

Introdução ao Teste de Software

Técnica de Teste Baseada em Erros

Simone do Rocio Senger de Souza
srocio@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação — ICMC/USP

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

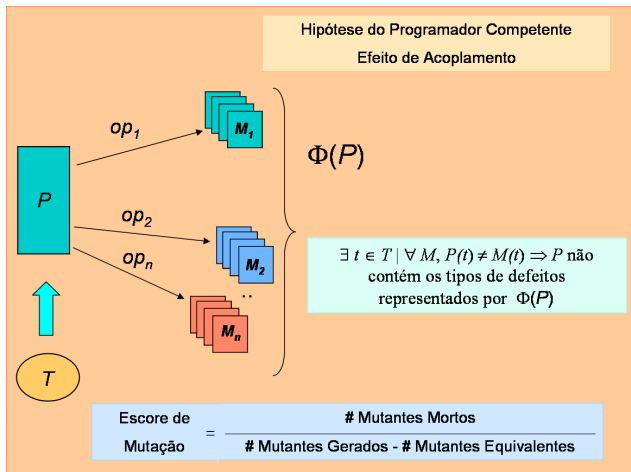
Resumo

Exercício

- Aulas Anteriores
- Técnica de Teste Baseada em Erros
- Análise de Mutantes
- Mutação de Interface
- Mutação em Especificação
- Ferramentas para o Teste de Mutação
- Exemplo: Identifier
- Resumo
- Exercício

- As técnicas de teste são definidas conforme o **tipo de informação** utilizada para realizar o teste.
- Contemplam diferentes perspectivas do software: **aspecto complementar!!!!**
 - **Técnica Funcional**
 - Os testes são baseados exclusivamente na **especificação de requisitos** do programa.
 - Nenhum conhecimento de como o programa está implementado é requerido.
 - **Técnica Estrutural**
 - Os testes são baseados na estrutura interna do programa, ou seja, na **implementação** do mesmo.

- Os requisitos de teste são derivados a partir dos **erros mais frequentes** cometidos durante o processo de desenvolvimento do software.
- Técnica **flexível**, podendo ser aplicada em diferentes contextos



Critério Análise de Mutantes

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

- Consiste na introdução de pequenos **desvios sintáticos** no programa em teste, os quais são responsáveis por modelar erros freqüentes de desenvolvimento.
- Encoraja o testador a construir casos de testes capazes de demonstrar que tais transformações resultam em programas **semanticamente incorretos**.
- Casos de teste que evidenciem as **diferenças de comportamento** entre o programa original (em teste) e os programas modificados.

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

Hipótese do Programador Competente

Programadores experientes escrevem programas corretos ou muito próximos do correto.

Efeito de Acoplamento

Casos de teste capazes de revelar erros simples são tão sensíveis que, implicitamente, também são capazes de revelar erros complexos.

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

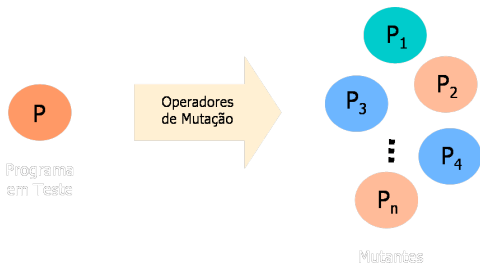
Resumo

Exercício

- 1 Geração dos Mutantes
- 2 Execução do Programa
- 3 Execução dos Mutantes
- 4 Análise dos Mutantes Vivos

- Geração dos Mutantes

Para modelar os desvios sintáticos mais comuns, **operadores de mutação** são aplicados a um programa, transformando-o em programas similares: **mutantes**.



Geração de Mutantes

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

Entende-se por **operador de mutação** as regras que definem as alterações a serem aplicadas ao programa P , dando origem a programas similares.

- Seleção dos operadores de mutação:
 - **Abrangente**
 - Capaz de modelar a maior parte dos erros.
 - **Pequena cardinalidade**
 - Problemas de custo: Quanto maior o número de operadores utilizados, maior o número de mutantes gerados.

Mutação de Comandos

SSDL Retira um comando de cada vez do programa

SWDD Troca o comando *while* por *do-while*

SMTC Interrompe a execução do laço após duas execuções

Mutação de Operadores

ORRN Troca operador relacional por operador relacional

OLBN Troca operador lógico por operador *bitwise*

OASN Troca operador aritmético por operador de deslocamento

Mutação de Constantes

Ccsr Troca referências escalares por constantes

Cccr Troca constante por constante

Mutação de Variáveis

VTWD Troca referência escalar pelo sucessor e predecessor

VDTR Requer valor negativo, positivo e zero para cada referência escalar

Programa modificado, resultante da aplicação dos operadores de mutação sobre o programa original.

- Assumindo a validade do Efeito de Acoplamento, apenas uma mutação de cada vez é aplicada ao programa em teste, ou seja, cada mutante contém apenas **uma transformação sintática**.
- Observa-se, entretanto, que **k transformações sintáticas** podem ser introduzidas no programa.
 - k -mutante.

- Troca do operador relacional `<` pelo operador relacional `<=` (**ORRN**).

```

{
    char  achar;
    int   length, valid_id;
    length = 0;
    printf ("Identificador: ");
    achar = fgetc (stdin);
    valid_id = valid_s(achar);
    if (valid_id)
        length = 1;
    achar = fgetc (stdin);
    while (achar != '\n')
    {
        if (!(valid_f(achar)))
            valid_id = 0;
        length++;
        achar = fgetc (stdin);
    }
    if (valid_id && (length >= 1) && (length <= 6))
        printf ("Valido\n");
    else
        printf ("Invalido\n");
}

```

Programa *Identifier* (função main)

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

- Execução do Programa
 - Execução do programa com os casos de teste.
- Execução dos Mutantes
 - Execução dos mutantes com os casos de teste.
 - Mutante morto
 - Mutante vivo

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

O resultado do mutante e o do programa original **diferem entre si** para algum caso de teste.

- Significa que o erro modelado pelo operador de mutação **não** está presente no programa.

- Considere um programa que calcula o fatorial de um número:

```

main () {
    int valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if (num >= 0) {
        while (num > 1) {
            fat = fat * num;
            num--; }
        printf("%d\n",fat); }
    else
        printf("Erro!\n"); }
    
```

Programa Fatorial

```

main () {
    int valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if (num <= 0) {
        while (num > 1) {
            fat = fat * num;
            num--; }
        printf("%d\n",fat); }
    else
        printf("Erro!\n"); }
    
```

Mutante Morto

- Considere um programa que calcula o fatorial de um número:

```

main () {
    int  valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if (num >= 0) {
        while (num > 1) {
            fat = fat * num;
            num--;
        }
        printf("%d\n",fat);
    }
    else
        printf("Erro!\n");
}

```

Programa Fatorial

```

main () {
    int  valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if (num <= 0) {
        while (num > 1) {
            fat = fat * num;
            num--;
        }
        printf("%d\n",fat);
    }
    else
        printf("Erro!\n");
}

```

Mutante Morto

- Suponha que a variável `num` assumo o valor **3**. Executando o programa original com esse valor, o resultado obtido é **6**.

- Considere um programa que calcula o fatorial de um número:

```
main () {
    int valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if (num >= 0) {
        while (num > 1) {
            fat = fat * num;
            num--;
        }
        printf("%d\n",fat);
    }
    else
        printf("Erro!\n");
}
```

Programa Fatorial

```
main () {
    int valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if {num <= 0} {
        while (num > 1) {
            fat = fat * num;
            num--;
        }
        printf("%d\n",fat);
    }
    else
        printf("Erro!\n");
}
```

Mutante Morto

- Suponha que a variável `num` assumo o valor **3**. Executando o programa original com esse valor, o resultado obtido é **6**.
- Por outro lado, executando o programa mutante, obtém-se uma mensagem de erro. Nesse caso, diz-se que o mutante foi morto pelo caso de teste **(3,6)**.

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Score de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

● Análise dos Mutantes Vivos

- Mutante equivalente
- Inclusão de novos casos de teste
- Escore de mutação
 - Medida de cobertura do teste de mutação!



Mutante Equivalente

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

O mutante e o programa original apresentam **sempre** o mesmo resultado, para qualquer caso de teste pertencente ao domínio de entrada.

- Considere um programa que calcula o fatorial de um número:

```

main () {
    int valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if (num >= 0) {
        while (num > 1) {
            fat = fat * num;
            num--; }
        printf("%d\n",fat); }
    else
        printf("Erro!\n"); }
    
```

Programa Fatorial

```

main () {
    int valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if (valor >= 0) {
        while (num > 1) {
            fat = fat * num;
            num--; }
        printf("%d\n",fat); }
    else
        printf("Erro!\n"); }
    
```

Mutante Equivalente

- Considere um programa que calcula o fatorial de um número:

```

main () {
    int valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if (num >= 0) {
        while (num > 1) {
            fat = fat * num;
            num--; }
        printf("%d\n",fat); }
    else
        printf("Erro!\n"); }
    
```

Programa Fatorial

```

main () {
    int valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if (valor >= 0) {
        while (num > 1) {
            fat = fat * num;
            num--; }
        printf("%d\n",fat); }
    else
        printf("Erro!\n"); }
    
```

Mutante Equivalente

- A troca do comando `if (num >= 0)` pelo comando `if (valor >= 0)` não altera os resultados produzidos pelo programa, que continua comportando-se conforme o esperado.

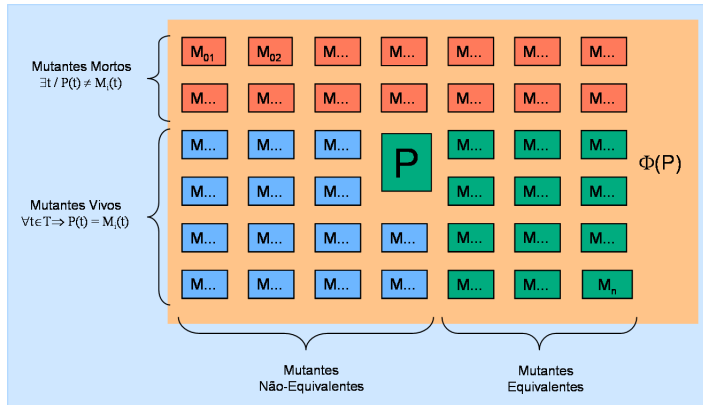
- Troca do operador lógico **&&** pelo operador aritmético ***** (**OLAN**).

```

{
    char  achar;
    int  length, valid_id;
    length = 0;
    printf ("Identificador: ");
    achar = fgetc (stdin);
    valid_id = valid_s(achar);
    if (valid_id)
        length = 1;
    achar = fgetc (stdin);
    while (achar != '\n')
    {
        if (!(valid_f(achar)))
            valid_id = 0;
        length++;
        achar = fgetc (stdin);
    }
    if (valid_id * (length >= 1) && (length < 6))
        printf ("Valido\n");
    else
        printf ("Invalido\n");
}

```

Programa *Identifier* (função main)



Mutantes de P

- Medida objetiva a respeito do **nível de confiança** da **adequação dos casos de teste** utilizados.
- Varia no intervalo entre 0 e 1.
 - Quanto **maior** o escore mais adequado é o conjunto de casos de teste.

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

- $DM(P,T)$: total de mutantes mortos pelo conjunto de casos de teste T .
- $M(P)$: total de mutantes gerados a partir do programa P .
- $EM(P)$: total de mutantes equivalentes ao programa P .

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

- Critério Análise de Mutantes

- Alta eficácia em revelar a presença de erros.
- **Limitação: alto custo de aplicação!!!**
 - Equivalência entre programas.
 - Grande número de mutantes gerados e que precisam ser executados.

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

- Abordagens Alternativas
 - Mutação Aleatória
 - Mutação Seletiva
 - Mutação Restrita
 - Conjunto Essencial de Operadores de Mutação

Viabilizar a aplicação do critério em ambientes reais de desenvolvimento de software.

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

- **Mutação Aleatória**
 - Apenas uma porcentagem dos mutantes gerados a partir de cada operador é considerada.
 - Estudos feitos por Mathur e Wong (1994) indicaram que um conjunto adequado a 10% dos mutantes obtiveram escores superiores a 0.99 em relação a todos os mutantes.

Mutação Seletiva

- Os operadores de mutação responsáveis pelo maior número de mutantes não são aplicados.

Tabela: Resultados obtidos por Offut et al (1996) - Mutação Seletiva

Categoria	Escore	Red. custo (%)
RE	0.9997	6.04
ES	0.9954	71.54
RS	0.9731	22.44

Mutação Restrita

- Operadores de mutação específicos são selecionados para serem utilizados na geração dos mutantes.

Tabela: Operadores de C selecionados por Wong et al (1997)

Operador	Descrição
OALN	Troca operador aritmético por operador lógico
OCNG	Insera negação lógica
OLAN	Troca operador lógico por operador aritmético
OLLN	Troca operador lógico por operador lógico
OLNG	Insera negação lógica em condições compostas
OLRN	Troca operador lógico por operador relacional
ORLN	Troca operador relacional por operador lógico
ORRN	Troca operador relacional por operador relacional
STRP	Requer a execução de todos os comandos do programa
VDTR	Requer valor neg., pos. e zero para referência escalar
VTWD	Troca referência escalar pelo seu sucessor e predecessor

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Passos de Aplicação

Operadores de Mutação

Mutante

Mutante Morto

Mutante Equivalente

Escore de Mutação

Abordagens Alternativas

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

Mutação Restrita

Tabela: Categorias de Mutação Restritiva por Wong et al (1997)

Categoria	Operadores
MUT1	OLLN, OLNG, ORRN
MUT2	OLLN, OLNG, ORRN, OCNG, ORLN, OLRN, OLAN, OALN
MUT3	VDTR, VTWD
MUT4	STRP
MUT5	OLLN, OLNG, ORRN, VDTR, VTWD
MUT6	OLLN, OLNG, ORRN, VDTR, VTWD, STRP

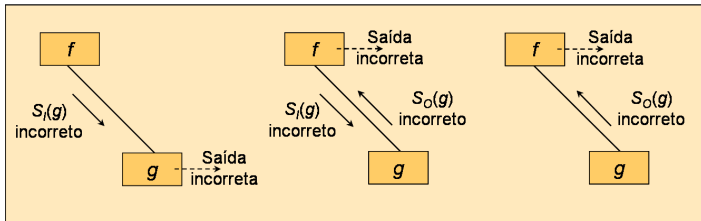
Conjunto Essencial de Operadores de Mutação

- Tipo de mutação restrita em o conjunto de operadores de mutação é escolhido sistematicamente

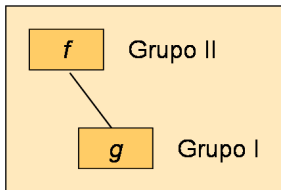
Tabela: Operadores Essenciais para C por Barbosa (1998)

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

- Estende os conceitos utilizados pela Análise de Mutantes para o teste de integração.
- **Idéias básicas:**
 - Aplicar os operadores somente nas partes relacionadas às interfaces dos módulos.
 - Chamadas de função, parâmetros, variáveis globais.
 - Restringir os operadores de mutação a fim de modelar somente os erros de integração.
 - Testar as conexões entre os módulos, separadamente, uma de cada vez.

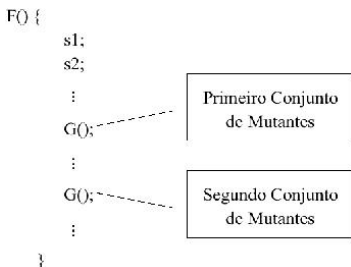


- Definição de dois grupos de operadores de mutação de interface.
 - Grupo I - função chamada
 - Comandos de interface, variáveis globais e de interface
 - Grupo II - função *chamadora*
 - Pontos de chamada de função



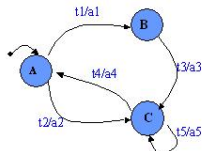
- Mutantes do Grupo-I

- Mutantes associados a uma dada chamada só podem ser mortos se tal ponto de chamada for executado!

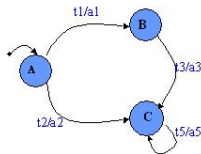


- Mapeamento das hipóteses básicas do critério Análise de Mutantes:
 - Hipótese do **projetista** competente.
 - Efeito de acoplamento.
- Identificação de erros típicos que podem ser cometidos durante a **especificação** do software.
- Trabalhos atuais exploram o Teste de Mutação para **especificações formais**.
 - Teste baseado em modelos.

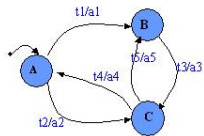
- A definição dos operadores de mutação no contexto de especificações formais baseia-se:
 - Modelos de erros de Chow para MEF:
 - Erros de transferência.
 - Erros de operação.
 - Erros de estados extras ou ausentes.
 - Operadores de mutação para linguagem C.
 - Operadores de mutação para expressões booleanas.



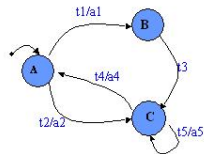
Especificação original



Mutante: Arco faltando



Mutante: Destino trocado



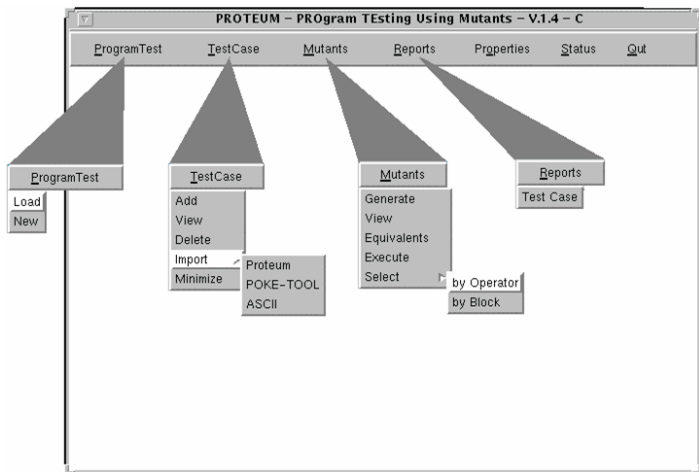
Mutante: Ação faltando

- **Mothra** - Georgia Institute of Technology (1980) (Fortran).
- **Proteum** (*PROgram TEsting Using Mutants*) - ICMC (1993).
- **μ Java (ou muJava)** - Korea Advanced Institute of Science and Technology e George Mason University (2003).
- **Milu**) - King's College London (2008) (C)

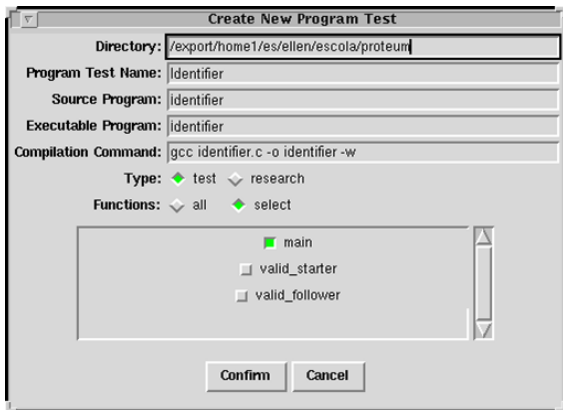
- **Proteum** (unidade) - Delamaro (1993), linguagem C.
- **Proteum/IM** (interface) - Delamaro (1997), linguagem C.
- **Proteum/FSM** (MEFs) - Fabbri (1996).
- **Proteum/ST** (Statecharts) - Sugeta (1999).
- **Proteum/PN** (Redes de Petri) - Simão (2000).

- Apóia a aplicação do critério Análise de Mutantes (unidade).
- Linguagem C
- Características
 - Orientada à sessão de teste.
 - Importação de casos de teste.
 - Inserção e remoção de casos de teste dinamicamente.
 - Casos de teste podem ser habilitados ou desabilitados.
 - Seleção dos operadores a serem utilizados.
 - **71 Operadores:** Comandos, Operadores, Variáveis e Constantes.
 - Geração de relatórios.
 - Versão com interface ou modo *script*.

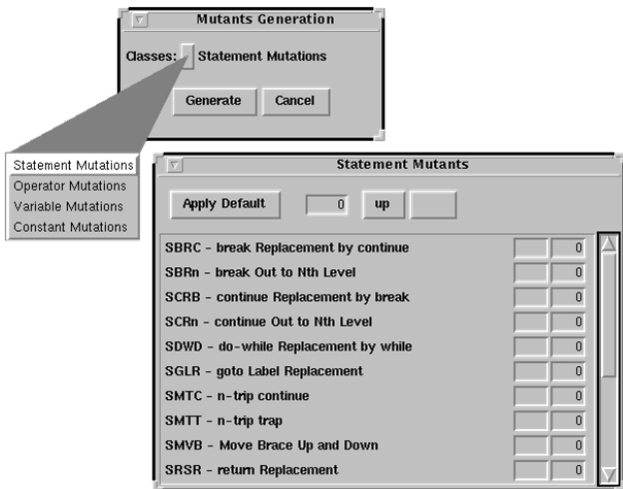
● Interface Gráfica



- Sessão de Teste



- Geração de Mutantes



Relatórios de Teste

Status

Directory:

Program Test Name:

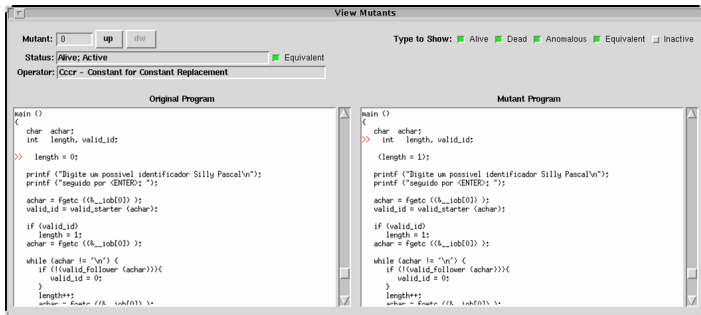
Source Program:

Executable Program:

Compilation Command:

Type: <input type="text" value="Test"/>	Test Cases: <input type="text" value="4"/>
Total Mutants: <input type="text" value="933"/>	Live Mutants: <input type="text" value="403"/>
Active Mutants: <input type="text" value="933"/>	Anomalous Mutants: <input type="text" value="0"/>
Equivalent Mutants: <input type="text" value="0"/>	MUTATION SCORE: <input type="text" value="0.568"/>

● Visualização de Mutantes



The screenshot shows the Proteum 'View Mutants' interface. At the top, there are controls for the mutant: 'Mutant: 0', 'up', and 'dnw' buttons. Below this, the 'Status' is set to 'Alive; Active' with an 'Equivalent' checkbox checked. The 'Operator' is 'Cccr - Constant for Constant Replacement'. On the right, there are checkboxes for 'Type to Show': 'Alive' (checked), 'Dead', 'Anomalous', 'Equivalent' (checked), and 'Inactive'.

The main area is split into two panes: 'Original Program' and 'Mutant Program'. Both panes display the following C code:

```
main ()
{
  char achar;
  int length, valid_id;
  >> length = 0;

  printf ("Digite um possível identificador Silly Pascal\n");
  printf ("seguido por <ENTER>: ");

  achar = fgetc ((B__ioB0));
  valid_id = valid_starter (achar);

  if (valid_id)
    length = 1;
  achar = fgetc ((B__ioB0));

  while (achar != '\n') {
    if (!valid_follower (achar)) {
      valid_id = 0;
    }
    length++;
    achar = fgetc ((B__ioB0));
  }
}
```

The 'Mutant Program' pane shows the same code as the 'Original Program' pane, indicating that the mutant is equivalent to the original.

O programa *Identifier* determina se um identificador é válido ou não. Um identificador válido deve começar com uma letra e conter apenas letras ou dígitos. Além disso, deve ter no mínimo um caractere e no máximo seis caracteres de comprimento.

● Identificadores Válidos

- abc12
- C4d5
- dcdf

● Identificadores Inválidos

- cont*1
- lsoma
- a123456

- Classes de Equivalência
(Particionamento em Classes de Equivalência)
- Classes Válidas e Inválidas

Condições de Entrada	Classes Válidas	Classes Inválidas
Tamanho t do identificador	$1 \leq t \leq 6$ (1)	$t > 6$ $t < 1$ (2) (3)
Primeiro caractere c é uma letra	Sim (4)	Não (5)
Só contém caracteres válidos	Sim (6)	Não (7)

- Conjunto de Casos de Teste

$T_0 = \{(a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido})\}$
(1, 4, 6) (5) (7) (2)

```

/* 01 */      {
/* 01 */      char achar;
/* 01 */      int length, valid_id;
/* 01 */      length = 0;
/* 01 */      printf ("Identificador: ");
/* 01 */      achar = fgetc (stdin);
/* 01 */      valid_id = valid_s(achar);
/* 01 */      if (valid_id)
/* 02 */          length = 1;
/* 03 */      achar = fgetc (stdin);
/* 04 */      while (achar != '\n')
/* 05 */          {
/* 05 */              if (!(valid_f(achar)))
/* 06 */                  valid_id = 0;
/* 07 */              length++;
/* 07 */              achar = fgetc (stdin);
/* 07 */          }
/* 08 */      if (valid_id && (length >= 1) && (length < 6))
/* 09 */          printf ("Valido\n");
/* 10 */      else
/* 10 */          printf ("Invalido\n");
/* 11 */      }

```

Implementação do Programa *Identifier* (função main)

- Associações Requeridas (**Todos-Potenciais-Usos**)

Associações Requeridas	T ₀	T ₁	T ₂	Associações Requeridas	T ₀	T ₁	T ₂
1) <1,(6,7),{ length }>		✓		17) <2,(6,7),{ length }>	✓		
2) <1,(1,3),{ achar, length, valid_id }>	✓			18) <2,(5,6),{ length }>	✓		
3) <1,(8,10),{ length, valid_id }>		✓		19) <3,(8,10),{ achar }>			✓
4) <1,(8,10),{ valid_id }>	✓			20) <3,(8,9),{ achar }>			✓
5) <1,(8,9),{ length, valid_id }>	*	*	*	21) <3,(5,7),{ achar }>	✓		
6) <1,(8,9),{ valid_id }>	✓			22) <3,(6,7),{ achar }>	✓		
7) <1,(7,4),{ valid_id }>	✓			23) <3,(5,6),{ achar }>	✓		
8) <1,(5,7),{ length, valid_id }>	✓			24) <6,(8,10),{ valid_id }>	✓		
9) <1,(5,7),{ valid_id }>	✓			25) <6,(8,9),{ valid_id }>	*	*	*
10) <1,(5,6),{ length, valid_id }>		✓		26) <6,(5,7),{ valid_id }>	✓		
11) <1,(5,6),{ valid_id }>	✓			27) <6,(5,6),{ valid_id }>			✓
12) <1,(2,3),{ achar, valid_id }>	✓			28) <7,(8,10),{ achar, length }>	✓		
13) <1,(1,2),{ achar, length, valid_id }>	✓			29) <7,(8,9),{ achar, length }>	✓		
14) <2,(8,10),{ length }>	*	*	*	30) <7,(5,7),{ achar, length }>	✓		
15) <2,(8,9),{ length }>		✓		31) <7,(6,7),{ achar, length }>			✓
16) <2,(5,7),{ length }>	✓			32) <7,(5,6),{ achar, length }>			✓

- Conjunto de Casos de Teste

T₀ = (a1, Válido), (2B3, Inválido), (Z-12, Inválido), (A1b2C3d, Inválido)

T₁ = T₀ U (1#, Inválido), (% , Inválido), (c, Válido)

T₂ = T₁ U (#-%, Inválido)

Status

Directory:

Test Session Name:

Source File:

Included Files:

Used Defines:

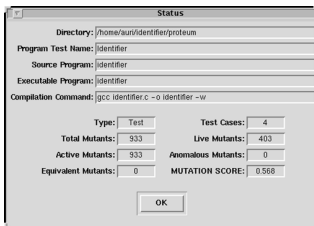
Compilation Command:

Function: Type: Total Test Case:

Criteria:

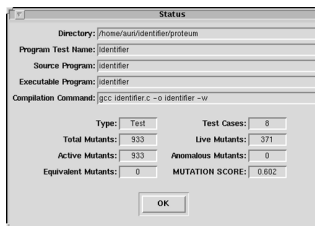
	Total	Exec	NExec	Infeas	Tot.Cover	Gr.Cover
<input checked="" type="checkbox"/> All Node	11	11	0	0	100.	
<input checked="" type="checkbox"/> All Edges	6	6	0	0	100.	
<input checked="" type="checkbox"/> All Potential Uses	32	29	3	0	90.62	89.46
<input checked="" type="checkbox"/> All Potential Uses/DU	32	29	3	0	90.62	89.46
<input checked="" type="checkbox"/> All Potential DU-paths	24	20	4	0	83.33	85.00

- Mutantes (**Análise de Mutantes**)
 - *Status* após T_0 (a) e T_1 e T_2 (b).



Status			
Directory:	/home/aur/identifier/ptreum		
Program Test Name:	Identifier		
Source Program:	Identifier		
Executable Program:	Identifier		
Compilation Command:	gcc identifier.c -o identifier -w		
Type:	Test	Test Cases:	4
Total Mutants:	933	Live Mutants:	403
Active Mutants:	933	Anomalous Mutants:	0
Equivalent Mutants:	0	MUTATION SCORE:	0.568

(a)



Status			
Directory:	/home/aur/identifier/ptreum		
Program Test Name:	Identifier		
Source Program:	Identifier		
Executable Program:	Identifier		
Compilation Command:	gcc identifier.c -o identifier -w		
Type:	Test	Test Cases:	8
Total Mutants:	933	Live Mutants:	371
Active Mutants:	933	Anomalous Mutants:	0
Equivalent Mutants:	0	MUTATION SCORE:	0.602

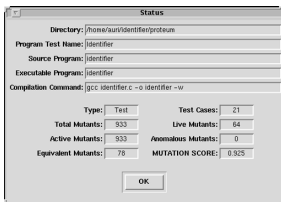
(b)

$$T_0 = \{(a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido})\}$$

$$T_1 = T_0 \cup \{(1\#, \text{Inválido}), (\%, \text{Inválido}), (c, \text{Válido})\}$$

$$T_2 = T_1 \cup \{(\#-\%, \text{Inválido})\}$$

- Mutantes (**Análise de Mutantes**)
 - *Status* após $T3$ (a) e $T4$ (b)



Directory: /home/aur/Identifier/ptreum

Program Test Name: Identifier

Source Program: Identifier

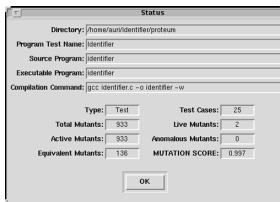
Executable Program: Identifier

Compilation Command: gcc Identifier.c -o Identifier -w

Type:	Test	Test Cases:	21
Total Mutants:	933	Live Mutants:	64
Active Mutants:	933	Anomalous Mutants:	0
Equivalent Mutants:	78	MUTATION SCORE:	0.925

OK

(a)



Directory: /home/aur/Identifier/ptreum

Program Test Name: Identifier

Source Program: Identifier

Executable Program: Identifier

Compilation Command: gcc Identifier.c -o Identifier -w

Type:	Test	Test Cases:	25
Total Mutants:	933	Live Mutants:	2
Active Mutants:	933	Anomalous Mutants:	0
Equivalent Mutants:	136	MUTATION SCORE:	0.997

OK

(b)

$T3 = T2 \cup \{(zzz, \text{Válido}), (aA, \text{Válido}), (A1234, \text{Válido}), (ZZZ, \text{Válido}), (AAA, \text{Válido}), (aa09, \text{Válido}), ([, \text{Inválido}), (\{, \text{Inválido}), (x/, \text{Inválido}), (x:, \text{Inválido}), (x18, \text{Válido}), (x[[, \text{Inválido}), (x\{\{, \text{Inválido})\}$

$T4 = T3 \cup \{(@, \text{Inválido}), (' , \text{Inválido}), (x@, \text{Inválido}), (x', \text{Inválido})\}$

- Mutantes Vivos
 - Mutante gerado pelo operador ORRN.

```

{
    char  achar;
    int  length, valid_id;
    length = 0;
    printf ("Identificador: ");
    achar = fgetc (stdin);
    valid_id = valid_s(achar);
    if (valid_id)
        length = 1;
    achar = fgetc (stdin);
    while (achar != '\n')
    {
        if (!(valid_f(achar)))
            valid_id = 0;
        length++;
        achar = fgetc (stdin);
    }
    if (valid_id && (length >= 1) && (length <= 6))
        printf ("Valido\n");
    else
        printf ("Invalido\n");
}
    
```



$t = \{(ABCDEF, \text{Válido})\}$ Saída obtida = **Inválido**

- Mutantes Vivos
 - Mutante gerado pelo operador VTWD (troca referência escalar por sucessor e predecessor).

```

{
    char  achar;
    int  length, valid_id;
    length = 0;
    printf ("Identificador: ");
    achar = fgetc (stdin);
    valid_id = valid_s(achar);
    if (valid_id)
        length = 1;
    achar = fgetc (stdin);
    while (achar != '\n')
    {
        if (!(valid_f(achar)))
            valid_id = 0;
        length++;
        achar = fgetc (stdin);
    }
    if (valid_id && (length >= 1) && (PRED(length) < 6))
        printf ("Valido\n");
    else
        printf ("Invalido\n");
}
    
```



Mutante *Error-Revealing*

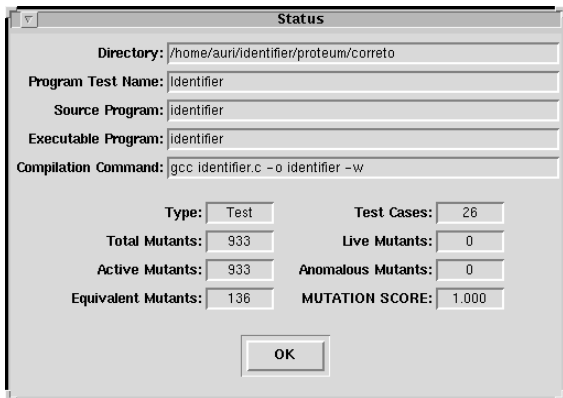
Um mutante é dito ser *error-revealing* se para qualquer caso de teste t tal que $P^*(t)$ diferente de $M^*(t)$ pudermos concluir que $P^*(t)$ não está de acordo com o resultado esperado, ou seja, **revela a presença de um erro**.

- Para qualquer caso de teste que diferencie o comportamento do mutante em relação ao programa original, também é possível concluir que o comportamento do programa original **não está de acordo** com o resultado esperado.

```
{
    char achar;
    int length, valid_id;
    length = 0;
    printf ("Identificador: ");
    achar = fgetc (stdin);
    valid_id = valid_s(achar);
    if (valid_id)
        length = 1;
    achar = fgetc (stdin);
    while (achar != '\n')
    {
        if (!(valid_f(achar)))
            valid_id = 0;
        length++;
        achar = fgetc (stdin);
    }
    if (valid_id && (length >= 1) && (length <= 6))
        printf ("Valido\n");
    else
        printf ("Invalido\n");
}
```

Programa *Identifier*: **Versão Corrigida**

- *Status* após $T5$ no programa corrigido.



The screenshot shows a dialog box titled "Status" with the following fields and values:

Directory:	/home/auri/identifier/teum/correto		
Program Test Name:	identifier		
Source Program:	identifier		
Executable Program:	identifier		
Compilation Command:	gcc identifier.c -o identifier -w		
Type:	Test	Test Cases:	26
Total Mutants:	933	Live Mutants:	0
Active Mutants:	933	Anomalous Mutants:	0
Equivalent Mutants:	136	MUTATION SCORE:	1.000

At the bottom of the dialog box is an "OK" button.

$T5 = T4 \cup \{(ABCDEF, \text{Válido})\}$

- Técnica de Teste Baseada em Mutação
 - **Erros mais freqüentes** cometidos pelo programador.
 - Operadores de Mutação.
 - Mutantes (vivo, morto, equivalente, *error-revealing*).
 - Escore de mutação.
 - **Aplicabilidade.**
 - Programa (unidade, interface).
 - Especificação.
 - **Limitação:** alto custo de aplicação.
 - Abordagens alternativas.

Introdução ao Teste de Software

Aulas Anteriores

Técnica de Teste Baseada em Erros

Análise de Mutantes

Mutação de Interface

Mutação em Especificação

Ferramentas para o Teste de Mutação

Exemplo: Identifier

Resumo

Exercício

