UMA VISÃO SOBRE ANÁLISE DE MUTANTES E O AMBIENTE DE TESTE PROTEUM

AN OVERVIEW OF MUTATION ANALYSIS AND OF THE PROTEUM TESTING TOOL

Prof. Dr. José Carlos MALDONADO*
Márcio Eduardo DELAMARO**
Marcos Lordello CHAIM***
Prof. Dr. Mario JINO****

ABSTRACT

With the advance in hardware technology, Mutation Analysis - an error based testing criterion - has been recently investigated by many researchers, and evidences have been found that it is an attractive and practical testing criterion for software production. The necessity and relevance of testing tools are widely recognized and efforts have been carried out for developing tools to support application of this criterion.

In this paper, Mutation Analysis, as well as related work, is reviewed, aiming at specifying the main aspects of a multilanguage testing tool, named Proteum (**Pro**gram **Te**sting **U**sing **M**utants), which supports application of Mutation Analysis criterion for testing C programs. In addition to features usually available in similar tools, Proteum offers facilities for conducting experimental studies. It has been used in some empirical studies; a brief view of results obtained is given.

KEY WORDS: error based testing, mutation analysis, testing criterion, testing tool.

RESUMO

Com o avanço da tecnologia de hardware, a Análise de Mutantes - um dos critérios de teste baseados em erros - tem sido mais recentemente investigada por diversos pesquisadores e se mostrado um critério de teste atrativo e factível para o uso na produção de software. A relevância e necessidade de ferramentas de teste são amplamente reconhecidas pela comunidade e esforços têm sido feitos para o desenvolvimento de ferramentas de apoio a este critério.

A Análise de Mutantes, bem como os principais trabalhos relacionados com este critério são revistos, procurando fornecer subsídios para evidenciar a necessidade de implementação de uma ferramenta de teste chamada Proteum (**Pro**gram **Te**sting **U**sing **M**utants). Essa ferramenta apóia a aplicação do critério Análise de Mutantes para programas C e, além de características comuns a outras ferramentas de teste, incorpora algumas características próprias como facilidades para a realização de experimentos empíricos. O Proteum foi utilizado em alguns experimentos empíricos cujos resultados são sucintamente descritos.

PALAVRAS-CHAVE: teste baseado em erros, análise de mutantes, critério de teste, ferramenta de teste.

(*) Professor do Departamento de Ciências da Computação e Estatística do Instituto de Ciências Matemáticas de São Carlos - ICMSC - USP.
(**) Aluno de Doutorado do Instituto de Física de São Carlos - IFSC - USP.
(***) Pesquisador do Centro Nacional de Pesquisa Tecnológica em Informática para Agricultura da Empresa Brasileira de Pesquisa Agropecuária - EMBRAPA.
(****) Professor do Departamento de Engenharia de Computação e Automação Industrial da Faculdade de Engenharia Elétrica - FEE - UNICAMP.

1 - INTRODUÇÃO

A Engenharia de Software constitui-se de um conjunto de métodos e técnicas que procuram dar ao desenvolvimento de software uma abordagem de engenharia de produto que requer planejamento, análise, projeto, implementação, teste e manutenção. Seu principal objetivo é produzir software de alta qualidade e com um custo baixo [PRE92].

Durante todo o ciclo de vida do software, paralelamente à sua construção, um conjunto de atividades denominadas de **Garantia de Qualidade de Software** é conduzido com o objetivo de assegurar que tanto o processo de desenvolvimento quanto o produto final atinjam níveis de qualidade especificados. Dentre essas atividades, a atividade de teste de software é uma das técnicas de validação mais utilizadas. A atividade de teste pode ser descrita essencialmente como: observação do comportamento do programa através de sua execução sobre um conjunto de dados.

A própria atividade de teste está sujeita a erros e deve ser submetida ao controle de qualidade, através da aplicação de técnicas de teste. Ferramentas de teste também são meios importantes para procurar níveis de qualidade satisfatórios na atividade de teste. Além disso, a disponibilidade de ferramentas que apoiem a aplicação de técnicas de teste permite que estudos sejam conduzidos com o intuito de avaliar e revelar características importantes das técnicas de teste e descobrir como elas se inter-relacionam, onde elas se sobrepõem e como elas podem se complementar.

Este artigo descreve um ambiente de teste denominado Proteum (**Pro**gram **Te**sting **U**sing **M**utants) que apóia a aplicação da **Análise de Mutantes**, um critério que se enquadra dentro da técnica de teste baseada em erros.

A Seção 2 apresenta uma revisão sobre técnicas e critérios de teste e a Seção 3 trata com mais detalhes o critério Análise de Mutantes. Na Seção 4 é descrita, através de um exemplo, a forma de aplicação deste critério dentro do ambiente de teste Proteum. A Seção 5 descreve alguns experimentos realizados utilizando-se a ferramenta e a Seção 6 traz as conclusões e trabalhos futuros.

2 - TESTE DE SOFTWARE

O teste de software, segundo Pressman [PRE92], é um elemento crítico para a garantia da qualidade de software e representa a última revisão da especificação, projeto e codificação. Apesar das limitações características da atividade de teste (em geral, não se consegue, através de teste, mostrar que um programa está correto), sua aplicação de maneira sistematizada e bem organizada pode conferir ao software algumas características mínimas que são importantes no estabelecimento da qualidade do produto e relevantes para o seu processo de evolução.

A atividade de teste não pode ser conduzida na força bruta. Torna-se necessária a aplicação de técnicas que dêem indicações de como testar o software, quando parar o teste e que forneçam uma medida do nível de confiança e qualidade alcançados com os testes realizados. Duas questões são importantes:

"Como os dados de teste devem ser selecionados?" e

"Como se pode dizer que um programa foi testado suficientemente?"

Assim, definem-se: **método de seleção de casos de teste,** como um procedimento para escolher casos de teste; e, **critério de adequação de casos de teste,** como um predicado usado para avaliar casos de teste. Em geral, um critério de adequação é um dos itens utilizados para decidir o encerramento das atividades de teste [MAL91].

Basicamente, pode-se classificar um critério dentro de três técnicas de teste: **funcional, estrutural** e **baseada em erros**. O que diferencia estas três técnicas é a origem da informação usada para estabelecer os requisitos para avaliar ou construir casos de teste.

A técnica funcional de teste, também chamada de "teste caixa preta", é usada principalmente na construção de casos de teste; utiliza essencialmente a especificação funcional do programa para escolher os casos de teste a serem empregados, não considerando detalhes de uma particular implementação. Como exemplos de critérios funcionais podem ser citados: Grafo de Causa-efeito, Particionamento em Classes de Equivalência e Análise de Valor Limite [PRE92].

Na técnica estrutural, às vezes chamada de "teste caixa branca", os aspectos de implementação são importantes na escolha dos casos de teste. Em geral, a técnica estrutural utiliza uma representação para programas conhecida como **grafo de fluxo de controle** ou **grafo de programa**; os critérios estruturais caracterizam-se através da determinação de quais componentes do grafo de programa devem ser exercitados pela aplicação dos casos de teste. Critérios estruturais podem basear-se em diferentes tipos de estruturas para determinar quais partes do programa têm sua execução requerida.

Critérios baseados no fluxo de controle usam apenas características do fluxo de execução, como comandos e desvios, para derivar os elementos requeridos. **Critérios baseados no fluxo de dados** exploram ainda associações entre pontos do programa onde atribui-se valor a uma variável e pontos do programa onde este valor é utilizado; com base nestas associações são determinados os elementos a serem exercitados. **Critérios baseados na complexidade** utilizam informação sobre a complexidade do programa para determinar os requisitos de teste [PRE92].

Os critérios que se enquadram na técnica de teste baseada em erros utilizam informação sobre erros mais freqüentes no processo de desenvolvimento de software e sobre tipos específicos de erros que se deseja localizar [DEM91]. Dentre estes critérios está a Análise de Mutantes, descrita na próxima seção.

3 - CRITÉRIO ANÁLISE DE MUTANTES

A Análise de Mutantes [DEM91, DEL93a] foi proposta por pesquisadores da Universidade YALE e do Georgia Institute of Technology no final da década de 70. Pode-se descrever a aplicação do critério da seguinte maneira: o testador fornece um programa **P** a ser testado e um conjunto de casos de teste **T** cuja qualidade deseja-se avaliar. O programa **P** é executado sobre **T**; se apresentar resultados incorretos, foi revelado um erro e o teste termina. Caso contrário, o programa pode conter erros que o conjunto **T** não conseguiu revelar.

São criados então programas P_1, P_2, ... P_k que são **mutantes** de P e que diferem de **P** apenas na existência de erros simples. Por exemplo, se **P** contém a expressão A < B, um dos mutantes P_i irá ter no lugar desta, a expressão A <= B. Em seguida, cada um dos mutantes é executado sobre **T**. Se um mutante P_i apresenta resultados diferentes de **P** diz-se que esse mutante está **morto**; nesse caso, **T** conseguiu identificar o "erro" no mutante ou, mais precisamente, conseguiu revelar a diferença entre **P** e P_i. Por outro lado, se P_i apresenta os mesmos resultados para todos os casos de teste de **T**, diz-se que ele continua **vivo**. Isto pode ocorrer por dois motivos: ou porque **T** não contém casos de teste capazes de distinguir P_i de **P** ou porque ambos os programas, apesar de sintaticamente diferentes, são funcionalmente equivalentes. No primeiro caso, novos casos de teste podem ser adicionados a **T** para "matar" o mutante. Já no caso de mutantes equivalentes, nenhum caso de teste será capaz de distingui-los, pois seus resultados serão sempre iguais aos resultados de **P**.

O objetivo é achar um conjunto de casos de teste que consiga matar o maior número possível de mutantes não equivalentes. Em outras palavras, a adequação de **T** é medida pela sua capacidade de "matar mutantes". Através do **escore de mutação** tem-se uma medida objetiva da adequação de **T**:

$$ms(T) = \frac{\#\ \textit{mutantes mortos por T}}{\#\ \textit{mutantes não equivalentes a P}}$$

Alguns dos pontos críticos na aplicação prática do critério Análise de Mutantes são sua complexidade e seu custo de aplicação, em termos de tempo. Mesmo para programas simples, pode ser gerado um número muito grande de mutantes, o que já descarta a possibilidade de aplicar-se o critério manualmente. Em muitos casos, mesmo com o auxílio de uma ferramenta de teste, tornam-se necessárias estratégias para minimizar o tempo de aplicação do critério. Algumas dessas estratégias podem ser encontradas na literatura e, em geral, procuram otimizar o custo de aplicação do critério, diminuindo de forma criteriosa o número de mutantes [MAR90, MAT93, WON94] ou utilizando arquiteturas de alto desempenho na condução do teste [MAT88, KRA91, CHO89].

4 - AMBIENTE PROTEUM E A APLICAÇÃO DA ANÁLISE DE MUTANTES

O Proteum [DEL93b, DEL94] é um ambiente de teste, implementado em plataforma de estações de trabalho SUN, que tem como objetivo apoiar a aplicação do critério Análise de Mutantes. Além disto, incorpora características que auxiliam na condução de experimentos empíricos, permitindo que avaliem-se as vantagens da aplicação do critério e que sejam realizados estudos comparativos com outros critérios.

Na definição do Proteum procurou-se levantar características que permitissem alcançar esses dois objetivos principais, além de identificar em outras ferramentas de teste existentes, aspectos e funcionalidades importantes para ambientes deste tipo.

Uma importante característica do Proteum é ser uma ferramenta multilinguagem. Com algum esforço de

programação e alteração de tabelas, principalmente as tabelas que dirigem a análise sintática, pode-se configurar a ferramenta para tratar diferentes linguagens alvo. Inicialmente o ambiente está configurado para tratar de programas escritos em linguagem C.

4.1- Sessões de Teste

Para realizar o teste de um programa **P**, utilizando como critério a Análise de Mutantes, um conjunto essencial de tarefas deve ser executado:

. definição do conjunto de casos de teste **T**;

. execução do programa P utilizando **T**;

. geração de mutantes;

. execução dos mutantes;

. comparação com os resultados de **P**;

. análise dos mutantes vivos; e

. cálculo do escore de mutação.

Para permitir que essas tarefas sejam realizadas, algumas automaticamente e algumas com a intervenção do testador, a ferramenta Proteum mantém uma base de dados que contém informação sobre os casos de teste a serem utilizados e sobre os mutantes gerados e seus estados atuais. Inicialmente, para testar um programa **P**, o usuário deve criar um teste para **P**, fornecendo à ferramenta dados pertinentes como: nome dos arquivos do programa fonte e do programa executável, diretório onde estão estes arquivos, etc. Por exemplo, para testar o programa da Figura 1 que está no arquivo **myprog.c**, o usuário deve selecionar no menu principal do Proteum o botão **Program Test** e em seguida a opção **New** (Figura 2).

```
#include   <stdio.h>

main()
{
int        a, b;

scanf("%d%d, &a, &b);
if (a > b)
   printf("\n%d eh maior que %d", a, b);
else
if (a < b)
   printf("\n%d eh menor que %d", a, b);
else
   printf("\n%d eh maior que %d", a, b);
return 0;
}
```

Figura 1 - Programa myprog.c

Feito isso, o usuário recebe um painel onde deve fornecer as características do teste que deseja criar (Figura 3).

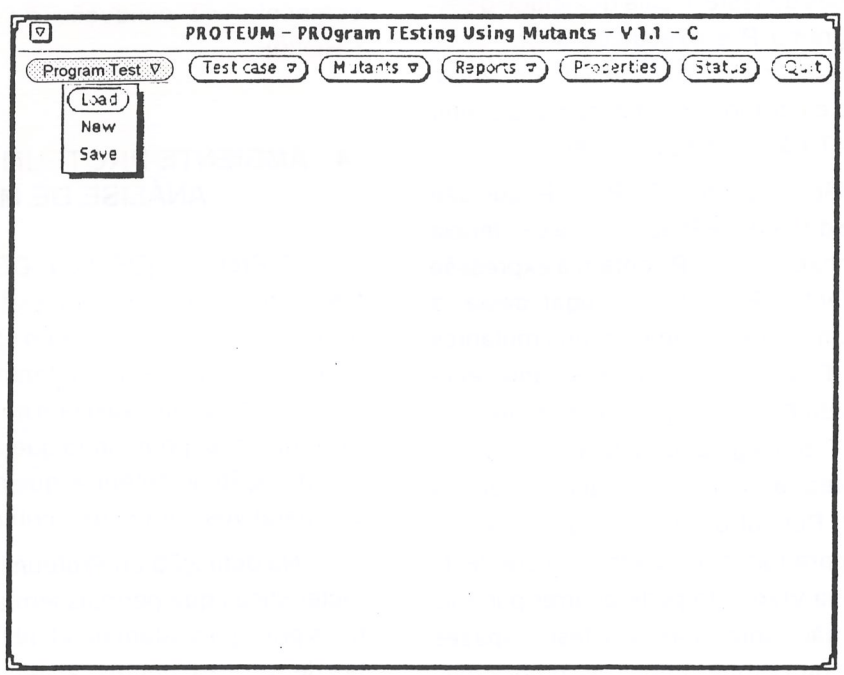


Figura 2 - Trabalhando com Sessões de Teste

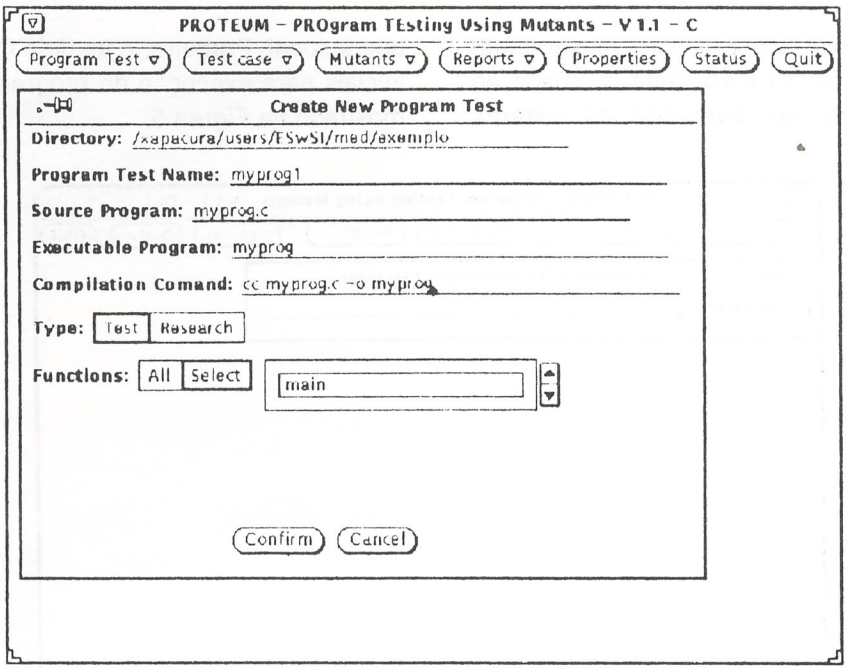


Figura 3 - Criando o Teste de um Programa

A operação do Proteum é dirigida por **sessões de teste**, ou seja, uma vez criado o teste de um programa, o testador pode interromper sua execução em qualquer ponto e futuramente retomá-la do ponto em que foi abandonada através da opção **Load**. Através da opção **Save** o testador garante que todas operações realizadas durante a sessão de teste serão realmente efetivadas e salvas para que a sessão possa ser retomada futuramente.

4.2 Casos de Teste

A definição do conjunto de casos de teste T é feita através de algumas operações, mostradas na Figura 4, que permitem ao testador incluir e excluir casos de teste ou determinar quais devem ser efetivamente avaliados. Embora seja possível apoiar de forma automatizada a geração de casos de teste [DEM91], o Proteum não o faz, ficando esta tarefa por conta do testador.

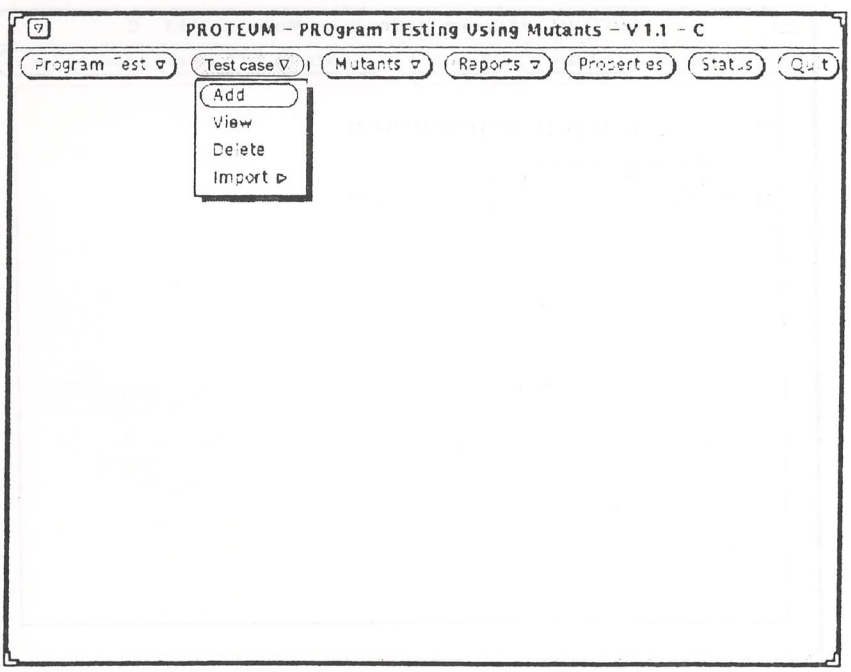


Figura 4 - Trabalhando com Casos de Teste

A inclusão de casos de teste é feita interativamente, de maneira simples, como se fosse uma chamada ao programa **P** através de um comando do sistema operacional. Primeiro, o usuário fornece os parâmetros iniciais para execução do programa, através do painel mostrado na Figura 5.

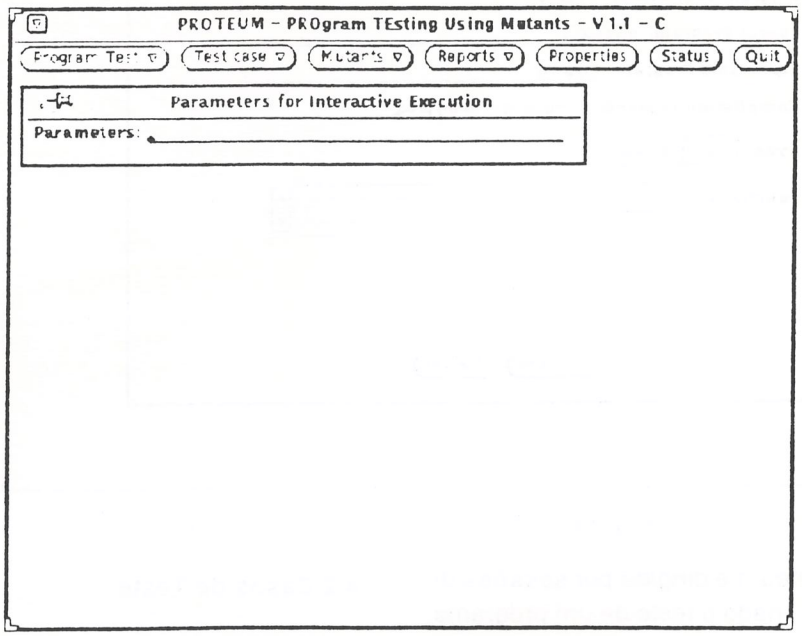


Figura 5 - Fornecendo Parâmetros para um Caso de Teste

Em seguida, inicia-se a execução do programa numa janela que simula um terminal, onde o usuário interage com o programa. Por exemplo, no caso do programa **myprog**, o programa aguarda a digitação dos dois número a serem comparados; é a situação que exibe a Figura 6.

Após fornecer todos os dados, ao final da execução, o usuário pode avaliar se o resultado apresentado é correto ou não, escolhendo entre incluir ou não o caso de teste. Para isso, o Proteum pede ao usuário a confirma-

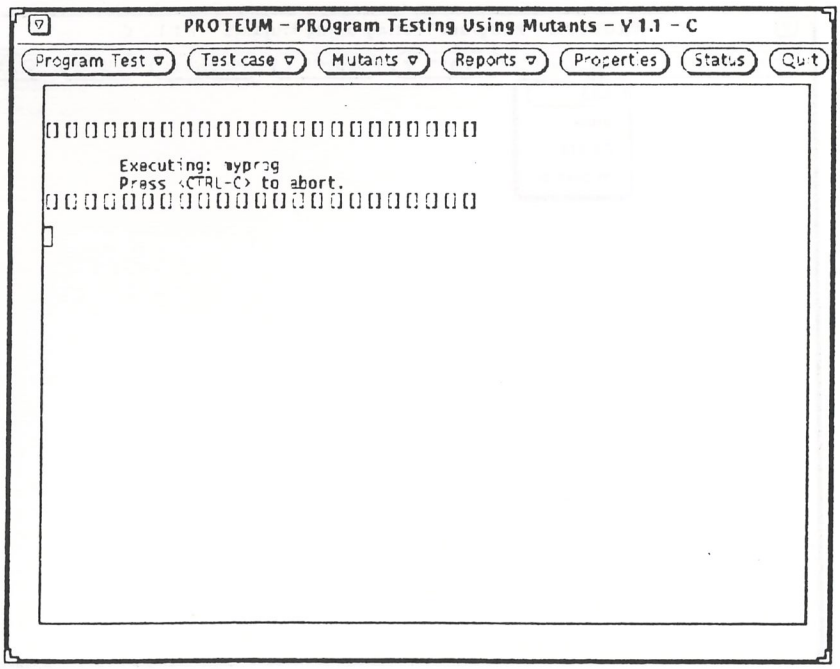


Figura 6 - Incluindo um Caso de Teste

ção, como mostra a Figura 7. Desta maneira o usuário tem a oportunidade de verificar se o resultado apresentado por **P** para o caso de teste fornecido é o esperado ou se um erro foi revelado.

Os dados fornecidos pelo usuário e os resultados apresentados na tela pelo programa em teste são armazenados pelo Proteum, caracterizando os casos de teste a serem empregados no teste corrente.

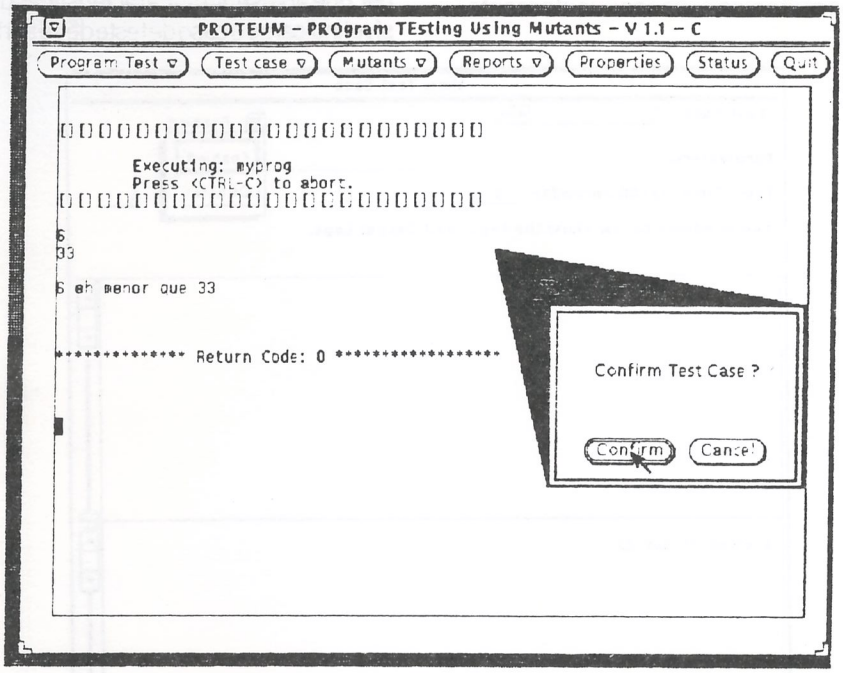


Figura 7 - Confirmando um Caso de Teste

Podem ser incluídos também casos de teste através da importação a partir de outras sessões de teste do Proteum ou da POKE-TOOL, ferramenta de teste que apóia a aplicação de critérios de teste baseados em fluxo de dados [CHA91] ou ainda a partir de arquivos ASCII convencionais.

A exclusão de casos de teste pode ser realizada física ou logicamente. Para eliminar fisicamente, através da opção **Delete**, o testador fornece os números dos casos de teste a serem eliminados (Figura 8), que então deixarão de pertencer ao conjunto **T**.

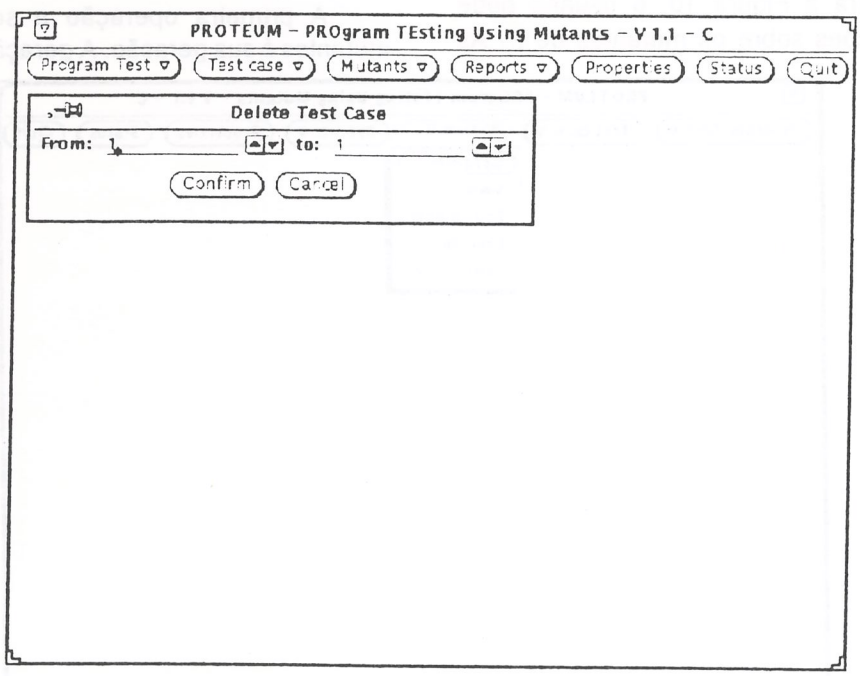


Figura 8 - Eliminando Casos de Teste

Tem-se também a possibilidade de desabilitar/habilitar, ou seja, excluir/reincluir logicamente casos de teste. Um caso de teste desabilitado continua fisicamente presente no conjunto **T** mas não logicamente, não sendo utilizado na execução dos mutantes e, conseqüentemente, na avaliação de **T**.

Para habilitar/desabilitar casos de teste, o testador deve selecionar a opção View do menu de casos de teste. Pode então selecionar o número do caso de teste a ser visualizado, através de um painel como o mostrado na Figura 9. Neste painel, que apresenta as características de um caso de teste, é possível indicar se o caso de teste deve ser habilitado ou desabilitado.

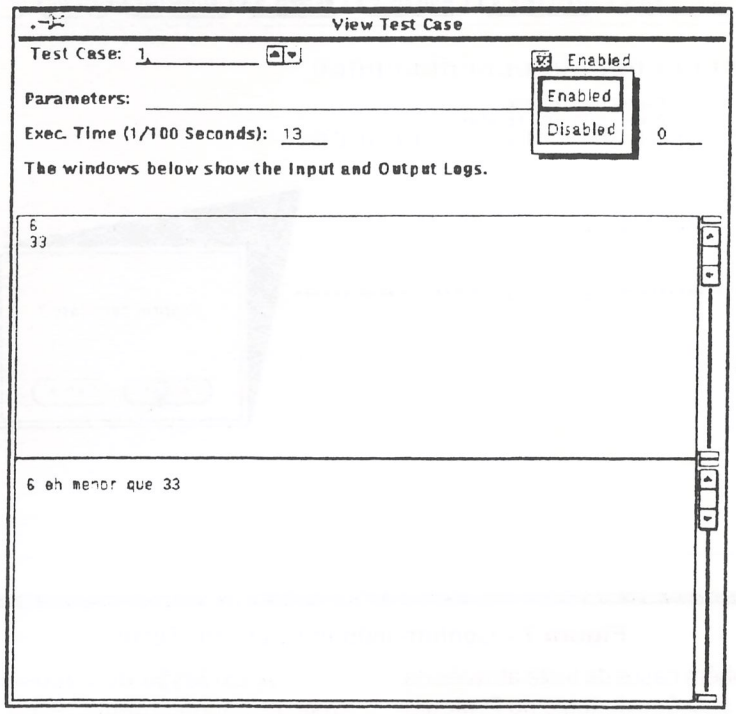


Figura 9 - Visualizando um Caso de Teste

4.3 - Mutantes

Conforme mostra a Figura 10, o usuário pode realizar cinco operações sobre os mutantes de **P**. As principais são: gerar (**Generate**), visualizar (**View**) e executar (**Execute**) os mutantes.

A primeira operação a ser efetuada sobre os mutantes é sua geração. A geração dos mutantes de **P** é

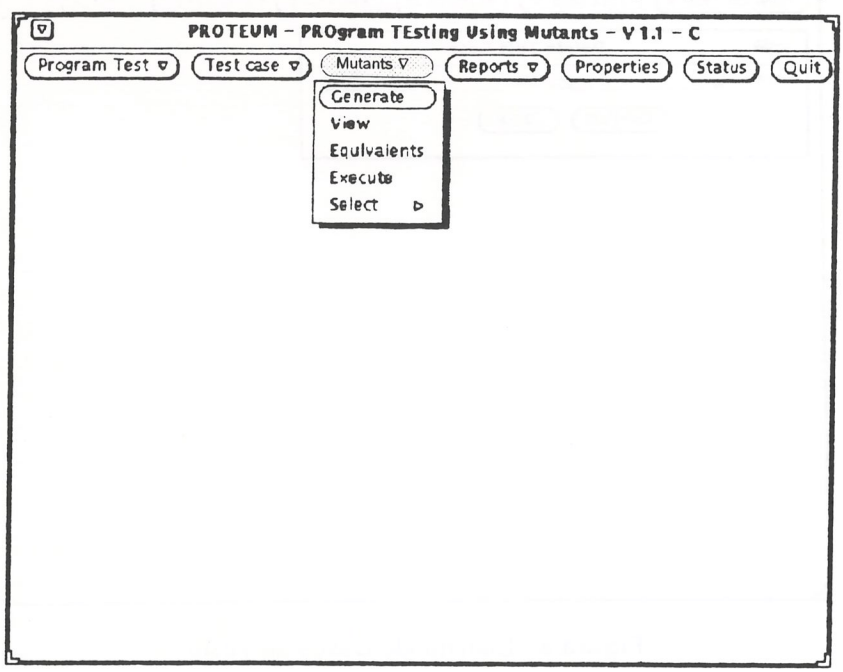


Figura 10 - Trabalhando com Mutantes

feita automaticamente. Cabe ao testador apenas selecionar quais os tipos de mutantes que devem ser gerados. Para determinar as mutações a serem feitas em **P** foi definido um conjunto de **operadores de mutação** [AGR89]; cada operador de mutação determina um tipo de alteração sintática que deve ser feita em **P** para criar os mutantes, procurando modelar uma classe específica de erros ou garantir características mínimas de qualidade ao conjunto **T** (por exemplo, a cobertura de todos os ramos do programa).

No Proteum foram implementados 71 operadores de mutação, divididos em quatro classes: mutações de comandos; de variáveis; de constantes; e de operadores. Para gerar os mutantes o testador pode selecionar, por classes ou individualmente, para cada operador de mutação, uma porcentagem do número de mutantes a serem gerados (Figura 11). Com isso implementou-se uma estratégia de minimização do número de mutantes com o objetivo de diminuir o custo de aplicação do critério. Experimentos têm mostrado que mesmo utilizando uma pequena porcentagem dos mutantes ou apenas alguns dos operadores de mutação, consegue-se obter bons casos de teste [MAT93].

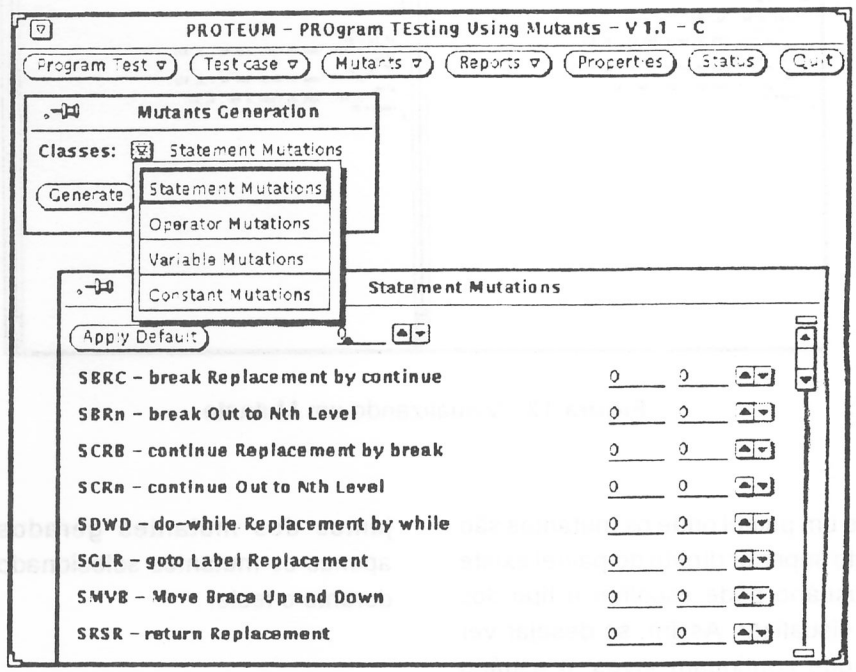


Figura 11 - Gerando Mutantes

O segundo passo para a avaliação do conjunto de casos de teste é a execução dos mutantes com este conjunto. A execução dos mutantes e comparação dos resultados também é feita de forma automática, sem intervenção do usuário. Um ponto importante a ser destacado é que ao efetuar a operação de execução dos mutantes, tem-se refletida a qualidade do conjunto de casos de teste existente no momento da execução. Isso permite que casos de teste sejam incluídos, excluídos, habilitados ou desabilitados e, ainda assim, mantenha-se a coerência entre o conjunto **T** e o estado dos mutantes. Por exemplo, considerando-se um mutante **M** que foi morto unicamente pelo caso de teste **N** que, em seguida, é desabilitado. Ao efetuar-se novamente a operação de execução dos mutantes, **M** deve ser "ressuscitado" pois o caso de teste que o matou não pertence mais ao conjunto de casos de teste (pelo menos logicamente).

Tradicionalmente, ao executar-se um mutante com um caso de teste que o mate, esse mutante deve ser descartado, sem que seja necessária sua execução com os demais casos de teste. No Proteum o usuário pode determinar um outro comportamento: aplicar sobre cada mutante todos os casos de teste, permitindo observar-se quais casos são eficientes para matar cada mutante. Este procedimento propicia o levantamento de dados importantes para condução de estudos empíricos.

A tarefa de analisar os mutantes vivos consiste em procurar identificar quais mutantes são equivalentes e quais podem ser mortos; esta tarefa, no Proteum, é feita totalmente pelo testador. No primeiro caso, os mutantes

podem ser marcados como equivalentes. Ao marcar um mutante como equivalente, o usuário altera o valor calculado para o escore de mutação, uma vez que o número de mutantes equivalentes é usado no cálculo. Já para os mutantes vivos não equivalentes, o testador pode construir novos casos de teste para eliminá-los e adicionar estes casos de teste a **T**. A análise dos mutantes é feita através da operação **View** do menu de mutantes. Assim como é feito para os casos de teste, o usuário pode visualizar um mutante fornecendo seu número ou "navegando" seqüencialmente através do conjunto de mutantes gerados, como mostra a Figura 12.

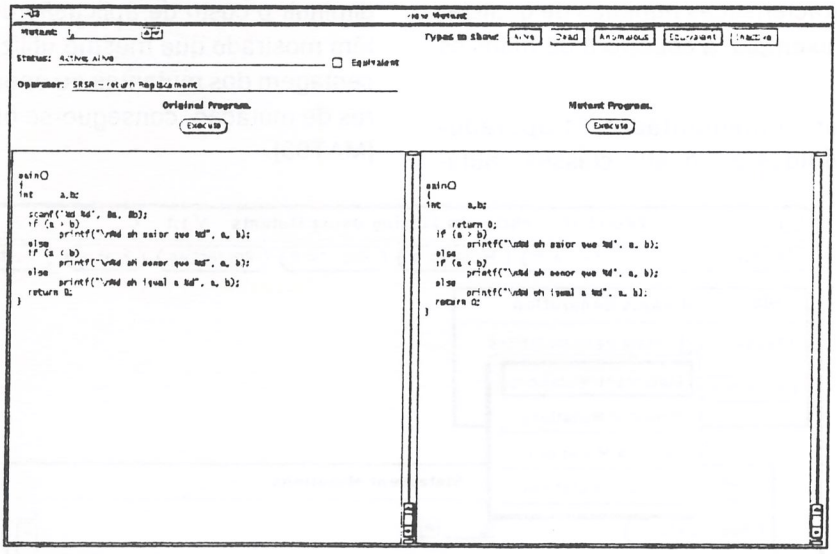


Figura 12 - Visualizando um Mutante

A Figura 12 exibe um painel onde os mutantes são apresentados. No canto superior direito do painel existe uma lista em que o usuário pode escolher o tipo dos mutantes que deseja visualizar. Assim, se desejar ver apenas os mutantes que continuam vivos, o usuário deve "desligar" os botões **Dead, Equivalent, Anomalous** (mutantes anômalos são aqueles que apresentam erros sintáticos, introduzidos pela mutação efetuada em **P**, e que por isso não podem ser executados; eles são desprezados no cálculo do escore de mutação). As janelas na base do painel mostram o programa original e o mutante. Sobre cada uma delas existem botões rotulados Execute; ao acioná-los o usuário inicia a execução interativa do programa correspondente podendo desta maneira "experimentar" casos de teste para tentar matar o mutante, antes de incluí-los.

A opção **Equivalent** do menu de mutantes é utilizada para que o testador, conhecendo a priori uma lista de mutantes equivalentes, possa marcá-los como equivalentes, todos de uma vez. A opção **Select** permite que sejam selecionados sub-conjuntos dos mutantes gerados, fazendo com que apenas os mutantes selecionados sejam considerados durante o teste.

4.4 - Outras Funções

Na seção anterior está descrita, basicamente, a forma de conduzir-se uma sessão de teste no Proteum. Algumas outras operações complementam estas operações principais. A primeira delas é iniciada ao selecionar-se o botão **Status**. O Proteum fornece informações sobre o andamento do teste. A Figura 13 mostra quais são essas informações; a mais importante delas é o escore de mutação que indica o quanto o conjunto de casos de teste está próximo da adequação para o programa em teste.

Dados sobre o andamento do teste podem também ser obtidos através de relatórios produzidos pelo Proteum. Atualmente o único tipo implementado é o relatório sobre casos de teste; nele o usuário obtém informação sobre o conteúdo e a capacidade de matar mutantes de cada

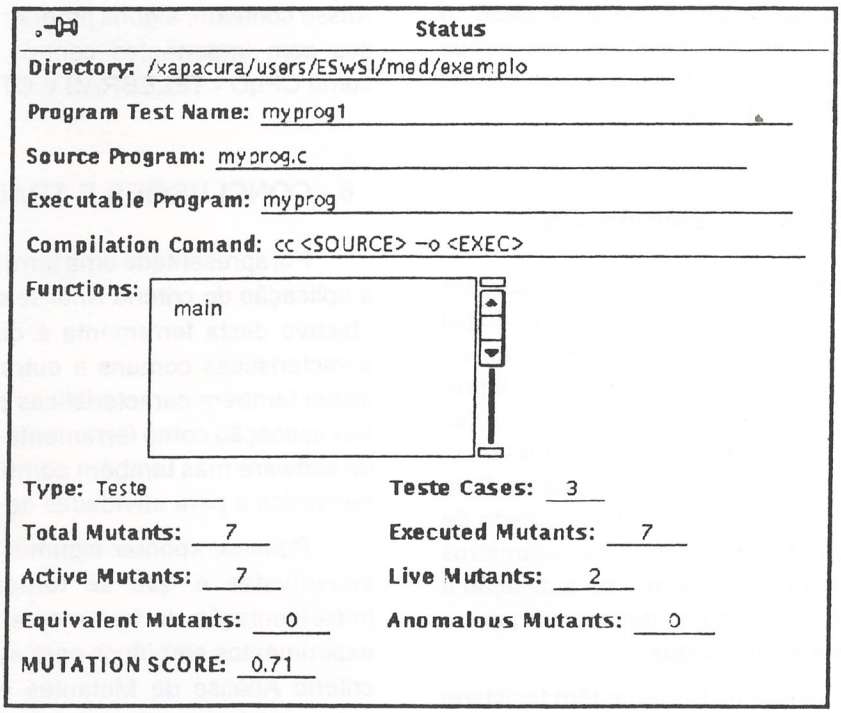
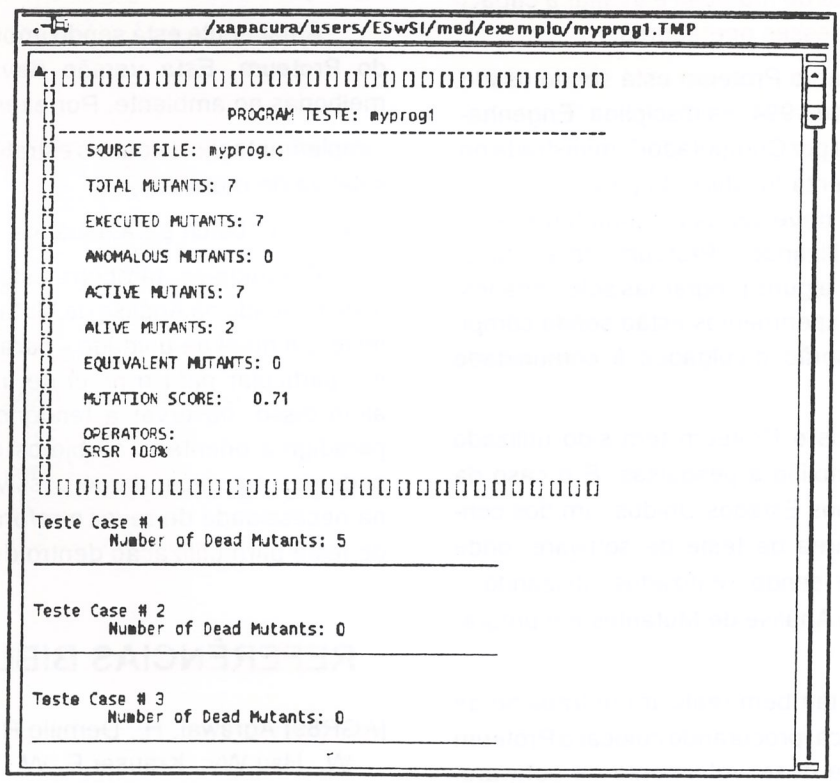


Figura 13 - Status de uma Sessão de Teste

caso de teste. O nível de detalhes de um relatório pode ser definido pelo usuário; a Figura 14 mostra um relatório que contém, para cada caso de teste, o número de mutantes mortos por ele.

O botão **Properties** permite ao usuário determinar algumas características operacionais do ambiente, como o diretório padrão, onde os arquivos de trabalho são armazenados. A opção **Quit** termina a execução da



ferramenta; se houver um teste sendo executado, o usuário deve escolher entre salvar as operações efetuadas durante a sessão de teste ou sair sem salvar, cancelando as operações realizadas.

5 - EXPERIMENTOS EMPÍRICOS

Alguns experimentos empíricos foram realizados utilizando o Proteum. O primeiro teve como principal objetivo avaliar a ferramenta tanto em termos funcionais como de desempenho [DEL93]. Foram testados 19 programas, em geral extraídos da literatura, com diferentes características. São programas de cálculo numérico, formatação de texto e um arbitrador de barramento. Este experimento serviu para demonstrar a factibilidade de utilização da Análise de Mutantes dentro de parâmetros aceitáveis de custo. Permitiu também uma avaliação a partir da qual novas características desejáveis para a ferramenta puderam ser identificadas.

Algumas características do Proteum têm facilitado a execução de alguns experimentos. É o caso, por exemplo, do trabalho apresentado no VIII SBES [WON94]. Esse experimento utilizou 10 diferentes programas para avaliar a capacidade de revelar erros de alguns subconjuntos de operadores de mutação. O procedimento básico foi o de aplicar sobre os programas em teste, que contêm erros, a Análise de Mutantes com diferentes conjuntos de operadores de mutação e avaliar a capacidade de revelar erros desses operadores.

Também no ensino o Proteum está sendo usado. No segundo semestre de 1994, na disciplina "Engenharia de Software Apoiada por Computador", ministrada no ICMSC-USP, a ferramenta foi utilizada para mostrar na prática o que é e como deve ser usada uma ferramenta de teste. Os alunos, utilizando o Proteum, desenvolveram experimentos com alguns programas selecionados; os resultados desses experimentos estão sendo compilados e brevemente serão divulgados à comunidade científica.

Em alguns centros o Proteum tem sido utilizado como ferramenta de auxílio a pesquisas. É o caso da Universidade Purdue nos Estados Unidos, um dos centros de destaque na área de teste de software, onde alguns trabalhos vêm sendo realizados utilizando o Proteum para aplicar a Análise de Mutantes em programas C.

Tem-se buscado também realizar um trabalho de transferência tecnológica, procurando colocar o Proteum em ambientes reais de desenvolvimento de software. Nesse contexto, alguns programas têm sido desenvolvidos com importantes centros de pesquisa nacionais como CPqD - TELEBRÁS e CTI.

6 - CONCLUSÕES E TRABALHOS FUTUROS

Foi apresentada uma ferramenta de teste que apóia a aplicação do critério Análise de Mutantes. O principal objetivo desta ferramenta é que ela, além de possuir características comuns a outras ferramentas de teste, inclua também características próprias, visando não só sua aplicação como ferramenta para o desenvolvimento de software mas também como instrumento para novas pesquisas e para atividades de ensino.

Pode-se apontar algumas atividades a serem desenvolvidas e que se tornaram possíveis com a implementação da ferramenta Proteum: realização de experimentos empíricos para avaliação da aplicação do critério Análise de Mutantes e sua comparação com outros critérios; utilização do Proteum em cursos de graduação e pós-graduação como instrumento para demonstração prática de conceitos ligados às atividades de desenvolvimento e teste de software; implementação de outras estratégias de minimização de mutantes; configuração para outras linguagens, com características diversas como C++, OCCAM e, em especial, linguagens de aplicação comercial como COBOL e CLIPPER.

Atualmente está sendo implementada a versão 1.2 do Proteum. Esta versão deve incorporar algumas melhorias no ambiente. Por exemplo:

. implementação de outras estratégias para minimização seletiva de mutantes;

. execução batch para usuários experientes

Pretende-se também estender os conceitos do teste baseado na Análise de Mutantes - que se limitam ao teste em nível de unidade - para outros níveis de teste, em particular para o nível de integração, procurando, além disso, observar a tendência atual que aponta o paradigma orientado a objetos como uma abordagem poderosa e prática, o que possivelmente deve implicar na necessidade de rever e reformular alguns conceitos de teste para utilização dentro desse paradigma.

REFERÊNCIAS BIBLIOGRÁFICAS

[AGR89] Agrawal, H., Demillo R. A., Hathaway B., Hsu W., Hsu Wy., Krauser E. W., Martin R. J., Mathur A.

P., Spafford E., "Design of Mutant Operators for the C Programming Language", Technical Report SERC-TR-41-P, Software Eng. Research Center, Purdue University, Mar 1989.

[CAR91] Carnassale, M., GFC - Uma Ferramenta Multilinguagem para Geração de Grafo de Programa, Dissertação de Mestrado, DCA/FEE/UNICAMP, Fev 1991.

[CHA91] Chaim, M. L., POKETOOL - Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados, DCA/FEE/UNICAMP, Abr 1991.

[CHO89] Choi, B. J., Mathur, A. P. e Pattison, B., "pMothra: Scheduling Mutants For Execution on a Hypercube", Third Symposium on Software Testing, Analysis and Verification, Key West, Dez 1989.

[DEL93a] Delamaro, M. E., "Uma Visão sobre a Aplicação da Análise de Mutantes", Notas do ICMSC, ICMSC-USP, São Carlos - SP, 1993.

[DEL93b] Delamaro, M. E., "Proteum: Um Ambiente de Teste Baseado na Análise de Mutantes", Dissertação de Mestrado, ICMSC-USP, São Carlos - SP, Out 1993.

[DEL94] Delamaro, M. E., "Proteum - Manual do Usuário, Versão 1.1 C", Relatório Técnico, ICMSC-USP, São Carlos - SP, 1994.

[DEM91] DeMillo, R. A. e Offut, A. J., "Constraint-Based Automatic Test Data Generation", IEEE Trans. on Software Eng., vol 17, Nº 9, Set 1991.

[KRA91] Krauser, E. W., Mathur, A. P. e Rego, V., "High Performance Software Testing on SIMD", IEEE Trans. on Software Eng., Vol 17, Nº 15, Mai 1991.

[MAL91] Maldonado, J. C., Critérios Potenciais Usos: uma Contribuição ao Teste Estrutural de Software, Tese de Doutorado, FEE/UNICAMP, Campinas - S. P., 1991.

[MAR90] Marshall, A. C., Hedley, D., Riddell, I. J. e Hennell, M. A., "Static Dataflow-aided Weak Mutation Analysis (SDAWM)", Information and Software Technology, vol 32, Nº 1, Jan/Fev 1990.

[MAT88] Mathur, A. P. e Krauser, E. W., "Modeling Mutation on Vector Processor", Proc. of the Second Workshop on Software Testing, Verification and Analysis, Banff-Canada, 1988.

[MAT93] Mathur, A. P., Wong, E. W., "Reducing the Cost of Mutation Testing: A Case Study", Tech Report SERC-TR-138-P, Software Engineering Research Center, Purdue University, Jun 1993.

[PRE92] Pressman, R. S., Software Engineering - A Practitioner's Approach (3rd. edition), McGraw-Hill, 1992.

[WON94] Wong, E. W., Maldonado, J. C., Delamaro, M. E., Mathur, A. P., "Constrained Mutation in C Programs", VII Simpósio Brasileiro de Eng. de Software, Curitiba - PR, Out 1994.