

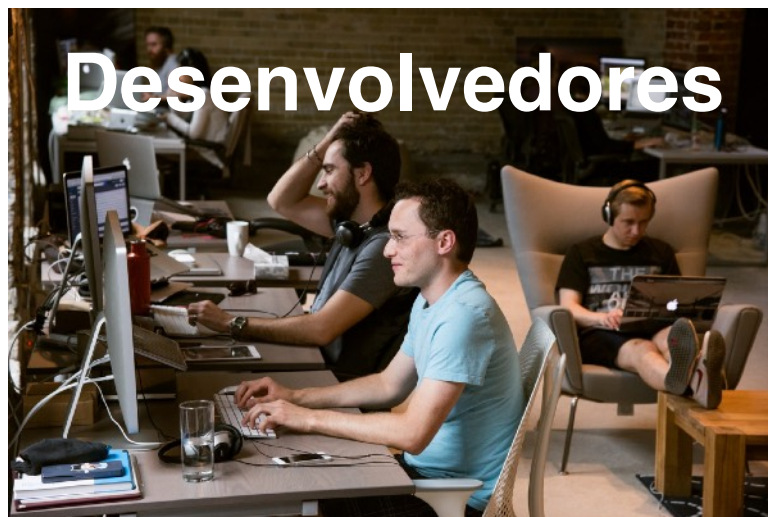
Engenharia de Software II

Qualidade, Métricas estáticas e dinâmicas,
Métricas de produto

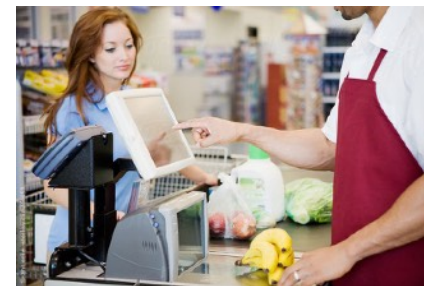
Prof. André Hora
DCC/UFMG
2019.1

Qualidade de Software

Conjunto de características a serem satisfeitas em um determinado grau, de modo que o software satisfaça às necessidades de seus **stakeholders**



Usuários finais



Qualidade de Software

- Conceitos fundamentais:
 - **Qualidade de produto**
 - Qualidade de processo
 - Teste de software

Agenda

1. Visão geral

2. Atributos de qualidade

3. Avaliação de qualidade

4. Métricas de produto

- Básicas
- Métricas OO

Qualidade de Produto

- Produto de qualidade:
 - Satisfaz as necessidades dos usuários
 - Produzido dentro dos limites de tempo e orçamento
- Qualidade de **produto** de software é fortemente afetada pela qualidade de **processo** de software
- Outro fator: realização de testes
 - Verifica se todos os requisitos foram implementados e encontra defeitos antes do produto ser utilizado

Conceito Subjetivo

- Código é legível?
- Padrões de programação são seguidos?
- Testes são adequados?
- Uso aceitável por usuário comum?

Agenda

1. Visão geral

2. Atributos de qualidade

3. Avaliação de qualidade

4. Métricas de produto

- Básicas
- Métricas OO

Atributos de Qualidade

- McCall define vários fatores relacionados a qualidade, em três visões:
 - **Operação:** utilização do produto (corretude, compatibilidade, eficiência, integridade e usabilidade)
 - **Revisão:** poder de mudança do produto (manutenabilidade, flexibilidade e testabilidade)
 - **Transição:** funcionamento em ambientes diferentes (portabilidade, interoperabilidade e reusabilidade)

Norma ISO 25010

- Atributos de qualidade da ISO 25010:
 - Funcionalidade
 - Usabilidade
 - Confiabilidade
 - Desempenho
 - Portabilidade
 - Manutenibilidade
 - Segurança
 - Compatibilidade

Norma ISO 25010

- **Confiabilidade:** avalia se ao longo do tempo o produto mantém um comportamento consistente
 - **Maturidade:** frequência com que o software apresenta defeitos
 - **Disponibilidade:** grau que software está disponível quando requisitado
 - **Tolerância a falhas:** como software reage na presença de falhas
 - **Recuperabilidade:** capacidade do software recuperar dados e re-estabelecer a situação normal após um desastre

Norma ISO 25010

- **Desempenho:** eficiência do uso de recursos
 - **Comportamento de tempo:** tempo que o sistema leva para processar suas tarefas
 - **Utilização de recursos:** espaço de armazenamento ou memória
 - **Capacidade:** limites máximos do produto

Norma ISO 25010

- **Portabilidade:** eficiência que o produto pode ser transferido para outro ambiente
- **Adaptabilidade:** facilidade de adaptar o software para outro ambiente
- **Instalabilidade:** facilidade de instalar o software
- **Substituibilidade:** grau que o sistema pode substituir outro

Norma ISO 25010

- **Usabilidade:** grau de satisfação do usuário com relação ao produto

?

Norma ISO 25010

- **Usabilidade:** grau de satisfação do usuário com relação ao produto
 - **Inteligibilidade:** facilidade para se entender o produto
 - **Apreensibilidade:** facilidade de aprendizado do sistema
 - **Operabilidade:** facilidade para se usar o produto
 - **Estética da interface gráfica:** grau que interface proporciona prazer ao usuário
 - **Acessibilidade:** grau que produto foi projetado para atender usuários com necessidade especiais

Norma ISO 25010

- **Segurança:** grau de proteção de acesso não autorizado e disponibilizado para acesso autorizado
- **Confidencialidade:** grau que dados são acessíveis para quem tem autorização
- **Integridade:** grau que dados são protegidos contra acessos por pessoas não autorizadas
- **Rastreabilidade:** grau que ações podem ser rastreadas para se comprovar quem fez a ação
- **Autenticidade:** grau que identidade do usuário pode ser provada

Norma ISO 25010

- **Manutenabilidade:** facilidade de se realizar alterações no sistema ou corrigir defeitos
- **Modularidade:** grau de modularidade de modo a minimizar o impacto das mudanças
- **Reusabilidade:** grau que partes do sistema podem ser reusadas
- **Testabilidade:** facilidade de se realizar testes de regressão nos sistema

Agenda

1. Visão geral
2. Atributos de qualidade
- 3. Avaliação de qualidade**
4. Métricas de produto
 - Básicas
 - Métricas OO

Modelo GQM

- GQM (Goal/Question/Metrics): abordagem para avaliar a qualidade de software
- Modelo de mensuração em três níveis:
 - **Conceitual** (goal/objetivos): definição dos objetivos
 - **Operacional** (question/questão): definição de um conjunto de questões para alcançar os objetivos
 - **Quantitativo** (metric/métrica): identificação de métricas que ajudam a responder as questões
- Ideia: derivar métricas a partir dos objetivos e questões

Modelo GQM

- **Objetivo:** o que se deseja melhorar
 - Ex: aumentar produtividade em X%
- **Questões:** refinamento do objetivo com questões
 - Ex: qual o número de linhas de código produzidas por desenvolvedor?
- **Métricas:** seleção de métricas apropriadas
 - Ex: LOC por desenvolvedor

Classes importantes

- **Objetivo:** identificar as classes responsáveis por 80% das modificações
- **Questões:**
 - Existem classes que dominam as modificações?
 - Como as modificações variam ao longo do tempo?
- **Métricas:**
 - Classes mais modificadas
 - Número total de modificações
 - Número de classes modificadas por mês

Devs chaves

- **Objetivo:** identificar os desenvolvedores responsáveis por 50% das modificações
- **Questões:**
 - Existem desenvolvedores que dominam as modificações?
 - Como as modificações variam ao longo do tempo?
- **Métricas:**
 - Desenvolvedores que mais realizam modificações
 - Número total de modificações
 - Número de classes modificadas por mês

G1: identificar as classes responsáveis por 80% das modificações

G2: identificar os desenvolvedores responsáveis por 50% das modificações

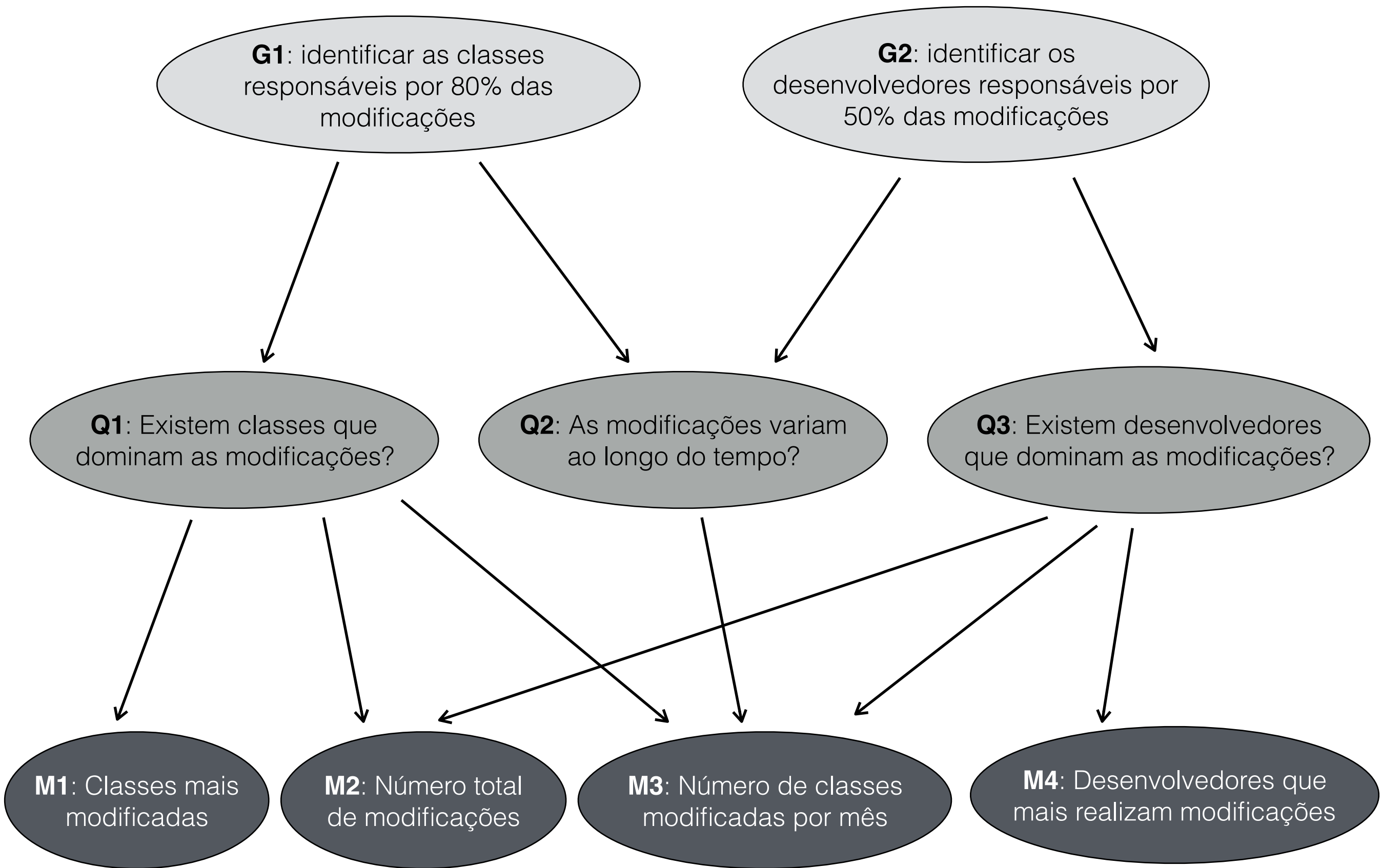
G1: identificar as classes responsáveis por 80% das modificações

G2: identificar os desenvolvedores responsáveis por 50% das modificações

Q1: Existem classes que dominam as modificações?

Q2: As modificações variam ao longo do tempo?

Q3: Existem desenvolvedores que dominam as modificações?



Agenda

1. Visão geral
2. Atributos de qualidade
3. Avaliação de qualidade
- 4. Métricas de produto**
 - Básicas
 - Métricas OO

Métricas de Produto

- Elemento chave para entender os **atributos de qualidade**
- Quantifica os atributos internos de um sistema
- Tipos: dinâmicas e estáticas
- Métricas básicas e OO

Tipos de Métricas

- **Métricas dinâmicas:** coletadas por medição durante a execução do sistema
 - Ex: tempo para completar uma determinada tarefa
 - Ajudam a avaliar a eficiência ou confiança de um sistema
 - Possui relacionamento direto com qualidade de produto
- **Métricas estáticas:** coletadas de artefatos do sistema tais como código e documentação
 - Ex: número de linhas de código, número de métodos
 - Possui relacionamento indireto com qualidade de produto
 - Sistema não é executado

Tipos de Métricas

- **Métricas dinâmicas:** coletadas por medição durante a execução do sistema
 - Ex: tempo para completar uma determinada tarefa
 - Ajudam a avaliar a eficiência ou confiança de um sistema
 - Possui relacionamento direto com qualidade de produto
- **Métricas estáticas:** coletadas de artefatos do sistema tais como código e documentação
 - Ex: número de linhas de código, número de métodos
 - Possui relacionamento indireto com qualidade de produto
 - Sistema não é executado

Agenda

1. Visão geral
2. Atributos de qualidade
3. Avaliação de qualidade
4. Métricas de produto
 - **Básicas**
 - Métricas OO

Métricas Básicas

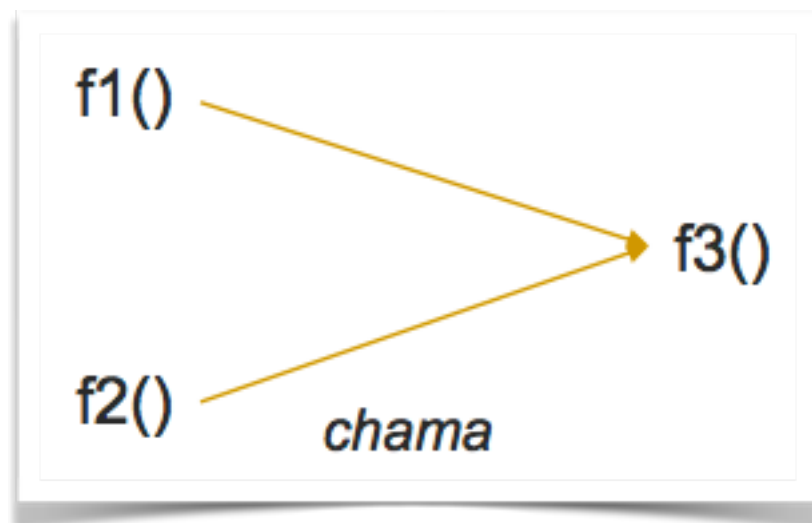
- Fan-in
- Fan-out
- Tamanho do Código
- Complexidade Ciclomática
- Profundidade de Aninhamento

Métricas Básicas

- **Fan-in**
- Fan-out
- Tamanho do Código
- Complexidade Ciclomática
- Profundidade de Aninhamento

Fan-in

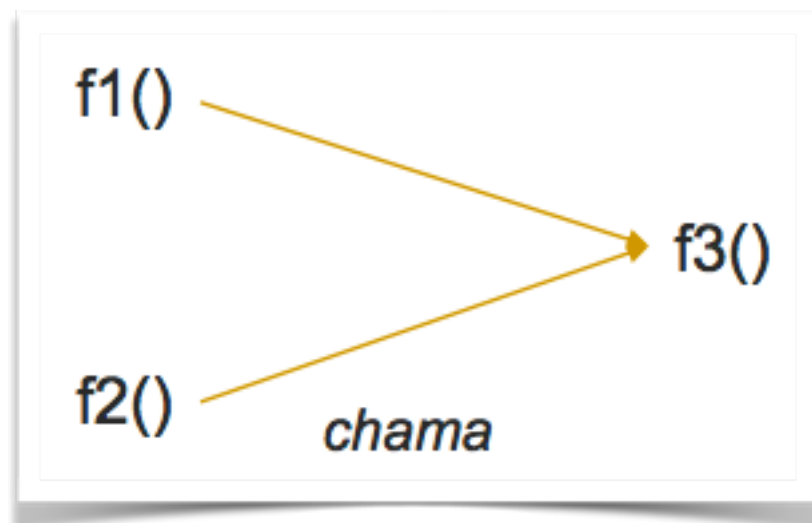
- Número de funções que chamam uma dada função
- Valor alto significa grande **impacto de mudanças** (propagação)



fan-in(f3) = ?

Fan-in

- Número de funções que chamam uma dada função
- Valor alto significa grande **impacto de mudanças** (propagação)



$$\text{fan-in}(f3) = 2$$

Android Ecosystem

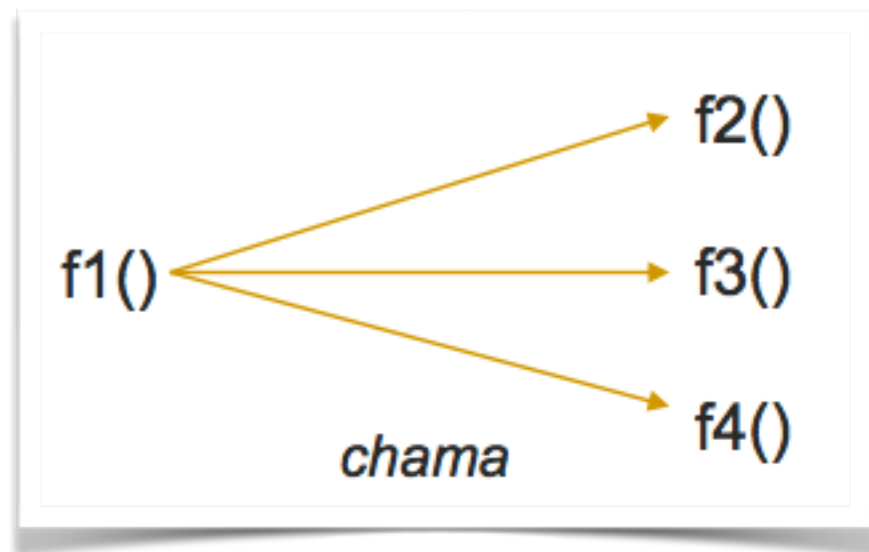


Impacto de mudanças



Fan-out

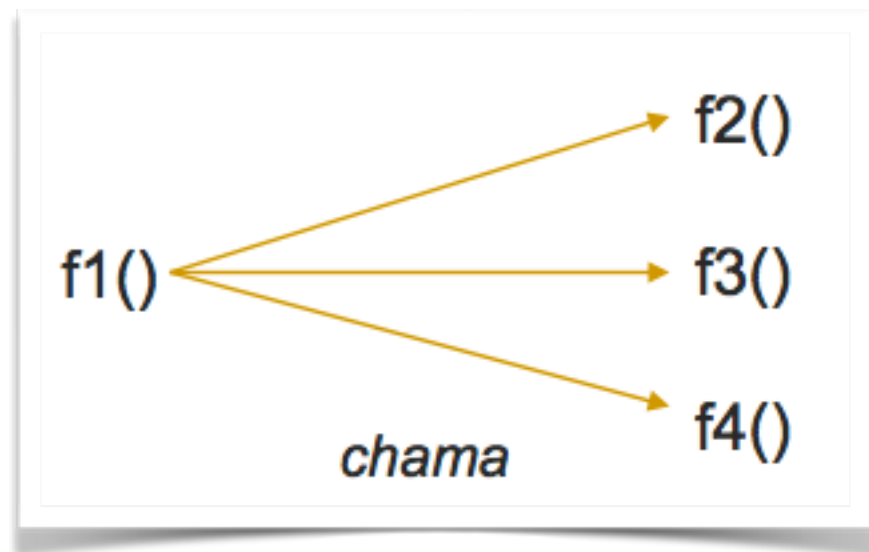
Qual o significado?



`fan-out(f1) = 3`

Fan-out

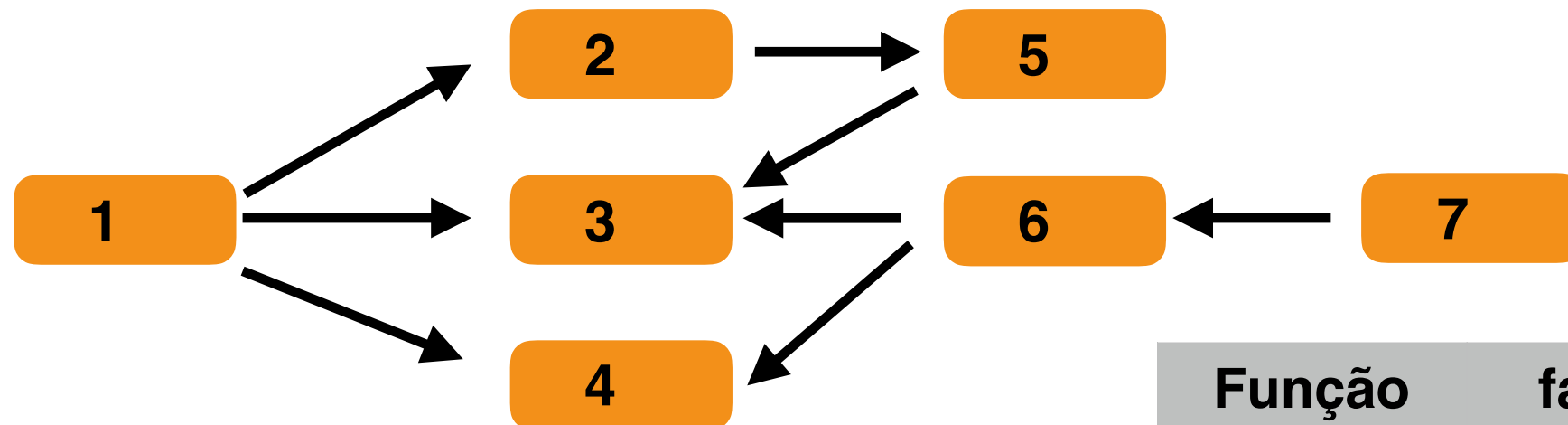
- Número de funções chamadas por uma dada função
- Valor alto significa grande **complexidade** da função



$$\text{fan-out}(f1) = 3$$

Exercício

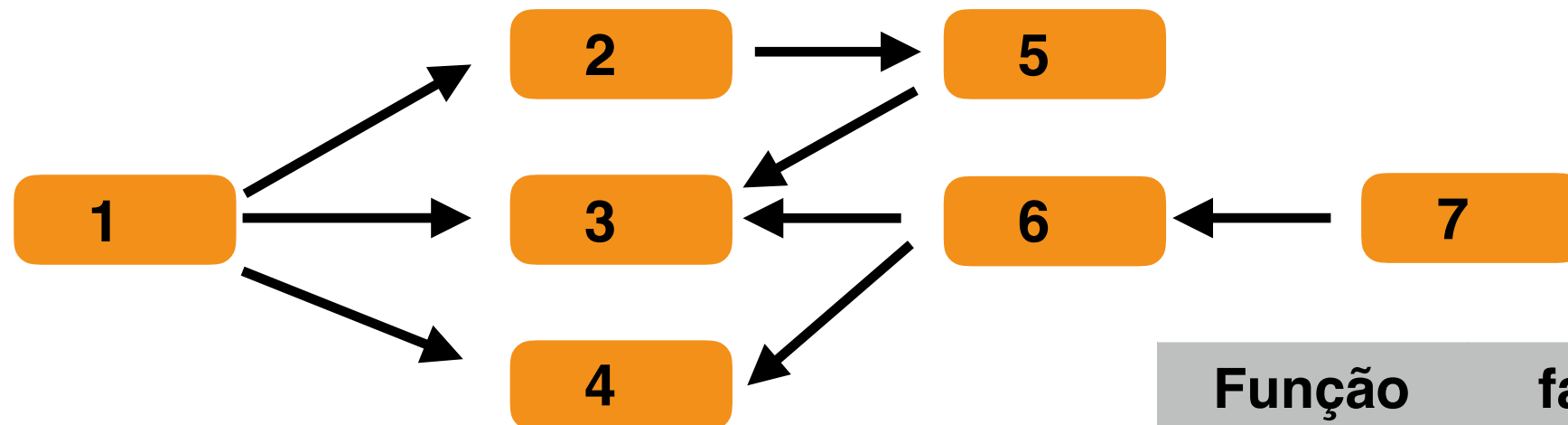
- a) Apresente as métricas **fan-in** e **fan-out** para cada função abaixo.
- b) Encontre a função mais **complexa** e com maior **impacto de mudança**.



Função	fan-in	fan-out
1		
2		
3		
4		
5		
6		
7		

Exercício

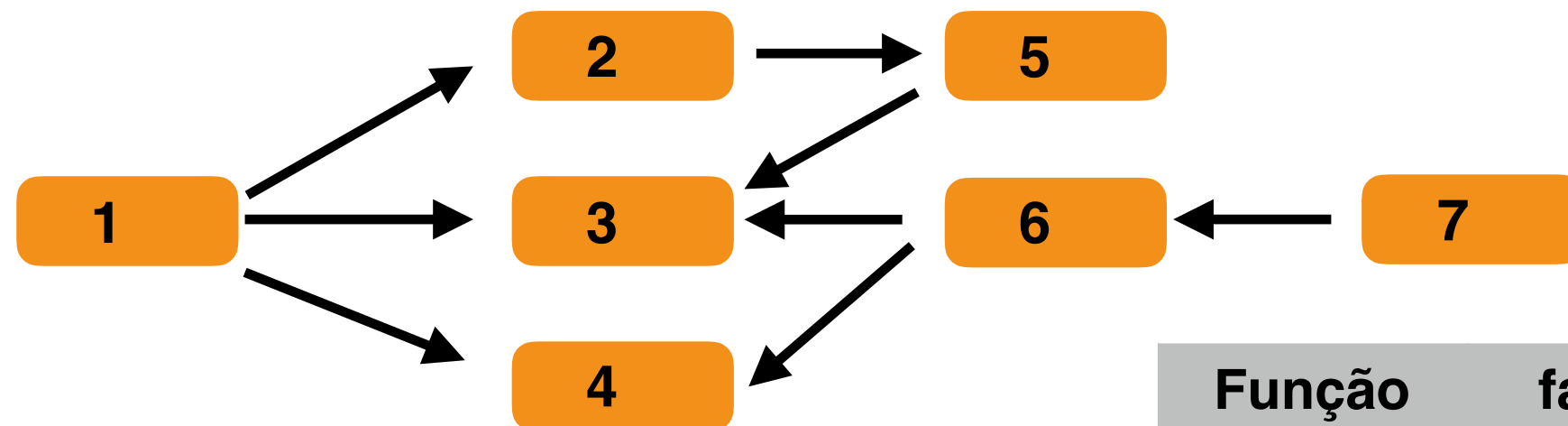
- a) Apresente as métricas **fan-in** e **fan-out** para cada função abaixo.
- b) Encontre a função mais **complexa** e com maior **impacto de mudança**.



Função	fan-in	fan-out
1	0	3
2	1	1
3	3	0
4	2	0
5	1	1
6	1	2
7	0	1

Exercício

- a) Apresente as métricas **fan-in** e **fan-out** para cada função abaixo.
- b) Encontre a função mais **complexa** e com maior **impacto de mudança**.



função mais complexa

função com maior impacto de mudanças

Função	fan-in	fan-out
1	0	3
2	1	1
3	3	0
4	2	0
5	1	1
6	1	2
7	0	1

Tamanho do Código

- Mede o tamanho do sistema
- Relacionado com a complexidade de manutenção, podendo indicar possíveis pontos de refatoração
- Em geral, quanto maior, mais complexo e propenso a erros será o componente
- LOC: número de linhas de código
 - Com ou sem documentação?

```
import lejos.nxt.*;

public class Hello
{
    /**
     * The main method is where your program starts
     */
    public static void main(String[] args) throws Exception
    {
        // makes a buzzing sound
        Sound.buzz();
        // shows text on column 3, row 4 of the LCD
        LCD.drawString("I am alive !!", 3, 4);
        // pauses 2000 ms (= 2sec)
        Thread.sleep(2000);
        // makes another buzzing sound
        Sound.buzz();
        // end of program
    }
}
```


cloc

- <https://github.com/AlDanial/cloc>
- cloc counts blank lines, comment lines, and physical lines of source code
- Many programming languages

```
spring-boot$ cloc *
```

```
5753 text files.
```

```
5619 unique files.
```

```
372 files ignored.
```

```
github.com/AlDanial/cloc v 1.76 T=40.60 s (133.3 files/s, 13479.1 lines/s)
```

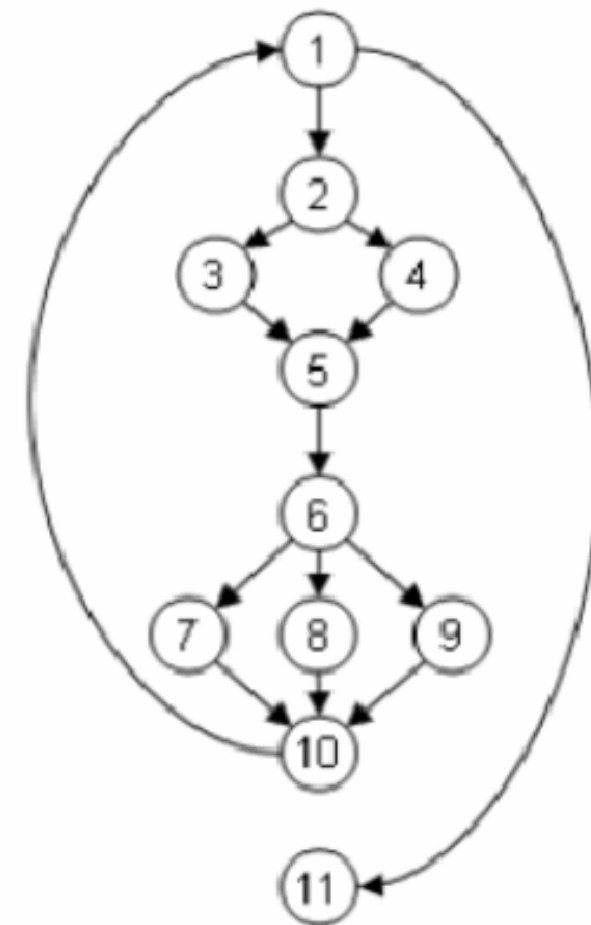
Language	files	blank	comment	code
Java	4539	67135	119600	272251
Maven	239	21	379	23063
AsciiDoc	76	6087	89	17603
JavaScript	5	3382	2897	15153
JSON	30	5	0	6220
Groovy	179	516	334	2342
XML	88	116	32	1674
YAML	48	34	3	1037
Kotlin	37	185	333	978
HTML	40	68	58	953
Velocity Template Language	15	135	0	909
Bourne Shell	44	160	96	898
CSS	5	19	48	560
SQL	34	102	83	395
Bourne Again Shell	2	49	50	308
DOS Batch	3	73	4	255
Ant	2	23	1	138
Freemarker Template	6	9	0	102
XSLT	4	2	0	86
Dockerfile	5	6	1	42
XSD	2	0	0	33
Mustache	6	9	0	32
JSP	3	4	0	31
Ruby	1	4	0	23
SUM:	5413	78144	124008	345086

Complexidade Ciclomática

- Indica a complexidade do código através da quantidade de caminhos de execução
- Mede a complexidade de controle do programa
 - if, while, for, etc
- Relacionada à facilidade de compreensão

Complexidade Ciclomática

Node	Statement
(1)	while(x<100){
(2)	if (a[x] % 2 == 0) {
(3)	parity = 0;
	}
(4)	else {
	parity = 1;
(5)	}
(6)	switch(parity){
	case 0:
(7)	println("a[" + i + "] is even");
	case 1:
(8)	println("a[" + i + "] is odd");
	default:
(9)	println("Unexpected error");
	}
(10)	x++;
	}
(11)	p = true;



Profundidade de Aninhamento (complexidade)

- Número de estruturas internas, como for, while e if aninhadas
- Valor alto indica dificuldade de compreensão
- Métrica de complexidade

```
1. public int method(int n) {  
2.     for(int i = 0; i < n; i++) {  
3.         for(int j = 0; j < n; j++) {  
4.             for(int k = 0; k < n; k++) {  
5.                 ...  
6.             }  
7.         }  
8.     }  
9. }
```

Tamanho dos Identificadores

- Relacionado com a legibilidade do código
- Quanto mais longo o identificados, melhor ele poderá ser entendido por desenvolvedores

```
for (int j=0; j<34; j++) {  
    s += (t[j]*4)/5;  
}
```

```
int realDaysPerIdealDay = 4;  
const int WORK_DAYS_PER_WEEK = 5;  
int sum = 0;  
for (int j=0; j < NUMBER_OF_TASKS; j++) {  
    int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;  
    int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);  
    sum += realTaskWeeks;  
}
```

Exercício

- Descreva as diferenças entre métricas estáticas e dinâmicas
- Apresente exemplos de métricas dinâmicas e estáticas

Exercício

- Descreva as diferenças entre métricas estáticas e dinâmicas
- Apresente exemplos de métricas dinâmicas e estáticas

Métricas estáticas: coletadas de artefatos do sistema tais como código e documentação, sem execução do sistema

Métricas dinâmicas: coletadas por medição durante a execução do sistema

Métricas estáticas: loc, métodos, atributos, complexidade ciclomática, fan-in, fan-out, acoplamento e coesão (estática), profundidade da herança, quantidade de subclasses diretas...

Métricas dinâmicas: desempenho, quantidade de defeitos, testes de software, complexidade (caminhos executados), acoplamento e coesão (dinâmica)...

Agenda

1. Visão geral
2. Atributos de qualidade
3. Avaliação de qualidade
4. Métricas de produto
 - Básicas
 - **Métricas OO**

Métricas OO

- Métricas de Chidamber-Kemerer (CK): específicas para sistemas orientado a objetos
 - Profundidade da Herança (DIT)
 - Número de Filhos (NOC)
 - Acoplamento entre Objetos (CBO)
 - Falta de Coesão em Métodos (LCOM)
 - Métodos Ponderados por Classes (WMC)
 - Resposta para Classe (RFC)

Métricas OO

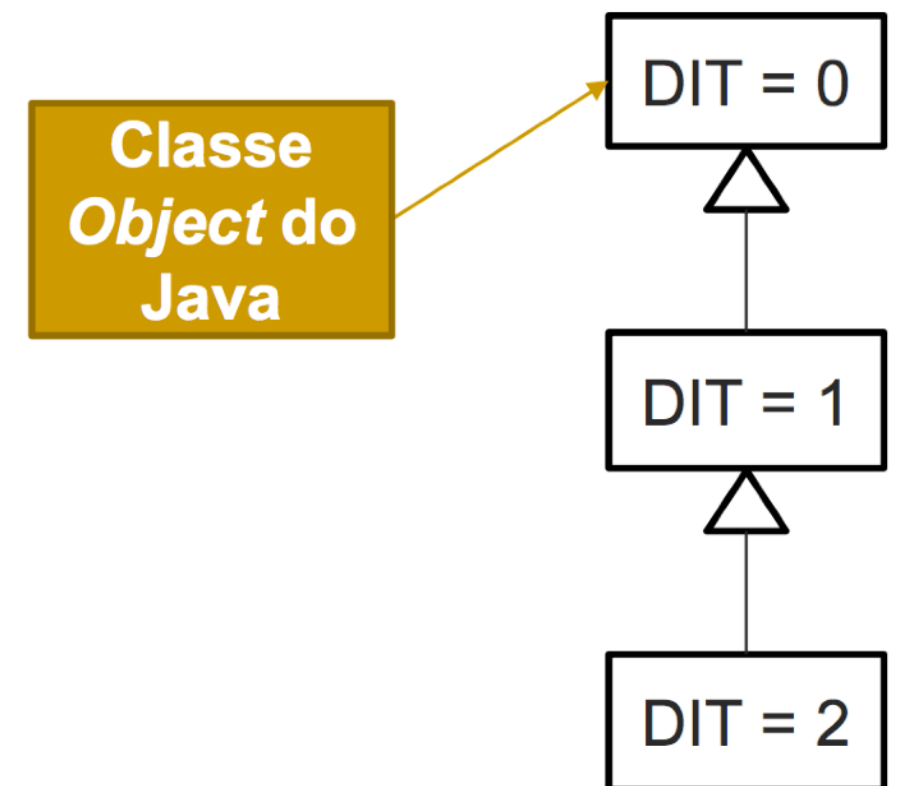
- Profundidade da Herança (DIT)
- Número de Filhos (NOC)
- Acoplamento entre Objetos (CBO)
- Falta de Coesão em Métodos (LCOM)
- Métodos Ponderados por Classes (WMC)
- Resposta para Classe (RFC)

Métricas OO

- **Profundidade da Herança (DIT)**
- Número de Filhos (NOC)
- Acoplamento entre Objetos (CBO)
- Falta de Coesão em Métodos (LCOM)
- Métodos Ponderados por Classes (WMC)
- Resposta para Classe (RFC)

Profundidade de Herança (DIT)

- Número de níveis em que uma classe herda métodos e atributos
- Quanto maior DIT:
 - Mais complexo o projeto
 - Mais difícil de se entender um módulo

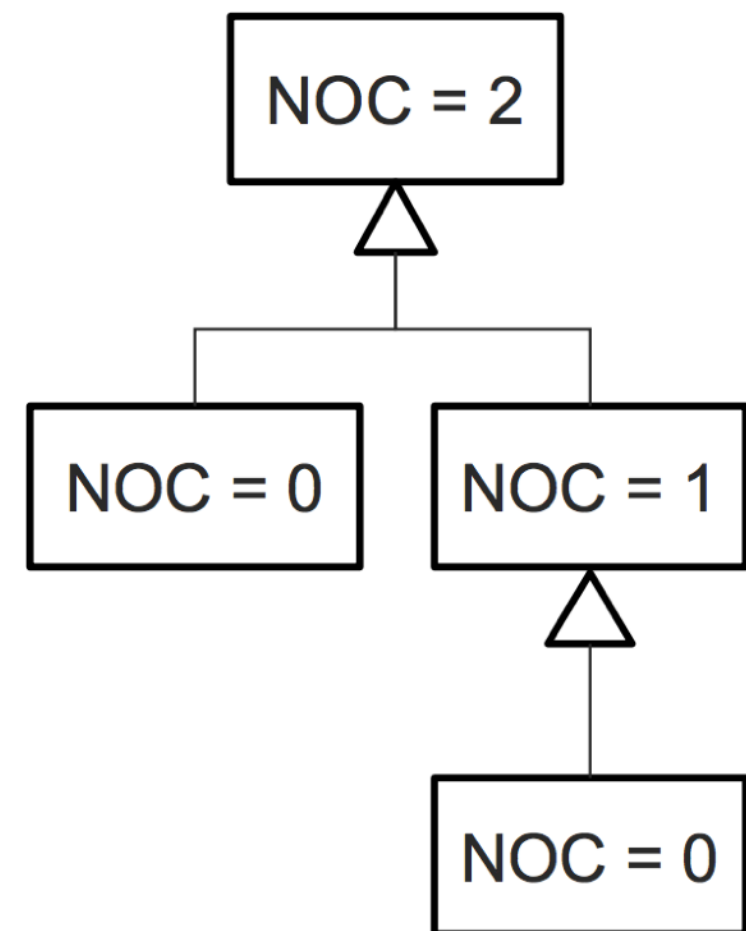


Métricas OO

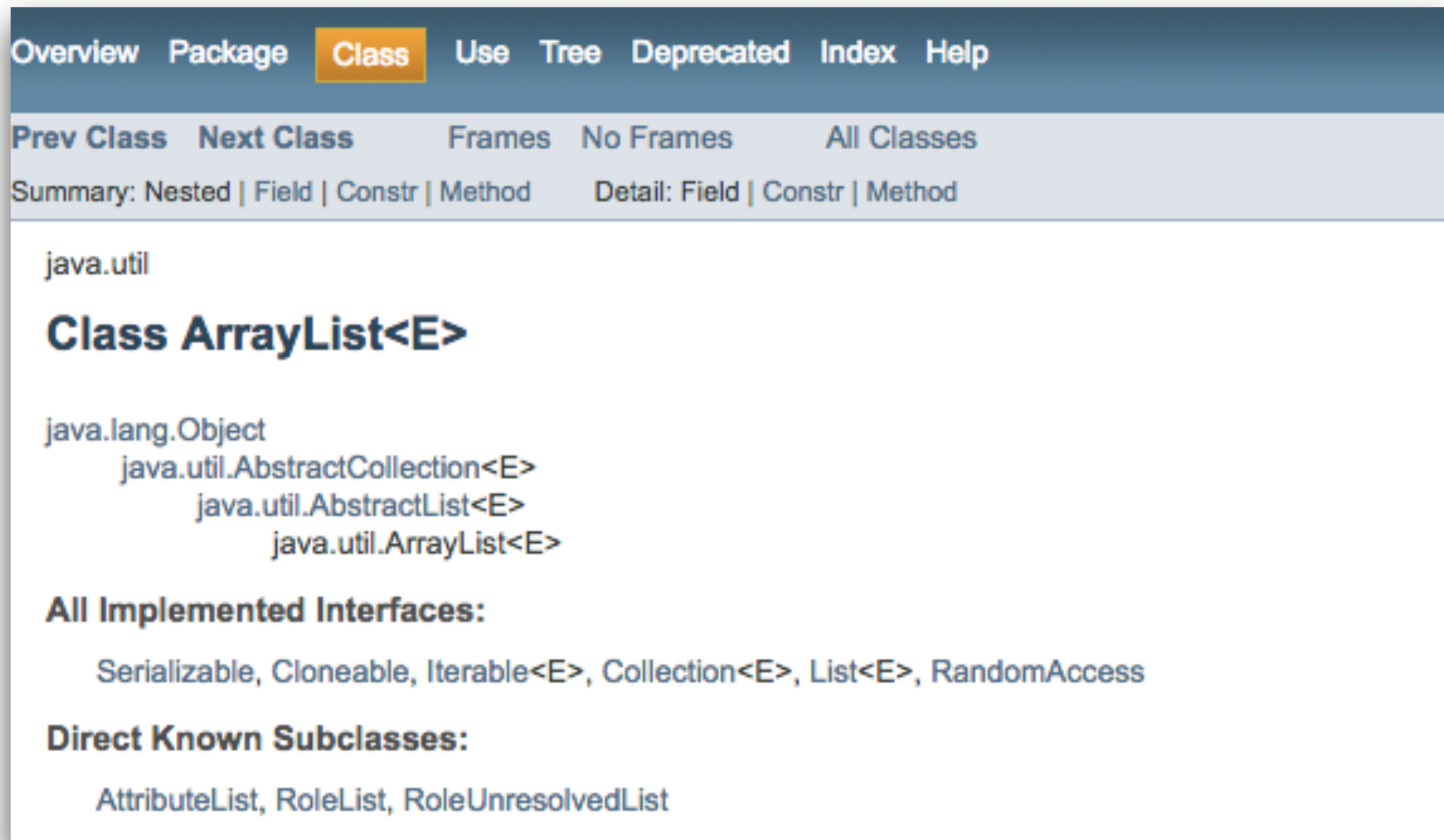
- Profundidade da Herança (DIT)
- **Número de Filhos (NOC)**
- Acoplamento entre Objetos (CBO)
- Falta de Coesão em Métodos (LCOM)
- Métodos Ponderados por Classes (WMC)
- Resposta para Classe (RFC)

Número de Filhos (NOC)

- Número de subclasses diretas
- Mede a largura da hierarquia de uma classe
- NOC alto pode indicar maior reuso



DIT e NOC de ArrayList?



The screenshot shows the Java API documentation for the `ArrayList<E>` class. The navigation bar at the top includes links for Overview, Package, Class (which is highlighted), Use, Tree, Deprecated, Index, and Help. Below this, there are links for Prev Class, Next Class, Frames, No Frames, and All Classes. A summary section shows links for Nested, Field, Constr, and Method, along with a detail section for Field, Constr, and Method. The main content area displays the package `java.util` and the class `ArrayList<E>`. It shows the inheritance hierarchy starting from `java.lang.Object` down to `java.util.ArrayList<E>`. Below the hierarchy, it lists all implemented interfaces: `Serializable`, `Cloneable`, `Iterable<E>`, `Collection<E>`, `List<E>`, and `RandomAccess`. Finally, it lists direct known subclasses: `AttributeList`, `RoleList`, and `RoleUnresolvedList`.

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Class ArrayList<E>

java.lang.Object
 java.util.AbstractCollection<E>
 java.util.AbstractList<E>
 java.util.ArrayList<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

DIT e NOC de ArrayList?

The screenshot shows the Java API documentation for the `ArrayList<E>` class. The navigation bar at the top includes tabs for Overview, Package, Class (selected), Use, Tree, Deprecated, Index, and Help. Below the navigation bar, there are links for Prev Class, Next Class, Frames, No Frames, and All Classes. The main content area shows the package `java.util` and the class `ArrayList<E>`. The inheritance hierarchy is listed as follows:

- `java.lang.Object`
- `java.util.AbstractCollection<E>`
- `java.util.AbstractList<E>`
- `java.util.ArrayList<E>`

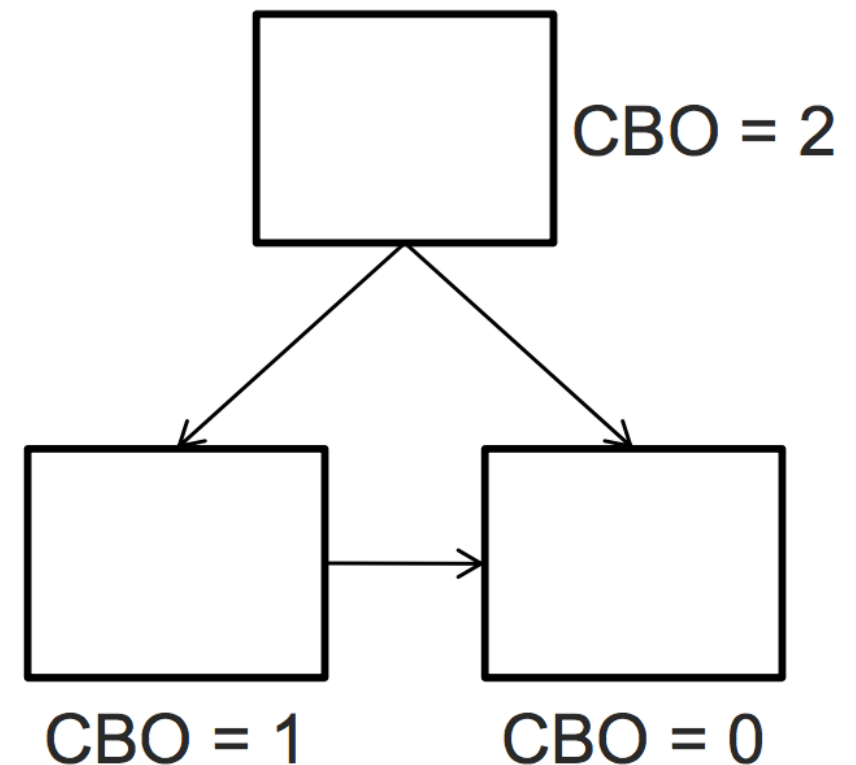
Next to the `java.util.ArrayList<E>` entry, the text **DIT = 3** is displayed in red. Below the inheritance hierarchy, the section **All Implemented Interfaces:** lists `Serializable`, `Cloneable`, `Iterable<E>`, `Collection<E>`, `List<E>`, and `RandomAccess`. The section **Direct Known Subclasses:** lists `AttributeList`, `RoleList`, and `RoleUnresolvedList`. Next to this section, the text **NOC = 3** is displayed in red.

Métricas OO

- Profundidade da Herança (DIT)
- Número de Filhos (NOC)
- **Acoplamento entre Objetos (CBO)**
- Falta de Coesão em Métodos (LCOM)
- Métodos Ponderados por Classes (WMC)
- Resposta para Classe (RFC)

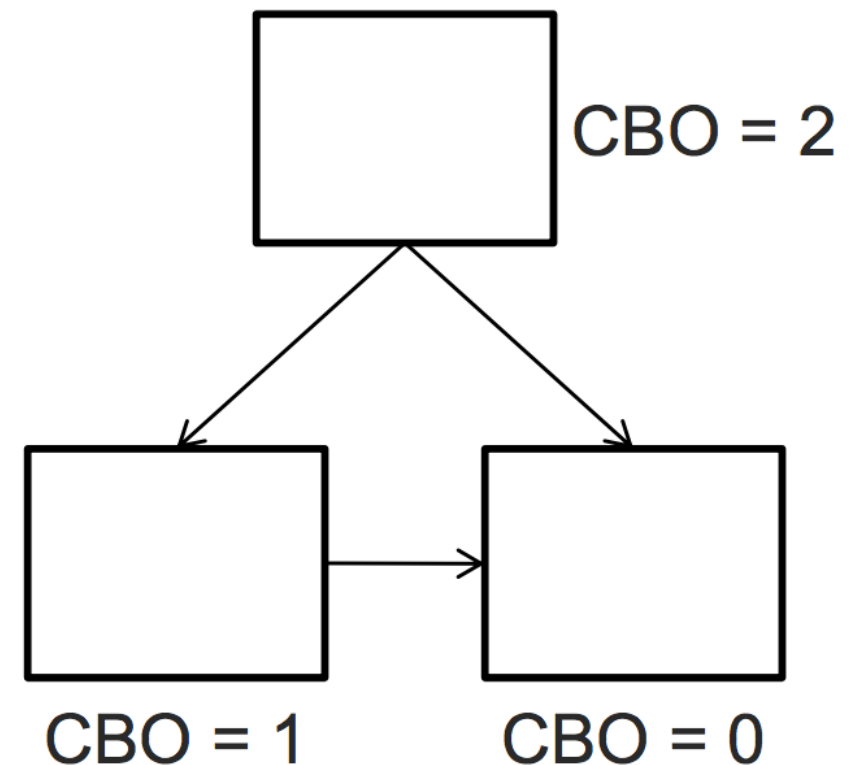
Acoplamento entre Objetos (CBO)

- Número de classes chamadas por uma classe
- Quanto mais acoplada uma classe, mais difícil de entender e manter



Acoplamento entre Objetos (CBO)

- Número de classes chamadas por uma classe
- Quanto mais acoplada uma classe, mais difícil de entender e manter



Devemos buscar baixo acoplamento!

CBO?

```
public class Customer {
    private String name;
    private List<Rental> rentals = new ArrayList<Rental>();
    public Customer(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void addRental(Rental rental) {
        rentals.add(rental);
    }
    public String statement() {
        double totalAmount = 0;
        int f = 0;
        String r = "Rental record for " + getName() + "\n";
        for (Rental rental : rentals) {
            double amount = 0;
            switch (rental.getMovie().getPriceCode()) {
                case Movie.REGULAR:
                    amount += 2;
                    if (rental.getDaysRented() > 2)
                        amount += (rental.getDaysRented() - 2) * 1.5;
                    break;
                case Movie.NEW_RELEASE:
                    amount += rental.getDaysRented() * 3;
                    break;
                case Movie.CHILDREN:
                    amount += 1.5;
                    if (rental.getDaysRented() > 3)
                        amount += (rental.getDaysRented() - 3) * 1.5;
                    break;
            }
            f++;
            if (rental.getMovie().getPriceCode() == Movie.NEW_RELEASE)
                f++;
            r += "\t" + rental.getMovie().getTitle() + "\t" + String.valueOf(amount) + "\n";
            totalAmount += amount;
        }
        r += "Amount owed is " + String.valueOf(totalAmount) + "\n";
        r += "You earned " + String.valueOf(f) + " frequent renter points";
        return r;
    }
}
```

CBO?

CBO: 5

- String
- List
- ArrayList
- Rental
- Movie

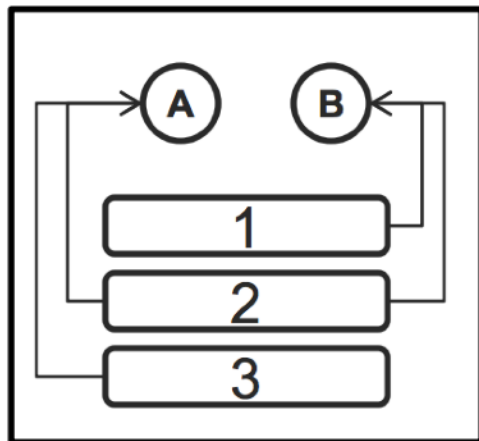
```
public class Customer {
    private String name;
    private List<Rental> rentals = new ArrayList<Rental>();
    public Customer(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void addRental(Rental rental) {
        rentals.add(rental);
    }
    public String statement() {
        double totalAmount = 0;
        int f = 0;
        String r = "Rental record for " + getName() + "\n";
        for (Rental rental : rentals) {
            double amount = 0;
            switch (rental.getMovie().getPriceCode()) {
                case Movie.REGULAR:
                    amount += 2;
                    if (rental.getDaysRented() > 2)
                        amount += (rental.getDaysRented() - 2) * 1.5;
                    break;
                case Movie.NEW_RELEASE:
                    amount += rental.getDaysRented() * 3;
                    break;
                case Movie.CHILDREN:
                    amount += 1.5;
                    if (rental.getDaysRented() > 3)
                        amount += (rental.getDaysRented() - 3) * 1.5;
                    break;
            }
            f++;
            if (rental.getMovie().getPriceCode() == Movie.NEW_RELEASE)
                f++;
            r += "\t" + rental.getMovie().getTitle() + "\t" + String.valueOf(amount) + "\n";
            totalAmount += amount;
        }
        r += "Amount owed is " + String.valueOf(totalAmount) + "\n";
        r += "You earned " + String.valueOf(f) + " frequent renter points";
        return r;
    }
}
```

Métricas OO

- Profundidade da Herança (DIT)
- Número de Filhos (NOC)
- Acoplamento entre Objetos (CBO)
- **Falta de Coesão em Métodos (LCOM)**
- Métodos Ponderados por Classes (WMC)
- Resposta para Classe (RFC)

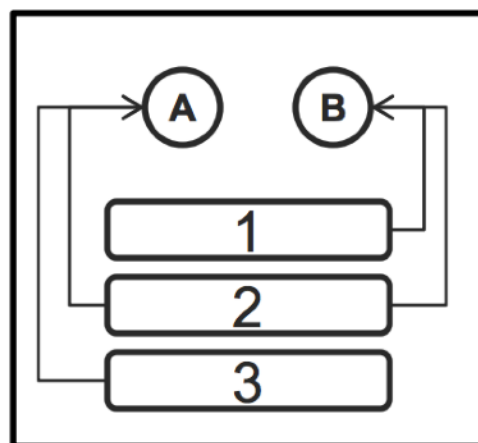
Falta de Coesão em Métodos (LCOM)

- Número de métodos na classes que não compartilham acesso a atributos
- Quanto maior LCOM, menos coesos são os métodos



Falta de Coesão em Métodos (LCOM)

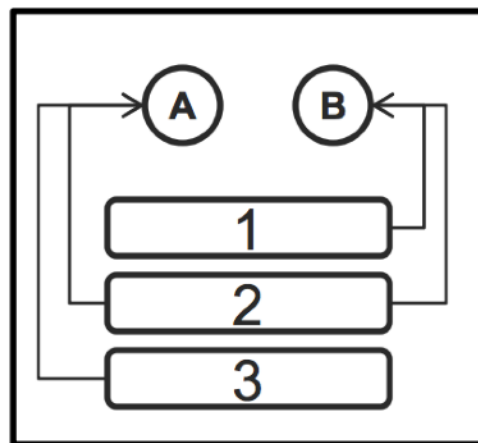
- Número de métodos na classes que não compartilham acesso a atributos
- Quanto maior LCOM, menos coesos são os métodos



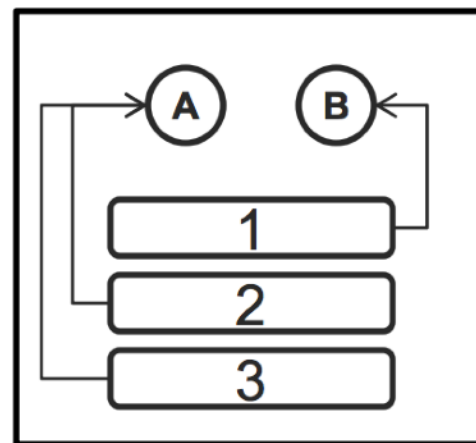
LCOM = 0

Falta de Coesão em Métodos (LCOM)

- Número de métodos na classes que não compartilham acesso a atributos
- Quanto maior LCOM, menos coesos são os métodos

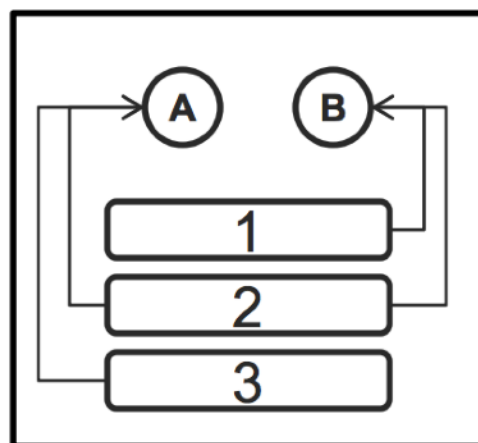


LCOM = 0

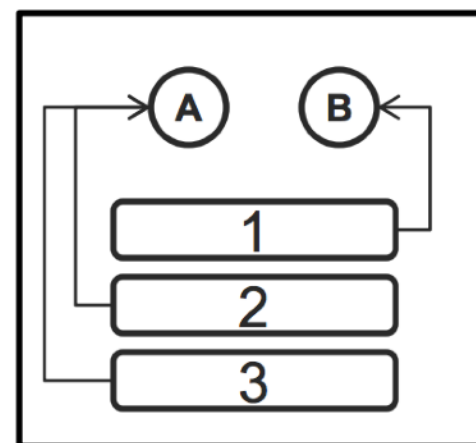


Falta de Coesão em Métodos (LCOM)

- Número de métodos na classes que não compartilham acesso a atributos
- Quanto maior LCOM, menos coesos são os métodos



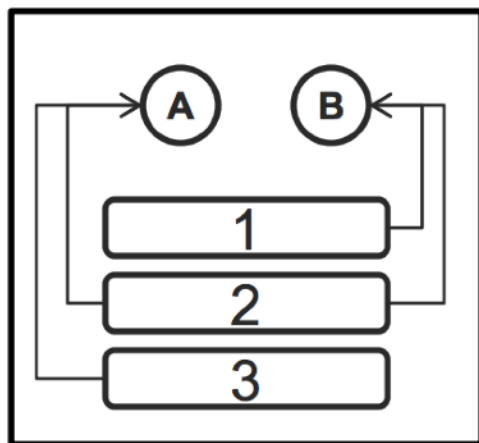
LCOM = 0



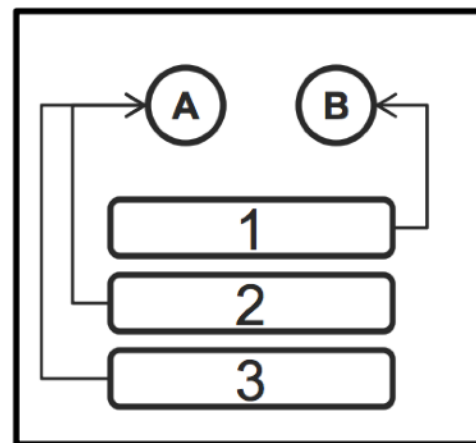
LCOM = 1
(o método 1 acessa B sozinho)

Falta de Coesão em Métodos (LCOM)

- Número de métodos na classes que não compartilham acesso a atributos
- Quanto maior LCOM, menos coesos são os métodos

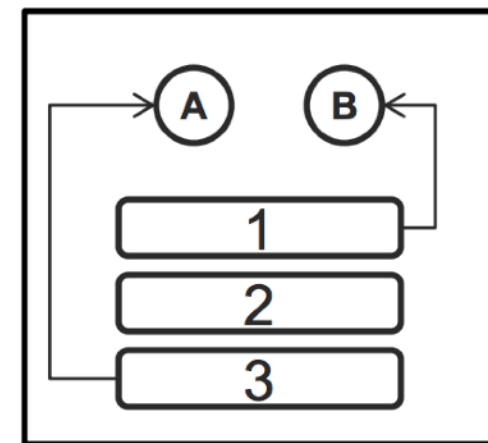


LCOM = 0



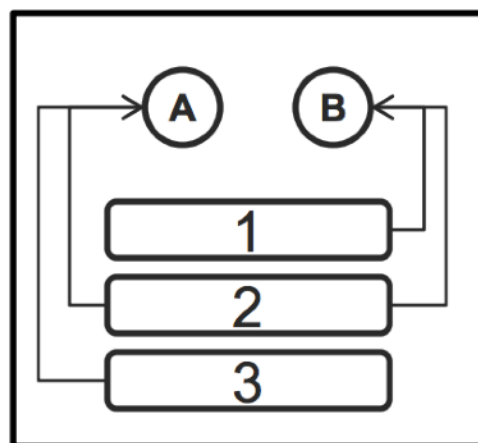
LCOM = 1

(o método 1 acessa B sozinho)

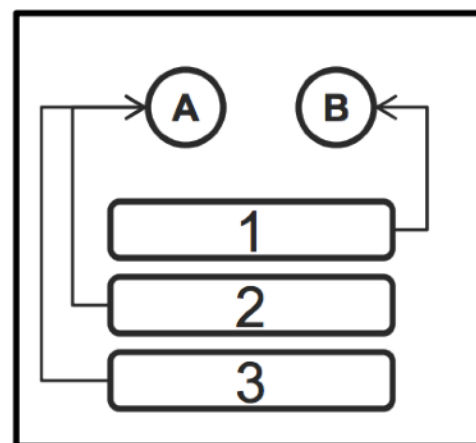


Falta de Coesão em Métodos (LCOM)

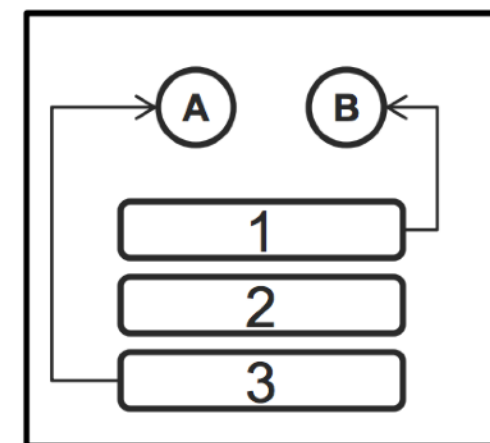
- Número de métodos na classes que não compartilham acesso a atributos
- Quanto maior LCOM, menos coesos são os métodos



LCOM = 0



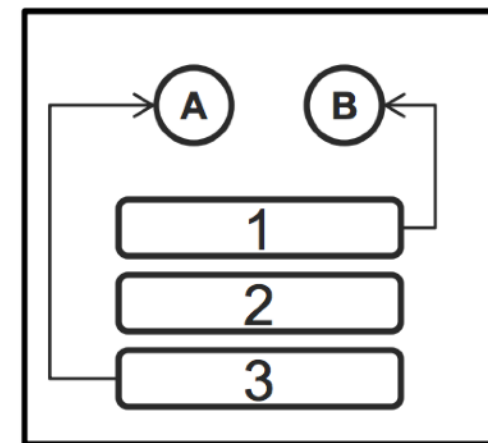
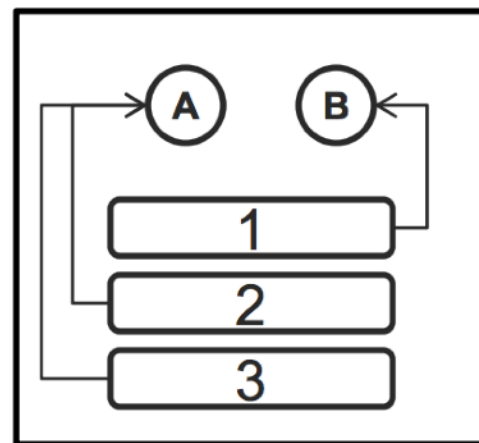
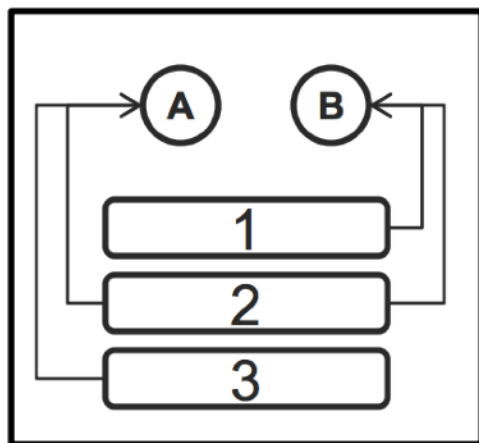
LCOM = 1
(o método 1 acessa B sozinho)



LCOM = 3
(nenhum dos 3 métodos compartilha A ou B)

Falta de Coesão em Métodos (LCOM)

- Número de métodos na classes que não compartilham acesso a atributos
- Quanto maior LCOM, menos coesos são os métodos



**Devemos buscar alta coesão!
(princípio da responsabilidade única)**

LCOM = 3
(nenhum dos 3 métodos
compartilha A ou B)

Métricas OO

- Profundidade da Herança (DIT)
- Número de Filhos (NOC)
- Acoplamento entre Objetos (CBO)
- Falta de Coesão em Métodos (LCOM)
- **Métodos Ponderados por Classes (WMC)**
- Resposta para Classe (RFC)

Métodos Ponderados por Classes (WMC)

- Mede o número de métodos em uma classe, ponderando pela complexidade do método
- Cada método da classe recebe um peso
- Peso pode ser LOC
- WMC alto indica complexidade

Métricas OO

- Profundidade da Herança (DIT)
- Número de Filhos (NOC)
- Acoplamento entre Objetos (CBO)
- Falta de Coesão em Métodos (LCOM)
- Métodos Ponderados por Classes (WMC)
- **Resposta para Classe (RFC)**

Resposta para Classe (RFC)

- Métodos que podem ser potencialmente invocados em resposta a uma mensagem para um objeto
- Valor alto significa complexidade da classe

```
public class A {  
    private B _aB;  
  
    public void methodA1() {  
        return _aB.methodB1();  
    }  
  
    public void methodA2(C aC) {  
        return aC.methodC1();  
    }  
}
```

RFC = ?

Resposta para Classe (RFC)

- Métodos que podem ser potencialmente invocados em resposta a uma mensagem para um objeto
- Valor alto significa complexidade da classe

```
public class A {  
    private B _aB;  
  
    public void methodA1() {  
        return _aB.methodB1();  
    }  
  
    public void methodA2(C aC) {  
        return aC.methodC1();  
    }  
}
```

RFC = 4

Resposta para Classe (RFC)

- Métodos que podem ser potencialmente invocados em resposta a uma mensagem para um objeto
- Valor alto significa complexidade da classe

```
public class A {  
    private B _aB;  
  
    public void methodA1 () {  
        return _aB.methodB1 ();  
    }  
  
    public void methodA2 () {  
        return _aB.methodB1 ();  
    }  
}
```

RFC = ?

Resposta para Classe (RFC)

- Métodos que podem ser potencialmente invocados em resposta a uma mensagem para um objeto
- Valor alto significa complexidade da classe

```
public class A {  
    private B _aB;  
  
    public void methodA1() {  
        return _aB.methodB1();  
    }  
  
    public void methodA2() {  
        return _aB.methodB1();  
    }  
}
```

RFC = 3