Métricas de Código-Fonte

Paulo Meirelles
paulormm@ime.usp.br

```
  def custosAPartirDoVertice(vertice):
1       custos = novo Lista(numeroDeVertices)
2       fila = nova FilaDePrioridades(numeroDeVertices)

3       for i in (1, numeroDeVertices):
4           custos[i] = -1

5       custos[vertice] = 0
6       fila.insere(nova Aresta(0,0))

7       while(fila.vazia()):
8           verticeDoMomento = fila.verticeDaArestaComCustoMinimo()
9           for aresta in (arestasDoVertice(verticeDoMomento)):
10              verticeDestino = aresta.verticeDestino()
11              custo = aresta.custo()

12              if(custos[verticeDestino] == -1):
13                  custos[verticeDestino] = custos[verticeDoMomento] + custo
14                  fila.insere(nova Aresta(verticeDestino, custos[verticeDestino]))

15              else if(custos[verticeDestino] > custos[verticeDoMomento] + custo):
16                  custos[verticeDestino] = custos[verticeDoMomento] + custo

17      return custos
```

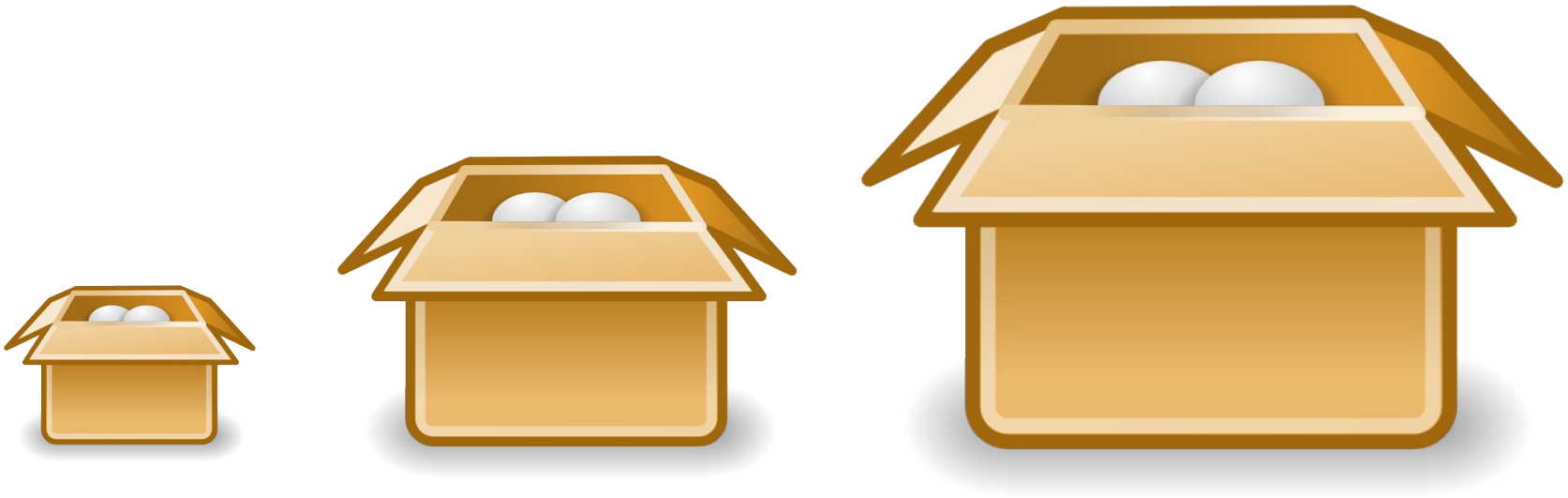
```
def custosAPartirDoVertice(vertice):
    inicializaCustos()
    inicializaFila()
    atualizaCustosAteAcabarVertices()
```

```
inicializaCustos(vertice):
 for i in (1, numeroDeVertices):
     custos[i] = -1
 custos[vertice] = 0
```

```
def inicializaFila(vertice):
    fila = nova FilaDePrioridades()
    fila.insere(nova Aresta(0,0))
```

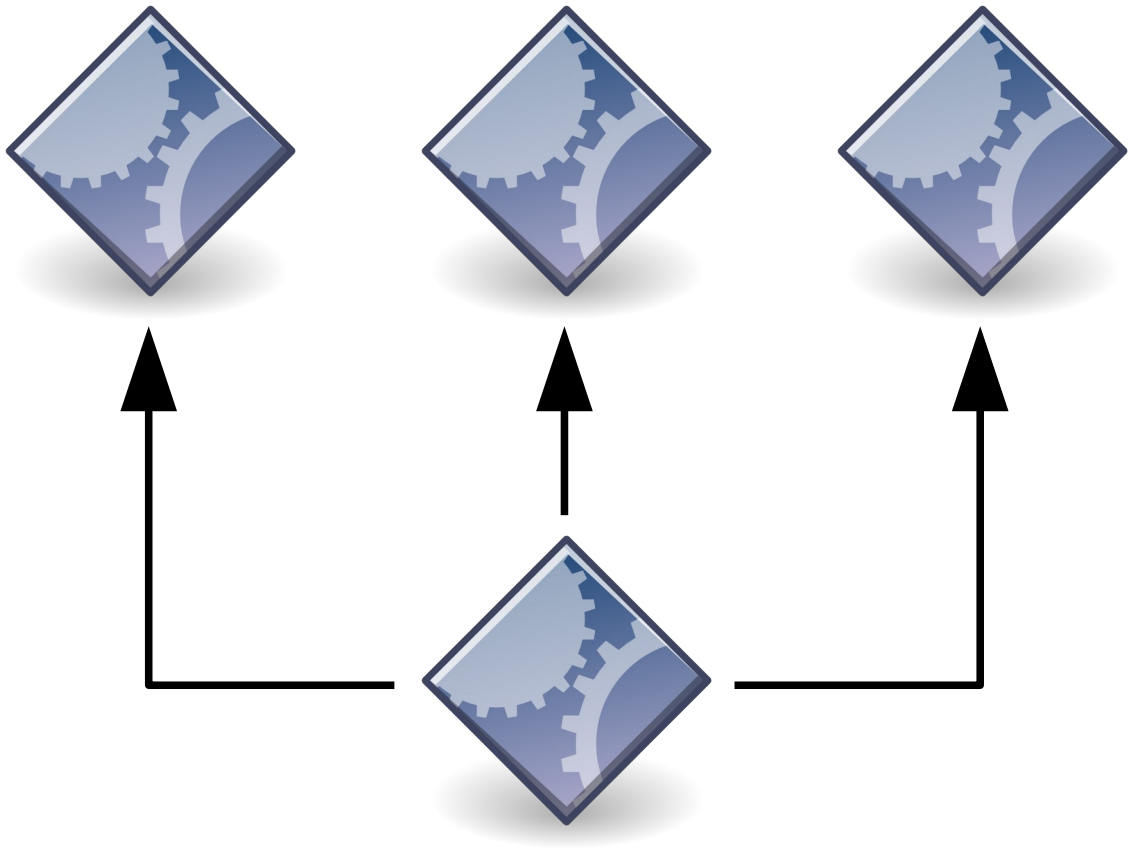
```
def atualizaCustosAteAcabarVertices():
    while(fila.vazia()):
        verticeDoMomento = fila.verticeDaArestaComCustoMinimo()
        atualizaCustosAPartirDe(vertice)
```

Tamanho



Coesão

Acoplamento



Métricas

Métricas de Software

Métricas de Software

Métricas de software têm como objetivo identificar, medir e consequentemente controlar os principais parâmetros que afetam o desenvolvimento de software ...

Métricas de Software

A necessidade do uso de métricas de software teve maior atenção quando se constatou a “crise do software”, devido à ineficácia da gerência do desenvolvimento da maior parte das soluções de software...

Métricas de Software

Métricas são utilizadas para estimar um cronograma e custos de desenvolvimento do software e para medir a produtividade e a qualidade do produto.

Métricas

Métricas de Código-Fonte

Métricas de Código-Fonte

- Métricas objetivas que tratam características (atributos) do código-fonte
  - Existem (propostas) dezenas
    - ~30 para medir a complexidade
    - ~70 no contexto da orientação a objetos

Métricas de Código-Fonte

- Métricas de tamanho
  - quantificar o tamanho e auxiliar as medições na fase de concepção do software
  - seguem o paradigma de desenvolvimento de software tradicional

Métricas de Código-Fonte

- Métricas de tamanho
  - Linhas de Código (LOC)
  - Pontos de Função (FP)

Métricas de Código-Fonte

- Métricas de complexidade
  - formas objetivas de medir a complexidade de um pedaço de software
  - alta complexidade deve ser evitada pois prejudica a compreensão do código e o torna mais suscetível a erros
  - Softwares grandes são mais suscetíveis a ter alta complexidade
  - Recomenda-se usá-las com as métricas de tamanho

Métricas de Código-Fonte

- Métricas de complexidade
  - Complexidade Ciclomática
    - v(G) ou McCabe
  - MaxNesting
  - Número de Nós
  - Fluxo de Informação

Métricas de Código-Fonte

- Métricas de Orientação a Objetos
  - orientação a objetos usa entidades e não algoritmos como componentes fundamentais
  - métricas de código-fonte para programas orientados a objetos deve ser diferente
  - Além de tamanho e complexidade, por exemplo, em um software OO é possível medir o uso da herança e o grau de interdependência entre as entidades

Métricas de Código-Fonte

- Métricas de Orientação a Objetos
  - Respostas para uma Classe (RFC)
  - Falta de Coesão entre Métodos (LCOM)
  - Acoplamento entre Objetos (CBO)
  - Número de Parâmetros por Método (NPM)
  - Número de Atributos Públicos (NPA)
  - Profundidade da Árvore de Herança (DIT)
  - Número de Filhos (NOC)

Métricas de Código-Fonte

Métricas usadas no contexto de sistemas de software orientados a objeto

```
1  public class HelloWorld {
2          // This comment will not count
3          public static void main(String args[]){
4                  Printer printer = new HelloWorldPrinter();printer.print();
5          }
6  }
7
8  class Printer {
9          private String message;
10
11         public Printer(String msg){
12                 message = msg;
13         }
14
15         public void print(){
16                 System.out.println(message);
17         }
18 }
19
20 class HelloWorldPrinter extends Printer {
21         public HelloWorldPrinter(){
22                 super("Hello World!");
23         }
24
25         public void doNothing(){
26                 // a method
27         }
28 }
```

Linhas de Código (LOC)

É a medida mais comum para o tamanho de um software. São contadas apenas as linhas executáveis, ou seja, são excluídas linhas em branco e comentários ...

Linhas de Código (LOC)

No exemplo, a classe HelloWorld tem LOC=3, contando as linhas 1, 3 e 4. A linha 4, apesar de possuir 2 instruções, conta como apenas 1 linha ...

Linhas de Código (LOC)

Para efetuar comparações entre sistemas usando LOC, é necessário que ambos tenham sido feitos na mesma linguagem de programação e que o estilo esteja normalizado …

Linhas de Código (LOC)

Os intervalos sugeridos para o LOC de uma classe são: até 70 (bom); entre 70 e 130 (regular); de 130 em diante (ruim).

```
1  public class HelloWorld {
2          // This comment will not count
3          public static void main(String args[]){
4                  Printer printer = new HelloWorldPrinter();printer.print();
5          }
6  }
7
8  class Printer {
9          private String message;
10
11         public Printer(String msg){
12                 message = msg;
13         }
14
15         public void print(){
16                 System.out.println(message);
17         }
18 }
19
20 class HelloWorldPrinter extends Printer {
21         public HelloWorldPrinter(){
22                 super("Hello World!");
23         }
24
25         public void doNothing(){
26                 // a method
27         }
28 }
```

AMLOC
Número médio de linhas por método

Indica se o código está bem distribuído entre os métodos. Quanto maior, mais pesados são os métodos. É preferível ter muitas operações pequenas e de fácil entendimento que poucas operações grandes e complexas ...

AMLOC
Número médio de linhas por método

No exemplo, a classe Printer tem AMLOC=3, pois possui 2 métodos e LOC= 6. Os intervalos sugeridos são: até 10 (bom); entre 10 e 13 (regular); de 13 em diante (ruim).

DIT
Profundidade da árvore de herança

É o número de superclasses e/ou classes ancestrais da classe sendo analisada. São contadas apenas as superclasses do sistema, ou seja, as classes de bibliotecas não são contabilizadas. No exemplo, a classe HelloWorldPrinter tem DIT = 1 e nas demais DIT = 0 ...

DIT
Profundidade da árvore de herança

Quanto maior for o valor DIT, maior é o número de atributos e métodos herdados, e portanto maior é a complexidade. Os intervalos sugeridos são: até 2 (bom); entre 2 e 4 (regular); de 4 em diante (ruim).

```
1  public class HelloWorld {
2          // This comment will not count
3          public static void main(String args[]){
4                  Printer printer = new HelloWorldPrinter();printer.print();
5          }
6  }
7
8  class Printer {
9          private String message;
10
11         public Printer(String msg){
12                 message = msg;
13         }
14
15         public void print(){
16                 System.out.println(message);
17         }
18 }
19
20 class HelloWorldPrinter extends Printer {
21         public HelloWorldPrinter(){
22                 super("Hello World!");
23         }
24
25         public void doNothing(){
26                 // a method
27         }
28 }
```

NPA
Número de atributos públicos

Os atributos de uma classe devem servir apenas às funcionalidades da própria classe. Portanto, as variáveis/atributos de classe devem ser ocultadas para evitar complexidade, pois fica difícil prever os efeitos colaterais de alterar atributos públicos ...

NPA
Número de atributos públicos

NPA mede o encapsulamento. O valor ideal dessa métrica é zero. Os intervalos sugeridos são: até 1 (bom); entre 1 e 9 (regular); de 9 em diante (ruim).

NPM
Número de métodos públicos

Representa o tamanho da interface da classe. Os métodos públicos representam os serviços que a classe disponibiliza ...

NPM
Número de métodos públicos

Valores altos para essa métrica indicam que a classe tem demasiadas funcionalidades e que poderia ser quebrada. Os intervalos sugeridos são: até 10 (bom); entre 10 e 40 (regular); de 40 em diante (ruim).

NP
Número de Parâmetros

Calcula o número de parâmetros de um método. Obtemos valor zero quando o método avaliado não possui parâmetro. Muitos parâmetros de um método pode indicar que ele está com mais de uma responsabilidade ...

LCOM4
Ausência de coesão em métodos

Seja M = {M1, …, Mn} o conjunto dos métodos da classe analisada. Dois métodos Mi e Mj estão relacionados se ambos acessam pelo menos um mesmo atributo da classe, ou se Mi chama ou é chamado por Mj . LCOM4 é a quantidade de partições de "M" formadas após separar os métodos em conjuntos de métodos relacionados ...

LCOM4
Ausência de coesão em métodos

No exemplo, a classe Printer tem LCOM4 = 1, pois todos os métodos acessam a variável message. HelloWorldPrinter tem LCOM4 = 2, pois o contrutor e o método doNothing() não estão relacionados ...

LCOM4
Ausência de coesão em métodos

Se uma classe tem diferentes conjuntos de métodos não relacionados entre si, é um indício de que a classe deveria ser quebrada em classes menores e mais coesas.
Os intervalos sugeridos são: até 2 (bom); entre 2 e 5 (regular); de 5 em diante (ruim).

```
1  public class HelloWorld {
2          // This comment will not count
3          public static void main(String args[]){
4                  Printer printer = new HelloWorldPrinter();printer.print();
5          }
6  }
7
8  class Printer {
9          private String message;
10
11         public Printer(String msg){
12                 message = msg;
13         }
14
15         public void print(){
16                 System.out.println(message);
17         }
18 }
19
20 class HelloWorldPrinter extends Printer {
21         public HelloWorldPrinter(){
22                 super("Hello World!");
23         }
24
25         public void doNothing(){
26                 // a method
27         }
28 }
```

```
classe Pilha
  int maxPosicoes
  int topo
  vetor[maxPosicoes] elementos

  vazia?():
    return topo == 0

  cheia?():
    return topo == maxPosicoes

  insere(Elemento elemento):
    levanta_excecao("Pilha cheia") if cheia?
    elementos[topo] = elemento
    topo += 1

  remove_topo():
    levanta_excecao("Pilha vazia") if vazia?
    topo -= 1
    elementos[topo]
```

CBO

Acoplamento (ligações, conexões) entre Objetos (classes)

Mede o acoplamento (conectividade) de uma classe. Se uma classe C1 acessa um método ou atributo da classe C2, dizemos que C1 é cliente da classe fornecedora C2...

CBO

Acoplamento (ligações, conexões) entre Objetos (classes)

$$cliente(C_i, C_j) = \begin{cases} 1 & \text{se } C_i \Rightarrow C_j \wedge C_i \neq C_j \\ 0 & \text{caso contrário} \end{cases}$$

$$\sum_{i=1}^{n} cliente(C_i, C_j)$$

- *n = número total de classes*

CBO

Acoplamento (ligações, conexões) entre Objetos (classes)

No nosso exemplo, CBO(Printer) é 2, pois ela é utilizada pelas outras duas classes. *HelloWorldPrinter* é filha e portanto cliente de *Printer* ...

CBO

Acoplamento (ligações, conexões) entre Objetos (classes)

Se o valor dessa métrica for grande, uma mudança na classe tem potencialmente mais efeitos colaterais, tornando mais difícil a manutenção. Os intervalos sugeridos são: até 2 (bom); entre 2 e 20 (regular); de 20 em diante (ruim).

```
1  public class HelloWorld {
2          // This comment will not count
3          public static void main(String args[]){
4                  Printer printer = new HelloWorldPrinter();printer.print();
5          }
6  }
7
8  class Printer {
9          private String message;
10
11         public Printer(String msg){
12                 message = msg;
13         }
14
15         public void print(){
16                 System.out.println(message);
17         }
18 }
19
20 class HelloWorldPrinter extends Printer {
21         public HelloWorldPrinter(){
22                 super("Hello World!");
23         }
24
25         public void doNothing(){
26                 // a method
27         }
28 }
```

MaxNesting
Nível Máximo de Estruturas Encadeadas

Calcula o nível máximo de estruturas encadeados presentes no corpo de um método. Seu valor varia entre zero e a quantidade total de quebras de fluxo do método ...

MaxNesting
Nível Máximo de Estruturas Encadeadas

Obtemos zero quando o método não possui controladores de fluxo (if, for, while etc). Atingimos o valor máximo quando todas as quebras condicionais presentes no método encontram-se em níveis diferentes da mesma estrutura.

CYCLO (McCabe)
Complexidade Ciclomática

Calcula o número de caminhos linearmente independentes no método analisado, conhecido como complexidade ciclomática ...

CYCLO (McCabe)
Complexidade Ciclomática

Obtemos 1 quando não há quebra do fluxo principal. Cada estrutura de controle de fluxo presente no corpo do método adiciona 1 no valor total da CYCLO ...

CYCLO (McCabe)
Complexidade Ciclomática

Quando os valores de MaxNesting e CYCLO são próximos, sabemos que muitos dos controladores de fluxo presentes no método estão encadeados em uma mesma estrutura.
Quando são muito distantes, podemos concluir que os controladores estão espalhados em estruturas diferentes e que elas não são muito profundas.

```
  def custosAPartirDoVertice(vertice):
1       custos = novo Lista(numeroDeVertices)
2       fila = nova FilaDePrioridades(numeroDeVertices)

3       for i in (1, numeroDeVertices):
4           custos[i] = -1

5       custos[vertice] = 0
6       fila.insere(nova Aresta(0,0))

7       while(fila.vazia()):
8           verticeDoMomento = fila.verticeDaArestaComCustoMinimo()
9           for aresta in (arestasDoVertice(verticeDoMomento)):
10              verticeDestino = aresta.verticeDestino()
11              custo = aresta.custo()

12              if(custos[verticeDestino] == -1):
13                  custos[verticeDestino] = custos[verticeDoMomento] + custo
14                  fila.insere(nova Aresta(verticeDestino, custos[verticeDestino]))

15              else if(custos[verticeDestino] > custos[verticeDoMomento] + custo):
16                  custos[verticeDestino] = custos[verticeDoMomento] + custo

17      return custos
```

```
custosAPartirDoVertice(vertice):
 custos = novo Lista(numeroDeVertices)
 fila = nova FilaDePrioridades(numeroDeVertices)

 for i in (1, numeroDeVertices):
      custos[i] = -1

 custos[vertice] = 0
 fila.insere(nova Aresta(0,0))

 while(fila.vazia()):
      verticeDoMomento = fila.verticeDaArestaComCustoMinimo()
      for aresta in (arestasDoVertice(verticeDoMomento)):
           verticeDestino = aresta.verticeDestino()
           custo = aresta.custo()

           if(custos[verticeDestino] == -1):
                custos[verticeDestino] = custos[verticeDoMomento] + custo
                fila.insere(nova Aresta(verticeDestino, custos[verticeDestino]))

           else if(custos[verticeDestino] > custos[verticeDoMomento] + custo):
                custos[verticeDestino] = custos[verticeDoMomento] + custo

 return custos
```

Alto LOC

Alta CYCLO

Alto MaxNesting

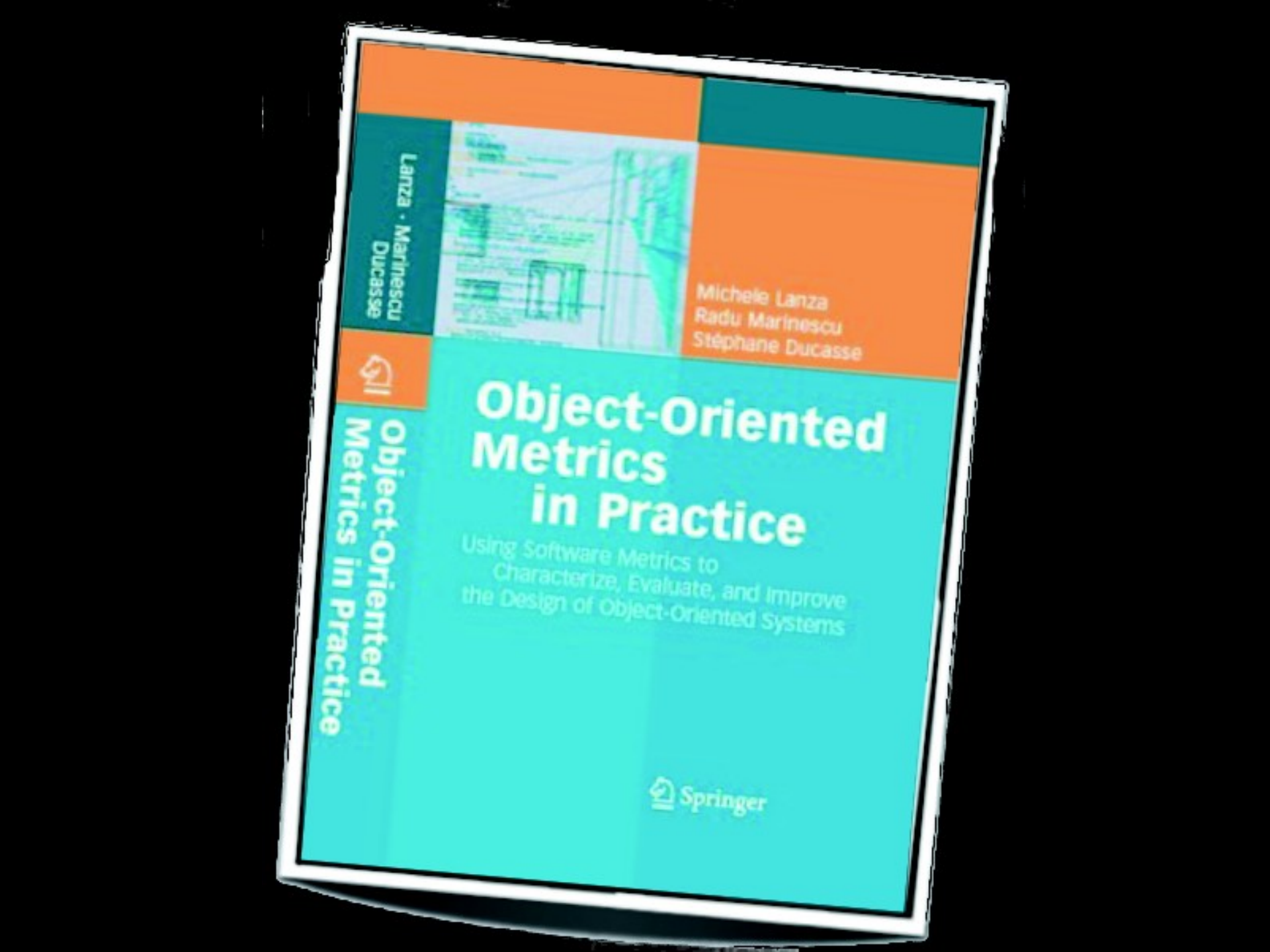
```
def custosAPartirDoVertice(vertice):
    inicializaCustos()
    inicializaFila()
    atualizaCustosAteAcabarVertices()
```

```
inicializaCustos(vertice):
 for i in (1, numeroDeVertices):
     custos[i] = -1
 custos[vertice] = 0
```

```
def inicializaFila(vertice):
    fila = nova FilaDePrioridades()
    fila.insere(nova Aresta(0,0))
```

```
def atualizaCustosAteAcabarVertices():
    while(fila.vazia()):
        verticeDoMomento = fila.verticeDaArestaComCustoMinimo()
        atualizaCustosAPartirDe(vertice)
```


PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Implementation Patterns
Kent Beck


Lanza · Marinescu
Ducasse
Object-Oriented
Metrics in Practice
Michele Lanza
Radu Marinescu
Stéphane Ducasse
Object-Oriented
Metrics
in Practice
Using Software Metrics to
Characterize, Evaluate, and Improve
the Design of Object-Oriented Systems
Springer

Métricas de Código-Fonte

Paulo Meirelles
paulormm@ime.usp.br