

Teste de Software

Parte 2

Engenharia de Software
Profa. Dra. Elisa Yumi Nakagawa
1º semestre de 2015

Técnica Estrutural (Caixa Branca)

- Baseada no conhecimento da estrutura interna (implementação) do programa
- Teste dos detalhes procedimentais
- A maioria dos critérios dessa técnica utilizam uma representação de programa conhecida como grafo de programa ou grafo de fluxo de controle

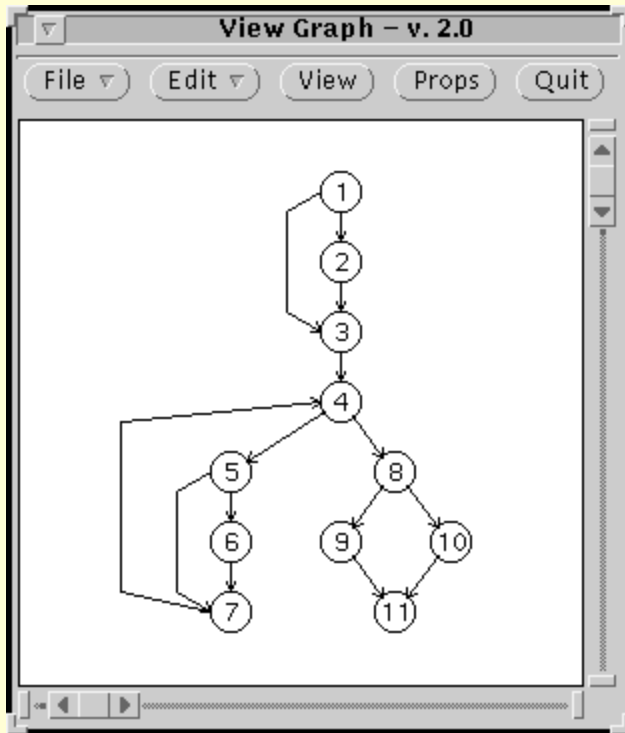
Técnica Estrutural

- Grafo de Programa
 - Nós: blocos “indivisíveis”
 - Não existe desvio para o meio do bloco
 - Uma vez que o primeiro comando do bloco é executado, os demais comandos são executados seqüencialmente
 - Arestas ou Arcos: representam o fluxo de controle entre os nós

Identifcier.c (função *main*)

```
/* 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 */ }
```

Técnica Estrutural



Grafo de Programa

- Detalhes considerados
 - nó
 - arco
 - caminho
 - simples (2,3,4,5,6,7)
 - completo (1,2,3,4,5,7,4,8,9,11)
 - fluxo de controle

Grafo de Programa do *identifier*
Gerado pela *View-Graph*

Identifier.c (função *main*)

```
/* 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 */ }
```

Caminho
Não-Executável

Identifier.c (funções valid_s () e valid_f ())

```
int valid_s (ch)
char ch;
{
    if (((ch >= 'A') && (ch <= 'Z')) ||
        ((ch >= 'a') && (ch <= 'z')))
        return (1);
    else
        return (0);
}
```

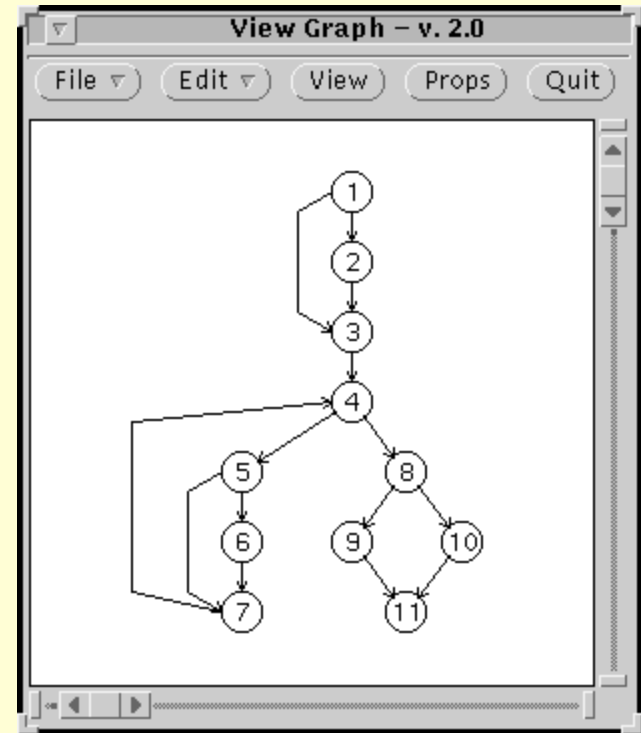
```
int valid_f(ch)
char ch;
{
    if (((ch >= 'A') && (ch <= 'Z')) ||
        ((ch >= 'a') && (ch <= 'z')) ||
        ((ch >= '0') && (ch <= '9')))
        return (1);
    else
        return (0);
}
```

Técnica Estrutural

- Critérios da Técnica Estrutural
 - Baseados em Fluxo de Controle
 - Todos-Nós, Todas-Arestas e Todos-Caminhos
 - Baseados em Fluxo de Dados
 - Critérios de Rapps e Weyuker
 - Todas-Defs, Todos-Usos, Todos-P-Usos e outros
 - Critérios Potenciais-Usos (Maldonado)
 - Todos-Potenciais-Usos, Todos-Potenciais-Usos/DU e outros
 - Baseados em Complexidade
 - Critério de McCabe

Técnica Estrutural

- Critérios Baseados em Fluxo de Controle
 - Todos-Nós
1,2,3,4,5,6,7,8,9,10,11
 - Todos-Arcos
 - arcos primitivos
<1,2>, <1,3>, <5,6>, <5,7>, <8,9>, <8,10>
 - Todos-Caminhos



Grafo de Programa do *identifier*
Gerado pela *ViewGraph*

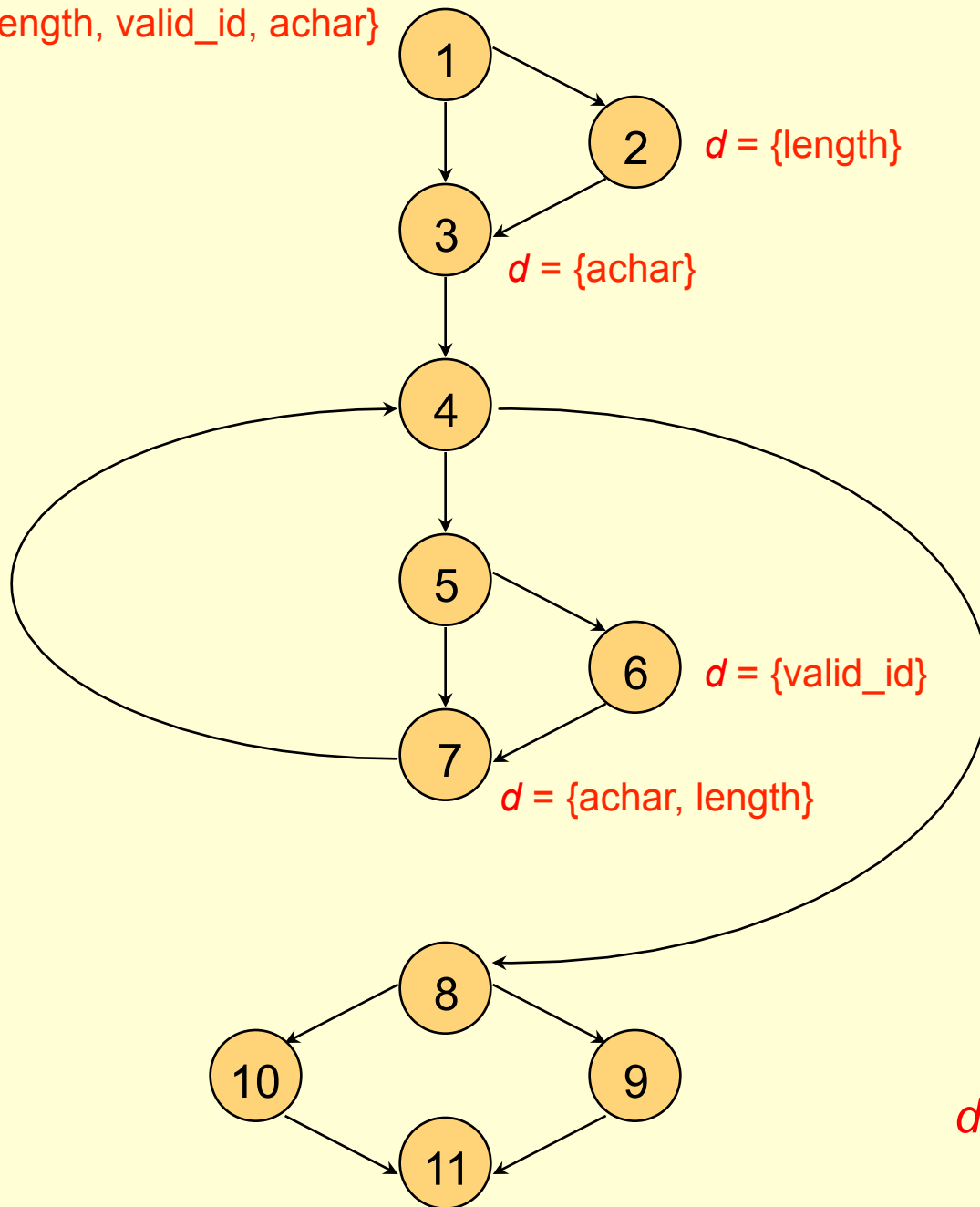
Técnica Estrutural

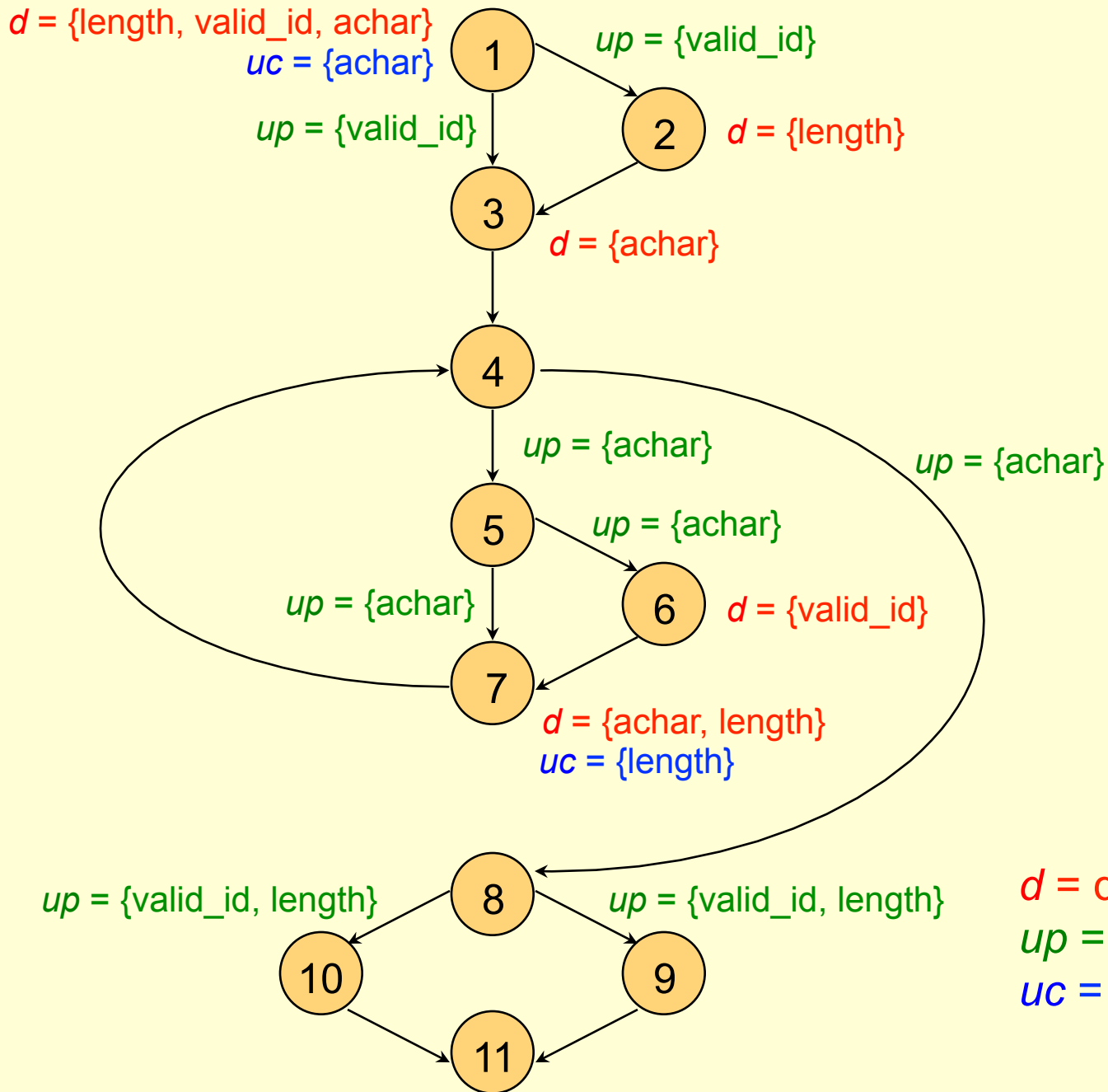
- Critérios Baseados em Fluxo de Dados
 - Rapps e Weyuker

Grafo Def-Uso: Grafo de Programa + Definição e Uso de Variáveis

- Definição
 - Atribuição de um valor a uma variável ($a = 1$)
- Uso
 - Predicativo: a variável é utilizada em uma condição
if ($a > 0$)
 - Computacional: a variável é utilizada em uma computação
 $b = a + 1$

$d = \{\text{length, valid_id, achar}\}$





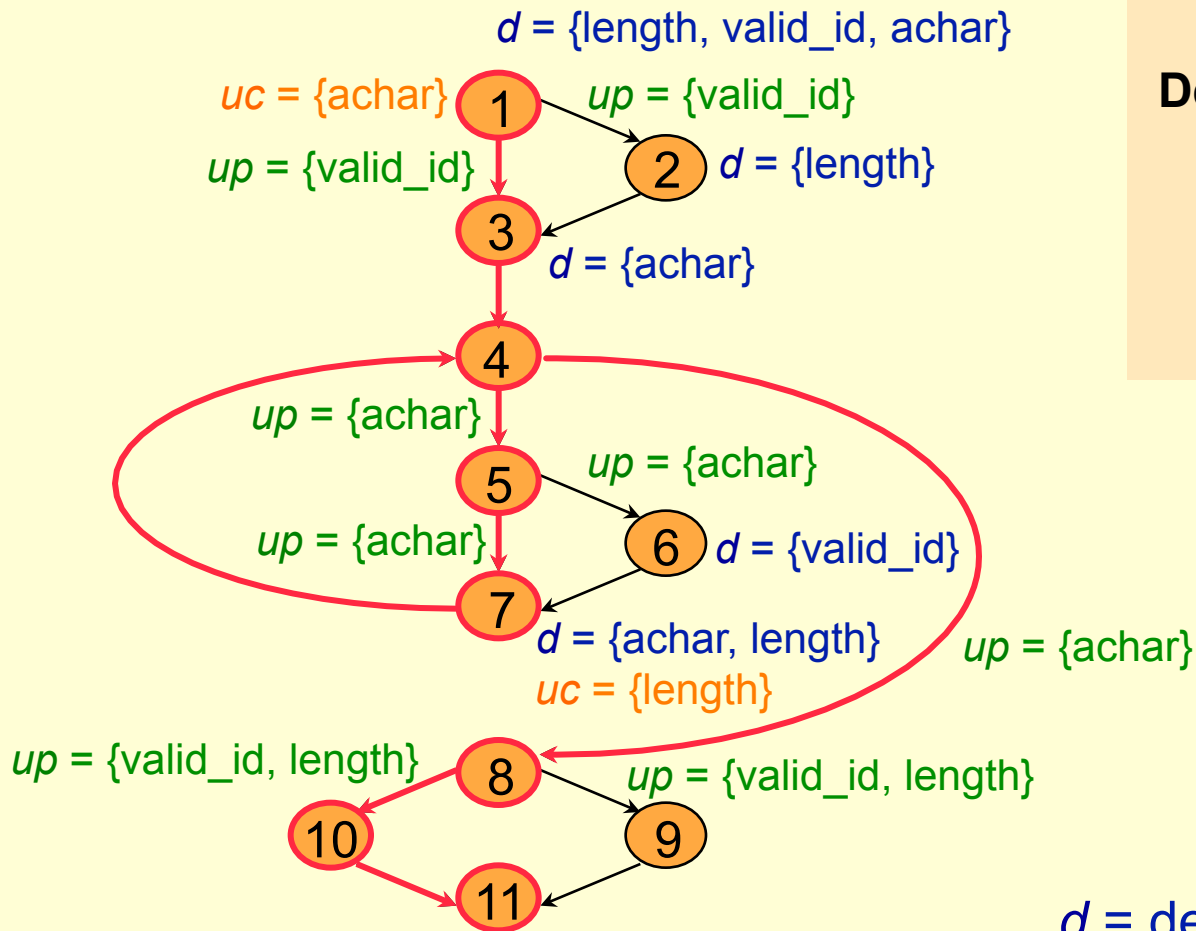
$d =$ definição
 $up =$ uso predicativo
 $uc =$ uso computacional

Técnica Estrutural

- Critérios Baseados em Fluxo de Dados
 - Rapps e Weyuker
 - Todas-Definições

Requer que cada definição de variável seja exercitada pelo menos uma vez, não importa se por um c-uso ou por um p-uso.

(1 3 4 5 7 4 8 10 11)



Grafo Def-Use do *identifier*

Rapps e Weyuker Todas-Definições

Definição de *length* no nó 1

- ✓ (1,7, *length*)
- × (1,(8,9), *length*)
- (1,(8,10), *length*)

d = definição

up = uso predicativo

uc = uso computacional

Técnica Estrutural

- Critérios Baseados em Fluxo de Dados
 - Rapps e Weyuker
 - Todos-Usos

Requer que todas as associações entre uma definição de variável e seus subseqüentes usos sejam exercitadas pelos casos de teste, através de pelo menos um caminho livre de definição.

(1 3 4 5 7 4 8 10 11)

(1 3 4 8 10 11)

$d = \{\text{length, valid_id, achar}\}$

$uc = \{\text{achar}\}$ 1 $up = \{\text{valid_id}\}$ 2 $d = \{\text{length}\}$

$up = \{\text{valid_id}\}$ 3 $d = \{\text{achar}\}$

4 $up = \{\text{achar}\}$ 5 $up = \{\text{achar}\}$

6 $d = \{\text{valid_id}\}$ 7 $d = \{\text{achar, length}\}$ $uc = \{\text{length}\}$ $up = \{\text{achar}\}$

8 $up = \{\text{valid_id, length}\}$ 9 $up = \{\text{valid_id, length}\}$

10 11

Grafo Def-Use do *identifier*

Rapps e Weyuker Todos-Usos

Definição de *length* no nó 1

- ✓ (1,7, *length*)
- ✗ (1,(8,9), *length*)
- ✓ (1,(8,10), *length*)

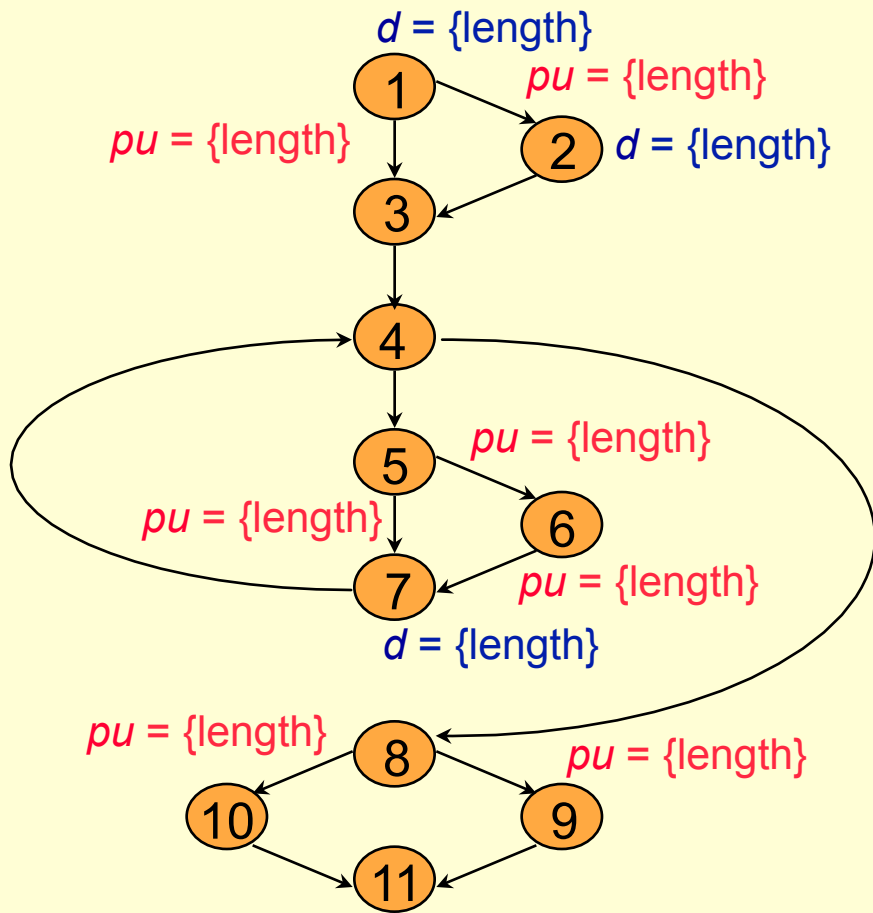
$d =$ definição
 $up =$ uso predicativo
 $uc =$ uso computacional

Técnica Estrutural

- Critérios Baseados em Fluxo de Dados
 - Potenciais-Usos

Grafo Def: Grafo de Programa + Definição de Variáveis

- Conceito de Potencial-Associação
 - Associações são estabelecidas sem a necessidade de um uso explícito



Grafo Def do *identifier*

d = definição

pu = potencial-uso

Potenciais-Usos

Todos-Potenciais-Usos

Definição de *length* no nó 1

$\langle 1, (1, 2), \{\text{length}\} \rangle$

$\langle 1, (1, 3), \{\text{length}\} \rangle$

$\langle 1, (5, 6), \{\text{length}\} \rangle$

$\langle 1, (5, 7), \{\text{length}\} \rangle$

$\langle 1, (6, 7), \{\text{length}\} \rangle$

$\times \langle 1, (8, 9), \{\text{length}\} \rangle$

$\langle 1, (8, 10), \{\text{length}\} \rangle$

➤ 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 }>			✓

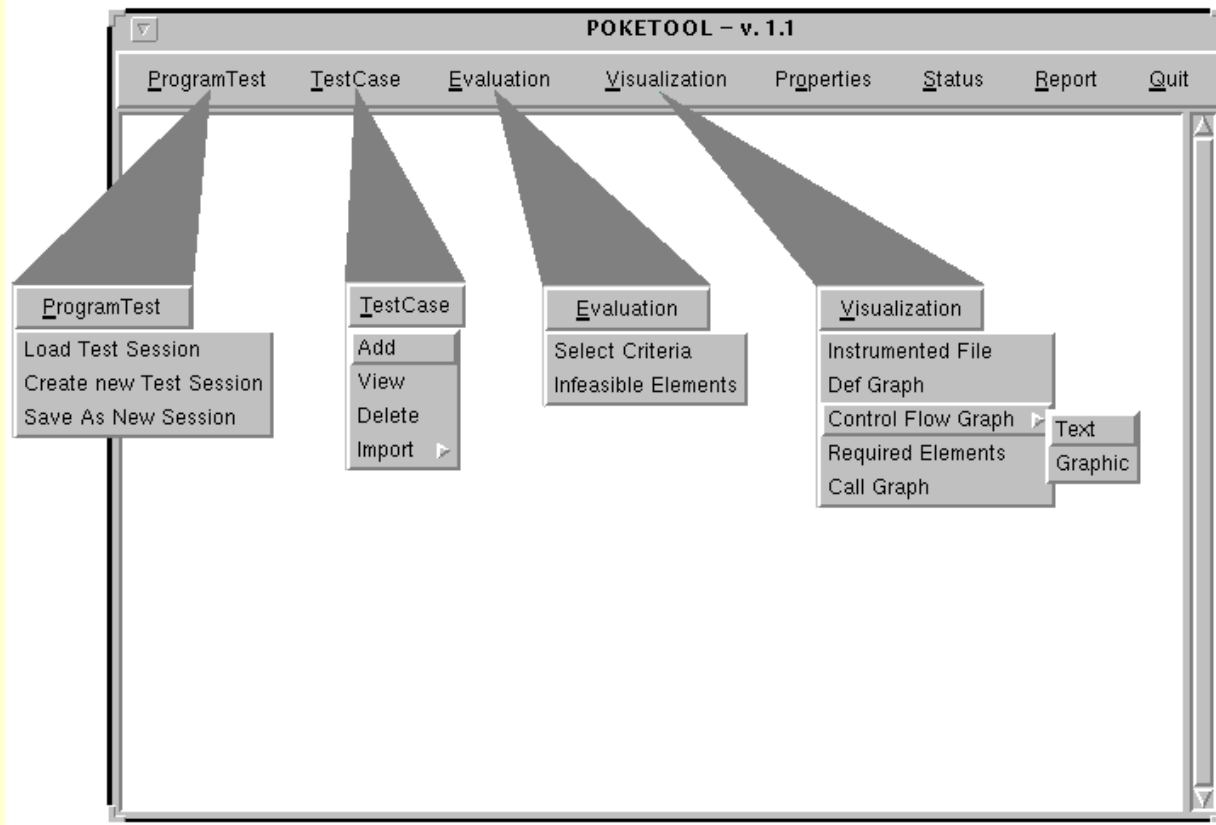
- $T_0 = \{ (a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido}) \}$
- $T_1 = T_0 \cup \{ (1\#, \text{Inválido}), (\%, \text{Inválido}), (c, \text{Válido}) \}$
- $T_2 = T_1 \cup \{ (\#-\%, \text{Inválido}) \}$

Técnica Estrutural

- Ferramenta *PokeTool*
 - Critérios Potenciais-Usos
 - Critérios de Rapps e Weyuker
 - Outros Critérios Estruturais
 - Todos-Nós, Todos-Arcos
 - Linguagem C
 - Outras Características
 - Importação de casos de teste
 - Inserção e remoção de casos de teste dinamicamente
 - Casos de teste podem ser habilitados ou desabilitados
 - Geração de relatórios

Técnica Estrutural

➤ *PokeTool*: Interface Gráfica



Técnica Estrutural

- *PokeTool*: Criando uma Sessão de Teste

Create New Test Session

Directory: /export/home1/es/ellen/escola/poketool

Test Session Name: l

Source Program: i.c

Included Files:

Used Defines:

Functions: main

Compilation Command: gcc <source> -o <exec> -w

Criteria:

- All Node
- All Edges
- All Potential Uses
- All Potential Uses/DU
- All Potential DU-paths

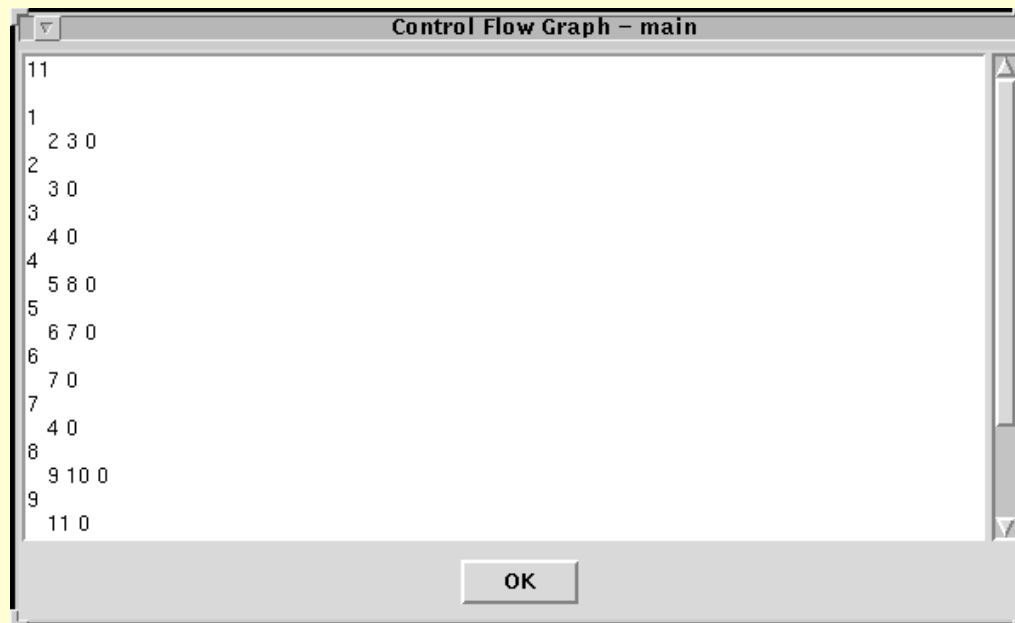
Type:

- test
- research

Confirm Cancel

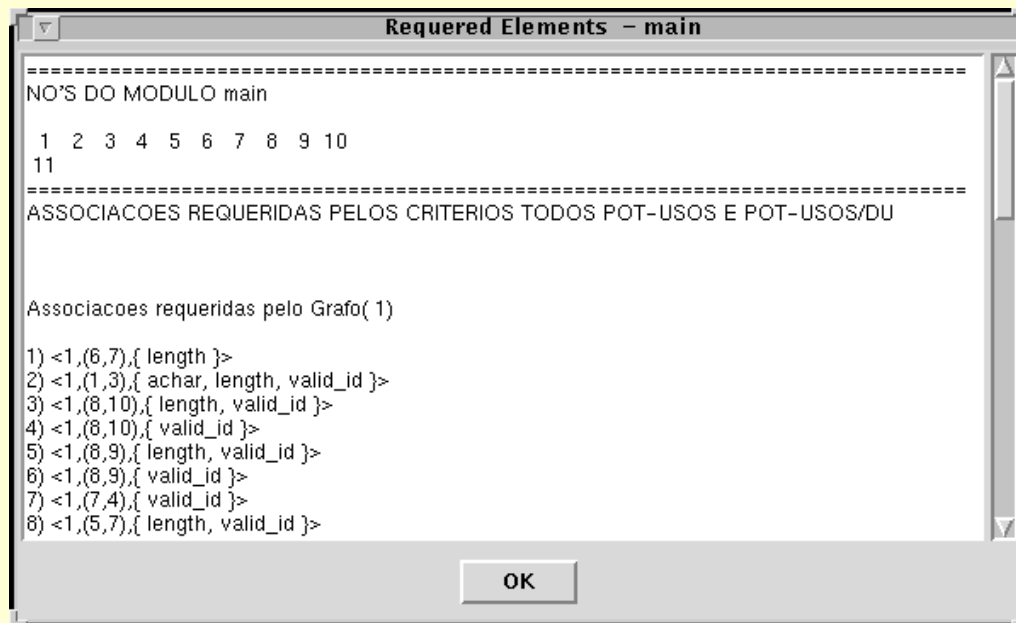
Técnica Estrutural

- *PokeTool*: Grafo de Programa



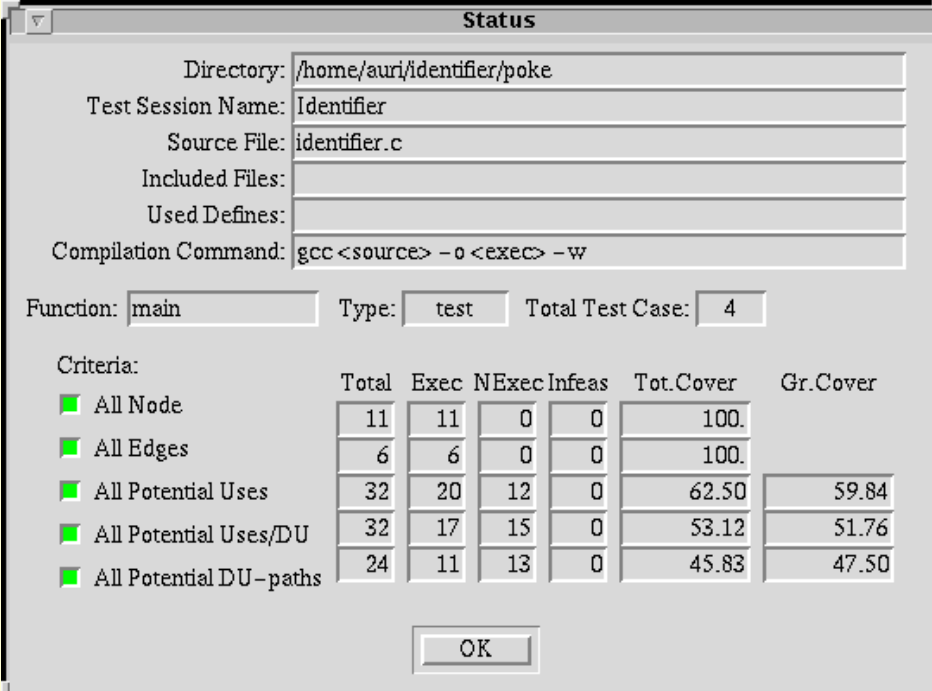
Técnica Estrutural

➤ *PokeTool*: Elementos Requeridos



Técnica Estrutural

➤ Status após T_0



The screenshot shows a 'Status' dialog box with the following fields:

- Directory: /home/auri/identifier/poke
- Test Session Name: Identifier
- Source File: identifier.c
- Included Files:
- Used Defines:
- Compilation Command: gcc <source> -o <exec> -w
- Function: main
- Type: test
- Total Test Case: 4

Criteria table:

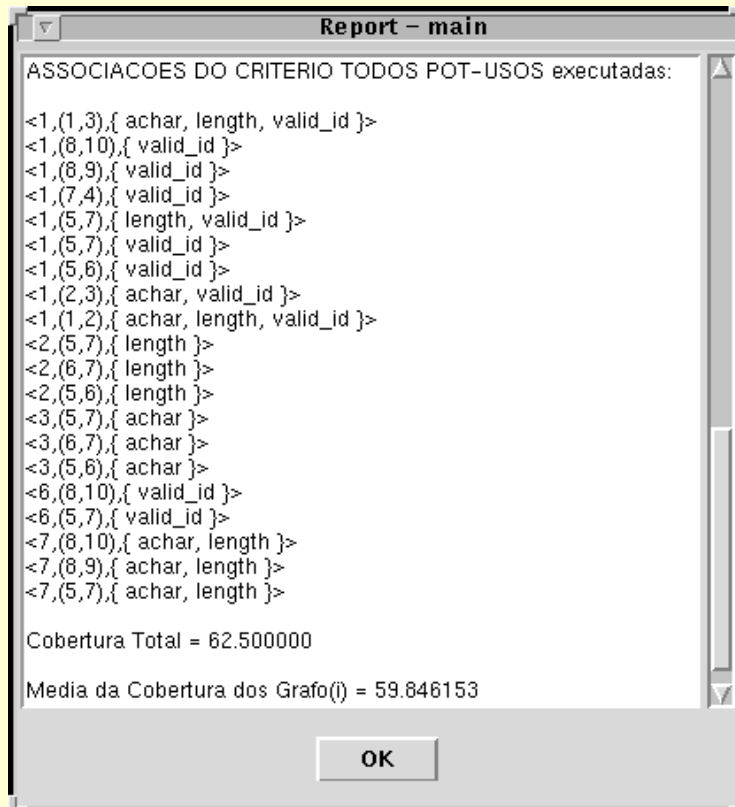
Criteria:	Total	Exec	NExec	Infeas	Tot.Cover	Gr.Cover
<input checked="" type="checkbox"/> All Node	11	11	0	0	100.	
<input checked="" type="checkbox"/> All Edges	6	6	0	0	100.	
<input checked="" type="checkbox"/> All Potential Uses	32	20	12	0	62.50	59.84
<input checked="" type="checkbox"/> All Potential Uses/DU	32	17	15	0	53.12	51.76
<input checked="" type="checkbox"/> All Potential DU-paths	24	11	13	0	45.83	47.50

OK

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

Técnica Estrutural

➤ *PokeTool*: Relatórios de Teste

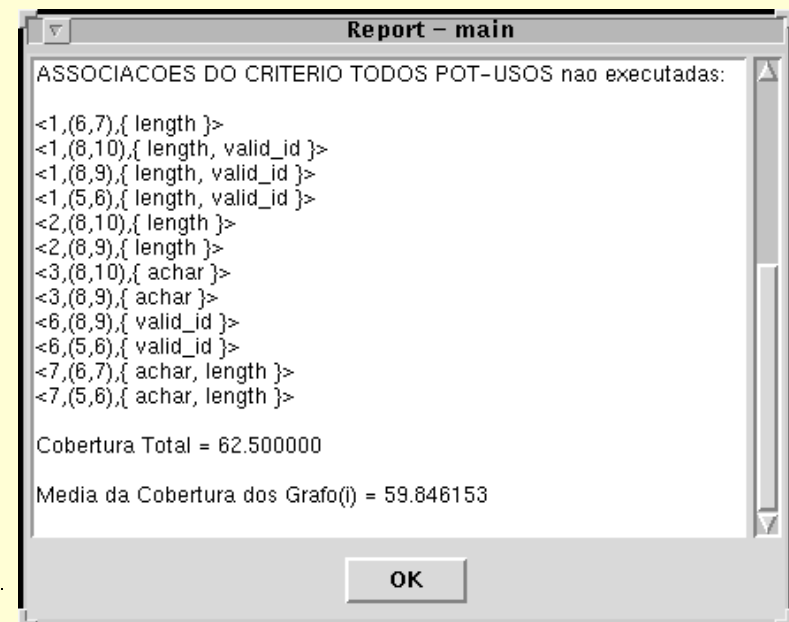


```
Report - main
ASSOCIACOES DO CRITERIO TODOS POT-USOS executadas:
<1,(1,3),{ achar, length, valid_id }>
<1,(8,10),{ valid_id }>
<1,(8,9),{ valid_id }>
<1,(7,4),{ valid_id }>
<1,(5,7),{ length, valid_id }>
<1,(5,7),{ valid_id }>
<1,(5,6),{ valid_id }>
<1,(2,3),{ achar, valid_id }>
<1,(1,2),{ achar, length, valid_id }>
<2,(5,7),{ length }>
<2,(6,7),{ length }>
<2,(5,6),{ length }>
<3,(5,7),{ achar }>
<3,(6,7),{ achar }>
<3,(5,6),{ achar }>
<6,(8,10),{ valid_id }>
<6,(5,7),{ valid_id }>
<7,(8,10),{ achar, length }>
<7,(8,9),{ achar, length }>
<7,(5,7),{ achar, length }>

Cobertura Total = 62.500000

Media da Cobertura dos Grafo(i) = 59.846153

OK
```



```
Report - main
ASSOCIACOES DO CRITERIO TODOS POT-USOS nao executadas:
<1,(6,7),{ length }>
<1,(8,10),{ length, valid_id }>
<1,(8,9),{ length, valid_id }>
<1,(5,6),{ length, valid_id }>
<2,(8,10),{ length }>
<2,(8,9),{ length }>
<3,(8,10),{ achar }>
<3,(8,9),{ achar }>
<6,(8,9),{ valid_id }>
<6,(5,6),{ valid_id }>
<7,(6,7),{ achar, length }>
<7,(5,6),{ achar, length }>

Cobertura Total = 62.500000

Media da Cobertura dos Grafo(i) = 59.846153

OK
```

Técnica Estrutural

➤ Status após T_1 (a) e T_2 (b)

Directory: /home/auri/identifler/poke
Test Session Name: Identifier
Source File: identifler.c
Included Files:
Used Defines:
Compilation Command: gcc <source> -o <exec> -w

Function: main Type: test Total Test Case: 7

Criteria:

	Total	Exec	NExec	Infeas	Tot.Cover	Gr.Cover
All Node	11	11	0	0	100.	
All Edges	6	6	0	0	100.	
All Potential Uses	32	26	6	0	81.25	76.46
All Potential Uses/DU	32	26	6	0	81.25	76.46
All Potential DU-paths	24	18	6	0	75.00	75.00

OK

(a)

Directory: /home/auri/identifler/poke
Test Session Name: Identifier
Source File: identifler.c
Included Files:
Used Defines:
Compilation Command: gcc <source> -o <exec> -w

Function: main Type: test Total Test Case: 8

Criteria:

	Total	Exec	NExec	Infeas	Tot.Cover	Gr.Cover
All Node	11	11	0	0	100.	
All Edges	6	6	0	0	100.	
All Potential Uses	32	29	3	0	90.62	89.46
All Potential Uses/DU	32	29	3	0	90.62	89.46
All Potential DU-paths	24	20	4	0	83.33	85.00

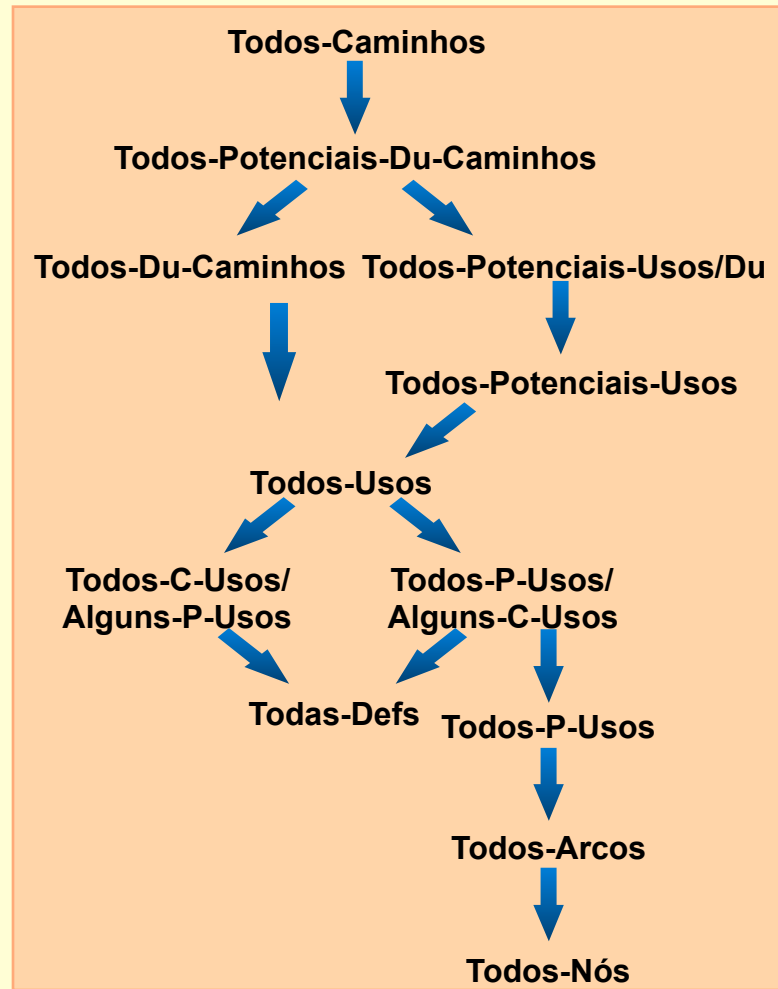
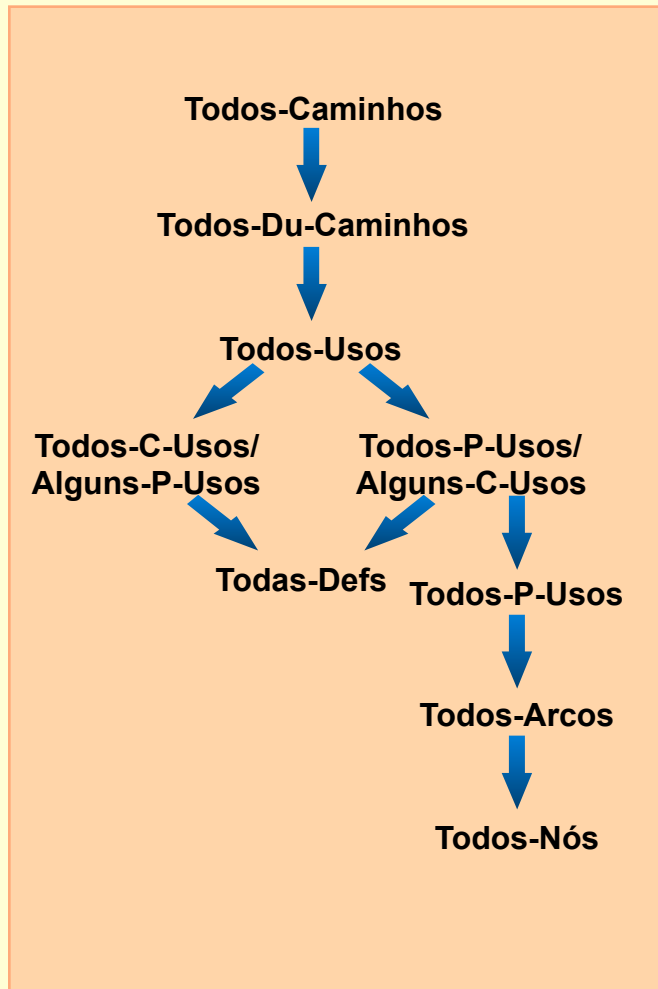
OK

(b)

- $T_1 = T_0 \cup \{(1\#, \text{Inválido}), (\%, \text{Inválido}), (c, \text{Válido})\}$
- $T_2 = T_1 \cup \{(\#-\%, \text{Inválido})\}$

Técnica Estrutural

➤ Hierarquia entre Critérios Estruturais



Técnica Baseada em Erros

- Os requisitos de teste são derivados a partir dos erros mais freqüentes cometidos durante o processo de desenvolvimento do software
- Critérios da Técnica Baseada em Erros
 - Semeadura de Erros
 - Teste de Mutação
 - Análise de Mutantes (unidade)
 - Mutação de Interface (integração)

Análise de Mutantes

➤ Hipótese do Programador Competente

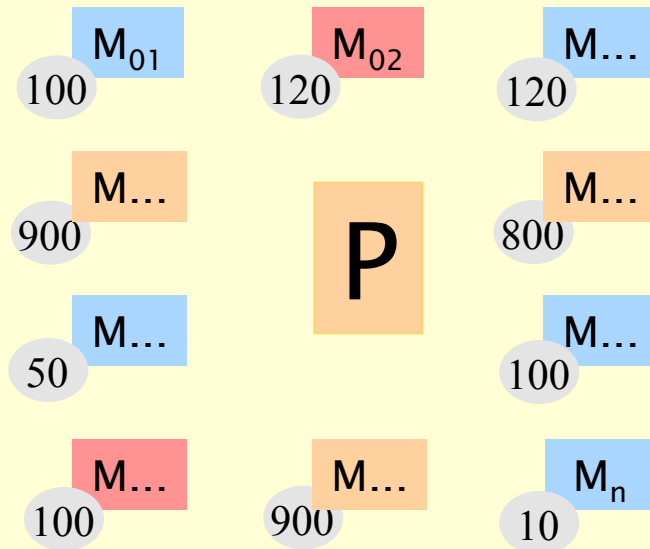
Programadores experientes escrevem programas corretos ou muito próximos do correto.

➤ Efeito de Acoplamento

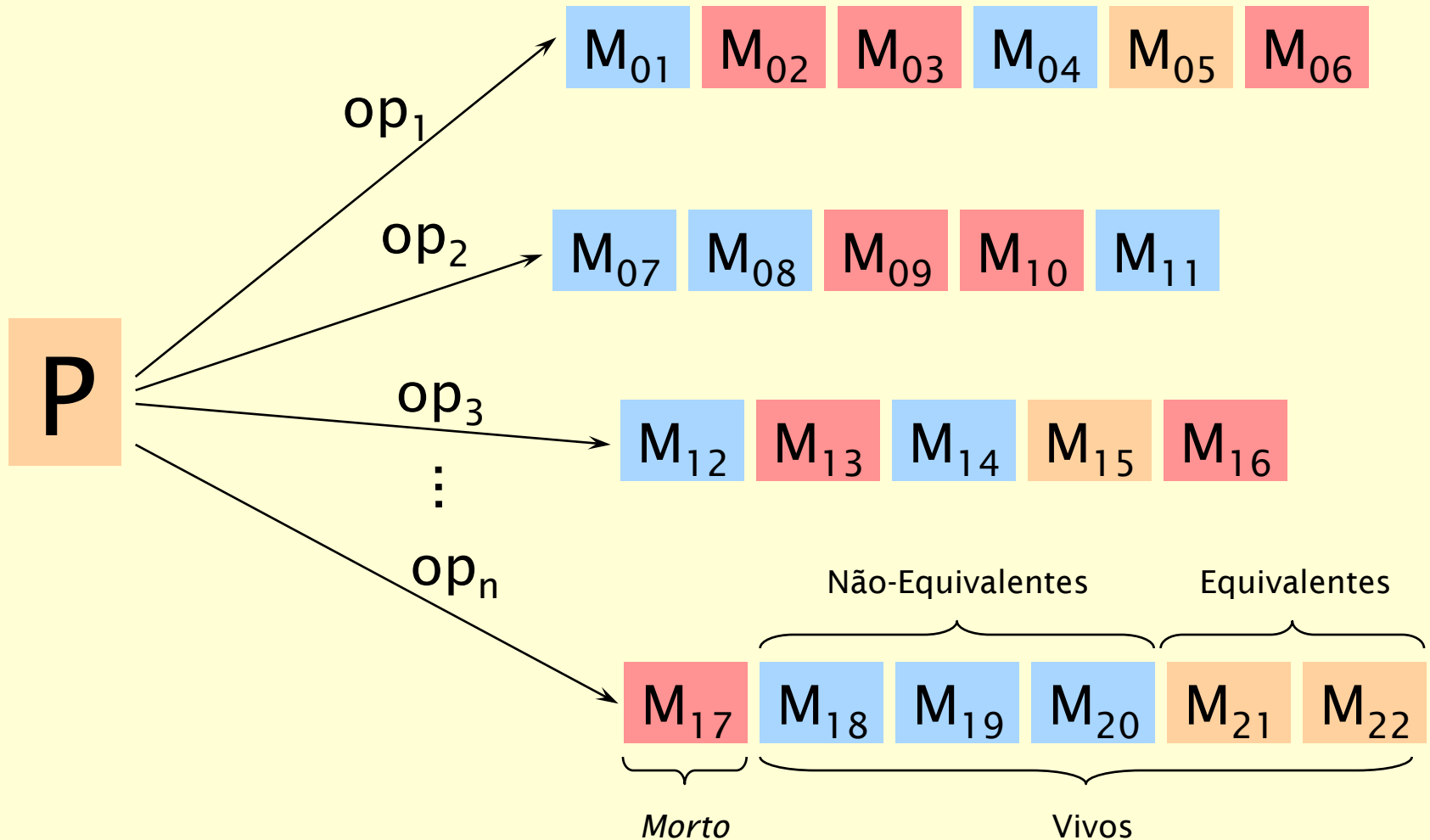
Casos de teste capazes de revelar erros simples são tão sensíveis que, implicitamente, também são capazes de revelar erros mais complexos.

Teste de Mutação

Freqüência de Execução
×
Determinação de Mutantes Equivalentes



Teste de Mutação

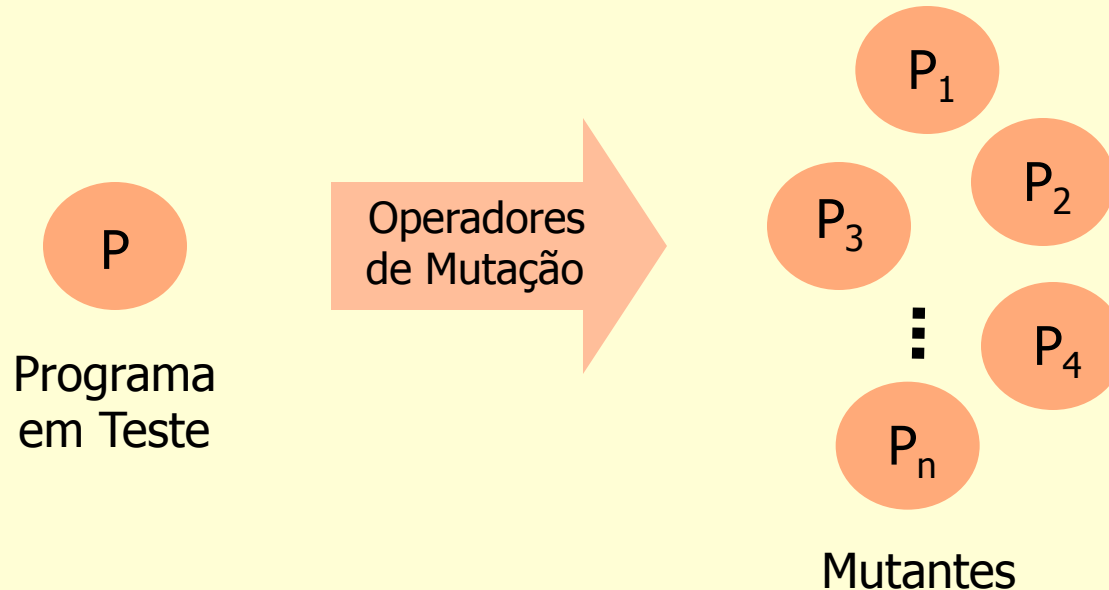


Análise de Mutantes

➤ Passos da Análise de Mutantes

1- Geração de Mutantes

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



Análise de Mutantes

- 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

Análise de Mutantes

➤ Exemplo de Mutantes

Mutante Gerado pelo Operador OLAN

```
if (valid_id * (length >= 1) && (length < 6) )  
    printf ("Valido\n");  
else  
    printf ("Invalido\n");
```

Mutante Gerado pelo Operador ORRN

```
if (valid_id && (length >= 1) && (length <= 6) )  
    printf ("Valido\n");  
else  
    printf ("Invalido\n");
```

Análise de Mutantes

➤ Passos da Análise de Mutantes

2 - Execução do Programa

- Execução do programa com os casos de teste

3 - Execução dos Mutantes

- Execução dos mutantes com os casos de teste
 - Mutante morto
 - Mutante vivo

4 - Análise dos Mutantes Vivos

- Mutante equivalente
- Inclusão de novos casos de teste

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

Análise de Mutantes

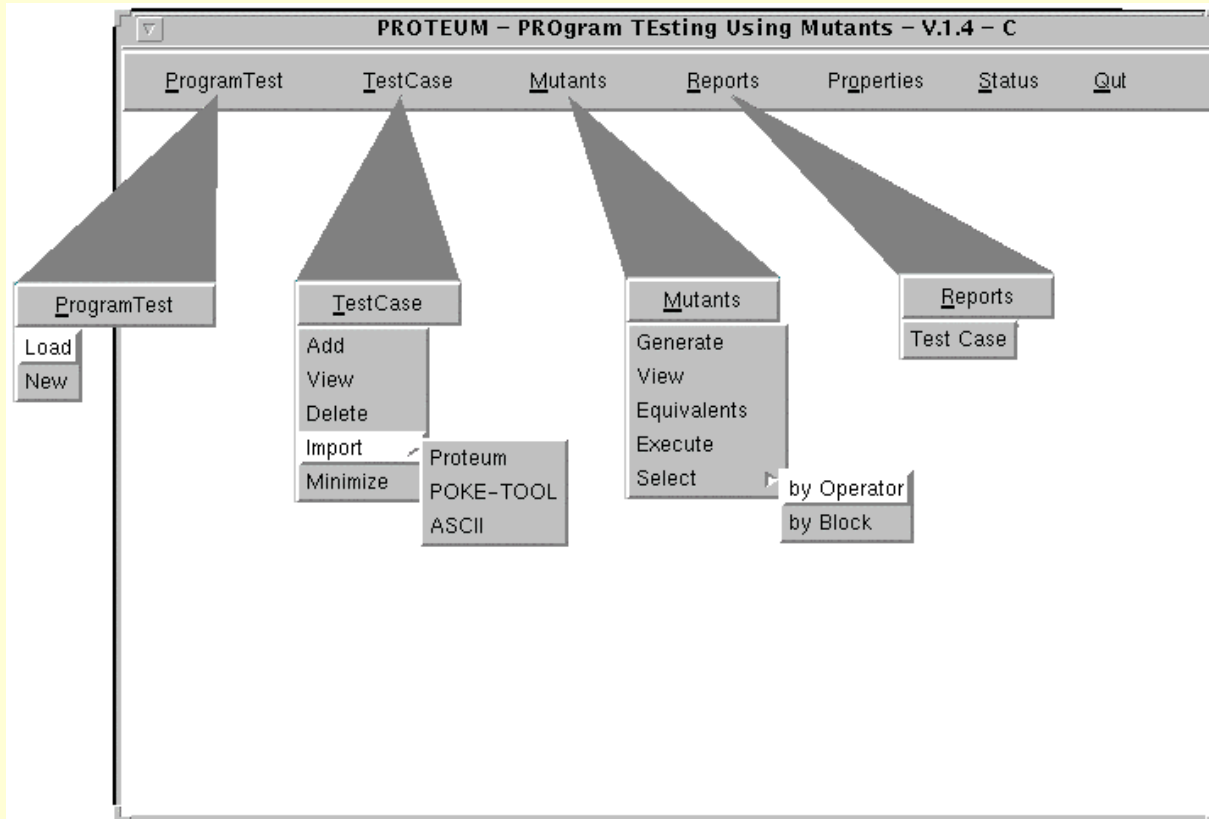
- Ferramenta *Proteum*
 - Critério Análise de Mutantes
 - Linguagem C
 - Outras Características
 - Importação de casos de teste
 - Inserção e remoção de casos de teste dinamicamente
 - Casos de teste podem ser habilitados ou desabilitados
 - Seleção dos operadores a serem utilizados
 - 71 operadores: comandos, operadores, variáveis e constantes
 - Geração de relatórios

Análise de Mutantes

- Ferramenta *Proteum*
 - Interface gráfica
 - Mais fácil
 - Constante interação com o testador
 - Scripts
 - Possibilitam a condução de uma sessão de teste de modo programado
 - Domínio dos conceitos de mutação e dos programas que compõem as ferramentas

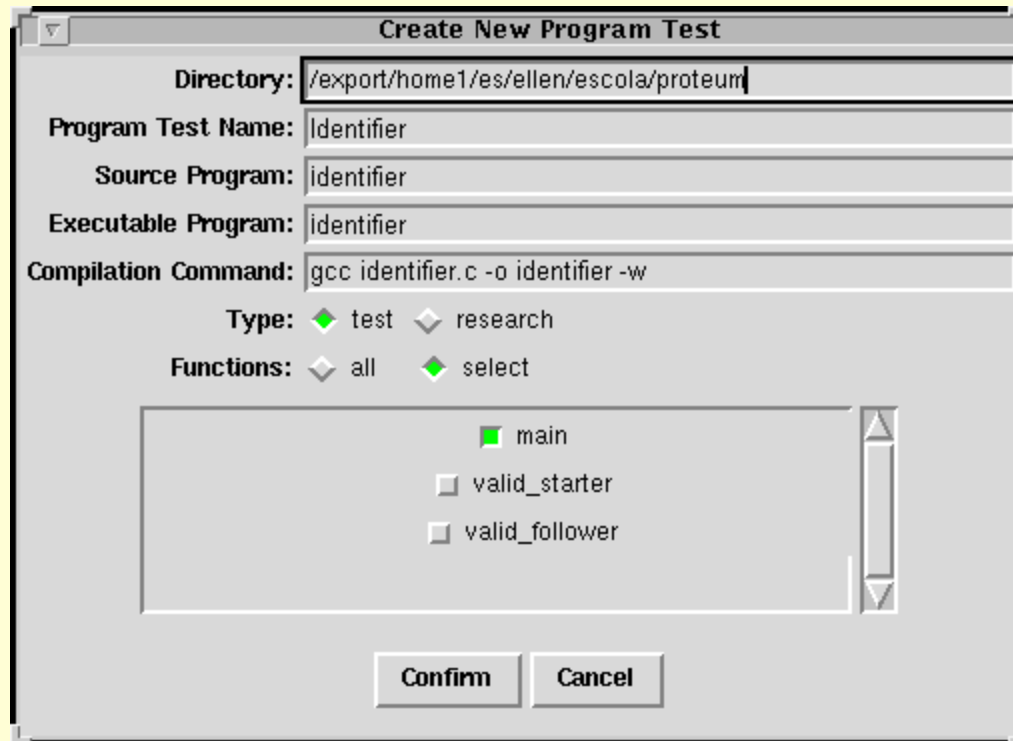
Análise de Mutantes

➤ *Proteum*: Interface Gráfica



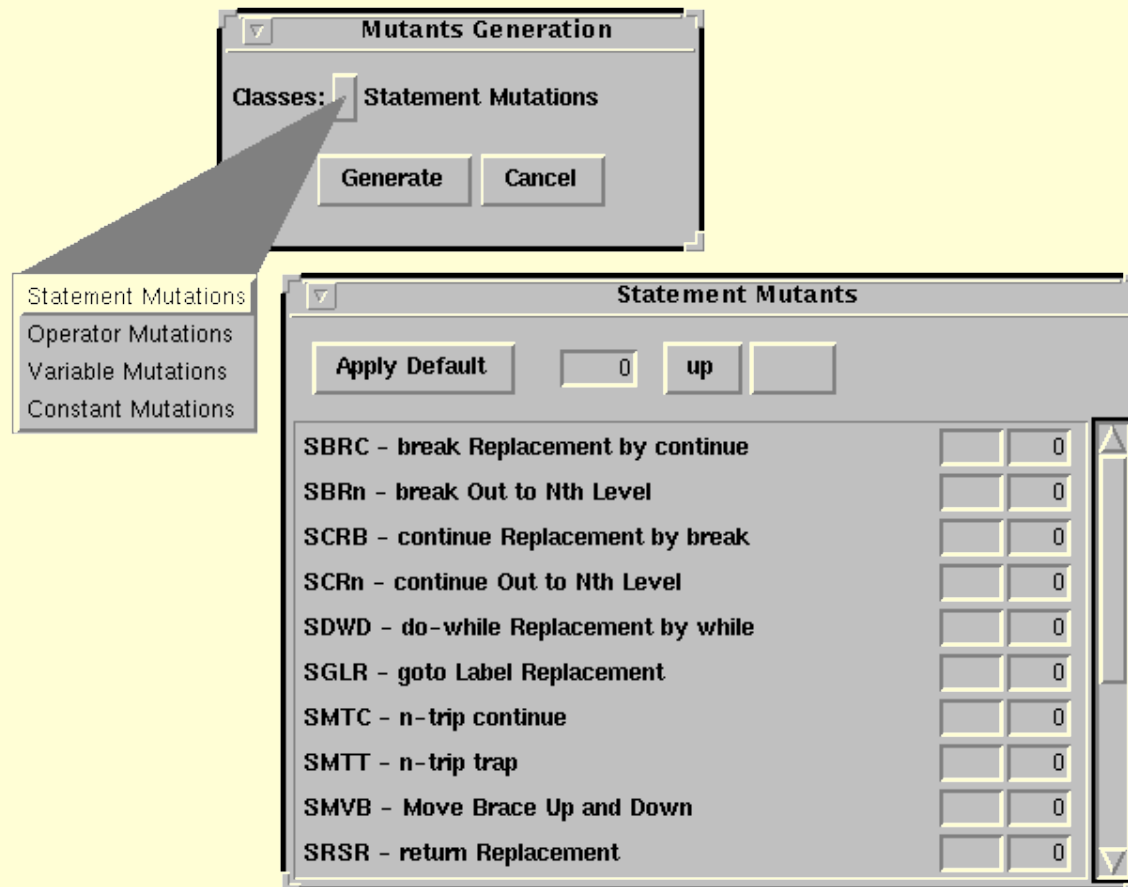
Análise de Mutantes

- *Proteum*: Criando uma Sessão de Teste



Análise de Mutantes

➤ *Proteum*: Gerando Mutantes



Análise de Mutantes

- *Status* após T_0 (a) e T_2 (b)

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

- Directory: /home/auri/identifier/teum
- 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

An OK button is located at the bottom center.

(a)

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

- Directory: /home/auri/identifier/teum
- 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

An OK button is located at the bottom center.

(b)

- $T_0 = \{ (a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido}) \}$
- $T_1 = T_0 \cup \{ (1\#, \text{Inválido}), (\%, \text{Inválido}), (c, \text{Válido}) \}$
- $T_2 = T_1 \cup \{ (\#-\%, \text{Inválido}) \}$

Análise de Mutantes

➤ *Proteum*: Visualização de Mutantes

The screenshot displays the Proteum software interface, titled "View Mutants". At the top, there are controls for selecting a mutant (Mutant: 0) and buttons for navigation (up, dw). Below this, the "Status" is set to "Alive; Active" with a checked "Equivalent" checkbox, and the "Operator" is "Cccr - Constant for Constant Replacement". To the right, the "Type to Show" section includes checkboxes for "Alive", "Dead", "Anomalous", "Equivalent", and "Inactive", all of which are currently checked.

The interface is split into two panes: "Original Program" and "Mutant Program".

Original Program:

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

  >> length = 0;

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

  achar = fgetc (&__iob[0] );
  valid_id = valid_starter (achar);

  if (valid_id)
    length = 1;
  achar = fgetc (&__iob[0] );

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

Mutant Program:

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

  (length = 1);

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

  achar = fgetc (&__iob[0] );
  valid_id = valid_starter (achar);

  if (valid_id)
    length = 1;
  achar = fgetc (&__iob[0] );

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

Análise de Mutantes

➤ Status após T_3 (a) e T_4 (b)

Directory:	/home/auri/identifer/proteum		
Program Test Name:	identifer		
Source Program:	identifer		
Executable Program:	identifer		
Compilation Command:	gcc identifer.c -o identifer -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

(a)

Directory:	/home/auri/identifer/proteum		
Program Test Name:	identifer		
Source Program:	identifer		
Executable Program:	identifer		
Compilation Command:	gcc identifer.c -o identifer -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

(b)

- $T_3 = T_2 \cup \{(zzz, \text{Válido}), (aA, \text{Válido}), (A1234, \text{Válido}), (ZZZ, \text{Válido}), (AAA, \text{Válido}), (aa09, \text{Válido}), ([, \text{Inválido}), (\{, \text{Inválido}), (x/, \text{Inválido}), (x:, \text{Inválido}), (x18, \text{Válido}), (x[, \text{Inválido}), (x\{\{, \text{Inválido})\}$
- $T_4 = T_3 \cup \{(@, \text{Inválido}), (` , \text{Inválido}), (x@, \text{Inválido}), (x` , \text{Inválido})\}$

Análise de Mutantes

➤ Mutantes Vivos

Mutante Gerado pelo Operador VTWD

```
if (valid_id && (length >= 1) && (PRED(length) < 6) )  
    printf ("Valido\n");  
else  
    printf ("Invalido\n");
```

Mutante Gerado pelo Operador ORRN

```
if (valid_id && (length >= 1) && (length <= 6) )  
    printf ("Valido\n");  
else  
    printf ("Invalido\n");
```

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

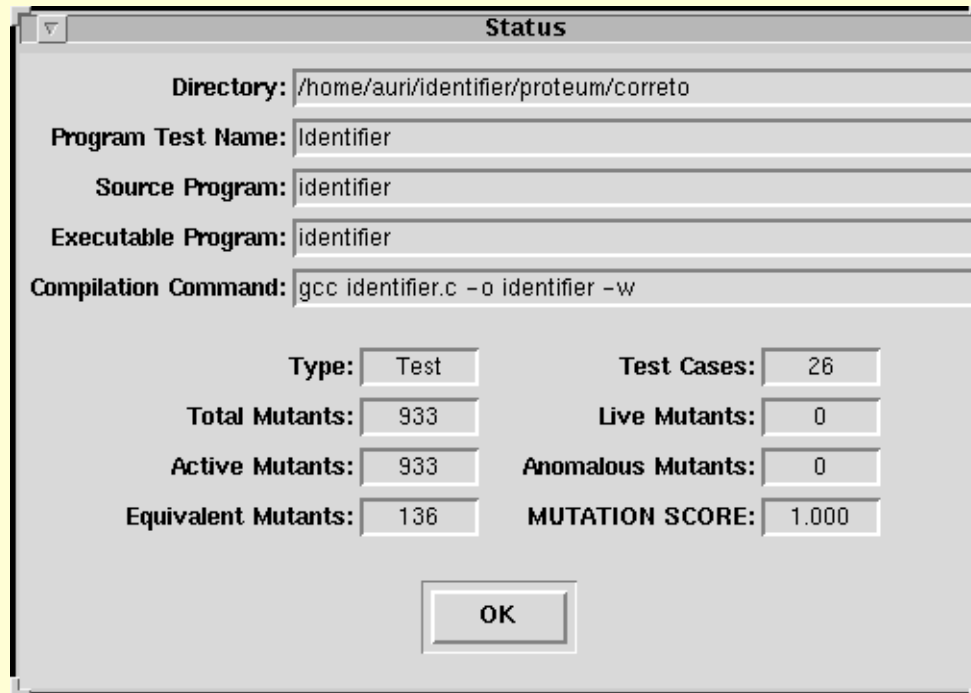
Identif.c (função *main*)

Versão Corrigida

```
/* 01 */ {
/* 01 */     char  achar;
/* 01 */     int   length, valid_id;
/* 01 */     length = 0;
/* 01 */     printf ("Digite um possível identificador\n");
/* 01 */     printf ("seguido por <ENTER>: ");
/* 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 */ }
```

Análise de Mutantes

- *Status* após T_5 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

OK

- $T_5 = T_4 \cup \{(ABCDEF, \text{Válido})\}$

Teste de Integração

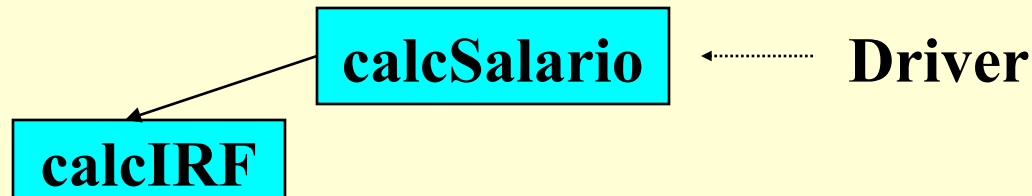
- Após testar cada módulo (teste de unidade), deve-se testar a integração entre os módulos (teste de integração)
 - O teste de unidade não garante que a integração das unidades terá sucesso
 - Por exemplo, funções, quando combinadas, podem produzir resultados inesperados

Teste de Integração

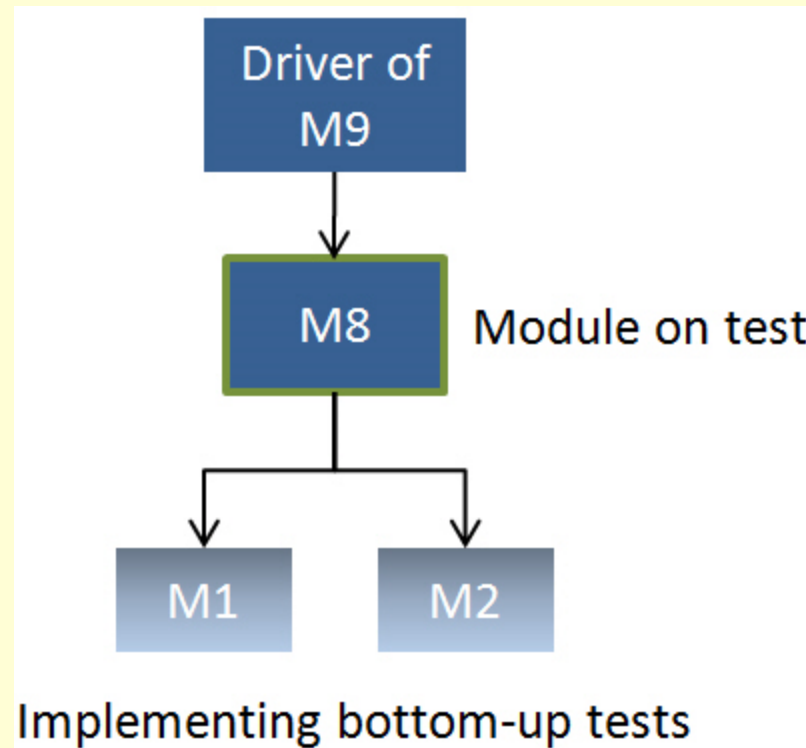
- Considere uma unidade $F \rightarrow$ referente a uma sub-rotina ou um método, por exemplo
- *Driver*
 - Unidade que coordena o teste de F , sendo responsável por:
 - ler os dados de teste fornecidos pelo testador,
 - repassar esses dados na forma de parâmetros para F ,
 - coletar os resultados relevantes produzidos por F e
 - apresentá-los para o testador
- *Stubs*
 - Unidade que simula o comportamento da unidade chamada por F com o mínimo de computação

Teste de Integração

- Exemplo de driver:
 - Função para calcular o salário (calcSalario), com base no salário base, número de horas, adicionais, desconto de IRF(sub-função calcIRF), desconto de INSS (sub-função calcINSS)
- Se ainda não temos o calcSalario, como testar calcIRF(salBruto)?
 - Driver
 - Fazer um falso calcSalario que inicializa as variáveis necessárias para poder chamar o calcIRF, chama calcIRF e apresenta o resultado.



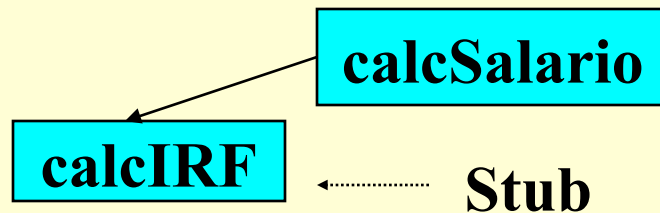
Teste de Integração



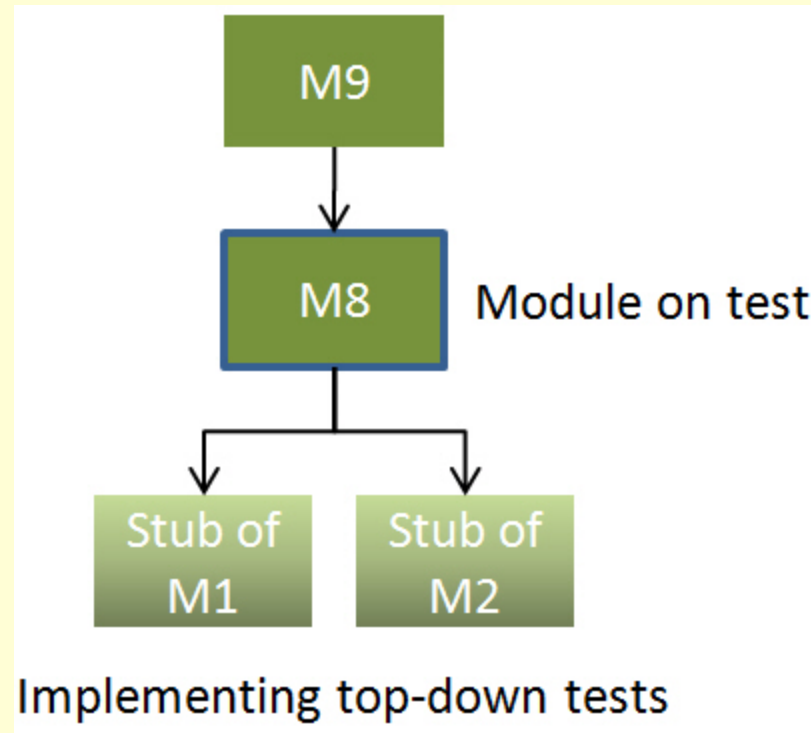
Teste bottom-up: começa pelo teste de unidade e depois integra

Teste de Integração

- Por outro lado, se ainda não temos o calcIRF, mas já temos calcSalario, como testar calcSalario?
 - Stub
 - Fazer um falso calcIRF que retorna um valor qualquer (fixo, por exemplo, 20,00)



Teste de Integração



Teste top-down: começa pelo teste das unidades de hierarquia mais alta e depois testa as unidades folha

Teste em OO

- Teste de unidade: métodos individualmente testados
- Teste de classe: testa a integração entre métodos de uma classe
- Teste de integração: testa a interação entre classes do sistema
- Teste de sistema: testa a funcionalidade do sistema como um todo

Conclusões

- A atividade de teste é fundamental no processo de desenvolvimento de software
 - Qualidade do produto
- Alto custo da atividade de teste
- Desenvolvimento e aplicação de técnicas e critérios de teste
- Desenvolvimento e utilização de ferramentas de teste
- Não existe um algoritmo de teste de propósito geral para provar a corretitude de um programa

Perspectivas

- Estratégias de Teste
- Teste de Integração
- Teste Orientado a Objetos e de Componentes
- Teste de Aspectos
- Teste de Especificação
- Teste de Sistemas Reativos
- Ambiente Integrado para Teste, Depuração e Manutenção de Software