FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUSTAVO MORAES HABERMANN

AVALIAÇÃO EXPERIMENTAL DE CRITÉRIOS DE TESTE DE ANÁLISE DE MUTAÇÃO

MARÍLIA
2005

GUSTAVO MORAES HABERMANN

AVALIAÇÃO EXPERIMENTAL DE CRITÉRIOS DE TESTE DE MUTAÇÃO

Dissertação apresentada ao Curso de Ciência da Computação do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Bacharel em Ciência da Computação. (Área de Concentração: Teste de Software).

Orientador:
Prof. Dr. Marcio Eduardo Delamaro

MARÍLIA
2005

HABERMANN, Gustavo Moraes e
Avaliação Experimental de Critérios de Teste de Mutação / Gustavo Moraes Habermann; Orientador: Prof. Dr. Marcio Eduardo Delamaro.
Marília, SP: [s.n.] , 2005.
115 f.

Dissertação (Bacharelado em Ciência da Computação) - Centro Universitário Eurípides de Marília - Fundação de Ensino Eurípides Soares da Rocha.

1. Engenharia de Software 2. Teste de Software 3. Mutação

CDD 005.1

GUSTAVO MORAES HABERMANN

AVALIAÇÃO EXPERIMENTAL DE CRITÉRIOS DE TESTE DE MUTAÇÃO

Banca examinadora da dissertação apresentada ao Curso de Ciência da Computação da UNIVEM,/F.E.E.S.R., para obtenção do Título bacharel em Ciência da Computação. Área de Concentração: Teste de Software.

Resultado: _____

ORIENTADOR: Prof. Dr. _____

1º EXAMINADOR:_____

2º EXAMINADOR:_____

Marília, 25 de novembro de 2005.

A todos que me ajudaram nessa jornada,
principalmente a meus pais pelo
esforço e confiança.

HABERMANN, Gustavo Moraes e. Avaliação Experimental de Critérios de Teste de Mutação / Gustavo Moraes Habermann; 2005. 115 f. Marília, SP.
Dissertação (Bacharelado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

RESUMO

Esse trabalho tem por objetivo aplicar os critérios de mutação tradicional e dual com o intuito de validar a utilização de *scripts* como ferramenta de teste, isso é feito com o auxílio de outra ferramenta de teste, a PROTEUMIM 2.0. Tudo isso para que seja possível apresentar técnicas mais eficazes que possam ser aplicadas aos testes de mutação, visando à diminuição de custos na aplicação de testes em softwares o que acarretaria em uma maior confiabilidade dos programas produzidos através dessa metodologia.

Palavras-chave: Engenharia de software, Teste de software, Mutação, Mutação dual.

HABERMANN, Gustavo Moraes e. Avaliação Experimental de Critérios de Teste de Mutação / Gustavo Moraes Habermann; 2005. 115 f. Marília, SP.
Dissertação (Bacharelado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ABSTRACT

This work has for objective to apply the criteria of traditional and dual mutation with intention to validate the use of scripts as test tool, this is made with the aid of another tool of test, PROTEUMIM 2.0. Everything this so that it is possible to present techniques more efficient than can be applied to the mutation tests, aiming at to the reduction of costs in the application of tests in softwares what it would cause a bigger trustworthiness of the programs produced through the used methodology.

Keywords: Engineering of software, Test of software, Mutation, dual Mutation.

SUMÁRIO

INTRODUÇÃO

A Engenharia de Software é a área da computação que visa aplicar conceitos de engenharia na produção de softwares com alto nível de qualidade e baixo custo (PRESSMAN, 2002). Esse conceito é implantado por meio da aplicação de critérios, da produção de ferramentas e da utilização de métodos específicos, tudo isso para que possamos atingir um alto índice de qualidade sem que isso acarrete em um alto custo no projeto.

Na Engenharia de Software uma das atividades mais importantes é a atividade de teste. Essa atividade visa proporcionar um alto grau de qualidade e com isso confiabilidade ao software produzido, entretanto para atingirmos essas metas através de teste é necessário hoje grande volume de recursos, além de tempo de teste. Existem casos em que o valor do teste de um software é mais caro do que a própria produção do software. Outro grande problema geralmente observado na atividade de teste é a grande quantidade de tempo utilizado para que possamos realizar um teste razoavelmente efetivo. A utilização da palavra razoável na frase anterior é utilizada, pois geralmente para que o teste de um software seja totalmente realizado o tempo necessário poderia ultrapassar uma década.

Uma outra realidade para a área de teste é que mesmo sendo uma atividade que vem sendo estudada e pesquisada há mais de vinte anos, ela ainda não possui uma grande quantidade de recursos muito efetivos que possam ser aplicados comercialmente durante o processo de produção de um software. Essa área de pesquisa necessita que muitos estudos sejam feitos e defrontados para obtermos uma certa comodidade para a aplicação de idéias mais efetivas na área.

Levando em consideração os fatos acima citados e também procurando contribuir de alguma forma para a área de teste esse projeto foi proposto. Ele visa utilizar o contexto da comparação de dois critérios de análise de mutação para criar uma estratégia eficaz e de baixo custo para a aplicação de testes.

Os experimentos serão realizados com a utilização de *scripts*, produzidos especialmente para esse trabalhos. Eles servirão para que os dois critérios de análise de mutação possam ser aplicados e comparados. Tudo isso propõe elevar a linguagem de *scripts* a um patamar de ferramenta de teste, pois por meio de sua utilização, a aplicação de testes que anteriormente poderia ser considerado impossível, ou que levaria uma enorme quantidade de tempo para ser realizado, pode através dessa ferramenta ser aplicados. Todo esse custo demonstrado na frase acima poderá ser visto através dos números apresentados adiante.

Em resumo, o que quer que se entenda sobre o exposto acima é que através da utilização da linguagem dos *scripts* será possível a realização de tarefas na atividade de teste que seriam impossíveis de serem realizadas manualmente, se caracterizando nisso a contribuição deste trabalho na área.

Objetivo

O objetivo desse projeto é avaliar e comparar técnicas de análise de mutantes através da utilização de *scripts* e da ferramenta de teste PROTEUM/IM 2.0, apresentando com isso meios mais eficazes de aplicar os testes de mutação. Nessas condições foram selecionados dois critérios de teste de mutação, o teste de mutação dual e o teste de mutação tradicional, também foram

escolhidos seis programas objetos para que esses critérios pudessem ser aplicados, tudo isso focando a utilização da linguagem dos *scripts* e da ferramenta de teste acima citada na realização dos experimentos.

Esses *scripts* são o foco principal deste trabalho. Eles serão utilizados em praticamente todas as etapas do experimento, começando pela aplicação do perfil operacional de cada programa objeto na criação dos casos de teste, passando pela criação das sessões de teste até o cruzamento dessas sessões. Provando com isso a importância desse tipo de aplicação na atividade de teste.

Motivação

Na engenharia de software uma das etapas mais custosas que temos é a que corresponde aos testes do software, ainda mais quando se trata de softwares utilizados nas chamadas áreas críticas, ou seja, nas áreas em que vidas humanas estão intimamente ligadas ao seu bom funcionamento.

Para tornar essa atividade menos dispendiosa e mais aplicável, esse trabalho foi realizado. Imaginem os custos de um teste completo de um programa se esse fosse feito manualmente, somente com a utilização da mão humana para a geração de todos os casos de teste, criação das sessões, criação e execução dos mutantes. Para qualquer pequeno programa a ser testado isso seria quase impossível, além do mais mesmo que um teste desse tipo fosse aprovado, provavelmente, levaria uma empresa à falência.

Organização do Trabalho

Nesse capítulo foram apresentados os contextos ao qual esse trabalho está inserido, também nele estão as motivações que nos levaram a essas linhas e o caminho que nos levarão aos resultados que buscamos apresentar. No Capítulo 1, serão focados os assuntos pertinentes ao Teste de Mutação, suas definições, seus usos e funções.

Já no Capítulo 2 será demonstrado em detalhes o que foi utilizado na aplicação dos experimentos, nele temos detalhes sobre a ferramenta de teste PROTEUM/IM 2.0 e os *scripts* utilizados. O Capítulo 3 será usado para a apresentação dos experimentos, como foram feitos e os seus resultados e por último tem-se o Capítulo 4 com as conclusões retiradas a partir deste trabalho e também propostas de novas idéias para a atividade de teste.

1. TESTE DE MUTAÇÃO

Neste capítulo serão apresentados alguns aspectos sobre a atividade de teste, mais especificamente sobre os testes de mutação que faz parte de um conjunto de técnicas de teste baseadas em erros ou defeitos. Também serão demonstrados outros dois tipos de abordagens, a funcional (caixa preta) e a estrutural (caixa branca).

1.1 Teste "Caixa Preta"

Segundo (PRESSMAN, 2002), os métodos de teste de caixa preta preocupam-se em enfocar os requisitos funcionais do software. Ou seja, esse tipo de atividade possibilita ao testador que derive conjuntos de condições de entrada que exercitem completamente todos os requisitos funcionais para um programa, visando descobrir erros nas seguintes categorias:

(1) Funções incorretas ou ausentes;

(2) Erros de interfaces;

(3) Erros nas estruturas de dados ou no acesso a bancos de dados externos;

(4) Erros de desempenho;

(5) Erros de inicialização e término;

Ao contrário do teste de caixa branca, que é executado cedo no processo de teste, o teste de caixa preta tende a ser aplicado durante as últimas etapas da atividade de teste, enfoca a

estrutura de controle, se concentrando no domínio de informação. Este teste foi projetado para responder às seguintes perguntas:

- Como a validade funcional é testada?
- Quais classes de entrada constituirão bons casos de teste?
- O sistema é particularmente sensível a certos valores de entrada?
- Como as fronteiras de uma classe de dados são isoladas?
- Quais índices e volumes de dados o sistema pode tolerar?
- Que efeitos terão combinações específicas de dados sobre a operação do sistema?

1.2 Teste “Caixa Branca”

A técnica de teste caixa branca, ou então teste estrutural é um método de projeto de casos de teste que usa a estrutura de controle de projeto procedimental para derivar casos de teste.

Os testes de estrutura são, em geral, aplicados à unidade de programas relativamente pequenos, como sub-rotinas, ou às operações associadas com um objeto. O testador pode analisar o código e utilizar conhecimentos sobre a estrutura de um componente, a fim de derivar os dados para o teste. A análise do código pode ser utilizada para descobrir quantos casos de teste são necessários para garantir que todos os requisitos sejam executados pelo menos uma vez durante o processo de teste.

A técnica estrutural apresenta uma série de limitações e desvantagens decorrentes das limitações inerentes às atividades de teste de programa enquanto estratégia de validação (WERNER, 2005 apud RAPPS, 1985; FRANKLF, 1987; HOWDEN, 1987; NTAFOS, 1988).

Na técnica de teste estrutural, os aspectos de implementação são fundamentais na escolha dos casos de teste. O teste estrutural baseia-se no conhecimento da estrutura interna da implementação.

Conforme definições de (PRESSMAN, 2002), os critérios de teste estrutural são, em geral, classificados em:

1. Critérios Baseados em Fluxo de Controle: utilizam apenas características de controle da execução do programa, como comandos ou desvios, para determinar quais estruturas são necessárias. Os critérios mais conhecidos dessa classe são:

    - Todos-Nós: exige que a execução do programa passe, ao menos uma vez, em cada vértice do grafo de fluxo, ou seja, que cada comando do programa seja executado pelo menos uma vez;
    - Todos-Arcos: requer que cada aresta do grafo, ou seja, cada desvio de fluxo de controle do programa, seja exercitada pelo menos uma vez; e
    - Todos-Caminhos: requer que todos os caminhos possíveis do programa sejam executados.

2. Critérios Baseados em Fluxo de Dados: utilizam informações do fluxo de dados do programa para determinar os requisitos de teste. Esses critérios exploram as interações que envolvem definições de variáveis e referências a tais definições para estabelecerem os

requisitos de teste. Exemplos dessa classe de critérios são os Critérios de Rapps e Weyuker e os Critérios Potenciais-Usos.

3. Critérios Baseados na Complexidade: utilizam informações sobre a complexidade do programa para derivar os requisitos de teste. Um critério bastante conhecido dessa classe é o Critério de McCabe, que utiliza a complexidade ciclomática do grafo de programa para derivar os requisitos de teste. Essencialmente, esse critério requer que um conjunto de caminhos linearmente independentes do grafo de programa seja executado.

Um problema relacionado ao teste estrutural é a impossibilidade, em geral, de se determinar automaticamente se um caminho é ou não executável, ou seja, não existe um algoritmo que dado um caminho completo qualquer decida se o caminho é executável e forneça o conjunto de valores que causam a execução desse caminho (VERGÍLIO, 1993).

1.3 Técnica de Teste Baseado em Erros

A técnica de teste baseado em erros propõe a utilização dos erros mais freqüentes que ocorrem na produção do software para que a partir deles sejam feitas regras para a aplicação dos testes. A ênfase desses testes é dada nos erros que o programador ou projetista pode cometer durante o desenvolvimento do software e nos aspectos abordados para que possam ser usados na detecção da ocorrência desses erros.

1.3.1 O critério de teste de Análise de Mutantes

O critério de teste de análise de mutantes, aqui também chamado, para fins de identificação, de critério de teste de análise de mutação tradicional ou normal, (MALDONADO, 1998). Surgiu na década de 70 na *Yale University* e *Georgia Institute of Technology* , contendo a mesma idéia usada no método clássico para a detecção de erros lógicos em circuitos digitais – o modelo de falha única (MALDONADO, 1998 apud FRIEDMAN, 1975). Esse critério utiliza um conjunto de programas com algumas pequenas alterações obtidas a partir de um determinado programa P para avaliar o quanto um conjunto de casos de testes T é adequado para o teste de P. O objetivo é encontrar um conjunto de casos de teste que consiga revelar, pela execução de P, a diferença de comportamento existentes entre P e seus programas mutantes (DELAMARO, 1998).

A idéia básica dessa técnica foi apresentada pela primeira vez por (DEMILLO, 1978) conhecida como hipótese do programador competente (*competent programmer hypothesis*), ela considera que programadores com mais experiência escrevem programas corretos ou muito próximos do correto. Assumindo que isso seja verdade, pode-se dizer que os 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 (AGRAWAL, 1998).

Uma outra hipótese explorada na aplicação do critério de Análise de Mutantes é o efeito de acoplamento (*coupling effect*) (DEMILLO, 1978), o qual assume que erros complexos estão

relacionados a erros simples. Relacionadas a essa hipótese já temos alguns experimentos que a validam, (ACREE, 1979) e (BUDD, 1980).

Levando em consideração as hipóteses acima descritas (Hipótese do programador competente e do Efeito de acoplamento), o critério de análise de mutantes é aplicado da seguinte maneira: 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 encontrado 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 P1;P2;::::;Pn denominados mutantes de P, diferindo de P apenas pela ocorrência de erros simples.

Os operadores de mutação (*mutant operators*) têm o objetivo de determinar os desvios sintáticos mais comuns, eles são aplicados em um programa P e depois através deles são gerados programas similares, chamados mutantes de P. Podemos dizer que os operadores de mutação são as "regras" que utilizamos para definir as alterações que serão aplicadas no programa original P. Esses operadores são construídos para satisfazerem a um entre dois propósitos (MALDONADO, 1998):

1. Induzir mudanças sintáticas simples com base nos erros típicos cometidos pelos programadores (por exemplo, trocar o nome de uma variável); ou
2. Forçar determinados objetivos de teste (como executar cada arco do programa), (MALDONADO, 1998 apud OFFUTT, 1996).

Após essas etapas, os mutantes são executados com o mesmo conjunto de casos de teste T. O objetivo é obter os casos de teste que resultem apenas em mutantes mortos (isto é para

algum caso de teste o resultado do mutante e o do programa original diferem entre si) e equivalentes (o programa original apresenta sempre o mesmo resultado, para qualquer d ∈ D); e neste caso 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 (MALDONADO, 1998 apud DEMILLO, 1978).

É importante destacar que a equivalência entre mutantes é uma questão muito trabalhosa, pois sua marcação deve ser feita manualmente, sendo que até hoje não existe um algoritmo confiável para sua automatização, necessitando sempre da ajuda do testador para reconhecê-los. Na Figura 1 existem três quadros que exemplificam um programa normal e seus mutantes normal e equivalente respectivamente (VINCENZI, 1998). No quadro **(a)** é apresentado o programa original que calcula faz o calculo de fibonacci, a figua **(b)** é um mutante deste programa, sua definição de mutante se dá pela mutação encontrada na linha indicada. Já no quadro **(c)** está representado um outro mutante também definido pela mutação que foi submetido na linha indicada, entretanto esse é o exemplo de um mutante equivalente, pois segue a definição acima apresentada.

Outro ponto importante, citado por (DEMILLO, 1980), é que a Análise de Mutantes oferece uma métrica objetiva do nível da efetividade dos casos de teste analisados, isso é feito através de um escore de mutação (*mutation score*). Esse escore é o resultado da relação entre o número de mutantes mortos com o número de mutantes gerados.

Ele é calculado da seguinte maneira:

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

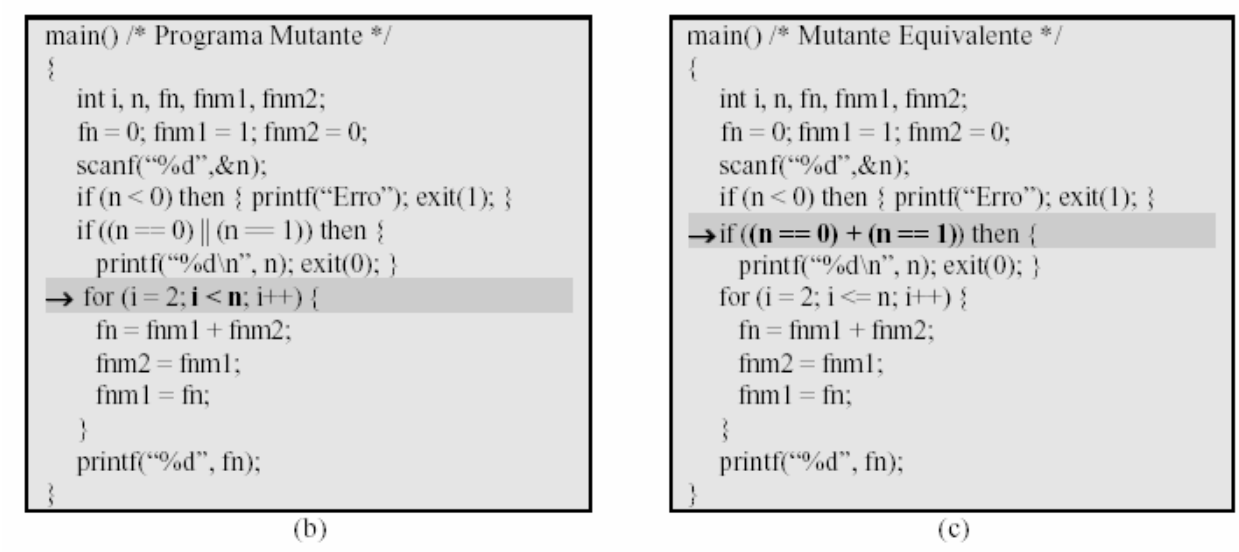
Onde :

- DM(P; T): número de mutantes mortos pelos casos de teste em T.
- M(P): número total de mutantes gerados.
- EM(P): número de mutantes gerados equivalentes a P.

Esse escore varia no intervalo entre 0 e 1 sendo que, quanto maior o escore mais adequado é o conjunto de casos de teste para o programa que está sendo testado. Podemos também observar na fórmula acima que somente DM(P; T) depende do conjunto de casos de teste utilizado e que EM(P) é obtido à medida que o testador, manualmente ou por meio da heurística, decide qual mutante vivo é equivalente (MALDONADO, 1998 apud SOUZA, 1996). Outro aspecto importante é o custo que o critério de análise de mutantes gera, porém a discussão desse assunto ficará para os capítulos posteriores.

```
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 = fnml;
      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.

1.3.2 O critério de teste de Análise de Mutação Dual

Segundo (DELAMARO, 2003), a idéia do teste de Mutação Dual consiste em utilizarmos os mutantes que são mortos mais facilmente para construir os casos de testes úteis. A proposta deste critério é selecionar os casos de teste que não distinguem mutantes, tendo sua notação apresentada da seguinte maneira: um caso de teste t tal que dado um mutante M_i , sendo $M_i(t) = P(t)$.

Em resumo, o critério de teste de mutação dual (*Dual Mutation Testing* - DMT) baseia-se em selecionar casos de testes que execute o objeto mutado e ainda produza o mesmo resultado que o programa original. Sua definição é apresentada a seguir.

Primeira definição: Considerando o programa a ser testado P, e um conjunto de teste T, um conjunto de mutantes M e o domínio de entradas D de P, então, de acordo com DMT nós temos que um mutante $M_i \in M$ é morto quando executado com T, se e somente se existe $t \in T$ e que satisfaça as duas condições :

- A execução de P com t alcança o comando que foi mutado para criar M_i .
- $M_i(t) = P_i(t)$;

A primeira definição foi usada para especificar quando um mutante é morto. A seguir veremos a definição de um mutante dual-equivalente, antes devemos esclarecer que as notações $t \succ M_i$ e $t \not\succ M_i$ indicam respectivamente que o caso de teste t matou o mutante M_i e que o caso de teste t não matou o mutante M_i .

Segunda definição: Um mutante $M_i \in M$ é dual-equivalente a P se e somente se

$$\forall \mathbf{t} \in \mathcal{D}, \mathbf{t} \not\succ M_i \quad \textbf{(1)}$$

A definição de **(1)** quer dizer que um mutante é dual-equivalente se o caso de teste t não executa o estado mutado, porém faz o mutante ter o mesmo comportamento do programa original.

Terceira definição: O escore de mutação dual do conjunto de teste T é dado por

$$ms(T,M,P) = \frac{\text{\# de mutantes mortos}}{|M| - \text{\# de mutantes dual-equiv.}} \quad \textbf{(2)}$$

Para não deixar dúvidas quanto aos critérios estudado anteriormente, observa-se que o exemplo, usado por (DELAMARO, 2003) para demonstrar suas características. Nesse exemplo

apresentado pela Figura 2 temos um programa com dois parâmetros inteiros de entrada (x e y), esses parâmetros podem ser maiores ou iguais a zero e foram usados para computar o valor de x^y como mostra o código do programa.

Na Tabela 1 temos três mutantes, gerados através de operadores de mutação implementados na Ferramenta PROTEUM/IM 2.0, mostrando como a mutação dual seleciona casos de teste muito específicos que seriam considerados inúteis na mutação tradicional. A próxima coluna mostra as condições requeridas como entrada para que ocorra a morte do mutante dual.

```
int pow(int x, int y)
{
int s = 1;
int i;

      for (i = 0; I < y; i++)
      {
            s *= x;
      }
      return x;
}
```

Figura 2 – Exemplo do programa que computa x^y .

Tabela 1 – Exemplo de mutantes duais e casos de teste que os mata.

Estado Original	**Estado Mutado**	**Operador de Mutação**	**Valores para matar os mutantes**
s *= x;	S *= 0;	CLSR	x = 0 y > 0
S *= x;	S *= 1;	VLSR	x = 1 y > 0
s *= x;	S *= -1;	CRCR	x = 1 y >= 2 y % 2 = 0
return x;	return -1;	CRCR	Equivalent

O último mutante mostra que a mutação dual não está livre dos mutantes equivalentes. Esse é um caso de teste que não faz o mutante retornar o mesmo valor que o programa original. Ele seria inútil para a mutação normal, no entanto é útil na a mutação dual.

2. A PROTEUM E OS EXPERIMENTOS

A seleção da ferramenta de teste PROTEUM/IM 2.0 para a realização desse experimento foi feita considerando os critérios analisados, a linguagem de programação em que os programas testados fora desenvolvidos e também o suporte que a ferramenta oferece para a realização dos experimentos.

Segundo (JORGE, 2001), em ferramentas tradicionalmente utilizadas para a aplicação dos critérios baseados em mutação, devem executar as atividades abaixo listadas:

- 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.

Também neste capítulo veremos as características dos experimentos realizados nesse trabalho a fim de instituir um roteiro para esse tipo de trabalho.

2.1 A ferramenta de teste PROTEUM/IM 2.0

A ferramenta de teste utilizada para esse experimento foi a PROTEUM/IM 2.0, pois ela possui um conjunto de características que a torna especialmente certa para esse tipo de trabalho,

pois além de possuir as propriedades acima citadas, ela ainda nos dá o suporte necessário para realizar os testes utilizando o critério de mutação dual, dando com isso mais crédito ao trabalho, pois tanto os testes feitos utilizando a mutação tradicional quanto os feitos em cima da mutação dual poderão ser realizados através dela e com isso poderemos descartar algum tipo de diferença de resultados decorrente da utilização de ferramentas diferentes.

De acordo com o manual da ferramenta de teste PROTEUM/IM 2.0 escrita por (BIANCHINI, 2000), ela é uma ferramenta baseada em mutação que consiste em um conjunto de módulos funcionais (programas) usados para a aplicação de testes através de *Shell scripts*, diminuindo o número de interações entre o usuário e a PROTEUM/IM 2.0 e economizando tempo. Essa ferramenta, diferentemente da versão Proteum, implementa o critério mutação de interface (DELAMARO, 2001), que é um critério de adequação inter-procedural, conveniente para ser usado ao nível de teste de integração.

Os testes feitos com essa ferramenta acontecem através de sessões de teste. De acordo com (JORGE, 2001) o estado de uma sessão de teste se dá por uma base de dados caracterizada 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 formada por uma seqüência de operações realizadas sobre essa base de dados através de chamadas aos programas que compõem as ferramentas (JORGE, 2001 apud DELAMARO, 1997).

Seguindo a idéia de (JORGE, 2001) a vantagem de utilizar-se sessões de teste é que o testador tem condições, através de operações que fazem parte da ferramenta, conseguir diferentes resultados provenientes de uma mesma sessão de teste, fazendo várias operações sobre a mesma. Considerando a ferramenta utilizada podemos citar 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.

Com isso pode-se aplicar diferentes combinações de casos de teste para avaliarmos diferentes conjuntos de mutantes em uma mesma sessão de teste.

2.2 Os objetos de estudo

Para realizar os experimentos apresentados a seguir foram usados 6 programa de domínio público, sendo que todos eles possuem a característica de manipularem datas e calendários. Esses programas foram todos escritos em linguagem C e possuem seus nomes relacionados abaixo:

- CALCDATE - Realiza operações entre datas.
- CALEN - Gera calendários para o objetivo de imprimi-los.
- CALEND –Gera calendários na tela.
- CAL – Gera o calendário de um mês ou ano específico.
- JDAY-JDATE – Retorna o número Juliano de uma determinada data.
- DATEPLUS - Faz operações de soma e subtração entre datas.

2.3 Plataforma e linguagem utilizada

Esse estudo foi totalmente realizado em plataforma linux, versão fedora 3, entre os motivos de sua escolha os que merecem ser listados são :

1. por ser uma software livre não acrescenta custo ao projeto;
2. possui a propriedade de ser multitarefa em modo texto;
3. ser confiável em relação a erros presentes no próprio sistema;
4. pode ser utilizado em máquinas não tão potentes , o que o torna muito portável.

A linguagem de *script* utilizada nos experimentos foi TCSH (The exTended C-*Shell*) , sua escolha foi feita por que essa linguagem é nativa da versão do linux acima descrita, isto é, ela é instalada junto com o sistema operacional e também por sua gramática ser simples e de fácil entendimento e programação. Além disso, uma outra vantagem destes *shell scripts* é que eles não precisam ser compilados, ou seja, basta apenas criar um arquivo texto qualquer, e inserir comandos a ele e para dar à este arquivo a definição de "*shell script*", teremos que incluir uma linha no começo do arquivo (#!/bin/tcsh) e torná-lo "executável", utilizando o comando chmod.

2.4 A Estratégia Utilizada nos Experimentos.

Para a realização dos experimentos foram seguidos os seguintes passos:

1. Identificação do perfil operacional de cada programa e criação dos *scripts* para a geração dos casos de teste a partir desses perfis.

2. Criação de uma sessão-raiz de teste para cada um dos seis programas anteriormente citados;

3. Após a criação das sessões-raiz, foram marcados os mutantes equivalentes de cada programa objeto. Essa marcação foi executada duas vezes, a primeira utilizando a definição de mutantes equivalentes de critério de mutação tradicional, a segunda foi feita utilizando a definição de equivalentes para o critério de mutação dual.

4. Com os mutantes equivalentes definidos em cada sessão-raiz e para cara critério, foi criado um diretório chamado "equivalentes" (em cada sessão-raiz). Esse diretório possuía dois sub-diretórios, um chamado dual e o outro normal, e cada um desses sub-diretórios possuía os casos de testes mais peculiares e os mutantes equivalentes referentes aos seus critérios.

5. Com a estrutura do experimento criada, em cada sessão-raiz de teste foram geradas 30 sub-sessões de teste para cada um dos dois critérios, lembrando que todo esse processo foi feito através de *scripts*. Após a criação de cada uma das 60 sub-sessões, acontecia a importação, para cada uma das sub-sessões, dos mutantes marcados como equivalentes e também já eram incluídos os casos de teste mais peculiares. E novamente utilizando os *scripts* de geração de casos de teste foram criados 15.000 casos de teste para cada uma das 60 sub-sessões de teste de todas as 6 sessões-raiz. É importante citar que a criação dos casos de teste é um processo aleatório baseado em um servidor de números, que será explicado na parte do detalhamento dos experimentos a seguir.

6. Em seguida foi feita a avaliação de cada sub-sessão criada, recolhendo dados sobre elas e posteriormente foi executado qualquer mutante, não morto durante o processo anterior, através da inclusão manual de casos de teste na sessão de teste. Tudo isso sendo feito com o auxílio da PROTEUM/IM 2.0, para que as sessões fossem preparadas para o cruzamento.

7. Para finalizar, foi realizado o cruzamento das sessões, isto é, dentro de cada uma das sub-sessões que foram inseridos os 15.000 casos de teste foi criado uma sub-sessão e dentro dela inserido os casos de teste válidos da sessão anterior. Em seguida esses casos de testes válidos foram usados para executar os mutantes através do critério de mutação contrário do que foi usado na sessão anterior. Isto é, se na sessão anterior foi utilizado o critério de teste de mutação normal para executar os mutantes, na sub-sessão criada foi usado o critério de teste de mutação dual. Isto foi feito para que pudesse ser verificado o quanto os casos de teste válidos para um critério são efetivos se utilizados em outro critério.

2.5 A descrição dos perfís operacionais

A identificação do perfil operacional foi feita através da observação do código dos programas objetos, isto foi feito pela observação do funcionamento de cada um dos programas para sabermos como funcionavam as estrutura de parâmetros, para que posteriormente fossem feitos os perfis operacionais. A descrição do funcionamento de cada programa assim como o perfil operacional gerado para cada um deles pode ser vista logo a seguir.

2.5.1 – Programa Calcdate

Esse programa calcula a diferença entre duas datas ou uma data e um número de dias, dependendo da opção desejada.

Argumentos:

calcdate mmddyy -o x - calcula data x dias depois da data original;

calcdate mmddyy -d mmddyy – calcula número de dias entre as duas datas;

Entradas válidas:

- 1º. Parâmetro: 010100 até 123199;
- 2º. Parâmetro: idem para data;
- -36524 ate 36524 para offset (obs:intervalo de valores validos para opção –o);

O perfil operacional do *script* utilizado para gerar os casos de teste para esse programa foi definido como sendo**:**

95% dos casos de teste válidos, isso representa que os valores de entrada utilizados como casos de testes estão dentro dos limites de cada parâmetro do programa.

Dos casos válidos:

- 50% com –d;
- 50% com –o;

05% dos casos não válidos, isso representa que os valores de entrada utilizados como casos de testes não estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado.

Dos casos não válidos:

- 1% com um único parâmetro.
- 1% sem parâmetro.

- 14% com mês não válido 1º parâmetro.
- 14% com ano não válido 1º parâmetro.
- 14% com dia não válido 1º parâmetro.
- 14% com mês não válido 2º parâmetro.
- 14% com ano não válido 2º parâmetro.
- 14% com dia não válido 2º parâmetro.
- 14% com offset não válido (obs: intervalo de valores validos para opção –o).

2.5.2- Programa Calen

Este programa gera calendários do tamanho da tela, embora o objetivo desse programa seja gerar esses calendários para impressoras matriciais. Ele possui 3 parâmetros de entradas mais algumas argumentos.

Argumentos:

calen y – mostra calendário do ano y.

calen m y – mostra calendário do mês m, ano y.

calen m y n – mostra n mêses a partir do mês m, ano y.

Outros argumentos aceitos:

-l -r : justificação à esquerda e direita.

-u -m : só letras maiúsculas ou letras de tamanhos misturados.

-o[seq] : usa uma seqüência de caracteres “seq” para formar o cabeçalho.

-bN : adiciona linhas em branco após fim de página.

Entradas válidas:

1º parâmetro:

- Pode ser somente o ano do calendário de 1753 a 9999 ou o mês de 1 a 12.

2º parâmetro:

- Se o primeiro parâmetro for um mês o segundo será um ano, também de 1753 a 9999.

3º parâmetro:

- Somente pode ser utilizado se usarmos o formato mês e ano, esse terceiro parâmetro é o numero de mêses que queremos gerar no calendário a partir do mês e ano especificados nos 1º e 2º parâmetros.

- As opções podem ser:

- -l (justifica as datas a esquerda) ou
- -r (justifica as datas a direita) ou
- nenhuma das opções, entretanto pode ter ainda as opções :
- -m (deixa os dias da semana em maiúsculas e minúsculas) ou
- -u (deixa os dias da semana em maiúsculas) ou
- nenhuma das opções, no entanto pode ter ainda as opções:
- -o[seq_caract] (seqüência de caracteres que forma o nome do mês) e também:
- -bN (acrescenta linhas em branco entre os calendários) ou também:
- nenhuma das opções opções descritas até agora.

O perfil operacional do *script* utilizado para gerar os casos de teste para esse programa foi definido como sendo**:**

95% dos casos de teste são válidos isso representa que os valores de entrada utilizados como casos de testes estão dentro dos limites de cada parâmetro do programa

Dos casos válidos:

- 34% com um parâmetro de ano.
- 33% com um parâmetro de mês e um de ano.
- 33% com um parâmetro de mês, um de ano e um de numero de mêses.

Obs: Para cada um dos casos válidos acima citados, cada um poderá ter a probabilidade de 20% de possuir uma das opções listadas abaixo:

- 20% de probabilidade de não possuir nenhuma opção.
- 20% de probabilidade de possuir somente uma opção.
- 20% de probabilidade de possuir duas opções.
- 20% de probabilidade de possuir três.
- 20% de probabilidade de possuir quatro opções.

5% Dos casos são não válidos, isso representa que os valores de entrada utilizados como casos de testes não estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado.

50% dos casos não válidos tem a probabilidade de ser :

- 10% com um único parâmetro ano inválido.
- 09% com o parâmetro mês inválido e o ano válido.
- 09% com o parâmetro mês válido e o ano inválido.
- 09% com os dois parâmetros mês e ano inválidos.
- 09% com o parâmetro mês inválido e os parâmetros ano e número de mêses válidos.

- 09% com o parâmetro mês válido, o ano inválido e o numero de mêses válido.
- 09% com o parâmetro mês válido, o ano válido e o número de mêses inválido.
- 09% com o parâmetro mês inválido, o ano inválido e o número de mêses válido.
- 09% com o parâmetro mês inválido, o ano válido e número de mêses inválido.
- 09% com o parâmetro mês válido, o ano inválido e o número de mêses inválido.
- 09% com os parâmetros mês, ano e número de mêses inválido.

50% dos casos não válidos têm a probabilidade de ser alguma das opções abaixo somente se estiverem acompanhadas de uma opção inválida, isto é, os parâmetros podem estar corretos, no entanto o que irá caracterizá-lo como inválido é a opção que o acompanha, pois esta será invalida.

- 34% com um parâmetro de ano.
- 33% com um parâmetro de mês e um de ano.
- 33% com um parâmetro de mês, um de ano e um de numero de mêses.

2.5.3- Programa Calend

Este programa apresenta calendários para serem visualizados na tela.O programa possui dois parâmetros:

- mês.
- ano da data que será gerado o calendário.

Argumentos:

calend mm – calendário do mês mm, ano corrente.

calend yyyy – calendário do ano yyyy.

calend mm yyyy – calendário do mês mm, ano yyyy.

calend yyyy mm – calendário do mês mm, ano yyyy.

Entradas válidas:

1º Parâmetro:

Se for valores de 1 a 12 -

Então é mostrado o calendário do mês indicado mais o anterior e o posterior do ano corrente.

Senão, Se for passado valores entre 13 e 9999

Então o valor passado é assumido como um ano e é mostrado o calendário daquele ano.

Senão

Como nenhum valor foi passado o programa considera que o mês e ano a ser utilizado são os correntes.

FimSe.

FimSe.

2º Parâmetro:

- O Segundo Parâmetro necessariamente será um ano se o primeiro for um valor de 1 a 12. Esse parâmetro poderá estar no intervalo de 0 a 9999. Agora se quisermos que o segundo parâmetro corresponda ao mês teremos que usar os valores para ano que estão no conjunto que corresponde entre 13 a 9999.

O perfil operacional do *script* utilizado para gerar os casos de teste para esse programa foi definido como sendo:

95% dos casos de testes são válidos isso representa que os valores de entrada utilizados como casos de testes estão dentro dos limites de cada parâmetro do programa

Dos casos válidos:

- 4% Com nenhum parâmetro.
- 24% Somente com valores de 1 a 12 no primeiro parâmetro.
- 24% Somente com valores de 13 a 9999 no primeiro parâmetro.
- 24% Com valores de 1 a 12 no primeiro parâmetro e de 0 a 9999 no segundo parâmetro.
- 24% com de 13 a 9999 no primeiro parâmetro e 1 a 12 no segundo parâmetro.

5% Dos casos são não válidos: isso representa que os valores de entrada utilizados como casos de testes não estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado.

Dos casos não válidos:

- 4% somente com um valor que não esteja ente 1 a 9999.
- 24% com um valor válido de mês no primeiro parâmetro e um parâmetro inválido para ano Isso significa que o ano não pode estar entre 0 e 9999.
- 24% com um valor válido para o ano no primeiro parâmetro, isso significa que o parâmetro deve ser 13 a 9999, e um parâmetro inválido de mês.
- 28% com os dois parâmetros inválidos, sendo o primeiro corresponde ao mês (significando que os valores não podem ser de 1 a 12, mais também não podem estar no conjunto de parâmetros válidos do ano.

2.5.4 – Programa Cal

Este programa gera o calendário de um mês ou ano específico, na tela seus parâmetros são mês e ano.

Argumentos:

cal – imprime mês corrente.

cal x – imprime calendário do ano x.

cal x y – imprime calendário do mês x, ano y.

O perfil operacional do *script* utilizado para gerar os casos de teste para esse programa foi definido como sendo:

95% dos casos de testes são válidos isso representa que os valores de entrada utilizados como casos de testes estão dentro dos limites de cada parâmetro do programa.

Dos casos válidos:

- 1% não possui parâmetros
- 49% possuem apenas um parâmetro
- 50% possuem dois parâmetros

5% Dos casos são não válidos, isso representa que os valores de entrada utilizados como casos de testes não estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado.

- 25% com um único parâmetro (ano) não valido.
- 25% com dois parâmetros, mês não válido.
- 25% com dois parâmetros, ano não válido.
- 25% com dois parâmetros, ambos não validos.

2.5.5 – Programa Jday-jdate

Esse programa aceita uma data como parâmetro e retorna o dia juliano, o dia juliano é o número de dias da data fornecida como parâmetro até uma data específica de um passado distante.

Argumentos:

jday-jdate x – calcula a data Gregoriana correspondente ao número Juliano x

jday-jdate mmddyy – calcula o número correspondente à data

Entradas válidas:

- 1º parâmetro: de 1 até 12 .
- 2º parâmetro: de 1 até 28 (fevereiro) ou 1 até 29 (fevereiro em ano bissexto) ou 1 a 30 para os mêses (abril, junho, setembro, novembro) ou de 1 a 31 (janeiro, março, maio, julho, agosto, outubro, dezembro).
- 3º parâmetro: de 0 até 522485619.

O perfil operacional do *script* utilizado para gerar os casos de teste para esse programa foi definido como sendo

95% dos casos de testes são válidos isso representa que os valores de entrada utilizados como casos de testes estão dentro dos limites de cada parâmetro do programa.

Dos casos válidos:

- 50% terão o formato mês dia e ano mmddyy.
- 50% terão um número do calendário Juliano.

5% Dos casos são não válidos, isso representa que os valores de entrada utilizados como casos de testes não estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado.

Dos casos não válidos:

- 01% sem parâmetro.
- 19% com um parâmetro.
- 19% com dois parâmetros.
- 19% com três parâmetros dois válidos e um inválido.
- 19% com três parâmetros um válido e dois inválidos.
- 19% com os três parâmetros inválidos.
- 04% com o número juliano inválido.

2.5.6 – Programa Dateplus

Esse programa faz operações de soma e subtração entre uma data fornecida como parâmetro para o programa e outros parâmetros também informados no momento da execução.

Argumentos:

- dateplus x y unidade – calcula a data representada pelo número x, incrementada ou decrementada de y unidades.
- as unidades podem ser: sec, min hour, day, week, mon, year.

O perfil operacional do *script* utilizado para gerar os casos de teste para esse programa foi definido como sendo:

95% dos casos de testes são válidos isso representa que os valores de entrada utilizados como casos de testes estão dentro dos limites de cada parâmetro do programa **Dos casos válidos:**

- 15% são no formato mmddaa operação + valor unidade.
- 15% são no formato mmddaa mais 2 vezes (operação + valor unidade).
- 15% são no formato mmddaa mais 3 vezes (operação + valor unidade).
- 15% são no formato mmddaa mais 4 vezes (operação + valor unidade).
- 15% são no formato mmddaa mais 5 vezes (operação + valor unidade).
- 15% são no formato mmddaa mais 6 vezes (operação + valor unidade).
- 10% são no formato mmddaa mais 7 vezes (operação + valor unidade).

Observação:

- operação - pode ser a subtração (-) ou a soma (+).
- valor - é a quantidade que será incrementada ou decrementada da data.
- unidade - é o tipo de unidade utilizada para especificar o valor (sec, min hour, day, week, mon, year) .

5% Dos casos são não válidos, isso representa que os valores de entrada utilizados como casos de testes não estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado

Dos casos não válidos:

- 50% com a data válida, mas com pelo menos uma opção inválida (operação + valor unidade), podendo ser da seguinte maneira:
  - 05% sem a data.
  - 15% somente com a data.

- 20% com a data, operação, valor e a unidade mas com essa opção inválida.
- 10% com a data mais 2 vezes (operação + valor unidade) mas com uma desses opções inválidas.
- 10% com a data mais 3 vezes (operação + valor unidade) mas com uma desses opções inválidas.
- 10% com a data mais 4 vezes (operação + valor unidade) mas com uma desses opções inválidas.
- 10% com a data mais 5 vezes (operação + valor unidade) mas com uma desses opções inválidas.
- 10% com a data mais 6 vezes (operação + valor unidade) mas com uma desses opções inválidas.
- 10% com a data mais 7 vezes (operação + valor unidade) mas com uma desses opções inválidas.

- 50% com a data inválida podendo ter até todas as opções válidas; sendo da seguinte maneira.
  - 05% sem a data.
  - 15% somente com a data.
  - 20% com a data, operação, valor e a unidade.
  - 10% com a data mais 2 vezes (operação + valor unidade).
  - 10% com a data mais 3 vezes (operação + valor unidade).
  - 10% com a data mais 4 vezes (operação + valor unidade).
  - 10% com a data mais 5 vezes (operação + valor unidade).

- 10% com a data mais 6 vezes (operação + valor unidade).
- 10% com a data mais 7 vezes (operação + valor unidade).

2.6 Os *scripts* de geração de casos de teste

Os *scripts* de geração de casos de teste foram feitos em linguagem *tcsh* e criados com base nos perfis operacionais acima descritos, como citado anteriormente, eles servem para gerar os casos de testes que irão ser importados pela ferramenta de teste PROTEUM/IM 2.0 para matar os mutantes criados. Cada caso de teste gerado por esses *scripts* é colocado em um arquivo texto, para que em seguida esses arquivos sejam lidos pela ferramenta e seu conteúdo usado como caso de teste.

Como uma das características mais importantes desses *scripts* é gerar os casos de teste aleatoriamente, então para isso foi utilizado um servidor de números, que nada mais é do que um programa que é deixado ativo em *background* no sistema, que quando chamado através de um comando próprio, retorna um número aleatório que é maior ou igual a um valor piso e menor que um valor teto. Quando ativamos esse servidor de número utilizamos o seguinte comando, "./server <semente> <porta>", onde a semente é o número em que o servidor irá se basear para gerar o número aleatório e a porta é onde esse serviço ficará ativo.

Então em síntese um *script* para geração de casos de teste, neste caso, nada mais é do que a implementação do perfil operacional do programa a ser testado utilizando-se como suporte

aleatoriedade na geração de números. Os seis *scripts* implementados aqui podem ser encontrados no Apêndice A.

2.7 A criação das sessões-raiz

A criação dessas sessões foi feita da seguinte maneira: primeiro foi criado um diretório com o nome do programa a ser testado (exemplo mkdir Cal) , em seguida foi copiado o programa fonte (exemplo Cal.c) para dentro desse diretório e para finalizar foi criado por meio da ferramenta de teste PROTEUM/IM 2.0 uma nova sessão de teste com o nome do programa a ser testado, isso acontece por meio dos seguintes passos :

1. Devemos criar a sessão de teste, para isso, já na Ferramenta PROTEUM/IM 2.0, acionamos a guia "*Program test*" em seguida a opção "*New*". Após isso é preenchidos os campos para a criação da sessão-raiz Cal.
    - *Directory: /home/teste/Cal*
    - *Program test name: Cal*
    - *Source Program: Cal*
    - *Executable Program: Cal*
    - *Compilation Command: gcc Cal.c –o Cal –w*
2. Após gerar uma nova sessão é feita a criação dos mutantes para esse programa. Para isso deve-se ir para guia "*Mutants*", depois "*Generate Unit*" e é pressionado o botão "*Continue*". Na próxima tela teremos um *list box* com as quatro classes

para que seja definido os operadores de mutação (ver os operadores utilizados, para a geração dos mutantes nos experimentos, no Apêndice B), e as porcentagens para cada tipo de operador (“*operators*”,”*statements*”,”*variables*”,”*constants*”). Depois de configurar as quantidades e os níveis de operação é só clicar no botão “*Generate*” para que os mutantes sejam gerados.

3. Por ultimo, é interessante verificar se os mutantes foram criados e também se a sessão de teste está com o escore de mutação igual a zero. Para isso deve-se clicar na guia “*Status*” e verificar os campos “*Total de mutants*” que deverá ser diferente de zero e “*Score*” que deverá ser igual a zero.

2.8 A Identificação dos equivalentes

A identificação dos mutantes equivalentes é a parte mais trabalhosa nesse tipo de experimento, pois mesmo sendo feita por meio da ferramenta da teste e de *scripts*, ela depende totalmente da análise humana para a sua identificação. Nesse experimento tudo isso foi feito utilizando a estratégia descrita abaixo, sendo que todo esse procedimento que será mostrado a seguir foi feito para as 6 sessões-raiz de teste :

1. Utilizando um *script* auxiliar, (ver o *script* no Apêndice C), foram criadas duas sub-sessões de teste. Isso foi feito criando-se dois subdiretórios e fazendo uma cópia da sessão raiz para dentro de cada um deles (uma para que fosse aplicado o critério de mutação normal e outra para aplicar o critério de mutação dual). Em seguida o *script* auxiliar utiliza o *script* de geração de casos de teste para

produzir 500 casos de teste (pois a PROTEUM/IM 2.0 somente importa 512 casos de teste por vez). Após isso acontecer o script auxiliar importa os casos de teste para a sessão de teste, e com eles, o *script* faz a execução dos mutantes (é importante salientar que essa operação é feita até que o número de casos de teste criados e importados chegue a 15.000). Para fazer a importação e execução dos mutantes é utilizado no *script* auxiliar comandos da ferramenta de teste, entretanto para isso é utilizado o modo texto da ferramenta.

2. Em seguida entramos em cada sub-sessão e utilizamos a PROTEUM/IM 2.0 em modo gráfico para auxiliar na marcação dos mutantes equivalente e também para matar os mutantes que ainda estão vivos, mas que também não são equivalentes.
3. Após todos os mutantes serem mortos é criada uma sub-pasta chamada de "equivalentes" a partir da pasta principal, e dentro dessa sub-pasta são criadas duas pastas chamadas normal e dual, em seguida dentro da sub-pasta normal são colocados os arquivos em que estão armazenados as marcações que indicam quais mutantes executados em modo normal eram equivalentes e também os arquivos contendo somente os casos de testes que precisaram ser inseridos manualmente, e assim também foi feita na pasta chamada dual só que respeitando os critérios de mutação dual. Com isso pode-se terminar a fase de preparação para a realização dos experimentos e o que dá uma estrutura como a que está apresentada na Figura 3.

A representação da estrutura dos diretórios e das sessões de testes até agora mencionadas podem ser vistas na Figura 3.

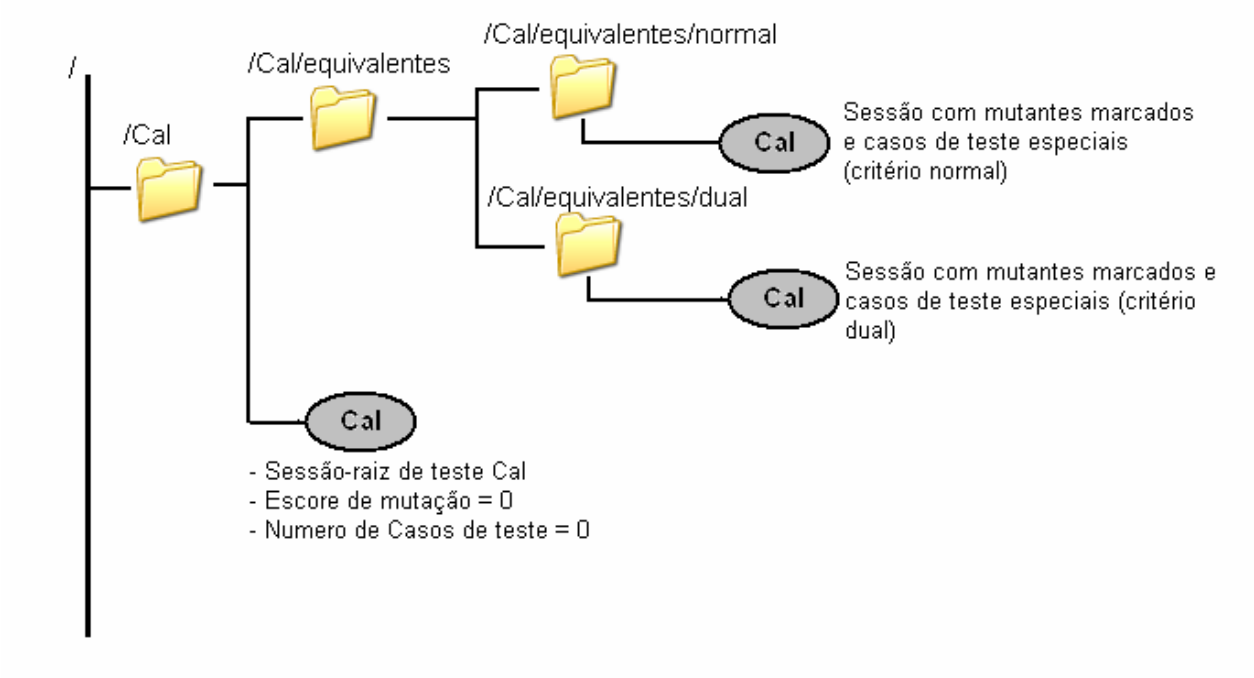


Figura 3 – Representação da estrutura dos diretórios e também das sessões de testes preparadas para a aplicação dos experimentos.

2.9 A criação das sessões de teste

Antes de iniciarmos esta parte é conveniente explicarmos um componente importante deste experimento: o servidor de números. Esse software tem a função de gerar aleatoriamente um número dentro de um intervalo especificado, ele é ativado e fica em segundo plano onde oferece seu serviço através de uma porta especificada no momento em que é inicializado.

Partindo de que o ambiente dos experimentos estejam prontos, a próxima tarefa foi a de preparar o *script* auxiliar, que é apresentado no apêndice C, para gerar 30 sessões de teste de cada critério. Essa implementação foi feita mudando-se o número de interações do laço principal e também criando um vetor com 30 sementes, que serão usadas na inicialização do servidor de números a cada interação do laço. Essa mudança de sementes a cada interação é feita para que

seja garantidas a aleatoriedade no momento da criação dos casos de teste e a independência entre as sessões.

Após a execução dessa parte do experimento, cada sessão-raiz de teste ficou com as estruturas mostradas na Figura 4. Com de 60 sub-sessões de teste onde foram importadas para cada uma delas 15.000 casos de teste, entretanto das 60 sessões, 30 foram utilizadas para a aplicação do critério de mutação normal e as outras 30 para o critério de mutação dual (Exemplo : MATAMUTANTE_01 e MATAMUTANTEDUAL_01).

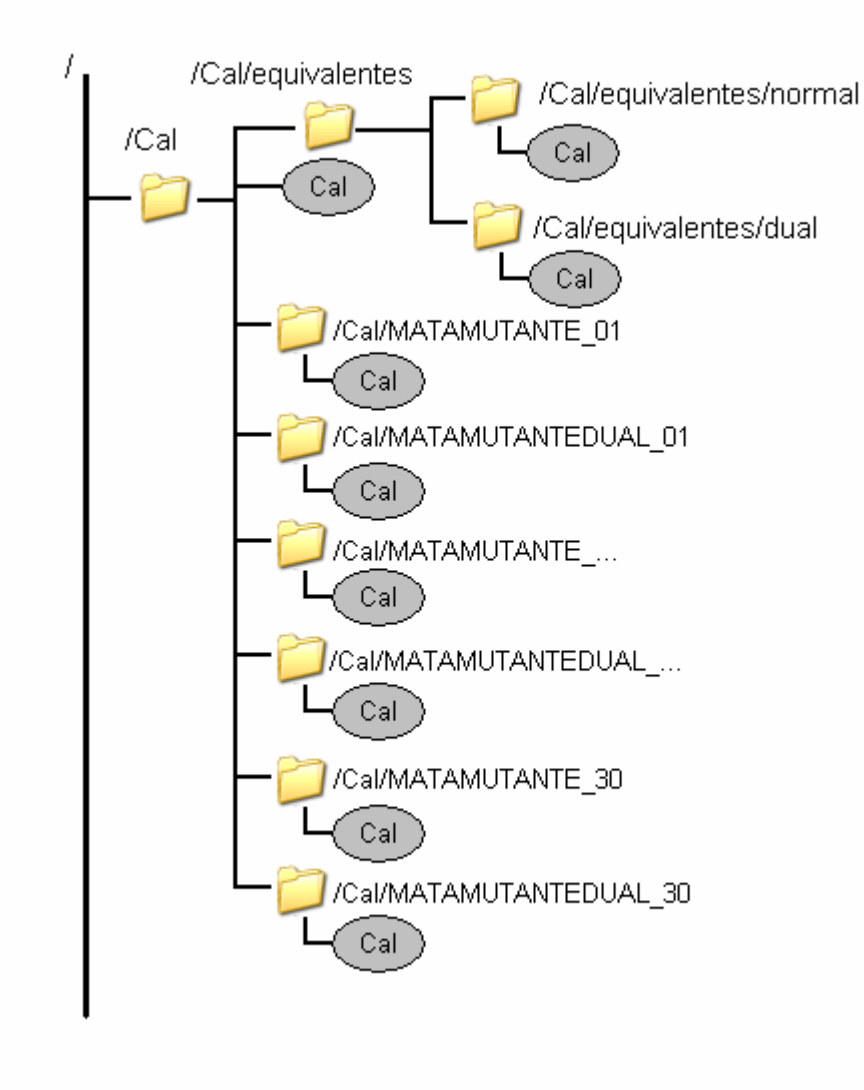


Figura 4 representa a sessão de teste Cal após a primeira etapa dos experimentos

2.10 O cruzamento dos mutantes

Após a criação das 360 sessões de teste foi preciso utilizar novamente a ferramenta PROTEUM/IM 2.0 para matar os mutantes ainda vivos em cada sessão. Essa tarefa é necessária para que possa ser feita a última etapa dos experimentos, que é a de cruzar os casos de teste. A partir do ponto em que todas as sessões de teste estão com os escore de mutação em 100 % é utilizado um outro *script* auxiliar para cruzarmos as sessões. Esse *script* funciona da seguinte maneira: O *script* auxiliar é executado a partir da sessão-raiz de teste e dentro do diretório, por exemplo, MATAMUTANTE_01, dentro deste diretório ele cria um outro diretório chamado dual (como o diretório corrente chama-se MATAMUTANTE_01, quer dizer que nesta sessão de teste foi aplicado o critério de mutação normal então no momento do cruzamento será criado dentro desta pasta um outro diretório com o nome do critério contrário que será aplicado). Após isso é criada uma nova sessão de teste dentro da pasta dual, e são importados para dentro dessa pasta os casos de teste válidos que estavam na sub-sessão anterior, que seguindo o exemplo anteriormente dado e representado pela Figura 3 seriam os casos de teste válidos da sessão de teste MATAMUTANTE_01.

Tudo isso é feito para verificarmos o quanto os casos de teste válidos para um certo critério são efetivos quando utilizados na aplicados de outro critério de teste. Toda a estrutura obtida através dos experimentos acima descritos está mostrada na Figura 5.

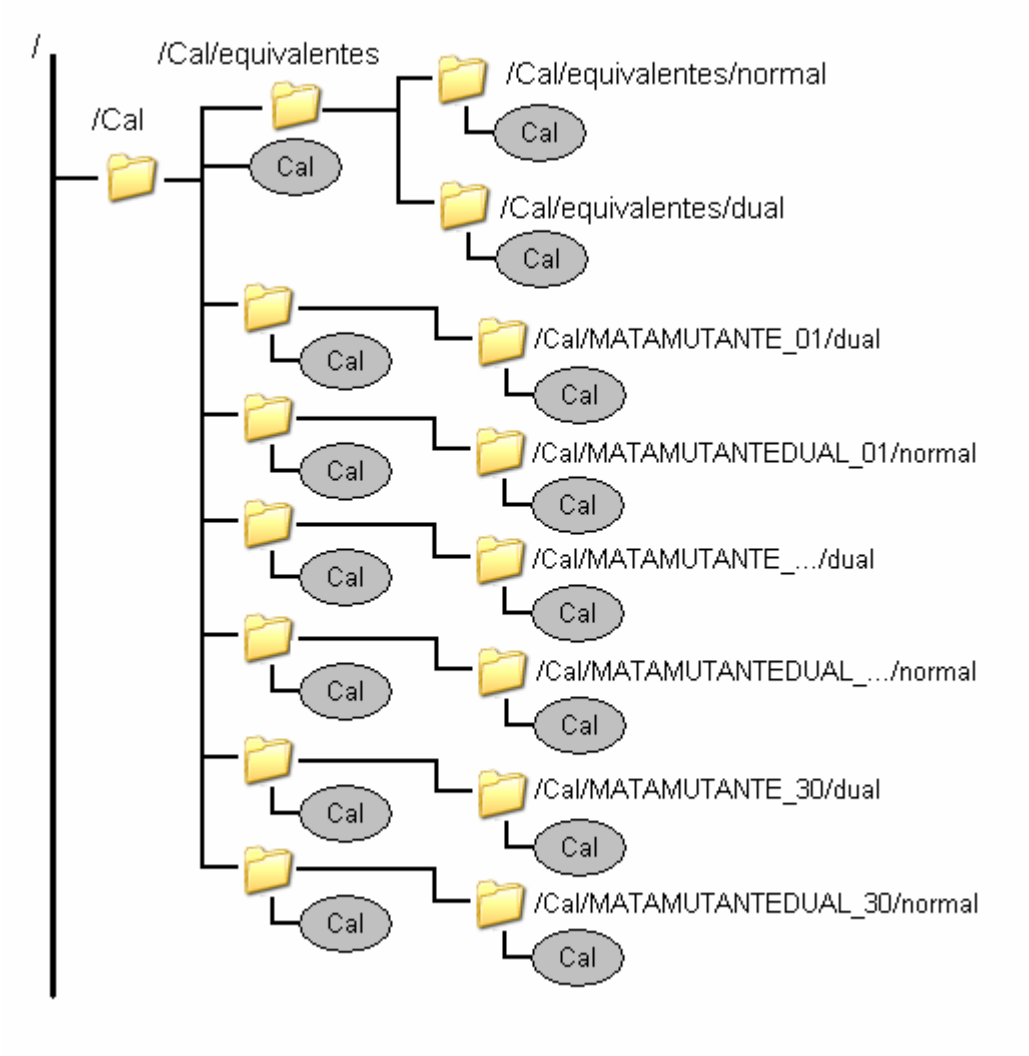


Figura 5 – Estrutura final obtida através dos experimentos descritos na seção 3.5.

3. RESULTADOS

Neste capítulo são apresentados os resultados obtidos por meio dos experimentos explicados no capítulo anterior. Mesmo esses resultados sendo provenientes dos experimentos demonstrados eles estão publicados na tese de mestrado de (WERNER, 2005). Esses resultados serão as bases para as conclusões tomadas no capítulo posterior, eles estão divididos em 3 partes, a primeira irá expor o que foi obtido durante a primeira fase dos experimentos (criação das 360 sessões de teste), a segunda irá cuidar dos resultados dos cruzamentos e a terceira é referente aos resultados metodológicos obtidos pelos experimentos.

3.1 As sessões de teste criadas

Na Tabela 2, serão apresentadas as informações colhidas nas sessões de teste antes do cruzamento, são elas:

- Número de mutantes gerados;
- Número de casos de teste válidos para cada um dos critérios;
- Número de mutantes mortos;
- Número de mutantes equivalentes;
- Escore de mutação apresentado ao final dessa parte dos experimentos.

A seguir está apresentada a descrição dos resultados:

Tabela 2 – Representa os resultados retirados da primeira fase do experimento

Programa	Número de mutantes							Escore de Cobertura	
	Gerados	Casos de Testes Requeridos		Mortos		Equivalentes			
		Normal	Dual	Normal	Dual	Normal	Dual	Normal (%)	Dual (%)
Calcdate	5.532	75 [a]	37 [b]	4.818	5.255	714 [a]	277 [b]	87,09	94,99
Calen	5.186	30 [a]	15 [b]	4.701	5.176	485 [a]	10 [b]	90,65	99,81
Calend	4.835	26 [a]	23 [b]	4.294	4.449	541 [a]	386 [b]	88,81	92,02
Cal	4.068	76 [a]	22 [b]	3.742	3.896	326 [a]	172 [b]	91,99	95,77
Jday	2.610	34 [a]	10 [b]	2.540	2.576	70 [a]	34 [b]	97,32	98,70
Dateplus	1.836	32 [a]	3 [b]	1.440	1.445	396 [a]	391 [b]	78,43	78,70

Médias com letras distintas diferem-se pelo Teste de Tukey e pelo Teste de Duncan (P<0,005).

3.2 O cruzamento das sessões de teste

Na Tabela 3, estão ilustrados os resultados do cruzamento entre os critérios, em que utilizou-se os conjuntos de casos de teste válidos de uma sessão, por exemplo, em que foi utilizando o critério de mutação dual, para executar os mutantes através do critério de mutação normal.

Tabela 3 – Medias dos resultados obtidos com os cruzamentos

Programas	Casos de Testes Requeridos		Escore de Cruzamento	
	Normal	Dual	Normal	Dual
Calcdate	64	38	0,825730 [a]	0,690660 [b]
Calen	14	12	0,892673 [a]	0,519887 [b]
Calend	25	24	0,884740 [a]	0,518610 [b]
Cal	43	14	0,942834 [a]	0,445192 [b]
Jday	24	9	0,959063 [a]	0,524340 [b]
Dateplus	8	3	0,784314 [a]	0,556609 [b]

Médias com letras distintas diferem-se pelo Teste de Tukey e pelo Teste de Duncan (P<0,005).

3.3 Os *scripts* criados

O resultados mais importantes deste trabalho são os *scripts* que foram gerados para a realização dos experimentos, pois através deles que pudemos realizar em tempo hábil todos as suas etapas.

No Apêndice A, estão os *scripts* criados a partir dos perfis operacionais dos seis programas objetos. O seu objetivo foi o de gerar um determinado número de casos de teste, onde cada um deles é armazenado em um arquivo texto diferente para que possa ser importado pela ferramenta de teste PROTEUM/IM 2.0.

No Apêndice C, são executados os dois *scripts* auxiliares para a criação das sessões de teste, o primeiro *script* foi utilizado para gerar uma sessão para a utilização de cada critério de todos os seis programas objetos e a inclusão dos 15.000 casos de teste em cada uma das sessões. O segundo *script* foi utilizado, depois que os mutantes equivalentes foram marcados, para a criação das 60 sessões de teste de cada um dos 6 programas objetos, ele também incluiu 15.000 casos de teste em cada uma das sessões criadas.

Por último, foi utilizado o *script* de cruzamento, apresentado no Apêndice D, para o cruzamento das sessões de teste e apresentação dos seus resultados. Também é importante expor como resultados a metodologia que descrevemos para aplicar os experimentos, pois foi através da seqüência das directivas apresentadas é que conseguimos o resultado esperado. Sendo isso uma prática que deveria ser mais utilizada se considerando que tais procedimentos tornam-se muito mais ágeis quando temos uma linha para seguir.

4. CONCLUSÕES

Através da metodologia utilizada foi possível chegar a diversas observações positivas sobre a atividade de teste realizada neste trabalho, sendo que a primeira é sobre a utilização de *scripts* para a mecanização da criação de casos de teste. Essa abordagem se mostrou altamente confiável, pouco custosa e rápida. Isso pode ser visto facilmente ao notarmos a quantidade de casos de teste utilizada no procedimento citado anteriormente. Esse resultado também pode ser aplicado aos *scripts* auxiliares, utilizados tanto para manipular os casos de teste quanto para a obtenção dos resultados, pois se essa atividade fosse feita manualmente seria impossível realizá-la no tempo em que foi feita.

Também é necessário dizer que seria importante para esse tipo de atividade algum tipo de algoritmo que possa ser implementado e que consiga identificar os mutantes equivalentes, por se tratar da parte mais custosa da atividade realizada.

Seria interessante também a proposta de um algoritmo que pudesse, por meio de um interpretador de comandos, identificar automaticamente o perfil operacional do programa a ser testado. Isso poderia ser feito analisando os parâmetros de entrada do programa, dando ênfase a tipos e intervalos de valores, a fim de tornar esse trecho da atividade de teste menos custosa e manual.

Por fim, é importante citar a importância deste tipo de trabalho para a atividade de teste, pois poderá servir de modelo para experimentos ou até mesmo testes futuros ficando aqui registrado entre outras coisas a eficácia da utilização de *scripts* neste tipo de atividade.

BIBLIOGRAFIA

ACREE, A. T.; BUDD, T. A.; DEMILLO R. A.; LIPTON R. J.; and SAYWARD F. G. ***Mutation analysis. Technical Report*** GIT-ICS-79/08, Georgia Institute of Technology, Atlanta, GA, September 1979.

AGRAWAL, H.; DEMILLO, R. A.; HATHAWAY, R.; HSU, W.; KRAUSER, E. W.; MARTIN, R. J.; MATHUR, A. P. and SPAFFORD, E. H.; ***Design of mutant operators for the C programming language.*** Technical Report SERC-TR41-P, Software Engineering Research Center, Purdue University, West Lafayette, IN, March 1989.

BIANCHINI[1], S. L.; VINCENZI[1], A. M. R.; DELAMARO[2], M. E.; SIMÃO[1], A. S.; BARBOSA[1], E. F.; MALDONADO[1], J. C.; ***PROTEUM/IM Version 2.0 C – User´s Guide***, ICMC/USP[1] , São Carlos - SP ; FIM/UNIVEM[2] , Marília – SP. 2000.

BUDD T. A. ***Mutation Analysis of Program Test Data***. PhD thesis, Yale University, New Haven, CT, 1980.

DELAMARO, M. E.. Mutação **de Interface: Um Critério de Adequação Interrocedimental para o Teste de Integração**, Tese (Doutorado), Instituto de Física de São Carlos - Universidade de São Paulo, São Carlos – SP . 1997.

DELAMARO, M. E.; ***Dual Mutation : The " Save the Mutants "*** Approach, UNIVEM, Marília – SP, 2003.

DELAMARO, M. E.; MALDONADO, J. C. and MATHUR, A. P. ***Interface mutation: An approach for integration testing***. *IEEE Transactions on Software Engineering*, 27(3):228.247, Março/2001.

DELAMARO, M.E. MALDONADO, A. PASQUINI, MATHUR, A. P. ***Interface Mutation Adequacy Criterion: Na Empirical Evaluatino***.1998.

DEMILLO, R. A. ***Mutation analysis as a tool for software quality assurance***. In *OMPSAC80*, Chicago, IL, October 1980.

DEMILLO, R. A., LIPTON R. J., SAYWARD, F. G., ***Hints on test data selection : Help for the Practicing Programmer***. IEEE Computer, 11(4):34.43, April 1978.

FRANKL,F. G. The ***Use of Data Flow Information for the Selection and Evaluation of Software Test Data***. PhD thesis, Universidade de New York, New York, NY, October 1987.

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

HOWDEN, W. E. ***Functional Program Testing and Analysis***. McGrall-Hill, New York,1987.

JORGE, R. F.; VINCENZI, A. M. R.; DELAMARO, M. E.; MALDONADO, J. C.; **Relatório dos Operadores de Mutação implementados nas ferramentas Proteum e PROTEUM/IM**., ICMC/USP, p 34, São Carlos – SP, 16/10/2001.

MALDONADO,J.C; BARBOSA,E.F; VINCENZI,A.M.R; DELAMARO,M.E; SOUZA,S.R.S; JINO, M.; **Introdução ao Teste de Software**, Relatório Técnico do ICMC – USP , São Carlos, 1998.

MYERS,G., **The Art of Software Testing**, Wiley, 1979.

NTAFOS, S. C. ***A comparison of some structural testing strategies***. IEEE Transactions on Software Engineering, 14(6):868-873, July 1988.

OFFUTT, A. J.; ROTHERMEL, G.; LEE, A.; UNTCH, R. H.; and ZAPF, C.; ***An experimental determination of sufficient mutant operators***. *ACM Transactions on Software Engineering Methodology*, 5(2):99 ñ118, April 1996.

PRESSMAN,R.S; **Engenharia de Software**;.McGraw-Hill, 5.ed. p. 816 ,Rio Janeiro,2002.

RAPPS, S. and WEYUKER, E. J. ***Selecting software test data using data flow information***. IEEE Transactions on Software Engineering, 11(4):367-375, April 1985.

SOUZA, S. R. S.; **Avaliação do custo e eficácia do critério análise de mutantes na atividade de teste de programas**. Master's thesis, ICMC-USP, S,,o Carlos ñ SP, June 1996.

VERGÍLIO, S. R.; MALDONADO, J. C.; e JINO,M. **Uma estratégia para a geração de dados de teste**. In VII Simpósio Brasileiro de Engenharia de Software, pg. 307-319, Rio de Janeiro, RJ, 10/1993.

VINCENZI, A. M. R., **Subsidio para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação**, ICMC/USP, São Carlos- SP, 1998.

WERNER, C.; DELAMARO, M. E.; Avaliação **Experimental do Critério de Teste Mutação Dual** – PPGCC/UNIVEM, Marília – SP, 10/2005.

APÊNDICE A

Programa Calcdate.

```
#!/bin/tcsh
# $1 - número da porta do servidor de números aleatórios
# $2 - nome do arquivo a gravar
# $3 - número de casos de teste a gerar
#
#------------------------------------------------------------
#-- Entradas válidas:
#-- 1o. parâmetro: 010100 até 123199
#-- 2o. parâmetro: idem para data
#--              -36524 até 36524 para offset
#-- Perfil operacional: calcdate
#--          95% dos casos de teste são validos
#--          Dos casos válidos:
#--                      50% com -d
#--                      50% com -o
#--
#--          Dos casos não válidos:
#--                      1% com um Único parâmetro
#--          1% sem parâmetros
#--                      14% com mês não válido 1o.  parâmetro
#--                      14% com ano não válido 1o.  parâmetro
#--                      14% com dia não valido 1o.  parâmetro
#--                      14% com mês não válido 2o.  parâmetro
#--                      14% com ano não válido 2o.  parâmetro
#--                      14% com dia não valido 2o.  parâmetro
#--          14% com offset inválido
#--
#------------------------------------------------------------

if ( $EXPERIMENTHOME == "") then
  echo "Variável EXPERIMENTHOME não setada"
  exit
endif
set tc1 = $2
set SCRIPTDIR = $EXPERIMENTHOME/scripts
@ N = $3
@ i = 1
```

```
while ( $i <= $N  )
            @ x = `$SCRIPTDIR/client_int $1 0 100`
            #echo "Valido/invalido: $x"

            if ( $x < 95 ) then
            # caso de teste valido
          @ d = `$SCRIPTDIR/client_int $1 1 32`
          @ m = `$SCRIPTDIR/client_int $1 1 13`
          @ a = `$SCRIPTDIR/client_int $1 0 100`
          if ( $d < 10 ) set d = "0$d"
          if ( $m < 10 ) set m = "0$m"
          if ( $a < 10 ) set a = "0$a"

              @ tipo = `$SCRIPTDIR/client_int $1 0 100`
              # echo "Tipo: $tipo"
              if ( $tipo >= 50 ) then
                echo -n "$m$d$a -d " > $tc1$i
            @ d1 = `$SCRIPTDIR/client_int $1 1 32`
            @ m1 = `$SCRIPTDIR/client_int $1 1 13`
            @ a1 = `$SCRIPTDIR/client_int $1 0 100`
            if ( $d1 < 10 ) set d1 = "0$d1"
            if ( $m1 < 10 ) set m1 = "0$m1"
            if ( $a1 < 10 ) set a1 = "0$a1"
                echo "$m1$d1$a1" >> $tc1$i
              else
                @ ofs = `$SCRIPTDIR/client_int $1 0 36525`
            if ( $tipo % 2 == 0 ) @ ofs = -$ofs
                echo "$m$d$a -o $ofs" > $tc1$i
              endif
            else
            # caso não valido
              @ tipo = `$SCRIPTDIR/client_int $1 0 100`
              #echo "Tipo: $tipo"
              @ flag1 = `$SCRIPTDIR/client_int $1 0 2`
              @ flag2 = `$SCRIPTDIR/client_int $1 0 2`
          @ d = `$SCRIPTDIR/client_int $1 1 32`
          if ( $d < 10 ) set d = "0$d"
          @ m = `$SCRIPTDIR/client_int $1 1 13`
          if ( $m < 10 ) set m = "0$m"
          @ a = `$SCRIPTDIR/client_int $1 0 100`
           if ( $a < 10 ) set a = "0$a"
          @ d1 = `$SCRIPTDIR/client_int $1 1 32`
          if ( $d1 < 10 ) set d1 = "0$d1"
          @ m1 = `$SCRIPTDIR/client_int $1 1 13`
          if ( $m1 < 10 ) set m1 = "0$m1"
```

```
@ a1 = `$SCRIPTDIR/client_int $1 0 100`
if ( $a1 < 10 ) set a1 = "0$a1"

      # tipo 0 : sÃ³ um parâmetro
        if ( $tipo == 0 ) then
          echo "$m$d$a " > $tc1$i
        endif
      # tipo 1: nenhum parâmetro
        if ( $tipo == 1 ) then
          echo " " > $tc1$i
       endif
      # tipo 2: dia não valido
       if ( $tipo >= 2 && $tipo <= 15 ) then

          @ decide = `$SCRIPTDIR/client_int $1 0 100`

          if ($decide == 0) then
                  @ d = 0
          else
                  @ d = `$SCRIPTDIR/client_int $1 32 100`
          endif
          echo "$m$d$a -d $m1$d1$a1" > $tc1$i
        endif
      # tipo 3: mês não valido
       if ( $tipo >= 16 && $tipo <= 29 ) then

          @ decide = `$SCRIPTDIR/client_int $1 0 100`

          if ($decide == 0) then
                  @ m = 0
          else
                  @ m = `$SCRIPTDIR/client_int $1 13 100`
          endif

         echo "$m$d$a -d $m1$d1$a1" > $tc1$i
        endif
      # tipo 4: ano não valido
       if ( $tipo >= 30 && $tipo <= 43 ) then

          @ decide = `$SCRIPTDIR/client_int $1 0 100`

          if ($decide == 0) then
                  @ a = 0
          else
                  @ a = `$SCRIPTDIR/client_int $1 100 1000`
```

```
    endif

    echo "$m$d$a -d $m1$d1$a1" > $tc1$i
  endif
# tipo 5: dia 2 não valido
  if ( $tipo >= 44 && $tipo <= 57 ) then

     @ decide = `$SCRIPTDIR/client_int $1 0 100`

    if ($decide == 0) then
            @ d1 = 0
    else
            @ d1 = `$SCRIPTDIR/client_int $1 32 100`
    endif

    echo "$m$d$a -d $m1$d1$a1" > $tc1$i
  endif
# tipo 6: mês 2 não valido
  if ( $tipo >= 58 && $tipo <= 71 ) then

    @ decide = `$SCRIPTDIR/client_int $1 0 100`

    if ($decide == 0) then
            @ m1 = 0
    else
            @ m1 = `$SCRIPTDIR/client_int $1 13 100`
    endif
    echo "$m$d$a -d $m1$d1$a1" > $tc1$i
  endif
# tipo 7: ano 2 não valido
  if ( $tipo >= 72 && $tipo <= 85 ) then

     @ decide = `$SCRIPTDIR/client_int $1 0 100`

    if ($decide == 0) then
            @ a1 = 0
    else
            @ a1 = `$SCRIPTDIR/client_int $1 100 1000`
            if ( $flag1 ) @ a1 = - $a1

    endif


    echo "$m$d$a -d $m1$d1$a1" > $tc1$i
  endif
```

```
            # tipo 8: offset não valido
              if ( $tipo >= 86 ) then

                  @ decide = `$SCRIPTDIR/client_int $1 0 2`

                if ($decide == 0) then
                        @ a1 = `$SCRIPTDIR/client_int $1 36525 100001`
                        @ a1 = - $a1

                else
                        @ a1 = `$SCRIPTDIR/client_int $1 36525 100001`

                endif
                echo "$m$d$a -o $a1" > $tc1$i
              endif
      endif
            @ i = $i + 1
end
```

Programa Calen.

```
#!/bin/tcsh
# $1 - número da porta do servidor de números
aleatórios
# $2 nome do arquivo a gravar
# $3 - número de casos de teste a gerar
#
#------------------------------------------------------------
#-- Entradas válidas:
#-- Disposição das opções de entrada e parâmetros :
#--
#--          As opções podem ser:
#--          -l (justifica as datas a esquerda) ou
#--          -r (justifica as datas a direita) ou
#--          nenhuma dessas opções mais
#--          -m (deixa os dias da semana em maiúsculas e minúsculas) ou
#--          -u (deixa os dias da semana em maiúsculas) ou
#--          nenhuma dessas opções mais
#--          -o[seq_caract] (seqüência de caracteres que forma o nome do mês) mais
#--          -bN (acrescenta linhas em branco entre os calendários) mais
#--          nenhuma dessas opções.
#--          Os parâmetros podem ser
#--          1o parâmetro: Pode ser somente o ano do calendário de 1753 a 9999
#--          ou o mês de 1 a 12.
#--          2o parâmetro: se o primeiro parâmetro for um mês o segundo será um
#--          ano também de 1753 a 9999
#--          3o parâmetro: só pode ser usado se usamos o formato mês e ano , esse
#--          terceiro parâmetro é o numero de meses que queremos gerar o
#--          calendário a partir do mês e ano especificados nos 1o e 2o
#--          parâmetros.
#--
#--
#--          Perfil operacional: calen.c
#--
#--          90% dos casos de teste são validos
#--          Dos casos válidos:
#--                    34% com um parâmetro de ano
#--                    33% com um parâmetro de mês e um de ano
#--                    33% com um parâmetro de mês, um de ano e um de numero de mêses
#--                    Obs: cada um dos casos válidos acima citados, cada um poderá ter a probabilidade de -#--          20% d
#--                    opções listadas abaixo:
#--          20% de probabilidade de não possuir nenhuma opção.
```

```
#--          20% de probabilidade de possuir somente uma opção.
#--          20% de probabilidade de possuir duas opções.
#--          20% de probabilidade de possuir três.
#--          20% de probabilidade de possuir quatro opções.
#--
#--          10% Dos casos são não válidos:
#--
#--
#--            50% dos casos não válidos tem a probabilidade de ser :
#--
#--                        10% com um único parâmetro ano inválido
#--                        09% com o parâmetro mês inválido e o ano válido
#--                        09% com o parâmetro mês válido e o ano inválido
#--                        09% com os dois parâmetros mês e ano inválidos
#--
#--                        09% com o parâmetro mês inválido e os parâmetros ano e numero_de_mêses válidos
#--                        09% com o parâmetro mês válido, o ano inválido e o numero_de_mêses válido
#--                        09% com o parâmetro mês válido, o ano válido e o número_de_mêses inválido
#--
#--                        09% com o parâmetro mês inválido, o ano inválido e o número_de_mêse válido
#--                        09% com o parâmetro mês inválido, o ano válido e o número_de_mêses inválido
#--                        09% com o parâmetro mês válido, o ano inválido e o número_de_mêses inválido
#--                        09% com os parâmetros mês, ano e número_de_mêses inválido.
#--
#--          50% dos casos não válido tem a probabilidade de ser o quer for escrito abaixo somente se um ou #--
mais opções forem inválidas assim mesmo que um ou todos os parâmetros sejam válidos
#--          queremos uma opção inválida para a caracterização de um caso de teste não válido.
#--
#--                        34% com um parâmetro de ano
#--                        33% com um parâmetro de mês e um de ano
#--                        33% com um parâmetro de mês, um de ano e um de numero de meses
#--
#--          Aqui mostraremos como que as opções determinarão como os casos de teste não válidos serão
#--          serão escolhidos:
#--          conforme as opções vão sendo escolhidas, e se alguma delas for inválida será atribuído um valor #--          pos
#--          decisão que verifica se o valor gerado é menor que 50% só será permitido a execução entre
#--          nessa opção se ocorreu pelo menos a de uma opção inválida , caso contrário seria
#--          descaracterizado o caso de teste com inválido, pois temos de um a três parâmetros válidos nessa #--
opção.
#--
#--
#--
#-------------------------------------------------------------------------------
```

```
if ( $EXPERIMENTHOME == "") then
  echo "Variável EXPERIMENTHOME não setada"
  exit
endif
set tc1 = $2
set SCRIPTDIR = $EXPERIMENTHOME/scripts
@ N = $3
@ i = 1
while ( $i <= $N  )
            @ x = `$SCRIPTDIR/client_int $1 0 100`
            #echo "Valido/invalido: $x"

            @ flag1 = `$SCRIPTDIR/client_int $1 0 5`
            @ flag2 = `$SCRIPTDIR/client_int $1 0 5`
            @ flag3 = `$SCRIPTDIR/client_int $1 0 5`
            @ flag4 = `$SCRIPTDIR/client_int $1 0 5`
            @ flag5 = `$SCRIPTDIR/client_int $1 0 5`



            if ( $x < 90 ) then
            # caso de teste valido

              @ mês = `$SCRIPTDIR/client_int $1 1 13`
              @ ano = `$SCRIPTDIR/client_int $1 1753 10000`
              @ numeroMêses = `$SCRIPTDIR/client_int $1 1 10000`

               if ($flag1 == 0) then
                        @ chave1 = `$SCRIPTDIR/client_int $1 0 2`
                        if ( $chave1 == 0) then
                          set opcao1 = "-l"
                        else
                          set opcao1 = "-r"
                        endif
             else
                        if ( $flag1 == 1 ) then
                           @ chave1 = `$SCRIPTDIR/client_int $1 0 2`

                           if ( $chave1 == 0) then
                             set opcao1 = "-u"
                           else
                             set opcao1 = "-m"
                           endif
                        else
                           if ($flag1 == 2) then
```

```
                    set opcao1 = "-o[111]"
                else
                    if ($flag1 == 3) then
                      set opcao1 = "-bN"
                    else
                      set opcao1 = " "
                    endif
                endif

            endif
endif

if( $flag2 == 4) then
  set opcao2 = " "
else

  while ( $flag1 == $flag2 )
    @ flag2 = `$SCRIPTDIR/client_int $1 0 4`
  end
  if ($flag2 == 0) then
            @ chave1 = `$SCRIPTDIR/client_int $1 0 2`

            if ( $chave1 == 0) then
              set opcao2 = "-l"
            else
              set opcao2 = "-r"
            endif
  else
            if ( $flag2 == 1 ) then
              @ chave1 = `$SCRIPTDIR/client_int $1 0 2`
              if ( $chave1 == 0) then
                set opcao2 = "-u"
              else
                set opcao2 = "-m"
              endif
            else
              if ($flag2 == 2) then
                    set opcao2 = "-o[111]"
              else
                    if ($flag2 == 3) then
                      set opcao2 = "-bN"
                    endif
              endif

            endif
```

```
  endif
endif

if ($flag3 == 4) then
  set opcao3 = " "
else
  while ($flag1 == $flag3 || $flag2 == $flag3)
    @ flag3 = `$SCRIPTDIR/client_int $1 0 4`
  end
  if ($flag3 == 0) then
            @ chave1 = `$SCRIPTDIR/client_int $1 0 2`
            if ( $chave1 == 0) then
              set opcao3 = "-l"
            else
              set opcao3 = "-r"
            endif
  else
            if ( $flag3 == 1 ) then
               @ chave1 = `$SCRIPTDIR/client_int $1 0 2`
               if ( $chave1 == 0) then
                 set opcao3 = "-u"
               else
                 set opcao3 = "-m"
               endif
            else
               if ($flag3 == 2) then
                        set opcao3 = "-o[111]"
               else
                        if ($flag3 == 3) then
                           set opcao3 = "-bN"
                        endif
               endif

            endif
  endif
 endif


 if ($flag4 == 4) then
  set opcao4 = " "
 else
  while ( $flag1 == $flag4 || $flag2 == $flag4 || $flag3 == $flag4 )
    @ flag4 = `$SCRIPTDIR/client_int $1 0 4`
  end
  if ($flag4 == 0) then
```

```
            @ chave1 = `$SCRIPTDIR/client_int $1 0 2`
            if ( $chave1 == 0) then
              set opcao4 = "-l"
            else
              set opcao4 = "-r"
            endif
    else
            if ( $flag4 == 1 ) then
              @ chave1 = `$SCRIPTDIR/client_int $1 0 2`

              if ( $chave1 == 0) then
                set opcao4 = "-u"
              else
                set opcao4 = "-m"
              endif

            else

               if ($flag4 == 2) then
                        set opcao4 = "-o[111]"
               else
                        if($flag4 == 3) then
                          set opcao4 = "-bN"
                        endif
               endif
            endif
     endif
   endif

   @ tipo = `$SCRIPTDIR/client_int $1 0 100`
   # echo "Tipo: $tipo"

   if ( $tipo < 34 ) then
            echo "$opcao1 $opcao2 $opcao3 $opcao4 $ano" > $tc1$i
   endif
   if ( $tipo > 33 && $tipo < 77 ) then
            echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano" > $tc1$i
   endif
   if ( $tipo > 66 && $tipo < 100 ) then
            echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano $numeroMêses" > $tc1$i
   endif
 else

   @ mês = `$SCRIPTDIR/client_int $1 1 13`
   @ ano = `$SCRIPTDIR/client_int $1 1753 10000`
```

```
@ numeroMêses = `$SCRIPTDIR/client_int $1 1 10000`

@ n_op_validas = `$SCRIPTDIR/client_int $1 0 5`
@ cont = 0

@ controle1 = 0
@ controle2 = 0
@ controle3 = 0
@ controle4 = 0
if ($n_op_validas == 0 ) then
  set opcao1 = "-f"
  set opcao2 = "-g"
  set opcao3 = "-h"
  set opcao4 = "-i"
else
  if($n_op_validas == 1) then
          @ controle1 = 1
          set opcao2 = "-g"
          set opcao3 = "-h"
          set opcao4 = "-i"
  else
    if($n_op_validas == 2) then

            @ controle1 = 1
            @ controle2 = 1
            set opcao3 = "-h"
            set opcao4 = "-i"

          else
            if($n_op_validas == 3) then

                    @ controle1 = 1
                    @ controle2 = 1
                    @ controle3 = 1
                    set opcao4 = "-i"
            else
                    @ controle1 = 1
                    @ controle2 = 1
                    @ controle3 = 1
                    @ controle4 = 1
            endif
          endif
  endif
endif
```

```
if($controle1 == 1) then
  if ($flag1 == 0) then
            @ chave1 = `$SCRIPTDIR/client_int $1 0 2`
            if ( $chave1 == 0) then
              set opcao1 = "-l"
            else
              set opcao1 = "-r"
            endif
else
            if ( $flag1 == 1 ) then
               @ chave1 = `$SCRIPTDIR/client_int $1 0 2`

               if ( $chave1 == 0) then
                 set opcao1 = "-u"
               else
                 set opcao1 = "-m"
               endif

            else

                if ($flag1 == 2) then
                          set opcao1 = "-o[111]"
                else
                          if ($flag1 == 3) then
                            set opcao1 = "-bN"
                          else
                            set opcao1 = " "
                          endif
                endif

            endif
  endif
endif
if( $controle2 == 1 ) then
 if( $flag2 == 4) then

  set opcao2 = " "
 else

  while ( $flag1 == $flag2 )
    @ flag2 = `$SCRIPTDIR/client_int $1 0 4`
  end
  if ($flag2 == 0) then
            @ chave1 = `$SCRIPTDIR/client_int $1 0 2`
```

```
            if ( $chave1 == 0) then
              set opcao2 = "-l"
            else
              set opcao2 = "-r"
            endif
   else
            if ( $flag2 == 1 ) then
               @ chave1 = `$SCRIPTDIR/client_int $1 0 2`

               if ( $chave1 == 0) then
                 set opcao2 = "-u"
               else
                 set opcao2 = "-m"
               endif

            else

               if ($flag2 == 2) then
                        set opcao2 = "-o[111]"
               else
                        if ($flag2 == 3) then
                           set opcao2 = "-bN"
                        endif
               endif

            endif
   endif
 endif
endif
if ($controle3 == 1) then
 if ($flag3 == 4) then

   set opcao3 = " "
 else
   while ($flag1 == $flag3 || $flag2 == $flag3)
     @ flag3 = `$SCRIPTDIR/client_int $1 0 4`
   end
   if ($flag3 == 0) then
            @ chave1 = `$SCRIPTDIR/client_int $1 0 2`

            if ( $chave1 == 0) then
              set opcao3 = "-l"
            else
              set opcao3 = "-r"
            endif
```

```
  else
            if ( $flag3 == 1 ) then
              @ chave1 = `$SCRIPTDIR/client_int $1 0 2`

              if ( $chave1 == 0) then
                set opcao3 = "-u"
              else
                set opcao3 = "-m"
              endif

            else

               if ($flag3 == 2) then
                        set opcao3 = "-o[111]"
               else
                        if ($flag3 == 3) then
                          set opcao3 = "-bN"
                        endif
               endif

            endif
  endif
 endif
endif
if ($controle4 == 1) then
 if ($flag4 == 4) then
  set opcao4 = " "

 else
  while ( $flag1 == $flag4 || $flag2 == $flag4 || $flag3 == $flag4 )
    @ flag4 = `$SCRIPTDIR/client_int $1 0 4`
  end
  if ($flag4 == 0) then
            @ chave1 = `$SCRIPTDIR/client_int $1 0 2`

            if ( $chave1 == 0) then
              set opcao4 = "-l"
            else
              set opcao4 = "-r"
            endif
  else
            if ( $flag4 == 1 ) then
              @ chave1 = `$SCRIPTDIR/client_int $1 0 2`

              if ( $chave1 == 0) then
```

```
                    set opcao4 = "-u"
                  else
                    set opcao4 = "-m"
                  endif

                else

                  if ($flag4 == 2) then
                        set opcao4 = "-o[111]"
                  else
                        if($flag4 == 3) then
                          set opcao4 = "-bN"
                        endif
                  endif
                endif
          endif
         endif
        endif

        @ tipo1 = `$SCRIPTDIR/client_int $1 0 100`

        if ($tipo1 < 50 && $n_op_validas < 4 ) then
          @ tipo = `$SCRIPTDIR/client_int $1 0 100`
          # echo "Tipo: $tipo"

          if ( $tipo < 34 ) then

                echo "$opcao1 $opcao2 $opcao3 $opcao4 $ano" > $tc1$i
          endif

          if ( $tipo >33 && $tipo < 67 ) then

                echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano" > $tc1$i
          endif

          if ( $tipo >66 ) then

                echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano $numeroMêses" > $tc1$i
endif

        else

          @ tipo = `$SCRIPTDIR/client_int $1 0 100`

          @ mêsInicial = 1
```

```
@ mêsFinal = 13
@ limiteInicial1 = -1000
@ limiteFinal1 = 1000

@ anoInicial = 1753
@ anoFinal = 10000
@ limiteInicial2 = -1000
@ limiteFinal2 = 20000

@ numeroInicial = 1
@ numeroFinal = 10001
@ limiteInicial3 = -1000
@ limiteFinal3 = 20000

@ mês = 1
@ ano = 1753
@ numeroMêses = 1


if ($tipo < 10) then
                                # ano inválido
        while ( $ano >= $anoInicial && $ano < $anoFinal )
          @ ano = `$SCRIPTDIR/client_int $1 $limiteInicial2 $limiteFinal2`

        end
        echo "$opcao1 $opcao2 $opcao3 $opcao4 $ano" > $tc1$i
endif
if ($tipo >= 10 && $tipo < 19) then
                                        # mês invalido ano valido

        @ ano = `$SCRIPTDIR/client_int $1 $anoInicial $anoFinal`

        while ( $mês >= $mêsInicial && $mês < $mêsFinal )
          @ mês = `$SCRIPTDIR/client_int $1 $limiteInicial1 $limiteFinal1`

        end
        echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano" > $tc1$i
endif
if ($tipo >= 19 && $tipo < 28) then
                                        # mês valido ano invalido
        @ mês = `$SCRIPTDIR/client_int $1 $mêsInicial $mêsFinal`

        while ( $ano >= $anoInicial && $ano < $anoFinal )
          @ ano = `$SCRIPTDIR/client_int $1 $limiteInicial2 $limiteFinal2`
```

```
        end
        echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano" > $tc1$i
endif
if ($tipo >= 28 && $tipo < 37) then

                                        # mês invalido ano invalido
        while ( $ano >= $anoInicial && $ano < $anoFinal )
          @ ano = `$SCRIPTDIR/client_int $1 $limiteInicial2 $limiteFinal2`

        end

        while ( $mês >= $mêsInicial && $ano < $mêsFinal )
          @ mês = `$SCRIPTDIR/client_int $1 $limiteInicial1 $limiteFinal1`

        end
        echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano" > $tc1$i
endif
if ($tipo >= 37 && $tipo < 46) then
                                # mês invalido ano valido e numero de meses valido
        @ ano = `$SCRIPTDIR/client_int $1 $anoInicial $anoFinal`

        while ( $mês >= $mêsInicial && $mês < $mêsFinal )
          @ mês = `$SCRIPTDIR/client_int $1 $limiteInicial1 $limiteFinal1`

        end

        @ numeroMêses = `$SCRIPTDIR/client_int $1 $numeroInicial $numeroFinal`

        echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano $numeroMêses" > $tc1$i
endif
 if ($tipo >= 46 && $tipo < 55) then
                                # mês valido ano invalido e numero de meses valido
        @ mês = `$SCRIPTDIR/client_int $1 $mêsInicial $mêsFinal`

        while ( $ano >= $anoInicial && $ano < $anoFinal )
          @ ano = `$SCRIPTDIR/client_int $1 $limiteInicial2 $limiteFinal2`

        end

        @ numeroMêses = `$SCRIPTDIR/client_int $1 $numeroInicial $numeroFinal`

        echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano $numeroMêses" > $tc1$i
endif

if ($tipo >= 55 && $tipo < 64) then
```

```
                                # mês valido ano valido e numero de meses invalido
        @ mês = `$SCRIPTDIR/client_int $1 $mêsInicial $mêsFinal`
        @ ano = `$SCRIPTDIR/client_int $1 $anoInicial $anoFinal`
        while ( $numeroMêses >= $numeroInicial && $numeroMêses < $numeroFinal )
          @ numeroMêses = `$SCRIPTDIR/client_int $1 $limiteInicial3 $limiteFinal3`

        end

        echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano $numeroMêses" > $tc1$i
endif
if ($tipo >= 64&& $tipo < 73) then

                                # mês invalido ano invalido e numero de meses valido
        while ( $ano >= $anoInicial && $ano < $anoFinal )
          @ ano = `$SCRIPTDIR/client_int $1 $limiteInicial2 $limiteFinal2`

        end

        while ( $mês >= $mêsInicial && $ano < $mêsFinal )
          @ mês = `$SCRIPTDIR/client_int $1 $limiteInicial1 $limiteFinal1`

        end

        @ numeroMêses = `$SCRIPTDIR/client_int $1 $numeroInicial $numeroFinal`

        echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano $numeroMêses" > $tc1$i
endif
 if ($tipo >= 73 && $tipo < 82) then
                                # mês invalido ano valido e numero de meses invalido
        @ ano = `$SCRIPTDIR/client_int $1 $anoInicial $anoFinal`

        while ( $mês >= $mêsInicial && $mês < $mêsFinal )
          @ mês = `$SCRIPTDIR/client_int $1 $limiteInicial1 $limiteFinal1`

        end

        while ( $numeroMêses >= $numeroInicial && $numeroMêses < $numeroFinal )
          @ numeroMêses = `$SCRIPTDIR/client_int $1 $limiteInicial3 $limiteFinal3`

        end
        echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano $numeroMêses" > $tc1$i
endif
if ($tipo >= 82 && $tipo < 91) then
                                # mês valido ano invalido e numero de meses invalido
        @ mês = `$SCRIPTDIR/client_int $1 $mêsInicial $mêsFinal`
```

```
                while ( $ano >= $anoInicial && $ano < $anoFinal )
                  @ ano = `$SCRIPTDIR/client_int $1 $limiteInicial2 $limiteFinal2`

                end

                while ( $numeroMêses >= $numeroInicial && $numeroMêses < $numeroFinal )
                  @ numeroMêses = `$SCRIPTDIR/client_int $1 $limiteInicial3 $limiteFinal3`

                end

                echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano $numeroMêses" > $tc1$i
            endif
            if ($tipo >= 91 && $tipo < 100) then
                                    # mês invalido ano invalido e numero de meses invalido

                while ( $mês >= $mêsInicial && $mês < $mêsFinal )
                  @ mês = `$SCRIPTDIR/client_int $1 $limiteInicial1 $limiteFinal1`

                end

                while ( $ano >= $anoInicial && $ano < $anoFinal )
                  @ ano = `$SCRIPTDIR/client_int $1 $limiteInicial2 $limiteFinal2`

                end

                while ( $numeroMêses >= $numeroInicial && $numeroMêses < $numeroFinal )
                  @ numeroMêses = `$SCRIPTDIR/client_int $1 $limiteInicial3 $limiteFinal3`

                end

                echo "$opcao1 $opcao2 $opcao3 $opcao4 $mês $ano $numeroMêses" > $tc1$i
            endif
          endif
        endif
        @ i = $i + 1
end
```

Programa Calend.

```
#!/bin/tcsh
# $1 - número da porta do servidor de números aleatórios
# $2 nome do arquivo a gravar
# $3 - número de casos de teste a gerar
#
#-------------------------------------------------------------------------------------------
#--
#--Entradas Válidas:
#--
#--1o Parâmetro:
#--
#--Se for valores de 1 a 12
#--          Então é mostrado o calendário do mês indicado mais o anterior e o posterior do ano
#--    corrente.
#--Senão
#--          Se for passado valores entre 13 e 9999
#--                    Então o valor passado é assumido como um ano e aparece o calendário daquele ano.
#--          Senão
#--               Como nenhum valor foi passado o programa considera que o mês e ano a ser utili-
#--                    zado são os correntes.
#--          FimSe
#--FimSe
#--
#--2o Parâmetro: O Segundo Parâmetro necessariamente será um ano se o primeiro for um valor de 1 a
#--12, esse parâmetro poderá estar no intervalo de 0 a 9999. Agora se quisermos
#--que o segundo parâmetro corresponda ao mês teremos que usar os valores para ano que estão no
#--conjunto que corresponde entre 13 9999.
#--
#--
#--Perfil Operacional : pcal
#--
#--          90% dos casos de testes são válidos
#--          Dos casos válidos:
#--                    4%  Com nenhum parâmetro.
#--                    24% Somente com valores de 1 a 12 no primeiro parâmetro.
#--                    24% Somente com valores de 13 a 9999 no primeiro parâmetro.
#--                    24% Com valores de 1 a 12 no primeiro parâmetro e de 0 a 9999 no
#--                              segundo parâmetro.
#--                    24% 13 a 9999 no primeiro parâmetro e 1 a 12 no segundo parâmetro.
#--
#--          10% dos Casos de testes não válidos
#--          Dos casos de testes não válidos:
```

```
#--                    24% Somente com um valor que não seja de 1 9999.
#--                    24% Com um valor válido de mês no primeiro parâmetro e um inválido numérico de
#--                          ano, isso quer dizer que não pode ser de 0 a 9999.
#--                    24% Com um valor válido de ano no primeiro parâmetro, isso quer dizer que tem que
#--                          ser de 13 a 9999, e um parâmetro inválido de mês.
#--                    28 % Com os dois parâmetros inválidos, sendo o primeiro correspondente ao mês (isso
#--                          quer dizer que não pode ser de 1 a 12, mas também não pode estar no
#--                          conjunto de parâmetros válidos do ano
#--
#-------------------------------------------------------------------------------------------------


if ( $EXPERIMENTHOME == "") then
  echo "Variável EXPERIMENTHOME não setada"
  exit
endif

set tc1 = $2
set SCRIPTDIR = $EXPERIMENTHOME/scripts
@ N = $3
@ i = 1

while ( $i <= $N  )

        @ x = `$SCRIPTDIR/client_int $1 0 100`
        #echo "Valido/invalido: $x"

                #-- Variáveis que recerão os valores válidos e serão usadas no
                #-- primeiro parâmetro.

        @ m1 = `$SCRIPTDIR/client_int $1 1 13`
        @ y1 = `$SCRIPTDIR/client_int $1 13 10000`

                #-- Variáveis que recerão os valores válidos e serão usadas no
                #-- segundo parâmetro.

        @ y2 = `$SCRIPTDIR/client_int $1 1  10000`


        if ( $x < 90 ) then
        # caso de teste valido

                @ tipo = `$SCRIPTDIR/client_int $1 0 100`
                # echo "Tipo: $tipo"
                if ( $tipo < 4 ) then
```

```
                                        #-- sem parâmetros
                echo " " > $tc1$i
        endif
        if ( $tipo >= 4 && $tipo < 28 ) then
                                        #-- com valores de 1 a 12 no primeiro parâmetro

                echo "$m1" > $tc1$i

        endif

        if (  $tipo >= 28 && $tipo < 52 ) then
                                #-- com valores de 13 a 9999 no primeiro parâmetro
                echo "$y1" > $tc1$i

        endif
        if (  $tipo >= 52 && $tipo < 76 ) then
                                #-- com valores de 1 a 12 no primeiro parâmetro e de
                                #-- 0 a 9999 no segundo

                echo "$m1 $y2" > $tc1$i

        endif
        if ( $tipo >= 76 && $tipo < 100 ) then

        #-- com valores de 13 a 9999 no primeiro parâmetro e de 1 a 12 no segundo parâmetro

                echo "$y2 $m1" > $tc1$i

        endif
else
# caso de teste não valido

        @ tipo = `$SCRIPTDIR/client_int $1 0 100`
        # echo "Tipo: $tipo"
        @ flag1 = `$SCRIPTDIR/client_int $1 0 2`
        if (  $tipo >= 0 && $tipo < 24 ) then
                                        #-- Somente com um valor que não seja de 1 9999.

                @ decide = `$SCRIPTDIR/client_int $1 0 2`
                if ($decide == 0) then
                        @ aux = `$SCRIPTDIR/client_int $1 10000 11001`
                else
                        @ aux = `$SCRIPTDIR/client_int $1 0 501`
                        if ( $aux < 0 ) @ aux = - $aux
                endif
```

```
		echo "$aux" > $tc1$i
endif
if (  $tipo >= 25 && $tipo < 49 ) then
		#-- primeiro parâmetro mês válido e segundo parâmetro ano invalido

		@ decide = `$SCRIPTDIR/client_int $1 0 2`

		if ($decide == 0) then
			@ aux = `$SCRIPTDIR/client_int $1 10000 11001`
		else
			@ aux = `$SCRIPTDIR/client_int $1 0 501`
			if ( $aux < 0 ) @ aux = - $aux
		endif
		echo "$m1 $aux" > $tc1$i
endif
if (  $tipo >= 49 && $tipo < 73 ) then
			#-- primeiro parâmetro ano valido e segundo mês inváido
		@ decide = `$SCRIPTDIR/client_int $1 0 2`

		if ($decide == 0) then
			@ aux = `$SCRIPTDIR/client_int $1 13 101`
		else
			@ aux = `$SCRIPTDIR/client_int $1 0 101`
			if ( $aux < 0 ) @ aux = - $aux
		endif
		echo "$y1 $aux" > $tc1$i
endif
if ( $tipo >= 73) then
			#-- Com os dois parâmetros inválidos, sendo o primeiro
			#-- correspondente ao mês e o segundo ao ano
		@ decide = `$SCRIPTDIR/client_int $1 0 2`

		if ($decide == 0) then
			@ aux1 = `$SCRIPTDIR/client_int $1 10000 11001`

		else
			@ aux1 = `$SCRIPTDIR/client_int $1 0 501`
			if ( $aux1 < 0 ) @ aux1 = - $aux1

		endif
		@ decide = `$SCRIPTDIR/client_int $1 0 2`
		if ($decide == 0) then
			@ aux2 = `$SCRIPTDIR/client_int $1 10000 11001`
		else
			@ aux2 = `$SCRIPTDIR/client_int $1 0 501`
```

```
                                        if ( $au2x < 0 ) @ aux2 = - $aux2

                                    endif

                                    @ aux1 = 1

                                    @ aux2 = 1
                                    echo "$aux1 $aux2" > $tc1$i
                            endif
                    endif
                    @ i = $i + 1
end
```

Programa Cal.

```
#!/bin/tcsh
# $1 - número da porta do servidor de números aleatórios
# $2 nome do arquivo a gravar
#
#-------------------------------------------------------------
#-- Perfil operacional: Cal
#--          95% dos casos de teste são não validos
#--          Dos casos válidos:
#--                    1% não possue parâmetros
#--                    49% possuem apenas um parâmetro
#--                    50% possuem dois parâmetros
#--
#--          Dos casos não válidos:
#--                    25% com um único parâmetro (ano) não valido
#--                    25% com dois parâmetros, mês não válido
#--                    25% com dois parâmetros, ano não válido
#--                    25% com dois parâmetros, ambos não validos
#--
#-------------------------------------------------------------

if ( $EXPERIMENTHOME == "") then
  echo "Variável EXPERIMENTHOME não setada"
  return
endif

set tc1 = $2
set SCRIPTDIR = $EXPERIMENTHOME/scripts
@ N = $3
@ i = 1

while ( $i <= $N  )
          @ x = `$SCRIPTDIR/client_int $1 0 100`
          #echo "Valido/invalido: $x"
          if ( $x < 95 ) then
          # caso de teste valido
            @ tipo = `$SCRIPTDIR/client_int $1 0 100`
            # echo "Tipo: $tipo"
            if ( $tipo == 99 ) then
              echo "  " > $tc1$i
            endif
            if ( $tipo >= 50 && $tipo < 99) then
```

```
    @ ano = `$SCRIPTDIR/client_int $1 1 10000`
    echo "$ano" > $tc1$i
  endif
  if ( $tipo < 50) then
    @ mês = `$SCRIPTDIR/client_int $1 1 13`
    @ ano = `$SCRIPTDIR/client_int $1 1 10000`
    echo "$mês $ano" > $tc1$i
  endif
else
# caso não valido
  @ tipo = `$SCRIPTDIR/client_int $1 0 4`
  #echo "Tipo: $tipo"
  @ flag1 = `$SCRIPTDIR/client_int $1 0 2`
  @ flag2 = `$SCRIPTDIR/client_int $1 0 2`

# tipo 0 : só ano invalido
  if ( $tipo == 0 ) then
    @ ano = 1
    while ( $ano >= 1 && $ano < 10000 )
      @ ano = `$SCRIPTDIR/client_int $1 0`
            if ( $flag1 ) @ ano = - $ano
    end
    echo "$ano" > $tc1$i
  endif
# tipo 1: mês e ano, mês inválido
  if ( $tipo == 1 ) then
    @ mês = 1
    while ( $mês >= 1 && $mês <= 12)
      @ mês = `$SCRIPTDIR/client_int $1 0`
            if ( $flag1 ) @ mês = - $mês
    end
    @ ano = `$SCRIPTDIR/client_int $1 1 10000`
    echo "$mês $ano" > $tc1$i
 endif
# tipo 2: mês, ano, ano invalido
 if ( $tipo == 2 ) then
    @ ano = 1
    while ( $ano >= 1 && $ano < 10000)
      @ ano = `$SCRIPTDIR/client_int $1 0`
            if ( $flag1 ) @ ano = - $ano
    end
    @ mês = `$SCRIPTDIR/client_int $1 1 13`
    echo "$mês $ano" > $tc1$i
  endif
# tipo 3: mês e ano, ambis invalidos
```

```
            if ( $tipo == 3 ) then
               @ ano = 1
               while ( $ano >= 1 && $ano < 10000)
                  @ ano = `$SCRIPTDIR/client_int $1 0`
                        if ( $flag1 ) @ ano = - $ano
               end
               @ mês = 1
               while ( $mês >= 1 && $mês <= 12)
                  @ mês = `$SCRIPTDIR/client_int $1 0`
                        if ( $flag2 ) @ mês = - $mês
               end
               echo "$mês $ano" > $tc1$i
            endif
         endif
         @ i = $i + 1
end
```

Programa Jday-Jdate.

```
#!/bin/tcsh
# $1 - número da porta do servidor de números aleatórios
# $2 nome do arquivo a gravar
# $3 - número de casos de teste a gerar
#
#------------------------------------------------------------
#-- Entradas válidas:
#-- 1o. parâmetro: de 1 até 12 .
#-- 2o. parâmetro: de 1 até 28 (fevereiro) ou 1 até 29 (fevereiro em ano bissexto) ou 1 a 30 para os meses
#--(abril, junho,  setembro, novembro) ou de 1 a 31 ( janeiro, março, maio, julho, agosto, outubro, dezembro).
#-- 3o. parâmetro: de 0 até 522485619.
#--
#--
#-- Perfil operacional: jday-jdate
#--          95% dos casos de teste são validos
#--          Dos casos válidos:
#--                    50% terão o formato mês dia e ano mm dd aa
#--                    50% terão um número do calendário juliano
#--
#--          Dos casos não válidos:
#--          01% sem parâmetro
#--          19% com um parâmetros
#--                    19% com dois parâmetros
#--                    19% com três parâmetros dois válidos e um inválido
#--                    19% com três parâmetros um válido e dois invalidos
#--                    19% com os três parâmetro inválidos
#--                    04% com o numero juliano inválido
#--
#--
#-------------------------------------------------------------------------------------------------------------------

if ( $EXPERIMENTHOME == "") then
  echo "Variável EXPERIMENTHOME não setada"
  exit
endif

set tc1 = $2
set SCRIPTDIR = $EXPERIMENTHOME/scripts

@ N = $3
@ i = 1
```

```
echo "passei por aqui ----  1"
while ( $i <= $N  )

    @ x = `$SCRIPTDIR/client_int $1 0 100`
    #echo "Valido/invalido: $x"
    @ mês = `$SCRIPTDIR/client_int $1 1 13`
    @ ano = `$SCRIPTDIR/client_int $1 0 522485620`
    if ($mês == 2 ) then

      @ aux = $ano
      @ chave = 1
      while ( $chave == 1)
         @ aux1 = $aux
                @ aux = ($aux / 4)
                if( $aux1 == ($aux * 4)) then

                  @ controle = 1
                  @ chave = 0

                else
                  @ controle = 0
                  @ chave = 0
                endif
              end

              if ($controle == 1) then
                @ dia = `$SCRIPTDIR/client_int $1 1 30`
              else
                @ dia = `$SCRIPTDIR/client_int $1 1 29`

              endif

    else
       if (($mês == 4) || ($mês == 6) || ($mês == 9) || ($mês == 11) ) then
         @ dia = `$SCRIPTDIR/client_int $1 1 31`
              else
                @ dia = `$SCRIPTDIR/client_int $1 1 32`
              endif
    endif

    @ juliano = `$SCRIPTDIR/client_int $1 -1000000 1000000`


    if ( $x < 95 ) then
             # caso de teste valido
```

```
        @ tipo2 = `$SCRIPTDIR/client_int $1 0 100`

        if ($tipo2 < 50) then
          echo "$mês $dia $ano" > $tc1$i
        else
          echo "$juliano" > $tc1$i
        endif

else
        @ mêsInv = `$SCRIPTDIR/client_int $1 -1000 1001`

        while ( $mêsInv > 0 && $mêsInv < 13)
           @ mêsInv = `$SCRIPTDIR/client_int $1 -1000 1001`
        end

        @ diaInv = `$SCRIPTDIR/client_int $1 -1000 1001`

        while ( $diaInv > 0 && $diaInv < 32)
           @ diaInv = `$SCRIPTDIR/client_int $1 -1000 1001`
        end

        @ anoInv = `$SCRIPTDIR/client_int $1 -999999999 1000000000`

        while ( $anoInv > -1 && $anoInv < 522485620)
           @ anoInv = `$SCRIPTDIR/client_int $1 -999999999 1000000000`
        end

        @ decisao1 = `$SCRIPTDIR/client_int $1 0 2`
        @ decisao2 = `$SCRIPTDIR/client_int $1 0 2`
        @ decisao3 = `$SCRIPTDIR/client_int $1 0 2`

        @ flag1 = `$SCRIPTDIR/client_int $1 0 4`
        @ flag2 = `$SCRIPTDIR/client_int $1 0 3`
        @ flag3 = `$SCRIPTDIR/client_int $1 0 3`


        @ tipo = `$SCRIPTDIR/client_int $1 0 100`
        if ($tipo == 0) then

          echo " " > $tc1$i

        endif

        if ($tipo > 0 && $tipo < 20) then
```

```
  if ($decisao2 == 0) then

    @ parâmetro2 = $mês
  else
    @ parâmetro2 = $mêsInv

  endif
  echo "$parâmetro2" > $tc1$i

endif

if ($tipo > 19 && $tipo < 39) then

  if ($flag1 == 0) then
            echo "$mês $dia" > $tc1$i
  else
            if ($flag1 == 1) then
              echo "$mêsInv $diaInv" > $tc1$i
            else
              if ($flag1 == 2) then
                        echo "$mês $diaInv " > $tc1$i
              else
                        echo "$mêsInv $dia " > $tc1$i
              endif

            endif
  endif

endif
if ($tipo > 38 && $tipo < 58) then

  if ($flag2 == 0) then
            echo "$mêsInv $dia $ano" > $tc1$i
  else
            if ($flag2 == 1) then
              echo "$mês $diaInv $ano" > $tc1$i
            else
              echo "$mês $dia $anoInv" > $tc1$i

            endif
  endif
endif

if ($tipo > 57 && $tipo < 77) then
  if ($flag3 == 0) then
```

```
                echo "$mês $diaInv $anoInv" > $tc1$i
            else
                if ($flag3 == 1) then
                  echo "$mêsInv $dia $anoInv" > $tc1$i
                else
                  echo "$mêsInv $diaInv $ano" > $tc1$i

                endif
            endif

          endif

          if ($tipo > 76 && $tipo < 96 ) then

            echo "$mêsInv $diaInv $anoInv" > $tc1$i

          endif


          if ($tipo > 95) then
            @ julianoInv = `$SCRIPTDIR/client_int $1 -1000000 1000000`

            while ( $julianoInv > -1000001 && $julianoInv < 1000001)
              @ julianoInv = `$SCRIPTDIR/client_int $1 -9999999 10000000`
            end


            echo "$julianoInv" > $tc1$i

          endif


    endif
    @ i = $i + 1
end
```

Programa Dateplus.

```
#!/bin/tcsh
# $1 - número da porta do servidor de números
aleatórios
# $2 nome do arquivo a gravar
#
#-------------------------------------------------------
#-- Perfil operacional: Dateplus
#--
#--          95% dos casos de teste são não validos
#--          Dos casos válidos:
#--                    15% são no formato mmddaa operação+valor unidade
#--                    15% são no formato mmddaa mais 2 vezes (operação+valor unidade)
#--                    15% são no formato mmddaa mais 3 vezes (operação+valor unidade)
#--                    15% são no formato mmddaa mais 4 vezes (operação+valor unidade)
#--                    15% são no formato mmddaa mais 5 vezes (operação+valor unidade)
#--                    15% são no formato mmddaa mais 6 vezes (operação+valor unidade)
#--                    10% são no formato mmddaa mais 7 vezes (operação+valor unidade)
#--
#--
#--                    Obs: opeação - pode ser a subtração (-) ou a soma (+)
#--                        valor - é a quantidade que será incrementada ou decrementada da data
#--                        unidade - é o tipo de unidade utilizada para especificar o valor (sec, min hour, day, #--                    we
#--
#--          Dos casos não válidos:
#--              50% com a data válida, mas com pelo menos uma opção inválida (opração+valor unidade),
#--              podendo ser da seguinte maneira:
#--                    05% sem a data
#--                    15% somente com a data
#--                    20% com a data, operação , valor e a unidade mas com essa opção inválida
#--                    10% com a data mais 2 vezes (operação+valor unidade) mas com uma desses opções #--                    inváli
#--                    10% com a data mais 3 vezes (operação+valor unidade) mas com uma desses opções #--                    inváli
#--                    10% com a data mais 4 vezes (operação+valor unidade) mas com uma desses opções #--                    inváli
#--                    10% com a data mais 5 vezes (operação+valor unidade) mas com uma desses opções #--                    inváli
#--                    10% com a data mais 6 vezes (operação+valor unidade) mas com uma desses opções #--                    inváli
#--                    10% com a data mais 7 vezes (operação+valor unidade) mas com uma desses opções #--                    inváli
#--
#--
#--              50% com a data inválida, podendo ter até todas as opções válidas; sendo da seguinte maneira.
#--                    05% sem a data
#--                    15% somente com a data
#--                    20% com a data, operação , valor e a unidade
```

```
#--                     10% com a data mais 2 vezes (operação+valor unidade)
#--                     10% com a data mais 3 vezes (operação+valor unidade)
#--                     10% com a data mais 4 vezes (operação+valor unidade)
#--                     10% com a data mais 5 vezes (operação+valor unidade)
#--                     10% com a data mais 6 vezes (operação+valor unidade)
#--                     10% com a data mais 7 vezes (operação+valor unidade)
#--
#--
#--
#-- OBs: valoeres máximos :sec = 999974899, min = 33549097 , hour = 559157, day = 23298, week = 3329,
#-- mon = 15,  year = 2012963340
#--
#-------------------------------------------------------------------------------------------------------------------

if ( $EXPERIMENTHOME == "") then
  echo "Variável EXPERIMENTHOME não setada"
  exit
endif

set tc1 = $2
set SCRIPTDIR = $EXPERIMENTHOME/scripts

@ N = $3
@ i = 1

while ( $i <= $N  )
    @ x = `$SCRIPTDIR/client_int $1 0 100`
    #echo "Valido/invalido: $x"

    @ mês = `$SCRIPTDIR/client_int $1 1 13`
    if ($mês < 10) then
            set mês = "0$mês"
    endif

    @ ano = `$SCRIPTDIR/client_int $1 0 100`
    if ($ano < 10) then
            set ano = "0$ano"
    endif

    if ($mês == 02 ) then

      @ aux = $ano
      @ chave = 1
      while ( $chave == 1)
         @ aux1 = $aux
```

```
            @ aux = ($aux / 4)
            if( $aux1 == ($aux * 4)) then

              @ controle = 1
              @ chave = 0

            else
              @ controle = 0
              @ chave = 0
            endif
          end

          if ($controle == 1) then
            @ dia = `$SCRIPTDIR/client_int $1 1 30`
          else
            @ dia = `$SCRIPTDIR/client_int $1 1 29`

          endif

else
   if (($mês == 04) || ($mês == 06) || ($mês == 09) || ($mês == 11) ) then
     @ dia = `$SCRIPTDIR/client_int $1 1 31`
          else
            @ dia = `$SCRIPTDIR/client_int $1 1 32`
          endif
endif
if ($dia < 10) then

          set dia = "0$dia"
endif

@ contador = 1

while ( $contador < 8 )

          @ aux1 = `$SCRIPTDIR/client_int $1 1 8`
          @ chave1 = `$SCRIPTDIR/client_int $1 0 2`
          @ chave2 = `$SCRIPTDIR/client_int $1 0 2`

          if ($chave1 == 0) then
            if ($chave2 == 0) then
              set sinal = "+"
            else
              set sinal = "+."
            endif
```

```
else
  if ($chave2 == 0) then
    set sinal = "-"
  else
    set sinal = "-."
  endif
endif

if ($aux1 == 1) then
  @ valAux = `$SCRIPTDIR/client_int $1 1 999974899`
  set nome = "sec"
  set paramAux = "$sinal$valAux $nome"
else
  if ($aux1 == 2) then
            @ valAux = `$SCRIPTDIR/client_int $1 1 33549097`
            set nome = "min"
            set paramAux = "$sinal$valAux $nome"
  else
            if ($aux1 == 3) then
              @ valAux = `$SCRIPTDIR/client_int $1 1 559157`
              set nome = "hour"
              set paramAux = "$sinal$valAux $nome"
            else
              if ($aux1 == 4) then
                        @ valAux = `$SCRIPTDIR/client_int $1 1 23298`
                        set nome = "day"
                        set paramAux = "$sinal$valAux $nome"
              else
                        if ($aux1 == 5) then
                          @ valAux = `$SCRIPTDIR/client_int $1 1 3329`
                          set nome = "week"
                          set paramAux = "$sinal$valAux $nome"
                        else
                          if ($aux1 == 6) then
                                    @ valAux = `$SCRIPTDIR/client_int $1 1 15`
                                    set nome = "mon"
                                    set paramAux = "$sinal$valAux $nome"
                          else
                                    @ valAux = `$SCRIPTDIR/client_int $1 1 2012963340`
                                    set nome = "year"
                                    set paramAux = "$sinal$valAux $nome"
                          endif
                        endif
              endif
            endif
```

```
            endif
          endif

          if ($contador == 1) then
            set parâmetro1 = "$paramAux"
          else
            if ($contador == 2) then
                    set parâmetro2 = "$paramAux"
            else
                    if ($contador == 3) then
                      set parâmetro3 = "$paramAux"
                    else
                      if ($contador == 4) then
                              set parâmetro4 = "$paramAux"
                      else
                              if ($contador == 5) then
                                set parâmetro5 = "$paramAux"
                              else
                                if ($contador == 6) then
                                        set parâmetro6 = "$paramAux"
                                else
                                        set parâmetro7 = "$paramAux"
                                endif
                              endif
                      endif
                    endif
            endif
          endif



          @ contador = $contador + 1
end
if ( $x < 95 ) then
          # caso de teste valido

    @ tipo = `$SCRIPTDIR/client_int $1 0 100`

          if ( $tipo < 15 ) then

            echo "$mês$dia$ano $parâmetro1" > $tc1$i

          endif
    if ( $tipo > 14 && $tipo < 30) then
```

```
            echo "$mês$dia$ano $parâmetro1 $parâmetro2" > $tc1$i

          endif

  if ( $tipo > 29 && $tipo < 45) then

            echo "$mês$dia$ano $parâmetro1 $parâmetro2 $parâmetro3" > $tc1$i

          endif

  if ( $tipo > 44 && $tipo < 60) then

            echo "$mês$dia$ano $parâmetro1 $parâmetro2 $parâmetro3 $parâmetro4" > $tc1$i

          endif


  if ( $tipo > 59  && $tipo < 75 ) then

            echo "$mês$dia$ano $parâmetro1 $parâmetro2 $parâmetro3 $parâmetro4 $parâmetro5" > $tc1$i

          endif

  if ( $tipo > 74 && $tipo < 90) then

              echo "$mês$dia$ano $parâmetro1 $parâmetro2 $parâmetro3 $parâmetro4 $parâmetro5
              $parâmetro6" > $tc1$i

          endif


  if ( $tipo > 89 ) then

          echo "$mês$dia$ano $parâmetro1 $parâmetro2 $parâmetro3 $parâmetro4 $parâmetro5 $parâmetro6
          $parâmetro7" > $tc1$i

          endif
else

          @ controlData1 = `$SCRIPTDIR/client_int $1 0 2`
          @ controlData2 = `$SCRIPTDIR/client_int $1 0 2`
          @ controlData3 = `$SCRIPTDIR/client_int $1 0 2`

          if ($controlData1 == 1) then
            @ diaInv = `$SCRIPTDIR/client_int $1 32 100`
```

```
else
  set diaInv = "$dia"


endif

if ($controlData2 == 1) then
  @ mêsInv = `$SCRIPTDIR/client_int $1 13 100`

else
  set mêsInv = "$mês"

endif

if ($controlData3 == 1) then
  @ anoInv = `$SCRIPTDIR/client_int $1 100 10000`

  if ($anoInv < 1000) then
            set anoInv = "0$anoInv"

  endif
else
  set anoInv = "$ano"


endif

@ controlParamInv = 9

@ contInv = 1

while ( $contInv < 8 )

  @ auxInv1 = `$SCRIPTDIR/client_int $1 1 8`
  @ chaveInv1 = `$SCRIPTDIR/client_int $1 0 2`
  @ chaveInv2 = `$SCRIPTDIR/client_int $1 0 2`
  @ chaveInv3 = `$SCRIPTDIR/client_int $1 0 2`

  @ controlSinalInv = `$SCRIPTDIR/client_int $1 0 2`
  @ controlValorInv = `$SCRIPTDIR/client_int $1 0 2`
  @ controlOpcaoInv = `$SCRIPTDIR/client_int $1 0 2`


  if ($chaveInv1 == 0) then
    if ($chaveInv2 == 0) then
```

```
            if ($chaveInv3 == 0) then
        set sinalInv = "+"
            else
              set sinalInv = "*"
              @ controlParamInv = $contInv
            endif
  else
            if ($chaveInv3 == 0) then
        set sinalInv = "+."
            else
              set sinalInv = "*."
              @ controlParamInv = $contInv
            endif
  endif
else
  if ($chaveInv2 == 0) then
          if ($chaveInv3 == 0) then
      set sinalInv = "-"
            else
              set sinalInv = "/"
              @ controlParamInv = $contInv
            endif
  else
            if ($chaveInv3 == 0) then
      set sinalInv = "-."
            else
              set sinalInv = "/."
              @ controlParamInv = $contInv
    endif
   endif
endif

@ valorInv = `$SCRIPTDIR/client_int $1 0 2`
@ opcaoInv = `$SCRIPTDIR/client_int $1 0 2`
if ($auxInv1 == 1) then
  if ($valorInv == 0) then

    @ valAux = `$SCRIPTDIR/client_int $1 999974900 2000000000 `
            @ controlParamInv = $contInv
  else
            @ valAux = `$SCRIPTDIR/client_int $1 1 999974899`
  endif
  if ($opcaoInv == 0) then
    set nomeInv = "sec"
  else
```

```
    set nomeInv = "set"
        @ controlParamInv = $contInv
  endif
  set paramAux = "$sinalInv$valAux $nomeInv"

else
  if ($auxInv1 == 2) then
      if ($valorInv == 0) then
        @ valAux = `$SCRIPTDIR/client_int $1 1 33549097`
      else
        @ valAux = `$SCRIPTDIR/client_int $1 33549097 100000000`
        @ controlParamInv = $contInv
      endif
      if ($opcaoInv == 0) then
        set nomeInv = "min"
      else
        set nomeInv = "mit"
        @ controlParamInv = $contInv
      endif
      set paramAux = "$sinalInv$valAux $nomeInv"
  else
        if ($auxInv1 == 3) then
          if ($valorInv == 0) then
            @ valAux = `$SCRIPTDIR/client_int $1 1 559157`
          else
              @ valAux = `$SCRIPTDIR/client_int $1 559157 1000000`
              @ controlParamInv = $contInv
          endif
          if ($opcaoInv == 0) then
            set nomeInv = "hour"
          else
              set nomeInv = "hout"
              @ controlParamInv = $contInv
          endif
          set paramAux = "$sinalInv$valAux $nomeInv"
        else
          if ($auxInv1 == 4) then
              if ($valorInv == 0) then
                @ valAux = `$SCRIPTDIR/client_int $1 1 23298`
              else
                @ valAux = `$SCRIPTDIR/client_int $1 23298 50000`
                @ controlParamInv = $contInv
              endif
              if ($opcaoInv == 0) then
                set nomeInv = "day"
```

```
            else
              set nomeInv = "dat"
              @ controlParamInv = $contInv
            endif
            set paramAux = "$sinalInv$valAux $nomeInv"
else
            if ($auxInv1 == 5) then
              if ($valorInv == 0) then
                @ valAux = `$SCRIPTDIR/client_int $1 13329`
              else
                        @ valAux = `$SCRIPTDIR/client_int $1 3329 5000`
                        @ controlParamInv = $contInv
              endif
              if ($opcaoInv == 0) then
                set nomeInv = "week"
              else
                        set nomeInv = "weet"
                        @ controlParamInv = $contInv
              endif
              set paramAux = "$sinalInv$valAux $nomeInv"
            else
              if ($auxInv1 == 0) then
                        if ($valorInv == 0) then
                          @ valAux = `$SCRIPTDIR/client_int $1 1 15`
                        else
                          @ valAux = `$SCRIPTDIR/client_int $1 15 30`
                          @ controlParamInv = $contInv
                        endif
                        if ($opcaoInv == 0) then
                          set nomeInv = "mon"
                        else
                          set nomeInv = "mot"
                          @ controlParamInv = $contInv
                        endif
                        set paramAux = "$sinalInv$valAux $nomeInv"
              else
                        if ($valorInv == 0) then
                          @ valAux = `$SCRIPTDIR/client_int $1 1 2012963340`
                        else
                                  @ valAux = `$SCRIPTDIR/client_int $1 012963340
                                  2500000000`
                          @ controlParamInv = $contInv
                        endif
                        if ($opcaoInv == 0) then
                          set nomeInv = "year"
```

```
                                    else
                                      set nomeInv = "yeat"
                                      @ controlParamInv = $contInv
                                    endif
                                    set paramAux = "$sinalInv$valAux $nomeInv"
                                  endif
                                endif
                              endif
                            endif
                          endif
                        endif
                        if ($contInv == 1) then
                          set parâmetroInv1 = "$paramAux"
                        else
                          if ($contInv == 2) then
                                    set parâmetroInv2 = "$paramAux"
                          else
                                    if ($contInv == 3) then
                                      set parâmetroInv3 = "$paramAux"
                                    else
                                      if ($contInv == 4) then
                                                set parâmetroInv4 = "$paramAux"
                                      else
                                                if ($contInv == 5) then
                                                  set parâmetroInv5 = "$paramAux"
                                                else
                                                  if ($contInv == 6) then
                                                            set parâmetroInv6 = "$paramAux"
                                                  else
                                                            set parâmetroInv7 = "$paramAux"
                                                  endif
                                                endif
                                      endif
                                    endif
                          endif
                        endif
                        @ contInv = $contInv + 1
                      end

                      @ tipo4 = `$SCRIPTDIR/client_int $1 0 100`
                      @ tipo5 = `$SCRIPTDIR/client_int $1 0 100`

                      if ( ($controlData1 == 0 && $controlData2 == 0 && $controlData3 == 0) && $tipo4 < 50) then

                        if ( $tipo5 < 5) then
```

```
        echo " " > $tc1$i
endif

if ( $tipo5 > 4 && $tipo5 < 20) then

        echo "$mês$dia$ano" > $tc1$i
endif

if ( $tipo5 > 19 && $tipo5 < 40) then

        echo "$mês$dia$ano $parâmetroInv1" > $tc1$i
endif


if ( $tipo5 > 39 && $tipo5 < 50) then

        echo "$mês$dia$ano $parâmetroInv1 $parâmetroInv2" > $tc1$i
endif

if ( $tipo5 > 49 && $tipo5 < 60) then

        echo "$mês$dia$ano $parâmetroInv1 $parâmetroInv2 $parâmetroInv3" > $tc1$i
endif

if ( $tipo5 > 59 && $tipo5 < 70) then

        echo "$mês$dia$ano $parâmetroInv1 $parâmetroInv2 $parâmetroInv3 $parâmetroInv4" >
        $tc1$i
endif

if ( $tipo5 > 69 && $tipo5 < 80) then

        echo "$mês$dia$ano $parâmetroInv1 $parâmetroInv2 $parâmetroInv3 $parâmetroInv4
        $parâmetroInv5" > $tc1$i

endif

if ( $tipo5 > 79 && $tipo5 < 90) then

        echo "$mês$dia$ano $parâmetroInv1 $parâmetroInv2 $parâmetroInv3 $parâmetroInv4
        $parâmetroInv5 $parâmetroInv6" > $tc1$i
endif

if ( $tipo5 > 89 && $tipo5 < 100) then
```

```
        echo "$mês$dia$ano $parâmetroInv1 $parâmetroInv2 $parâmetroInv3 $parâmetroInv4
        $parâmetroInv5 $parâmetroInv6 $parâmetroInv7" > $tc1$i
    endif


else

    if ( $tipo5 < 5) then

            echo " " > $tc1$i
    endif

    if ( $tipo5 > 4 && $tipo5 < 20) then

            echo "$mêsInv$diaInv$anoInv" > $tc1$i
    endif

    if ( $tipo5 > 19 && $tipo5 < 40) then

            echo "$mêsInv$diaInv$anoInv $parâmetroInv1" > $tc1$i

    endif


    if ( $tipo5 > 39 && $tipo5 < 50) then

            echo "$mêsInv$diaInv$anoInv $parâmetroInv1 $parâmetroInv2" > $tc1$i
    endif

    if ( $tipo5 > 49 && $tipo5 < 60) then

        echo "$mêsInv$diaInv$anoInv $parâmetroInv1 $parâmetroInv2 $parâmetroInv3" > $tc1$i
    endif

    if ( $tipo5 > 59 && $tipo5 < 70) then

            echo "$mêsInv$diaInv$anoInv $parâmetroInv1 $parâmetroInv2 $parâmetroInv3
            $parâmetroInv4" > $tc1$i

    endif

    if ( $tipo5 > 69 && $tipo5 < 80) then
```

```
                    echo "$mêsInv$diaInv$anoInv $parâmetroInv1 $parâmetroInv2 $parâmetroInv3
                    $parâmetroInv4 $parâmetroInv5" > $tc1$i
                endif

                if ( $tipo5 > 79 && $tipo5 < 90) then

                    echo "$mêsInv$diaInv$anoInv $parâmetroInv1 $parâmetroInv2 $parâmetroInv3
                    $parâmetroInv4 $parâmetroInv5 $parâmetroInv6" > $tc1$i
                endif

                if ( $tipo5 > 89 && $tipo5 < 100) then

                    echo "$mêsInv$diaInv$anoInv $parâmetroInv1 $parâmetroInv2 $parâmetroInv3
                    $parâmetroInv4 $parâmetroInv5 $parâmetroInv6 $parâmetroInv7" > $tc1$i

                endif

            endif
    endif
    @ i = $i + 1
end
```

APÊNDICE B

O conjunto de operadores de mutação de unidade é classificada em quatro classes: Mutação de Constantes, Mutação de Comandos,Mutação de Operadores e Mutação de Variáveis.

Para a realizaçao dos experimentos desse trabalho, foram selecionados 65 operadores de unidade.Cada conjunto de operadores de mutação revela diferentes tipos de erros em um mesmo tipo de estrutura.

Operador	**Descrição**	**Porcentagem (%)**
CCCR	Troca constantes por todas as constantes do programa	100
CCSR	Troca referências escalares por constantes	100
CRCR	Troca cada referência escalar por constantes requeridas dependendo do tipo da referência	100
OAAA	Troca um operador aritmético com atribuições por outro operador aritmético com atribuições	100
OAAN	Troca um operador aritmético sem atribuições por outro operador aritmético sem atribuições	100
OABA	Troca um operador aritmético com atribuições por operador bitwise com atribuições	100
OABN	Troca um operador aritmético sem atribuições por um operador bitwise sem atribuições	100
OAEA	Troca um operador aritmético com atribuições por um operador plano	100
OALN	Troca um operador aritmético sem atribuições por um operador lógico	100
OARN	Troca um operador aritmético sem atribuições por um operador relacional	100
OASA	Troca um operador aritmético com atribuições por um operador de deslocamento com atribuições	100
OASN	Troca um operador aritmético sem	100

	atribuições por um operador de deslocamento sem atribuições	
OBAA	Troca um operador bitwise com atribuições por um operador aritmético com atribuições	100
OBAN	Troca um operador bitwise sem atribuições por um operador aritmético sem atribuições	100
OBBA	Troca um operador bitwise com atribuições por outro operador bitwise com atribuições	100
OBBN	Troca um operador bitwise sem atribuições por um outro operador bitwise sem atribuições	100
OBEA	Troca um operador bitwise com atribuições por um operador plano	100
OBLN	Troca um operador bitwise sem atribuições por um operador lógico	100
OBNG	Reverte o contexto das expressões comparando bitwise	100
OBRN	Troca um operador bitwise sem atribuições por um operador relacional	100
OBSA	Troca um operador bitwise com atribuições por um operador swift com atribuições	100
OBSN	Troca um operador bitwise sem atribuições por um operador de deslocamento sem atribuições	100
OCNG	Modela erros em comando de seleção e repetição, revertendo essas condições. Utilizado quando não envolve nenhum operador lógico	100
OCOR	Troca por um operador de cast por todos os outros tipos primitivos da linguagem	100
OEAA	Troca um operador plano por um operador aritmético com atribuições	100
OEBA	Troca um operador plano por um operador bitwise com atribuições	100
OESA	Troca um operador plano por um operador de deslocamento com atribuições	100
Oido	Modela erros que surgem com o uso incorreto dos operadores ++ e - -	100
OIPM	Revela erros no uso incorreto de expressões que envolvam ++, -- e operadores de indireção *.	100

OLAN	Troca um operador lógico por um operador aritmético sem atribuições	100
OLBN	Troca um operador lógico por um operador bitwise sem atribuições	100
OLLN	Troca um operador lógico por outro operador lógico	100
OLNG	Modela erros em comandos de seleção e repetição, revertendo essas condições	100
OLRN	Troca um operador lógico por um operador relacional	100
OLSN	Troca um operador lógico por um operador de deslocamento sem atribuições	100
ORAN	Troca um operador relacional por um operador aritmético sem atribuições	100
ORBN	Troca um operador relacional por um operador bitwise sem atribuições	100
ORLN	Troca um operador relacional por um operador lógico	100
ORRN	Troca um operador relacional por outro operador relacional	100
ORSN	Troca um operador relacional por um operador de deslocamento sem atribuições	100
OSAA	Troca um operador de deslocamento com atribuições por um operador aritmético com atribuições	100
OSAN	Troca um operador de deslocamento sem atribuições por um operador aritmético sem atribuições	100
OSBA	Troca um operador de deslocamento com atribuições por um operador bitwise com atribuições	100
OSBN	Troca um operador de deslocamento sem atribuições por um operador bitwise sem atribuições	100
OSEA	Troca um operador de deslocamento com atribuições por um operador plano	100
OSLN	Troca um operador de deslocamento sem atribuições por um operador lógico	100
OSRN	Troca um operador de deslocamento sem atribuições por um operador relacional	100
OSSA	Troca um operador de deslocamento com atribuições por outro operador de deslocamento com atribuições	100
OSSN	Troca um operador de deslocamento	100

	sem atribuições por outro operador de deslocamento sem atribuições	
SBRC	Troca os comandos *break* por comandos *continue*	100
SCRB	Troca os comandos *continue* por comandos *break*	100
SGLR	Troca os rótulos dos comandos do tipo *goto* por todos os outros rótulos que aparecem na mesma função	100
SRSR	Troca cada comando de uma função por todos os comando *return* que existem na mesma função	100
STRI	É projetado para garantir que a condição de cada comando *if* seja avaliado pelo menos uma vez pelo ramo verdadeiro e pelo ramo falso	100
VDTR	Requer valores negativos, positivos e zero para cada referência escalar	100
VSCR	São responsáveis pela mutação de referência e componentes de uma estrutura que são substituídos por referências aos demais componentes da mesma estrutura, respeitando os tipos de componentes	100
VTWD	Troca referência escalar pelo seu valor sucessor e predecesssor	100

APÊNDICE C

Esse *script* foi usado na geração e importação dos primeiros 15.000 casos de teste de cada sessão.

```
#!/bin/tcsh
# $1 = nome do programa a utilizar
# $2 = numero da porta a utilizar para servidor de numeros aleatorios
#############################################################

set prog = $1
set port = $2
if ( $EXPERIMENTHOME == "") then
   echo "Variavel EXPERIMENTHOME não setada"
   exit
endif
set SCRIPTDIR = $EXPERIMENTHOME/scripts


cd $EXPERIMENTHOME/$prog
if (-d MATAMUTANTE) then
   echo "Diretorio MATAMUTANTE jah existe"
   exit
endif

if (-d MATAMUTANTEDUAL) then
   echo "Diretorio MATAMUTANTEDUAL jah existe"
   exit
endif


mkdir MATAMUTANTE
mkdir MATAMUTANTEDUAL
cp $prog* ./MATAMUTANTE
cp __$prog* ./MATAMUTANTE
cp $prog* ./MATAMUTANTEDUAL
cp __$prog* ./MATAMUTANTEDUAL
@ cont = 0
#instalar servidor
$SCRIPTDIR/server 129 $port &
```

```
while ( 1 )
   echo "Gerando casos de teste..."
   ./SGeraTestCase $port tc 500
   cp tc* MATAMUTANTE
   cp tc* MATAMUTANTEDUAL
   rm tc*
   cd MATAMUTANTE
   tcase -ascii -f 1 -t 500 -p tc -trace -E $prog
-EE ${prog}_inst $prog
   echo "Executando mutantes, aguarde..."
   exemuta -exec -v "" -trace $prog
   muta -ms $prog
   list-good -d $prog >/dev/null
   tcase -z $prog

   cd ../MATAMUTANTEDUAL
   tcase -ascii -f 1 -t 500 -p tc -trace -E $prog
-EE ${prog}_inst $prog
   echo "Executando mutantes duais, aguarde..."
   exemuta -exec -v "" -trace -dual $prog
   muta -ms $prog
   list-good -d $prog >/dev/null
   tcase -z $prog

   @ cont = $cont + 500

   cd ..
   if ( -e stop.stop ) exit
   echo "Executados $cont casos de teste"
   echo -n "Sua chance de interromper. Dez segundos
para abortar "
   sleep 1
   echo -n "0..."
   sleep 2
   echo -n "2..."
   sleep 2
   echo -n "4..."
   sleep 2
   echo -n "6..."
   sleep 2
   echo -n "8..."
   sleep 2
   echo -n "10..."
   echo "Continuando..."
end
```

Esse outro *script* foi usado na geração e importação das 60 sessões de teste de cada um dos 6 programas objetos, assim como os casos de teste que foram inclusos nas sessões.

```
#!/bin/tcsh
# $1 = nome do programa a utilizar
# $2 = numero da porta a utilizar para servidor de números aleatórios
#############################################################

set prog = $1
set port = $2
if ( $EXPERIMENTHOME == "") then
   echo "Variável EXPERIMENTHOME não setada"
   exit
endif
set SCRIPTDIR = $EXPERIMENTHOME/scripts



cd $EXPERIMENTHOME/$prog

@ cont = 0
@ cont_sessao = 1

set semente = (1 2 7 12 17 18 55 71 88 93 100 102 106 123 140 147 169 215 249
252 267 268 281 275 279 297 302 311 345 367)


while ( $cont_sessao < 31 )

   mkdir MATAMUTANTE_$cont_sessao
   mkdir MATAMUTANTEDUAL_$cont_sessao
   cp $prog* ./MATAMUTANTE_$cont_sessao
   cp __$prog* ./MATAMUTANTE_$cont_sessao
   cp $prog* ./MATAMUTANTEDUAL_$cont_sessao
   cp __$prog* ./MATAMUTANTEDUAL_$cont_sessao

   @ cont_tcase = 0

   #instalar servidor

   $SCRIPTDIR/server $semente[$cont_sessao] $port &
```

```
while ( $cont_tcase < 15000 )


     echo "Gerando casos de teste...   "
     ./SGeraTestCase $port tc 500
     cp tc* MATAMUTANTE_$cont_sessao
     cp tc* MATAMUTANTEDUAL_$cont_sessao
     rm tc*
     cd MATAMUTANTE_$cont_sessao
     tcase -ascii -f 1 -t 500 -p tc -trace -E $prog -EE ${prog}_inst $prog
     echo "Executando mutantes, aguarde..."
     exemuta -exec -v "" -trace $prog
     muta -ms $prog
     list-good -d $prog >/dev/null
     tcase -z $prog

     cd ../MATAMUTANTEDUAL
     tcase -ascii -f 1 -t 500 -p tc -trace -E $prog -EE ${prog}_inst $prog
     echo "Executando mutantes duais, aguarde..."
     exemuta -exec -v "" -trace -dual $prog
     muta -ms $prog
     list-good -d $prog >/dev/null
     tcase -z $prog

     @ cont_tcase = $cont_tcase + 500


     cd ..
     if ( -e stop.stop ) exit

     echo "Executados $cont_tcase casos de teste"
     echo -n "Sua chance de interromper. Dez segundos para abortar "
     sleep 1
     echo -n "0..."
     sleep 2
     echo -n "2..."
     sleep 2
     echo -n "4..."
     sleep 2
     echo -n "6..."
     sleep 2
     echo -n "8..."
     sleep 2
     echo -n "10..."
```

```
        echo "Continuando..."


    end

    echo "Finalizando sessão de teste numero $cont_sessao "

    killall -9 server

    @ cont_sessao = $cont_sessao + 1

end
```

APÊNDICE D

```
#!/bin/tcsh
# $1 = nome do programa a utilizar
############################################################

set prog = $1

if ( $EXPERIMENTHOME == "") then
   echo "Variavel EXPERIMENTHOME não setada"
   exit
endif
set SCRIPTDIR = $EXPERIMENTHOME/scripts


cd $EXPERIMENTHOME/$prog

@ cont = 0
@ cont_sessao = 1


while ( $cont_sessao < 31 )

   echo "Copiando Arquivos da Sessao de teste NORMAL $cont_sessao para serem
cruzados"

   cd MATAMUTANTE_$cont_sessao
   mkdir DUAL
   cp $prog* ./DUAL
   cp __$prog* ./DUAL
   cd DUAL

   echo "Zerando o escore e executanto os mutantes em modo DUAL"

   date

   muta -create $prog
   muta-gen -O ../../../scripts/list_operators $prog

   exemuta -exec -v "" -trace -dual $prog

   muta -ms $prog
   date
```

```
   cd ..
   cd ..

   echo "Copiando Arquivos da Sessao de teste DUAL $cont_sessao para serem
cruzados"

   cd MATAMUTANTEDUAL_$cont_sessao
   mkdir NORMAL
   cp $prog* ./NORMAL
   cp __$prog* ./NORMAL
   cd NORMAL

   echo "Zerando o escore e executanto os mutantes em modo NORMAL"

   date

   muta -create $prog
   muta-gen -O ../../../scripts/list_operators $prog

   exemuta -exec -v "" -trace $prog

   muta -ms $prog

   date

   cd ..
   cd ..

   echo "Finalizando sessão de teste NORMAL e DUAL numero $cont_sessao "

   @ cont_sessao = $cont_sessao + 1

end
```