



# CBSoft

## II Congresso Brasileiro de Software: Teoria e Prática

**SBES**

**SBLP**

**SBMF**

**SBCARS**

**XXV Simpósio Brasileiro de Engenharia de Software**

**XV Simpósio Brasileiro de Linguagens de Programação**

**XIV Simpósio Brasileiro de Métodos Formais**

**V Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software**

Miniconferência Latino-Americana de Linguagens de Padrões para Programação

XVIII Sessão de Ferramentas | IV FEES | II Trilha da Indústria | I WTDSOFT

V LA-WASP | V WDDS | V SAST | II AutoSoft | II WB-DSDM | II WESB | I WBVS

# Minicurso

**Introdução a "Slicing" de Programas**

Marcelo d'Amorim, Elton Alves (UFPE)

Marcelo d'Amorim  
Elton Alves

## Introdução a Slicing de Programas



---

---

---

---



---

---

---

Marcelo d'Amorim  
Elton Alves

## Introdução a ~~Slicing~~ Extração de Dependência de Programas



---

---

---

---

---

---

---

### Motivação

- Informação de dependência é útil para uma série de aplicações

Cenário 1:

Modifiquei o código e não sei o que inspecionar ou testar. Que partes podem ter sido indiretamente afetadas pela mudança?

---

---

---

---

---

---

---

## Motivação

- Informação de dependência é útil para uma série de aplicações

Cenário 1: **Análise de Impacto**

Modifiquei o código e não sei o que inspecionar ou testar. Que partes podem ter sido indiretamente afetadas pela mudança?

Ver: Chianti [Ren et al., OOPSLA 2004]

---

---

---

---

---

---

---

---

## Motivação

- Informação de dependência é útil para uma série de aplicações

Cenário 2:

Tenho uma aplicação grande e uma propriedade específica a checar. Como ignorar partes irrelevantes?

---

---

---

---

---

---

---

---

## Motivação

- Informação de dependência é útil para uma série de aplicações

Cenário 2: **Slicing estático**

Tenho uma aplicação grande e uma propriedade específica a checar. Como ignorar partes irrelevantes?

Ver: Indus [Dwyer et al., TACAS 2006]

---

---

---

---

---

---

---

---

## Motivação

- Informação de dependência é útil para uma série de aplicações

Cenário 3:

A execução de um teste falha; viola uma asserção de estado. Mas o que produziu este estado indevido?

---

---

---

---

---

---

---

## Motivação

- Informação de dependência é útil para uma série de aplicações

Cenário 3:

Localização de Falta

A execução de um teste falha; viola uma asserção de estado. Mas o que produziu este estado indevido?

Ver: Tarantula [Jones et al., ICSE 2002]

---

---

---

---

---

---

---

## Motivação

- Informação de dependência é útil para uma série de aplicações

localização de faltas  
seleção de testes  
dataflow analysis  
model checking  
compreensão de código

---

---

---

---

---

---

---

## Agenda

- Parte 1: Conceitos
  - Representação de código, dependência de dados, dependência de controle, etc.
  - Slicing estático
  - Slicing dinâmico
    - Dados, full, e relevante

---

---

---

---

---

---

---

## Agenda

- Parte 2: Demo
- Parte 3: Aplicações
  - Localização de Faltas
  - Compreensão de Linhas de Produto de Software

---

---

---

---

---

---

---

## Agenda

- Parte 2: Demo
- Parte 3: Aplicações
  - Localização de Faltas
  - Compreensão de Linhas de Produto de Software



Elton Alves, mestrando



Sabrina Souto, doutoranda

---

---

---

---

---

---

---

## Parte 1: Básico

---

---

---

---

---

---

---

---

## Representações do programa

pgm. fonte      pgm. objeto      PDG  
                          pilha      SSA  
 3 endereços  
 árvore sintática      call-graph      CFG

---

---

---

---

---

---

---

---

## Representações do programa

pgm. fonte      pgm. objeto      PDG  
                          pilha      SSA  
 3 endereços  
 árvore sintática      call-graph      CFG

---

---

---

---

---

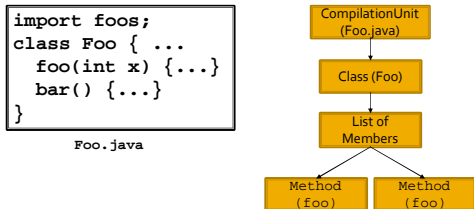
---

---

---

## Árvore Sintática

- Estrutura em árvore caracterizando estrutura de uma aplicação

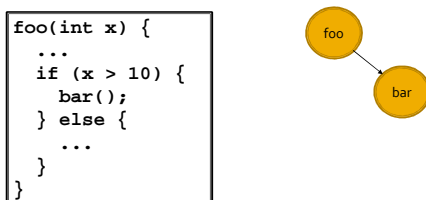


## Como obter árvore sintática

- Use um parser
  - JapaParser (para fonte Java)
    - <http://code.google.com/p/javaparser/>

## Call graph

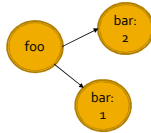
- Grafo direcionado que representa as chamadas de sub-rotinas



## Call graph

- Grafo direcionado que representa as chamadas de sub-rotinas

```
foo(int x) {
  ...
  if (x > 10) {
    thizz.bar();
  } else {
    ...
  }
}
```



Pode requerer análise de ponteiro!

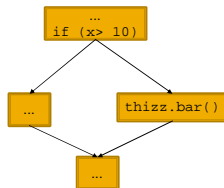
## Como obter o call graph

- SOOT: <http://www.sable.mcgill.ca/soot/>
- No Eclipse
  - Selecione um método
  - Menu "Navigate > Open Call Hierarchy"

## Control-flow graph (CFG)

- DAG que representa fluxo de controle de um procedimento

```
foo(int x) {
  ...
  if (x > 10) {
    thizz.bar();
  } else {
    ...
  }
}
```





## Como obter CFG

- Use um parser
  - JapaParser (para fonte Java)
    - <http://code.google.com/p/javaparser/>
  - BCEL (para bytecodes Java)
    - <http://www.sable.mcgill.ca/soot/>

## Dependência de dados

- Linha j depende de linha i se puder ler um valor escrito em i.

Pares uso-definição.

```
foo(int x) {
  ...
  if (x > 10) {
    x = x + 1;
  } else {
    x = 5;
  }
  print(x);
}
```

## Dependência de dados

- Linha j depende de linha i se puder ler um valor escrito em i.

Pares definição-uso.

```
foo(int x) {
  ...
  if (x > 10) {
    x = x + 1;
  } else {
    x = 5;
  }
  print(x);
}
```

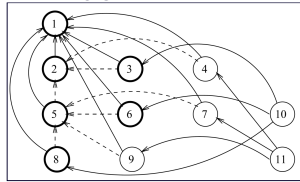
## Dependência de controle

- O comando j possui uma dependência de controle para i se sua execução de i determina a execução de j.

```
foo(int x) {
  ...
  if (x > 10) {
    x = x + 1;
  } else {
    x = 5;
  }
  print(x);
}
```

## Dados e controle juntos

DEPENDENCE GRAPH



```
begin
S1: read(X);
S2: if (X < 0)
    then
S3:   Y := f1(X);
S4:   Z := g1(X);
    else
S5:   if (X = 0)
        then
S6:   Y := f2(X);
S7:   Z := g2(X);
        else
S8:   Y := f3(X);
S9:   Z := g3(X);
        end if;
    end if;
S10: write(Y);
S11: write(Z);
end.
```

From "Dynamic Program Slicing", Agrawal and Horgan, PLDI'90

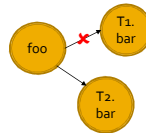
## Program Dependence Graph (PDG)

- Grafo de dependência para todo o programa
  - Baseada no flow graph do **programa**
    - Combina call graph e control-flow graph de funções
  - Fontes de imprecisão
    - Dynamic binding
    - Calling context

## Program Dependence Graph (PDG)

- Grafo de dependência por dependência
  - Baseada no flow graph de controle
  - Combina call graph e control flow graph
- Fontes de imprecisão
  - ➔ Dynamic binding
  - Calling context

```
foo(T x) {
  ...
  t.bar();
}
```

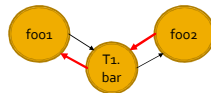


## Program Dependence Graph (PDG)

- Grafo de dependência por dependência
  - Baseada no flow graph de controle
  - Combina call graph e control flow graph
- Fontes de imprecisão
  - Dynamic binding
  - ➔ Calling context

```
foo1() {...
  (new T1()).bar();
}
```

```
foo2() {...
  (new T1()).bar();
}
```

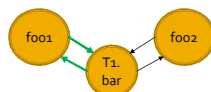


## Program Dependence Graph (PDG)

- Grafo de dependência por dependência
  - Baseada no flow graph de controle
  - Combina call graph e control flow graph
- Fontes de imprecisão
  - Dynamic binding
  - ➔ Calling context

```
foo1() {...
  (new T1()).bar();
}
```

```
foo2() {...
  (new T1()).bar();
}
```



## Como obter grafo de dependência

- Dentro de procedimentos
  - Reachable Definitions [Aho *et al.*, 2006]
  - Soot: <http://www.sable.mcgill.ca/soot/>
- Através de procedimentos
  - Imprecisão amplificada [Harrold and Soffa, TOPLAS 1996]
  - Indus: <http://indus.projects.cis.ksu.edu>

---

---

---

---

---

---

---

## Como obter grafo de dependência

- Dentro de procedimentos
  - Reachable Definitions [Aho *et al.*, 2006]
  - Soot: <http://www.sable.mcgill.ca/soot/>
- Através de procedimentos
  - Imprecisão amplificada [Harrold and Soffa, TOPLAS 1996]
  - Indus: <http://indus.projects.cis.ksu.edu>



Uma alternativa é usar probabilistic PDGs!  
[Baah *et al.*, TSE 2010]

---

---

---

---

---

---

---

## Slicing de programas

- O que é?
  - Técnica para filtrar linhas relevantes do program de acordo com algum **critério de pesquisa**

Programa




---

---

---

---

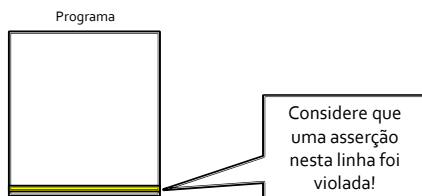
---

---

---

## Slicing de programas

- O que é?
  - Técnica para filtrar linhas relevantes do program de acordo com algum **critério de pesquisa**




---

---

---

---

---

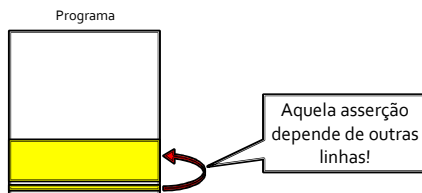
---

---

---

## Slicing de programas

- O que é?
  - Técnica para filtrar linhas relevantes do program de acordo com algum **critério de pesquisa**




---

---

---

---

---

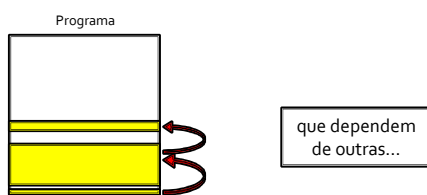
---

---

---

## Slicing de programas

- O que é?
  - Técnica para filtrar linhas relevantes do program de acordo com algum **critério de pesquisa**




---

---

---

---

---

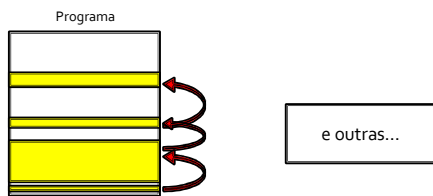
---

---

---

## Slicing de programas

- O que é?
  - Técnica para filtrar linhas relevantes do program de acordo com algum **critério de pesquisa**




---

---

---

---

---

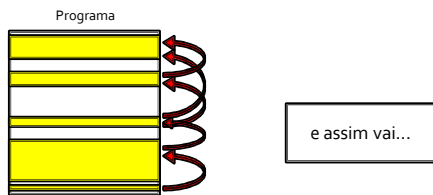
---

---

---

## Slicing de programas

- O que é?
  - Técnica para filtrar linhas relevantes do program de acordo com algum **critério de pesquisa**




---

---

---

---

---

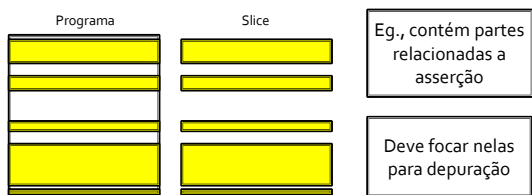
---

---

---

## Slicing de programas

- O que é?
  - Técnica para filtrar linhas relevantes do programa de acordo com algum **critério de pesquisa**




---

---

---

---

---

---

---

---

## Slicing estático (visão geral)

- Estático
  - Descobre dependências em **tempo de compilação**
  - Calcula o slice a partir do grafo de dependências
  - **Pode** levar em consideração apenas dependências alcançáveis a partir de uma função main

Problema : grafo de dependência é impreciso (inclui + ou – dependências que o ideal).

---

---

---

---

---

---

---

---

## Slicing dinâmico (visão geral)

- Dinâmico
  - Descobre dependências em **tempo de execução**
  - O slice é uma subgrafo do grafo estático de dep.
  - Leva em consideração uma função main

Problema : considera apenas função main.

---

---

---

---

---

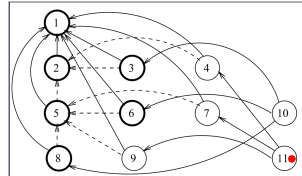
---

---

---

## Slicing estático

DEPENDENCE GRAPH



Valor de Z no ponto de interesse  
independe do valor de Y!

```

begin
S1:  read(X);
S2:  if (X < 0)
      then
S3:      Y := f1(X);
S4:      Z := g1(X);
      else
S5:      if (X = 0)
          then
S6:      Y := f2(X);
S7:      Z := g2(X);
          else
S8:      Y := f3(X);
S9:      Z := g3(X);
          end.if;
        end.if;
S10: write(Y);
S11: write(Z);
end.
  
```

From "Dynamic Program Slicing", Agrawal and Horgan, PLDI'90

---

---

---

---

---

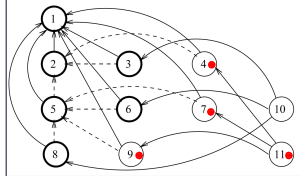
---

---

---

## Slicing estático

DEPENDENCE GRAPH



Valor de Z no ponto de interesse  
independe do valor de Y!

```

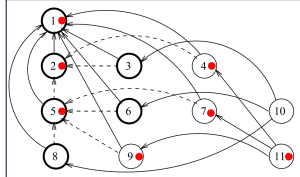
begin
S1:  read(X);
S2:  if (X < 0)
      then
S3:    Y := f1(X);
S4:    Z := g1(X);
      else
S5:    if (X = 0)
          then
S6:      Y := f2(X);
S7:      Z := g2(X);
          else
S8:      Y := f3(X);
S9:      Z := g3(X);
          end if;
        end if;
S10: write(Y);
S11: write(Z);
end.

```

From "Dynamic Program Slicing", Agrawal and Horgan, PLDI'90

## Slicing estático

DEPENDENCE GRAPH



Valor de Z no ponto de interesse  
independe do valor de Y!

```

begin
S1:  read(X);
S2:  if (X < 0)
      then
S3:    Y := f1(X);
S4:    Z := g1(X);
      else
S5:    if (X = 0)
          then
S6:      Y := f2(X);
S7:      Z := g2(X);
          else
S8:      Y := f3(X);
S9:      Z := g3(X);
          end if;
        end if;
S10: write(Y);
S11: write(Z);
end.

```

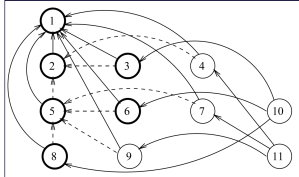
From "Dynamic Program Slicing", Agrawal and Horgan, PLDI'90

## Complexidade

- Imprecisão do grafo se acumula no tratamento do programa **inteiro**

No exemplo anterior,  
fizemos o slice  
de apenas uma  
função sem  
ponteiros!

DEPENDENCE GRAPH





## Complexidade

- Slices podem crescer muito rapidamente para programas arbitrários
  - Dynamic binding
  - Aliasing

[Binkley *et al.*, TOSEM 2007]

Independente disto, informação de dependência estática ainda pode ser útil para várias aplicações.

## Slicing Dinâmico

- Aplicação original é depuração automatizada de código



## Slicing Dinâmico

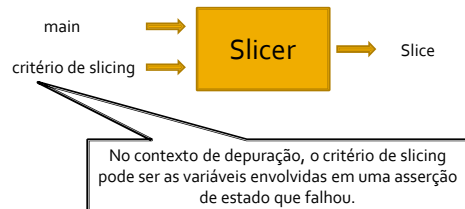
- Aplicação original é depuração automatizada de código



No contexto de depuração, a função main é o teste que falhou. O usuário usa o slicer para tentar descobrir a causa da falha.

## Slicing Dinâmico

- Aplicação original é depuração automatizada de código




---

---

---

---

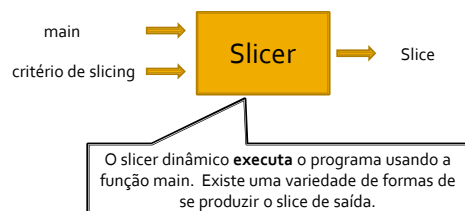
---

---

---

## Slicing Dinâmico

- Aplicação original é depuração automatizada de código




---

---

---

---

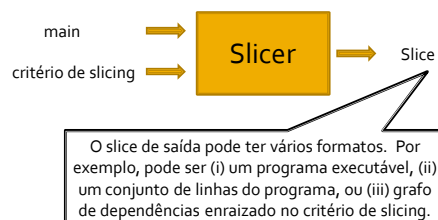
---

---

---

## Slicing Dinâmico

- Aplicação original é depuração automatizada de código




---

---

---

---

---

---

---

## Backward e Forward Slicing

- **Backward** (post-mortem): produz slice a partir do trace de instruções gerados na execução
- **Forward**: produz slice durante a execução
  - Cada valor associa-se a um conjunto
  - Conjunto inclui linhas de código
    - Representa fecho transitivo de dependências enraizadas naquele valor

---

---

---

---

---

---

---

---

## Tipos de slice

- Dados
- Dados + Controle
- Relevante

Também conhecido por Slice Full ou Executável

---

---

---

---

---

---

---

---

## Tipos de slice

- Dados
- Dados + Controle
- Relevante

```
foo() {
  x = 1;
  y = 0; // TOFIX: 1
  z = -1;
  if (y > 0) {
    z = 1;
  }
  if (z < 0) {
    z = z + 1;
  }
  x assert(z > 0);
}
```

---

---

---

---

---

---

---

---

## Tipos de slice

- Dados
- Dados + Controle
- Relevante

```
foo() {
  x = 1;
  y = 1; // FIXED
  z = -1;
  if (y > 0) {
    z = 1;
  }
  if (z < 0) {
    z = z + 1;
  }
  assert(z > 0);
}
```

## Tipos de slice

- Dados
- Dados + Controle
- Relevante

Erro de omissão!

```
foo() {
  x = 1;
  y = 0; // TOFIX: 1
  z = -1;
  if (y > 0) {
    z = 1;
  }
  if (z < 0) {
    z = z + 1;
  }
  assert(z > 0);
}
```

## Tipos de slice

- Dados
- Dados + Controle
- Relevante

Neste caso, o slice de dados não ajuda!

```
foo() {
  x = 1;
  y = 0; // TOFIX: 1
  z = -1;
  if (y > 0) {
    z = 1;
  }
  if (z < 0) {
    z = z + 1;
  }
  assert(z > 0);
}
```

## Tipos de slice

- Dados
- Dados + Controle
- Relevante

O slice completo também não.

```
foo() {
  x = 1;
  y = 0; // TOFIX: 1
  z = -1;
  if (y > 0) {
    z = 1;
  }
  if (z < 0) {
    z = z + 1;
  }
  assert(z > 0);
}
```

## Tipos de slice

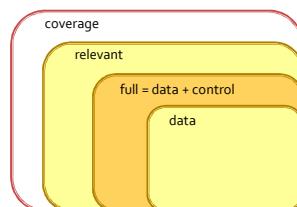
- Dados
- Dados + Controle
- Relevante

O relevante ajuda!  
Na verdade, é suficiente.

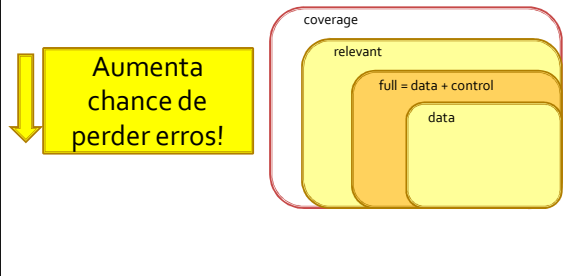
```
foo() {
  x = 1;
  y = 0; // TOFIX: 1
  z = -1;
  if (y > 0) {
    z = 1;
  }
  if (z < 0) {
    z = z + 1;
  }
  assert(z > 0);
}
```

## Precisão e Custo

Tamanho das caixas denotam tamanho **relativo** dos slices



### Precisão e Custo



---

---

---

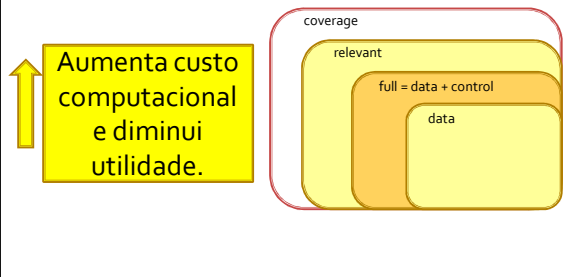
---

---

---

---

### Precisão e Custo



---

---

---

---

---

---

---

### Detalhes importantes

[Zhang *et al.*, AADBUG 2005]

- Apenas o slicer relevant slicer é completo
- Slicers relevant e completo requerem informação estática
- Slice de dados são bem menores que outros mas freqüentemente perdem erros

---

---

---

---

---

---

---

## Detalhes importantes

[Zhang *et al.*, AADBUG 2005]

- Apenas o slicer relevant slicer é completo
- Slicers relevant e completo requerem informação estática
- Slice de dados são bem menores que outros mas frequentemente perdem erros

Isto é crítico para depuração. A ausência de linhas não é tão crítica em outras aplicações. Por exemplo, compreensão de código. (mais adiante)

---

---

---

---

---

---

---

---

## Implementações existentes

- Várias implementações para C
- Para Java
  - JSlicer (<http://jslice.sourceforge.net>)
  - Extensão para Java PathFinder
    - Usada no nosso grupo (mas ainda não publicada)

---

---

---

---

---

---

---

---

## Agenda

- Parte 1: Conceitos
- Parte 2: Demo
- Parte 3: Aplicações

Demonstração de uma implementação de slicing dinâmico usando a infraestrutura do Java PathFinder.

---

---

---

---

---

---

---

---

## Agenda

- Parte 1: Conceitos
- Parte 2: Demo
- Parte 3: Aplicações

---

---

---

---

---

---

---

## Agenda

- Parte 1: Conceitos
- Parte 2: Demo
- Parte 3: Aplicações
  - ▢ Localização de Falhas
  - ▢ Compreensão de Linhas de Produto de Software



Elton Alves, mestrando  
Sabrina Souto, doutoranda

---

---

---

---

---

---

---

## Localização de Falhas

Problema:

Dado um teste falho (ou um conjunto deles), informar linhas do código suspeitas de causarem o erro.

Falha vs. Falta (ou defeito)

Ver estatísticas em livros como "CodeComplete" e "BeautifulTesting".

---

---

---

---

---

---

---



## Tarantula [Jones et al., ICSE 2002]

```

1  class BST {
2      Node root; int size;
3      static class Node {
4          int value; Node left, right;
5          Node(int value) {
6              this.value = value;
7          }
8      }
9      void insert(int value) {
10         if (root == null) {
11             root = new Node(value);
12             size++;
13             return;
14         }
15         Node current = root;
16         while (true) {
17             if (value < current.value) {
18                 if (current.left == null) {
19                     current.left = new Node(value);
20                     break;
21                 }
22                 current = current.left;
23             }
24             else {
25                 if (value >= current.value) {
26                     if (current.right == null) {
27                         current.right = new Node(value);
28                         break;
29                     }
30                     current = current.right;
31                 }
32                 else { return; }
33             }
34             size++;
35         }
36     }
37 }

```

Stmt	Test Cases					Stops	Ranks			
	5	4,3,2	6,7,8	3,3	3,3		R00	R1	R2	R3
6	•	•	•	•	•	0.500	13	10	-	-
10	•	•	•	•	•	0.500	13	10	-	-
11	•	•	•	•	•	0.500	13	10	-	-
12	•	•	•	•	•	0.500	13	10	-	-
13	•	•	•	•	•	0.500	13	-	-	-
14	•	•	•	•	•	0.577	8	6	-	-
15	•	•	•	•	•	0.577	8	6	-	-
16	•	•	•	•	•	0.577	8	-	-	-
17	•	•	•	•	•	0.000	20	-	-	-
18	•	•	•	•	•	0.000	20	-	-	-
19	•	•	•	•	•	0.000	20	-	-	-
20	•	•	•	•	•	0.000	20	-	-	-
21	•	•	•	•	•	0.000	20	-	-	-
23	•	•	•	•	•	0.707	4	3	3	1
24	•	•	•	•	•	0.707	4	-	-	-
25	•	•	•	•	•	0.707	4	-	-	-
26	•	•	•	•	•	0.707	4	3	3	-
27	•	•	•	•	•	0.000	20	-	-	-
28	•	•	•	•	•	0.000	20	-	-	-
32	•	•	•	•	•	0.577	8	6	4	-

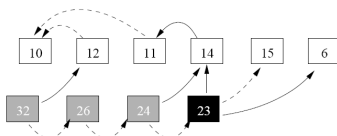
$$susp(s) = \frac{failed(s)}{\sqrt{totfailed * (failed(s) + passed(s))}}$$

## Slice de bst.size

```

BST bst = new BST(); bst.insert(3);
bst.insert(3); assert(bst.size==1); ✗

```



## Filtragem

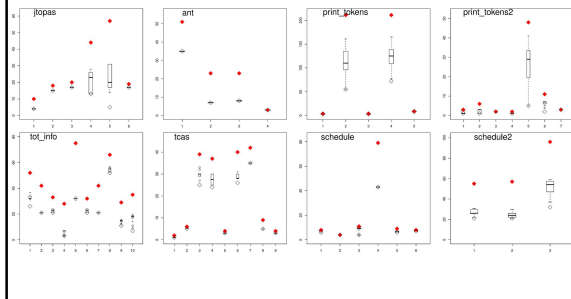
```

1  class BST {
2      Node root; int size;
3      static class Node {
4          int value; Node left, right;
5          Node(int value) {
6              this.value = value;
7          }
8      }
9      void insert(int value) {
10         if (root == null) {
11             root = new Node(value);
12             size++;
13             return;
14         }
15         Node current = root;
16         while (true) {
17             if (value < current.value) {
18                 if (current.left == null) {
19                     current.left = new Node(value);
20                     break;
21                 }
22                 current = current.left;
23             }
24             else {
25                 if (value >= current.value) {
26                     if (current.right == null) {
27                         current.right = new Node(value);
28                         break;
29                     }
30                     current = current.right;
31                 }
32                 else { return; }
33             }
34             size++;
35         }
36     }
37 }

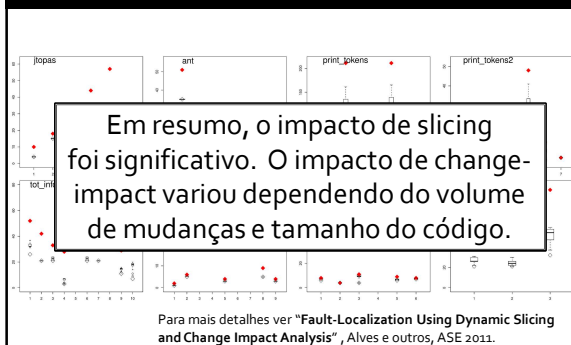
```

Stmt	Test Cases					Stops	Ranks			
	5	4,3,2	6,7,8	3,3	3,3		R00	R1	R2	R3
6	•	•	•	•	•	0.500	13	10	-	-
10	•	•	•	•	•	0.500	13	10	-	-
11	•	•	•	•	•	0.500	13	10	-	-
12	•	•	•	•	•	0.500	13	10	-	-
13	•	•	•	•	•	0.500	13	-	-	-
14	•	•	•	•	•	0.577	8	6	-	-
15	•	•	•	•	•	0.577	8	6	-	-
16	•	•	•	•	•	0.577	8	-	-	-
17	•	•	•	•	•	0.000	20	-	-	-
18	•	•	•	•	•	0.000	20	-	-	-
19	•	•	•	•	•	0.000	20	-	-	-
20	•	•	•	•	•	0.000	20	-	-	-
21	•	•	•	•	•	0.000	20	-	-	-
23	•	•	•	•	•	0.707	4	3	3	1
24	•	•	•	•	•	0.707	4	-	-	-
25	•	•	•	•	•	0.707	4	-	-	-
26	•	•	•	•	•	0.707	4	3	3	-
27	•	•	•	•	•	0.000	20	-	-	-
28	•	•	•	•	•	0.000	20	-	-	-
32	•	•	•	•	•	0.577	8	6	4	-

## Resultados



## Resultados



## Compreensão de LPS

- LPS é uma forma de se escrever código onde **variação** é uma abstração central
  - Um **produto** implementa uma seleção de **features**
  - Impacto
    - Metodologia adotada em alguns domínios
      - Exemplo: jogos, kernel de S.O.
    - Interesse crescente em comunidades acadêmicas

## Compreensão de LPS

- Compreender como features se relacionam é um caminho para entender LPS

Problema 1:

Como extrair dependências entre features?

Problema 2:

Como fazer isto de forma eficiente?  
(explosão combinatorial de produtos)

---

---

---

---

---

---

---

---

## Compreensão de LPS

- Compreender c  
um caminho pa

**Dynamic Data Slicing**

Problema 1:

Como extrair  
features?

**Seleção de Teste**

Problema 2:

Como fazer isto de forma eficiente?  
(explosão combinatorial de produtos)

---

---

---

---

---

---

---

---

## Resumo

- Conceitos básicos
- Demo
- Duas aplicações da tecnologia



pan.cin.ufpe.br

Elton Alves  
Marcelo d'Amorim

[erma@cin.ufpe.br](mailto:erma@cin.ufpe.br)  
[damorim@cin.ufpe.br](mailto:damorim@cin.ufpe.br)

---

---

---

---

---

---

---

---

XXV Simpósio  
Brasileiro de  
Engenharia de  
Software

XV Simpósio  
Brasileiro de  
Linguagens de  
Programação

XIV Simpósio  
Brasileiro de  
Métodos  
Formais

V Simpósio  
Brasileiro de  
Componentes,  
Arquiteturas e  
Reutilização de  
Software

Promoção



Realização



EACH



Patrocínio



Ministério da  
Educação



Organização



[www.each.usp.br/cbsoft2011](http://www.each.usp.br/cbsoft2011)