode construir um programa

especificando o teste a ser realizado e a ferramenta executa esse programa, reduzindo o tempo gasto na atividade de teste. Além disso, os *scripts* têm se mostrado de grande utilidade na condução de estudos empíricos que requerem que uma seqüência de passos seja efetuada várias vezes até que se alcance uma significância estatística dos resultados.

Cabe ressaltar que a elaboração de *scripts* exige um esforço de programação e completo domínio tanto dos conceitos sobre o teste baseado em mutação quanto dos próprios programas que compõem as ferramentas, devendo ser utilizado pelo testador mais experiente [DEL97c].

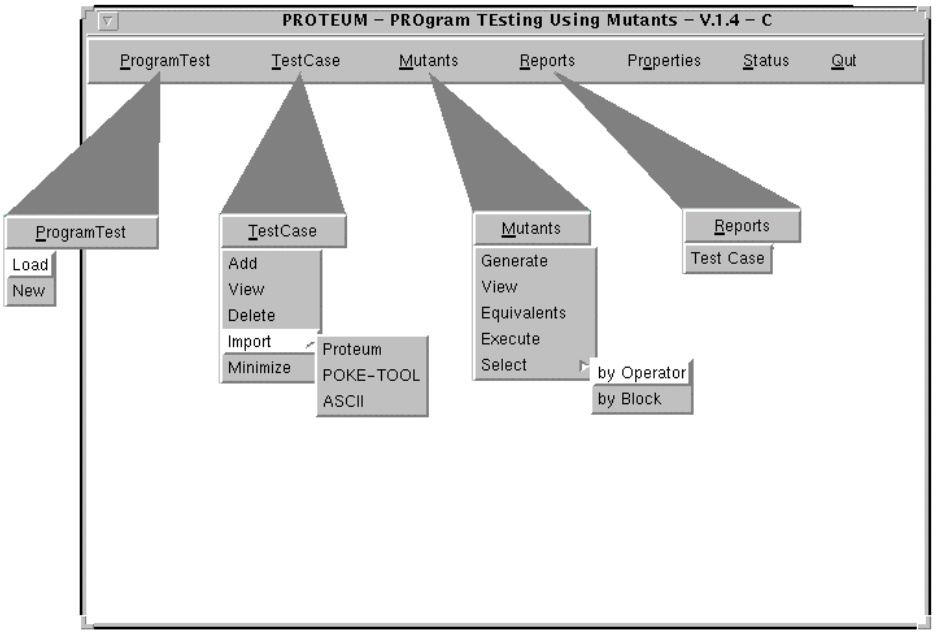


Figura 10 – Tela Principal da Ferramenta *Proteum* [DEL93c]

A seguir, a título de ilustração, são apresentadas algumas telas das ferramentas *Proteum* e *Proteum/IM*. Informações detalhadas sobre as ferramentas podem ser encontradas em [DEL96a, DEL97c].

Na Figura 11 é apresentada a tela de criação de uma sessão de teste na ferramenta *Proteum*. O testador deve indicar o nome da sessão de teste, nome do programa fonte a ser testado, o nome do programa executável e o comando de compilação que foi utilizado na compilação do programa fonte. Além disso, testador pode decidir pelo tipo de teste que deseja realizar (*test* ou *research*) e se deseja testar todas as unidades que compõem o programa ou alguma unidade específica. A diferença básica em relação à *Proteum/IM* é que ao invés de se escolher a unidade a ser testada deve-se escolher por conexões.

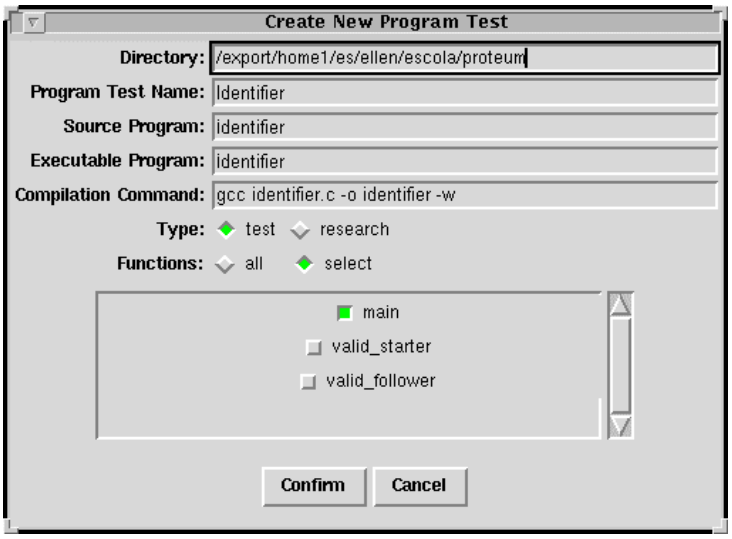
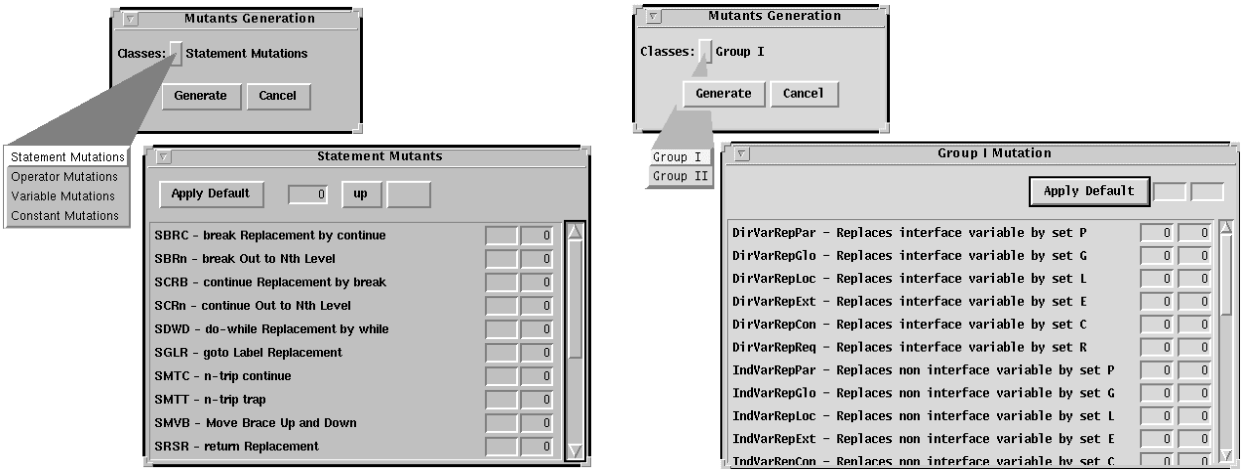


Figura 11 – Tela de Criação da Sessão de Teste

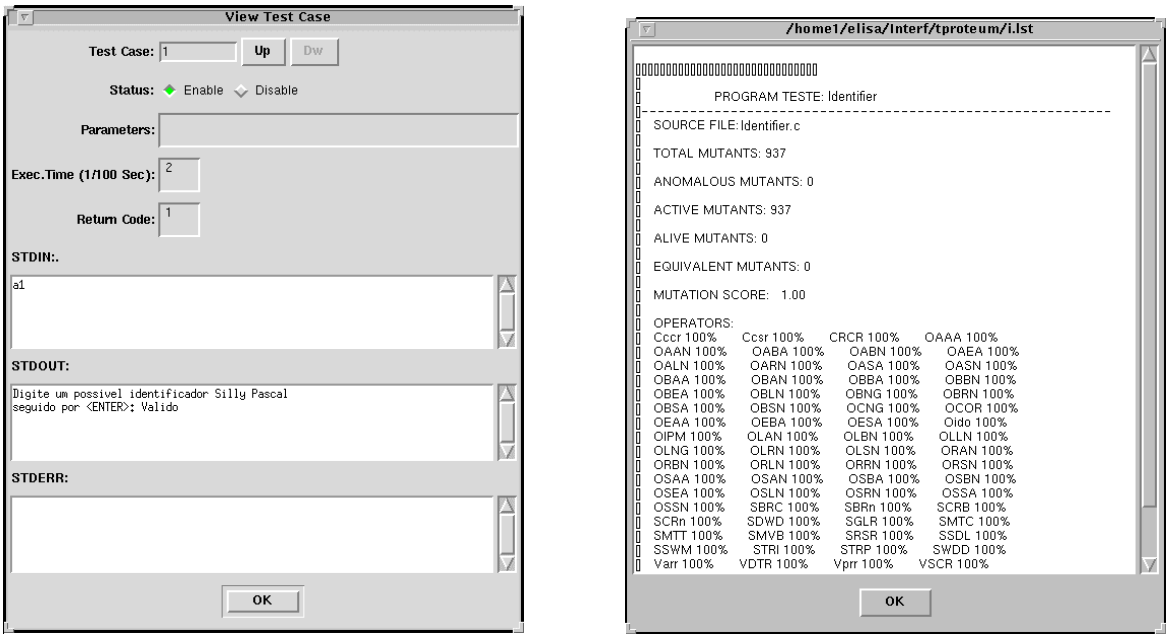


(a) (b)

Figura 12 – Geração de Mutantes: (a) Operadores *Proteum* e (b) Operadores da *Proteum/IM*

A Figura 12 ilustra a tela de geração de mutantes das ferramentas *Proteum* e *Proteum/IM*. Uma funcionalidade importante a ser observada é que ambas as ferramentas permitem que se especifique a porcentagem de mutantes a serem gerados por operador e, no caso da *Proteum/IM*, é possível ainda especificar a quantidade de mutantes que se deseja gerar por ponto de mutação.

Além dessas funcionalidades, as ferramentas permitem ainda a visualização dos casos de teste (Figura 13(a)), a geração de relatórios a respeito do andamento da atividade de teste (Figura 13(b)) e a visualização de mutantes (Figura 14), que permite ao testador comparar um mutante com o programa original e decidir pela equivalência ou não.



(a) (b)

Figura 13 – Telas da Ferramenta *Proteum*: (a) Visualização de Casos de Teste e (b) Relatório

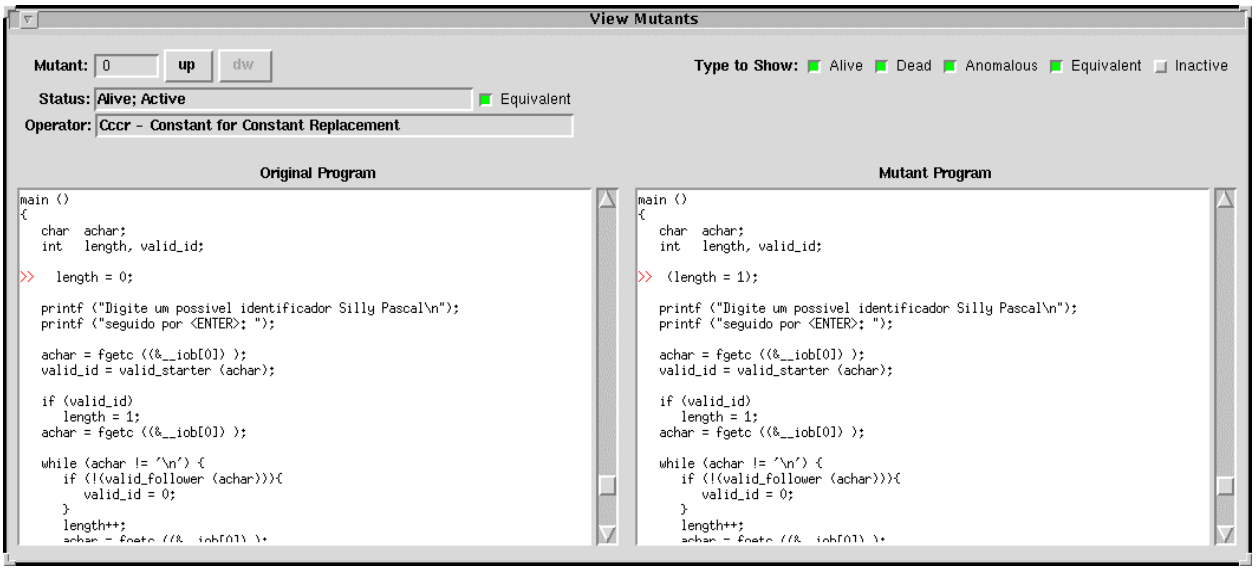


Figura 14 – Visualização de Mutante