Avoiding Code Pitfalls in Aspect-Oriented Programming

A. Santos, P. Alves, E. Figueiredo, and F. Ferrari

Science of Computer Programming (SCP, 2015)

Apresentação: Aline Brito

Programação Orientada a Aspecto (AOP)

Técnica que visa **melhorar a modularidade** do software através da separação de **interesses transversais** em unidades modulares chamadas **aspectos**

Objetivo

Investigar os tipos de **erros** cometidos por iniciantes na **Programação Orientada a Aspecto**

Experimento

Etapas

System	Round	Number of subjects	Grouping
ATM	First	11	Pairs
	Second	16	Individually
Chess	Third	15	Pairs
	Fourth	15	Individually
Telecom	Fifth	3	Pairs
	Sixth	19	Individually
Task Manager	Seventh	6	Pairs
	Eighth	13	Individually
Total number of subjects		98	

Etapas

System	Round	Number of subjects	Grouping
ATM			Pairs Individually
Chess			Pairs Individually
Telecom			Pairs Individually
	Sixth	19	
Task Manager	Seventh	6	Pairs Individually
	Eighth	13	
Total number of subjects		98	

Trabalho individual ou dupla

Etapas

System	Round	Number of subjects	Grouping
ATM	First Secon		
Chess	Third Fourt		
Telecom	Fifth Sixth		
Task Manager	Seven Eighth	15	individually
Total number of subjects		98	

4 aplicativos

ATM, Chess, Telecom, Task Manager

Sistema ATM

- Terminal de autoatendimento
- 600 linhas de código, 12 módulos
- Interesse refatorado: **Logging**

Sistema ATM

Interesse refatorado: **Logging**

- Registra todas as operações realizadas por um usuário de caixa eletrônico

- 19 linhas de código, 4 módulos

Sistema Chess

- Jogo de xadrez com uma interface gráfica
- 1.000 linhas de código, 13 módulos
- Interesse refatorado: **ErrorMessages**

Sistema Chess

Interesse refatorado: **ErrorMessages**

- Mostra mensagens quando um jogador tenta violar uma regra do jogo de xadrez

- 36 linhas de código, 8 módulos

Sistema Telecom

- Gerenciamento de chamadas telefônicas
- 200 linhas de código, 8 módulos
- Interesse refatorado: **Timing**

Sistema Telecom

Interesse refatorado: **Timing**

- Registra o tempo decorrido em uma determinada chamada

- 30 linhas de código, 4 módulos

Sistema Task Manager

- Gerenciamento de tarefas
- 209 linhas de código, 6 módulos
- Interesse refatorado: **Observer**

Sistema Task Manager

Interesse refatorado: **Observer**

- Notifica todas as classes que estão observando as tarefas quando ocorre uma mudança

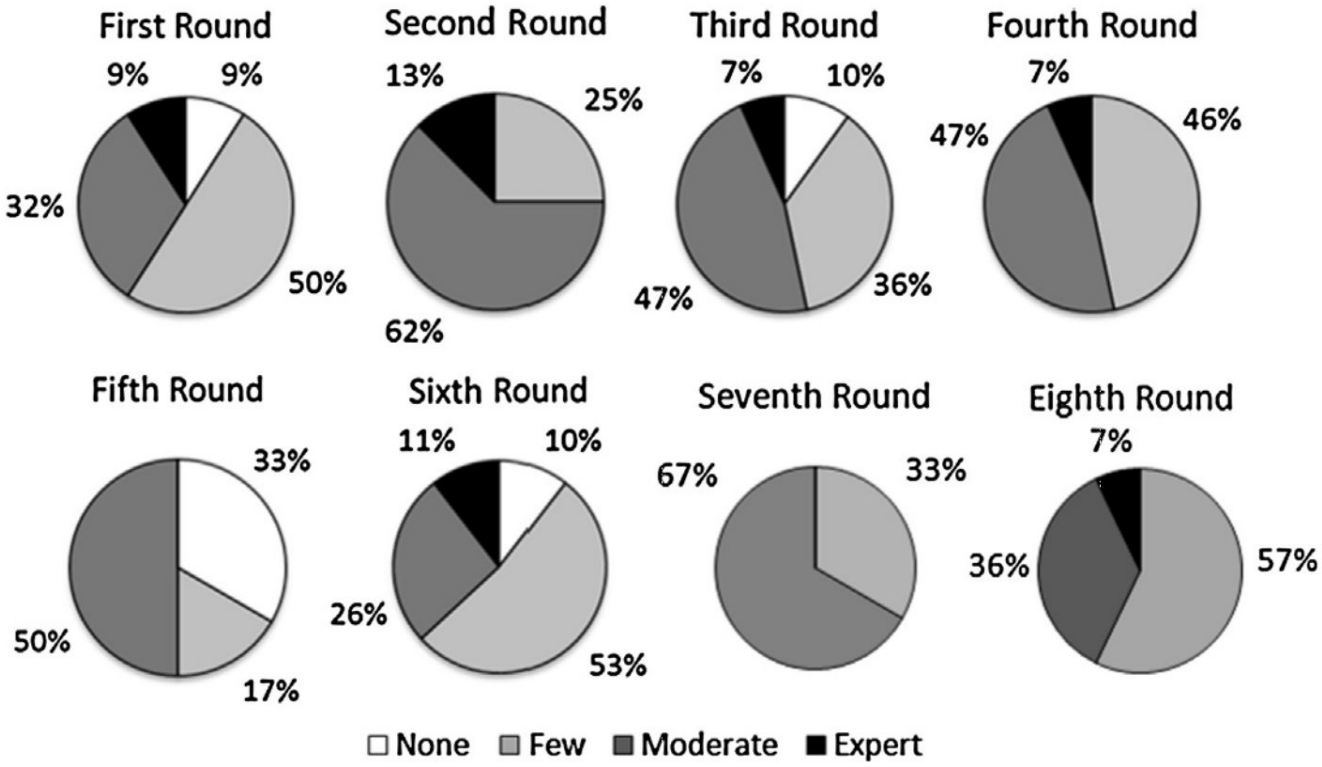
- 3 módulos

Perfil dos participantes

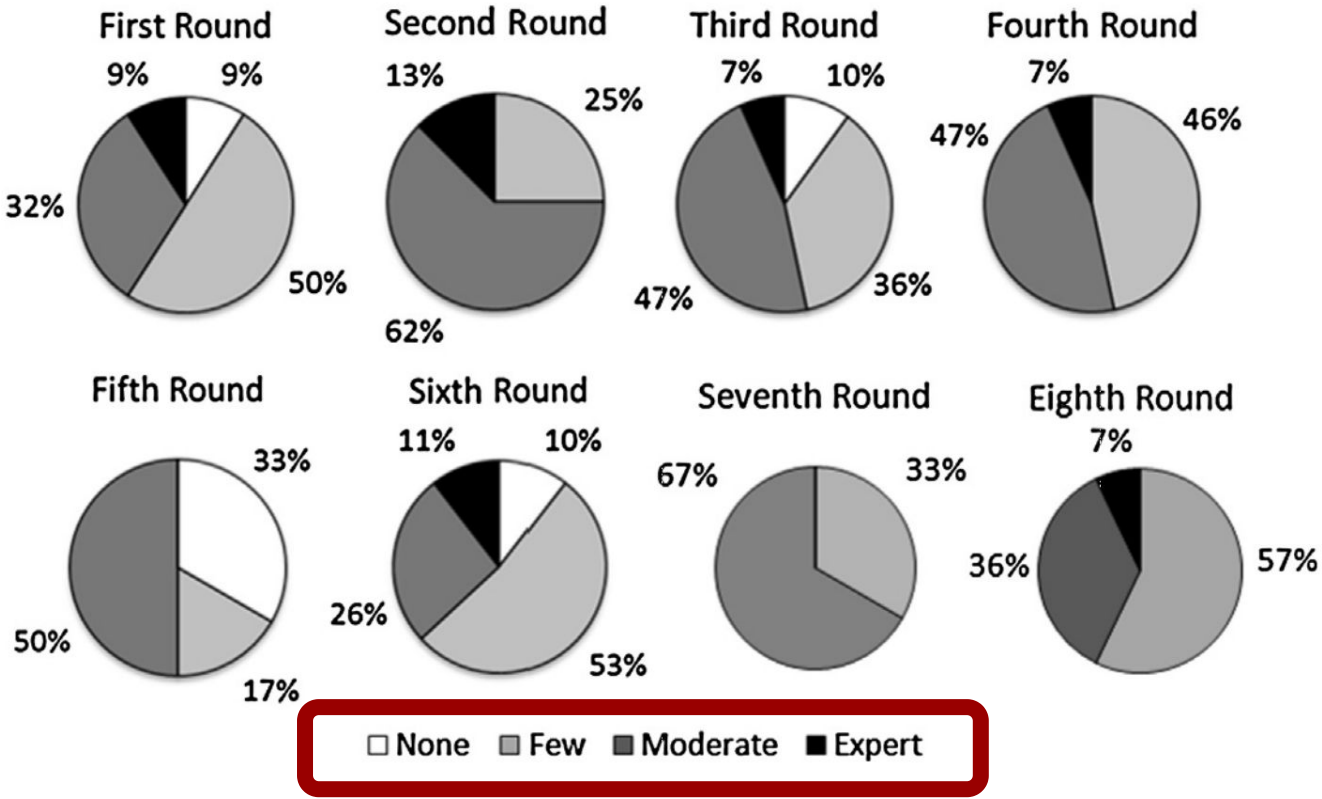
- Estudantes ou pessoas recém formadas

- Eles preencheram um **questionário** sobre seu **nível de conhecimento** em Programação Orientada a Objetos

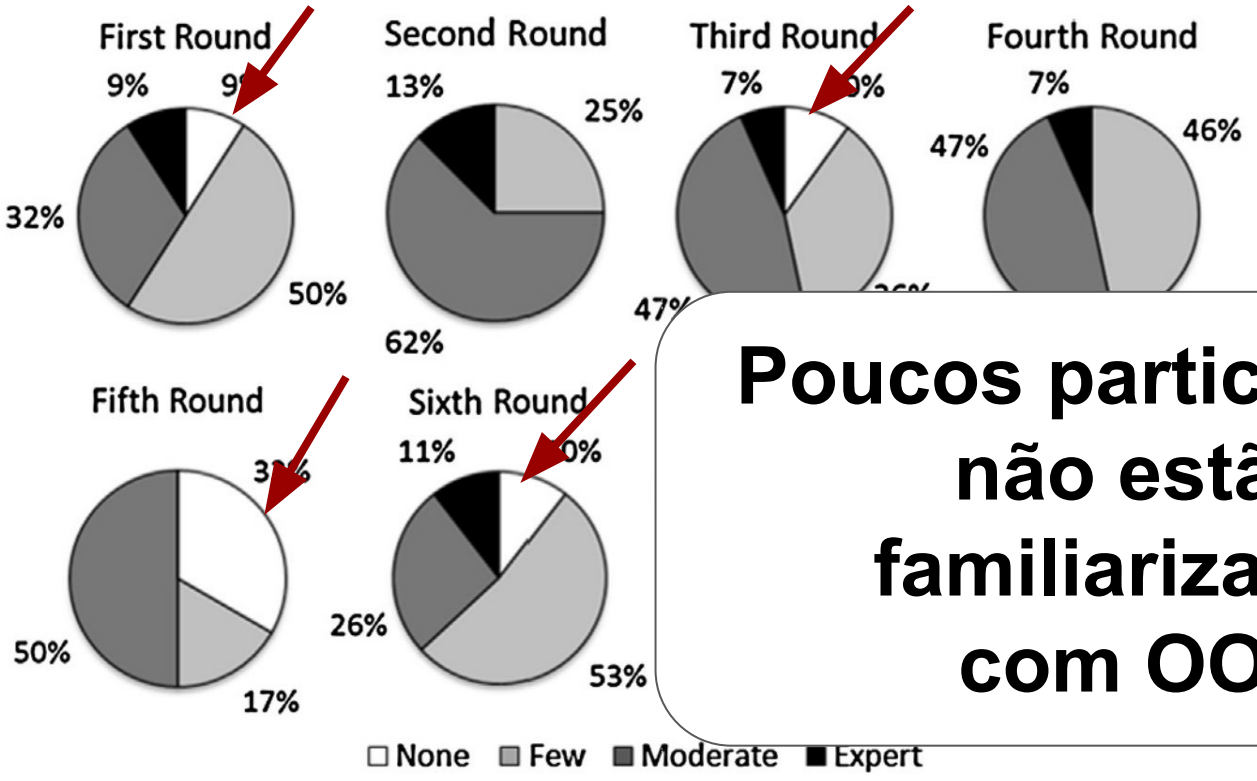
Perfil dos participantes



Perfil dos participantes



Perfil dos participantes



Poucos participantes não estão familiarizados com OOP

Tarefas

Refatorações usando a linguagem de programação **AspectJ**

Duas horas de **treinamento** sobre Programação Orientada a Aspecto e AspectJ

Tarefas

Foi solicitado a cada participante **refatorar um interesse** usando a IDE Eclipse e o *plugin* AJDT

Cada grupo recebeu o **código fonte** e a **descrição do interesse** transversal

Resultados

Erros Recorrentes

Erros recorrentes

- Lógica incorreta
- Uso incorreto do *Advice*
- Erros de compilação ou *warning*
- Interesse duplicado
- Excesso de refatorações
- Refatorações incompletas

Lógica Incorreta

O código refatorado se **comporta** de maneira **diferente** do original

Lógica Incorreta

```
public class ATM {
  private static final int LOG = 0;
  void performTransactions() {
    ...
    while (!userExited) {
      ...
      switch (mainMenuSelection) {
        ...
        case LOG:
          Logger.printLog();
          break;
      }
    }
  }
  ...
}
```

Antes:
Relatório de logs chamado dentro do da estrutura switch

Lógica Incorreta

<u>Depois</u>:
Relatório de logs sempre é exibido após o método performTransactions()

```
public aspect Logging {

  public pointcut performTransactions() :
    call (void ATM.performTransactions());

  after() returning () :
      performTransactions() {
    Logger.printLog();
  }
  ...
}
```

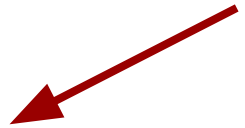
Uso incorreto do Advice

Uso de **tipo incorreto** de Advice ao refatorar o interesse

Uso incorreto do Advice

```
public class BankDatabase {

  public void credit(int userAccount, double amount) {
    getAccount(userAccount).credit(amount);
    Logger.log(userAccount + " made a deposit of " + amount);
  }
  ...
}
```



<u>Antes</u>:

Uma operação de depósito é registrada ao final do método credit()

Uso incorreto do Advice

Depois:
Operação de depósito registrada no **início** da transação

```
public pointcut credit(int userAccount,double amount):
  call (public void BankDatabase.credit(int, double))&&
  args(userAccount, amount);

before(int userAccount, double amount) :
    credit(userAccount, amount) {
  log(userAccount + " made a deposit of " +amount);
}
...
}
```

Erros de Compilação e Warning

Erros que levaram a uma **falha sintática** ou **warning** sinalizado pelo AspectJ

Erros de Compilação e Warning

```
public class Task implements Subject {
  ...
  public void createTask(String nName, String nNotes, String nTimeRequired) {
    name = nName;
    notes = nNotes;
    timeRequired = getTimeRequired(nTimeRequired);
    notifyObservers();
  }
  ...
}
```

(a) Before refactoring

```
public aspect TaskObserver {

  declare parents: Task implements Subject;

  pointcut subjectChange(Subject subject):
    call (void Task.createTask()) && target(subject);
  ...
}
```

(b) After refactoring

Ausência de parâmetros no método createTask()

Excesso de refatorações

Parte do código que **não pertence** a um interesse **é refatorado** para aspecto

Excesso de refatorações

```
public abstract class Connection {
  void complete() {
    state = COMPLETE;
    System.out.println("connection completed");
    timer.start();
  }
  ...
}
```

(a) Before refactoring

```
public aspect Timing {

  pointcut complete(Connection c):
      call(void Connection.complete())&& target(c);

  after(Connection c): complete(c) {
    System.out.println("connection completed");
    c.timer.start();
  }
  ...
}
```

(b) After refactoring

Excesso de refatorações

```
public abstract class Connection {
  void complete() {
    state = COMPLETE;
    System.out.println("connection completed");
    timer.start();
  }
  ...
}
```

(a) Before refactoring

<u>Antes</u>:

Interesse para refatorar: timer.start()

Excesso de refatorações

<u>Depois</u>:

Mensagem refatorada junto com o interesse

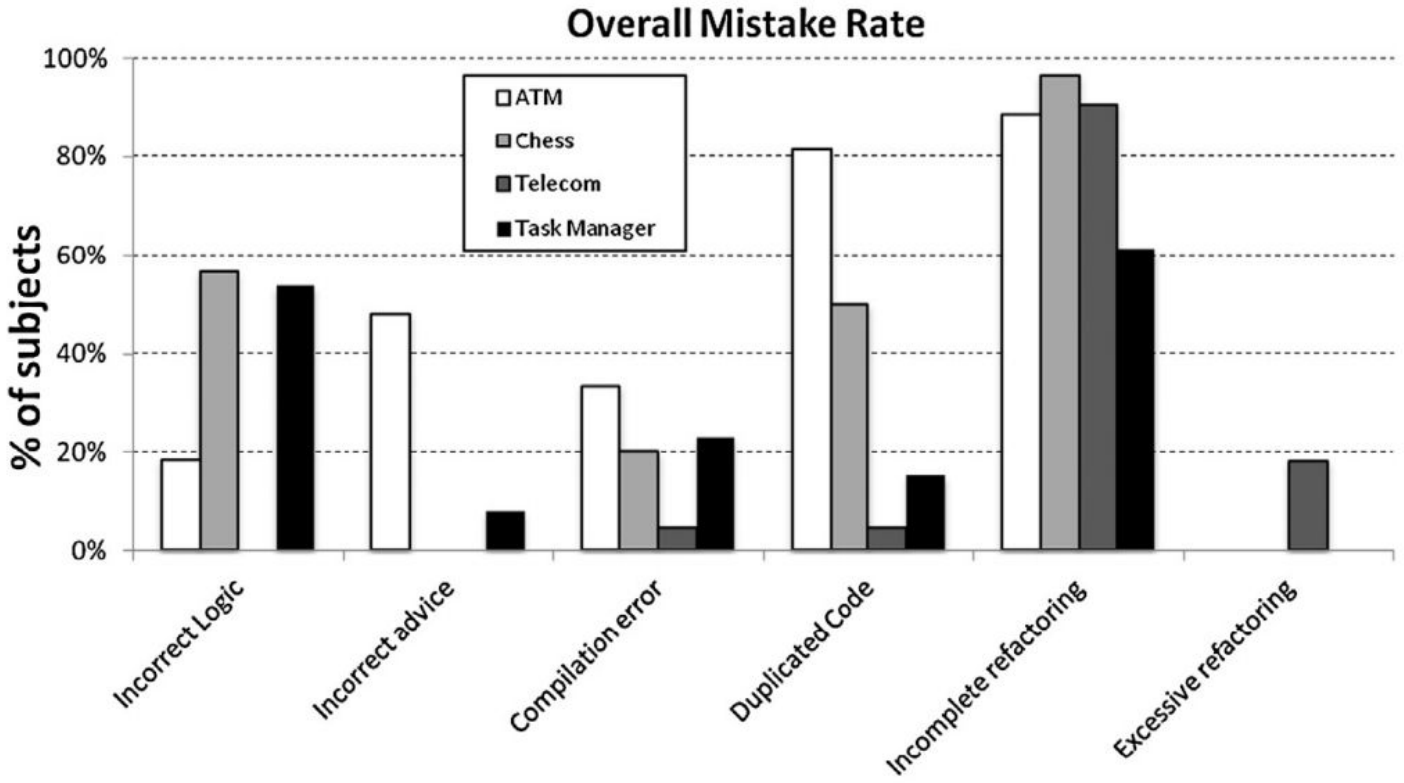
```
    pointcut complete(Connection c):
        call(void Connection.complete())&& target(c);

    after(Connection c): complete(c) {
      System.out.println("connection completed");
      c.timer.start();
    }
    ...
}
```

(b) After refactoring

Análise dos Resultados

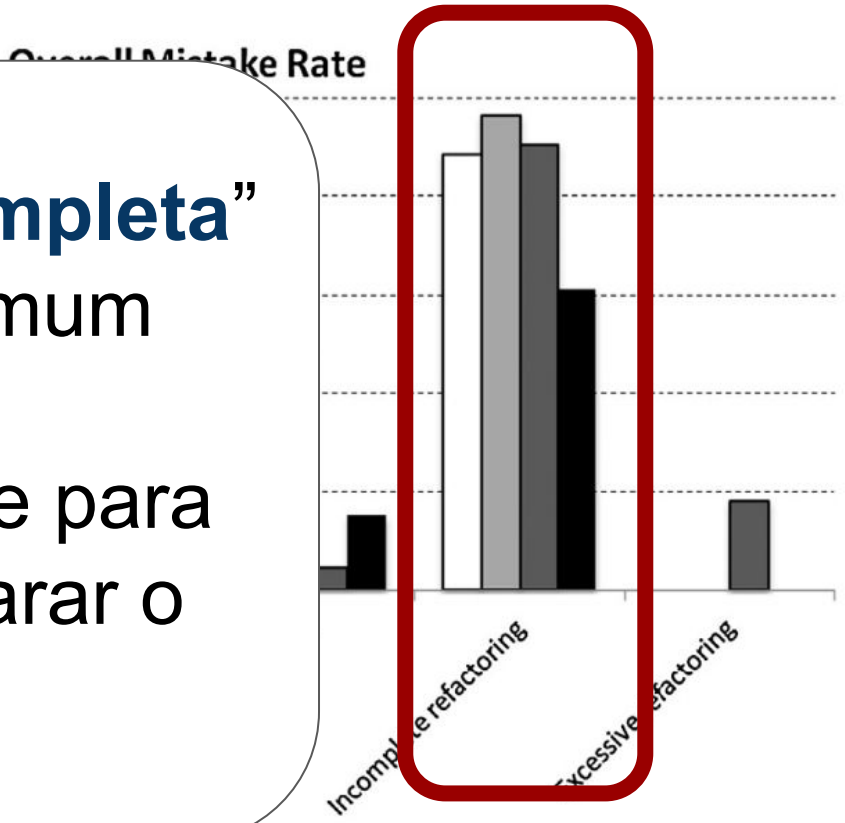
Erros por sistema



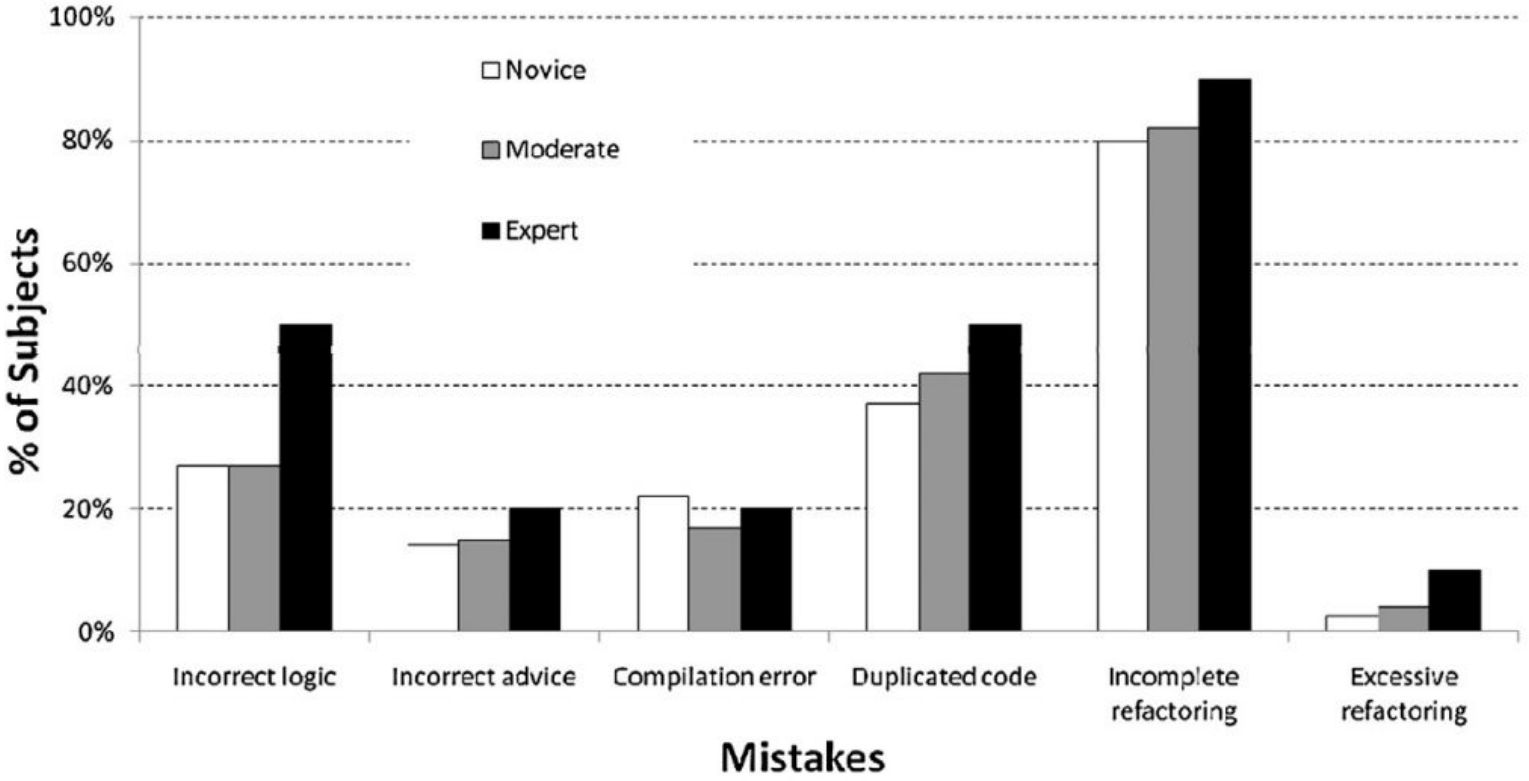
Erros por sistema

“**Refatoração incompleta**”
é o erro mais comum

Motivo: dificuldade para identificar ou separar o interesse

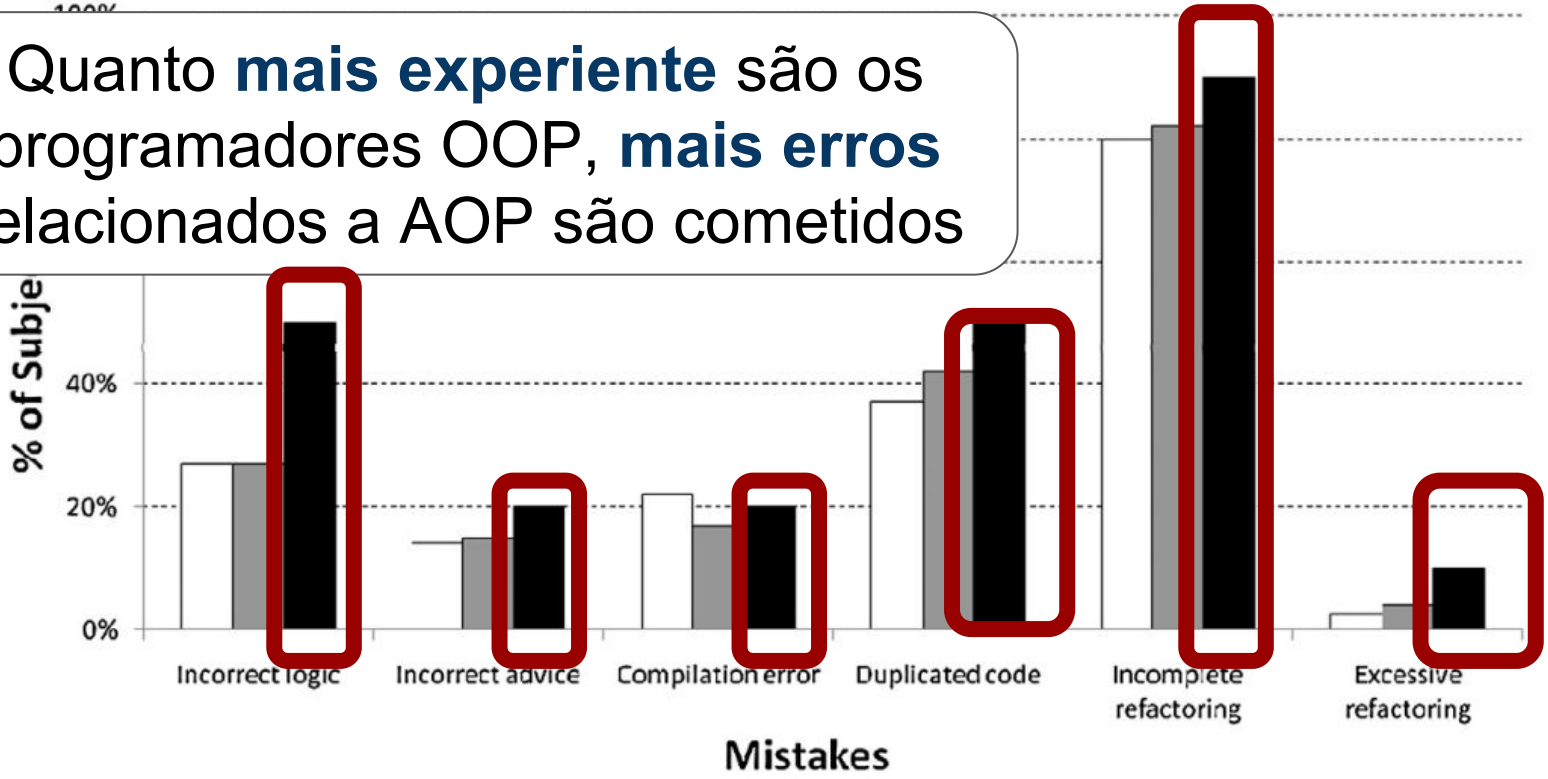


Erros por nível de conhecimento em POO

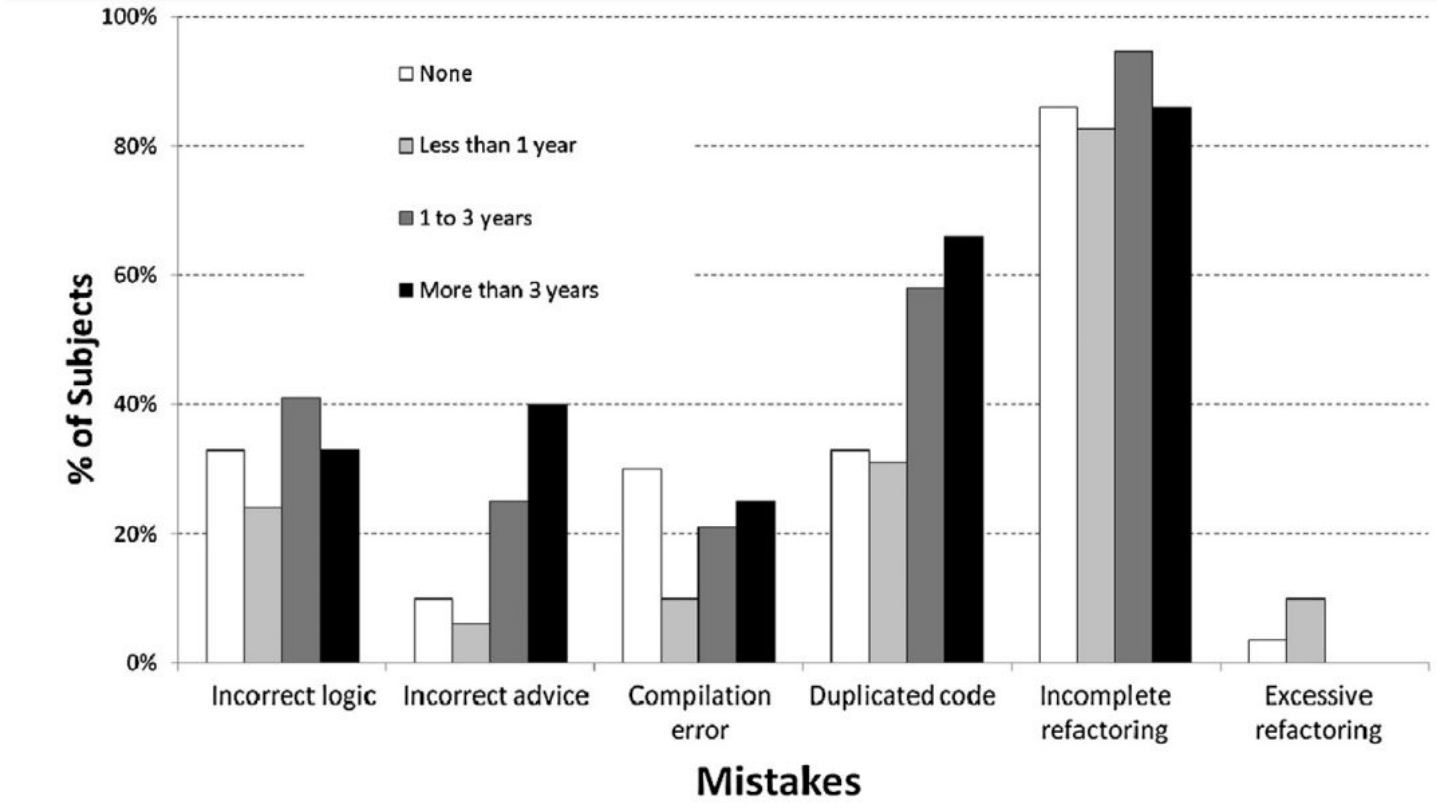


Erros por nível de conhecimento em POO

Quanto **mais experiente** são os programadores OOP, **mais erros** relacionados a AOP são cometidos



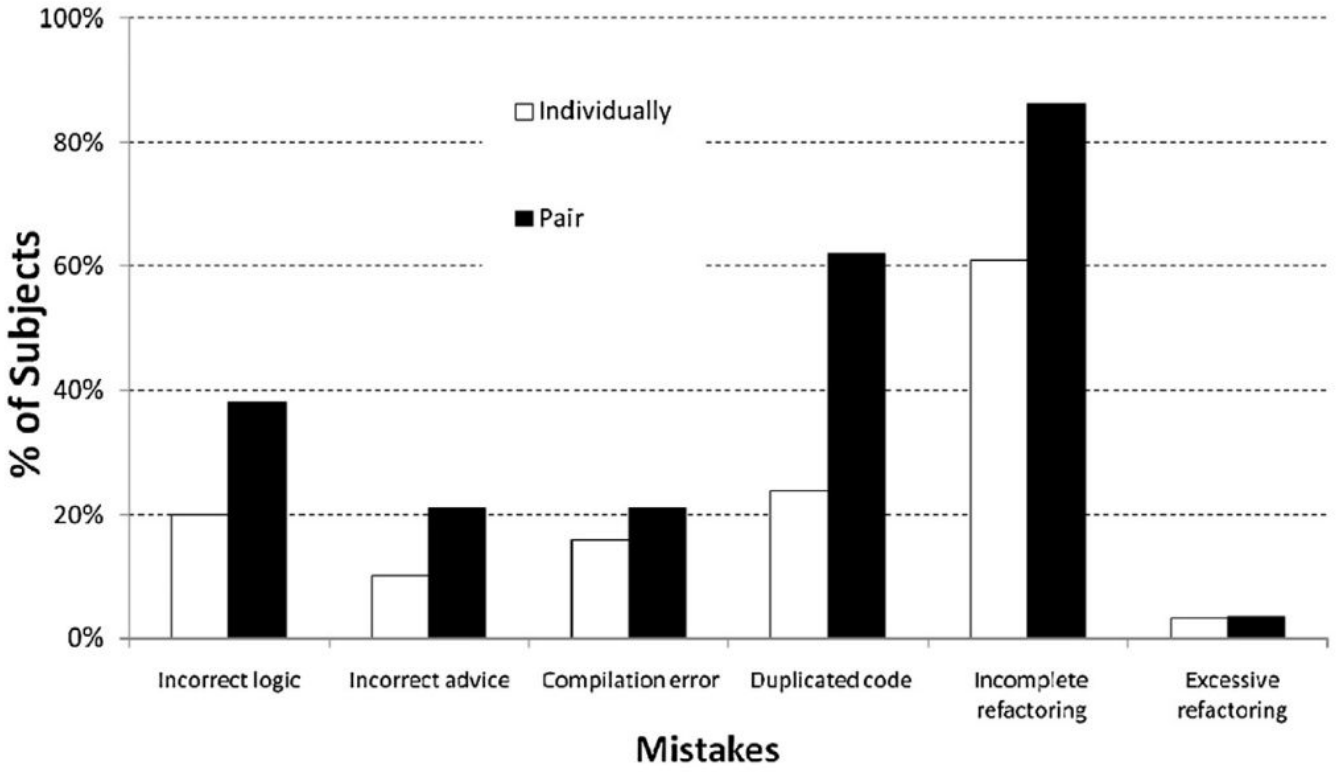
Erros por nível de experiência profissional



Erros por nível de experiência profissional

Os programadores **mais experientes** tiveram **menos erros de compilação** ou **refatorações em excesso**

Erros por grupo



Erros por grupo

A **programação em pares não é a melhor opção** em alguns casos

“Armadilhas” na AOP

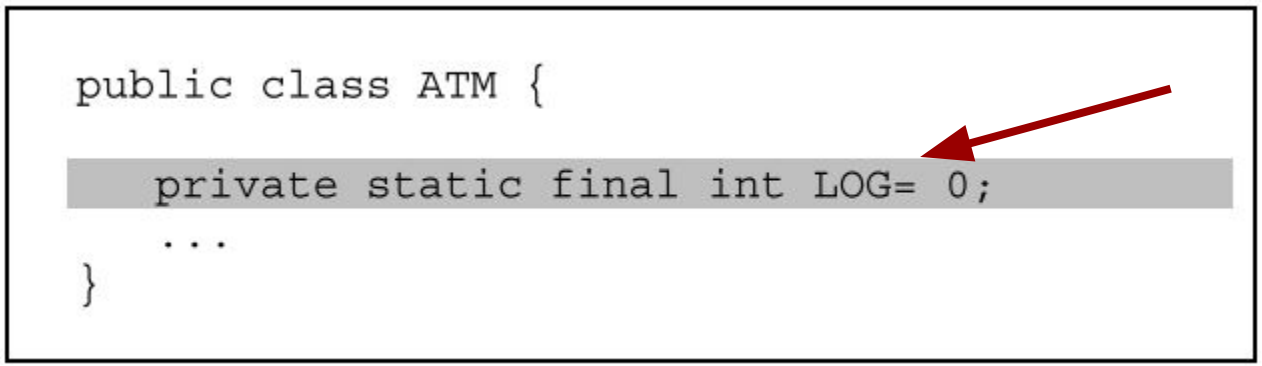
Constantes

Constantes implementando interesses

levaram a erros no experimento

Constantes

A constante LOG implementa o interesse Logging no sistema ATM



Atributos Non-Dedicated

Atributos com **tipos primitivos** ou **tipos não dedicados ao interesse** levaram a erros no experimento

Atributos Non-Dedicated

```
public abstract class Connection {
   Timer timer = new Timer();
   ...
}
```

(a) Attribute of a dedicated type

```
public class Costumer {
   ...
   long totalConnectionTime = 0;
   ...
}
```

(b) Attribute of a non-dedicated type

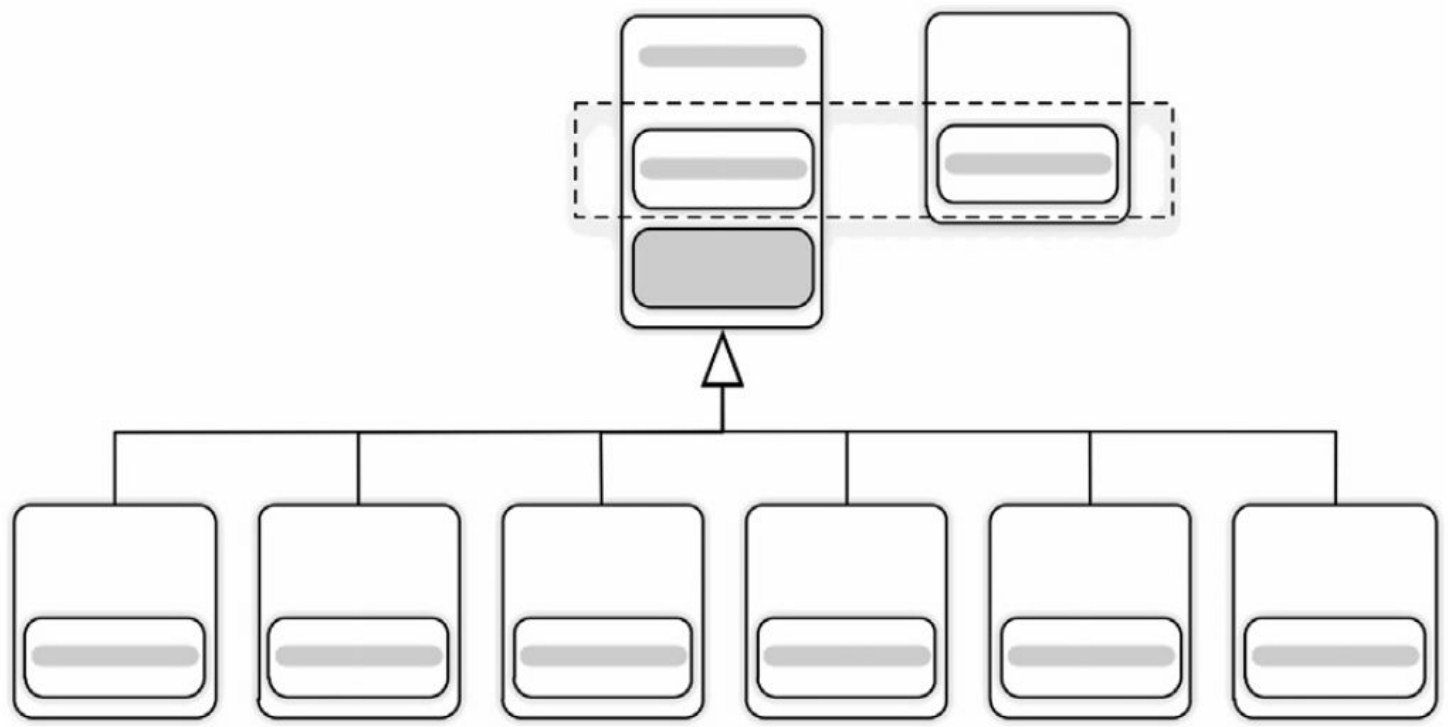
Dedicado ao interesse (poucos erros)

Tipo primitivo

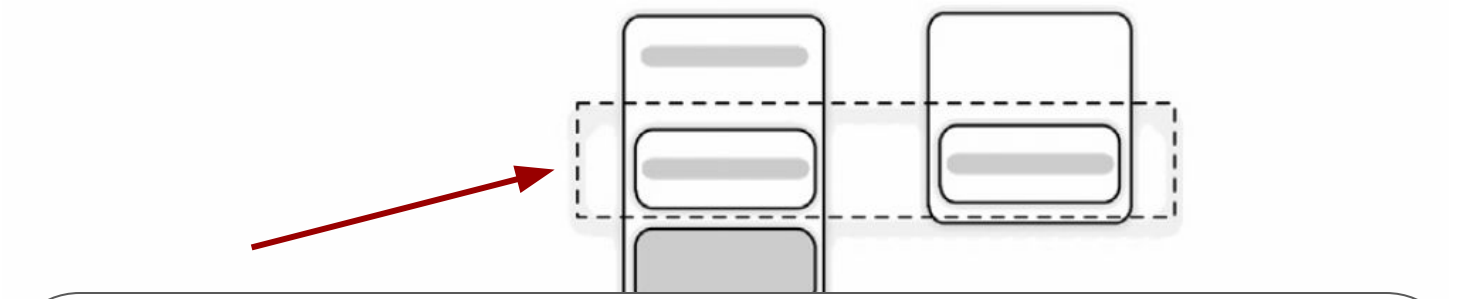
Árvores de Herança Disjuntas

A **falta de relacionamento** entre as árvores da herança pode levar a refatorações incompletas

Árvores de Herança Disjuntas



Árvores de Herança Disjuntas



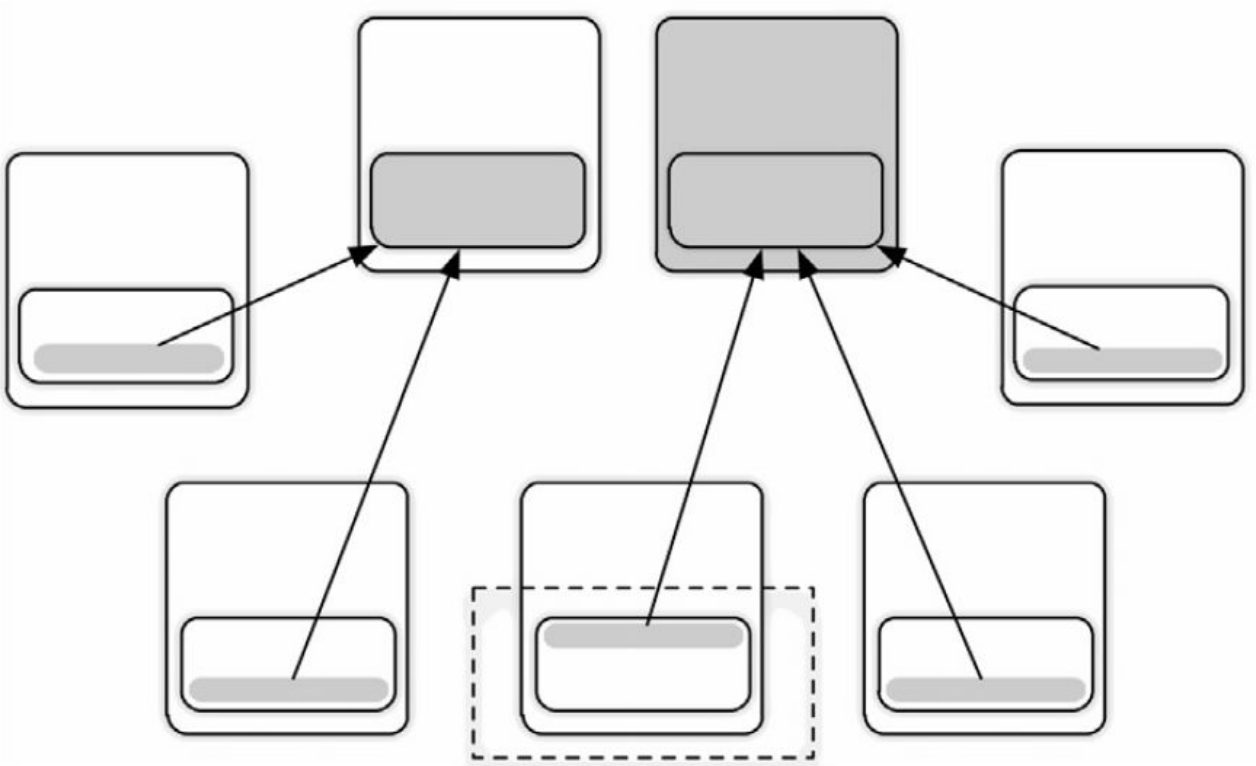
Duas ou mais árvores de herança

(propenso ao erro)

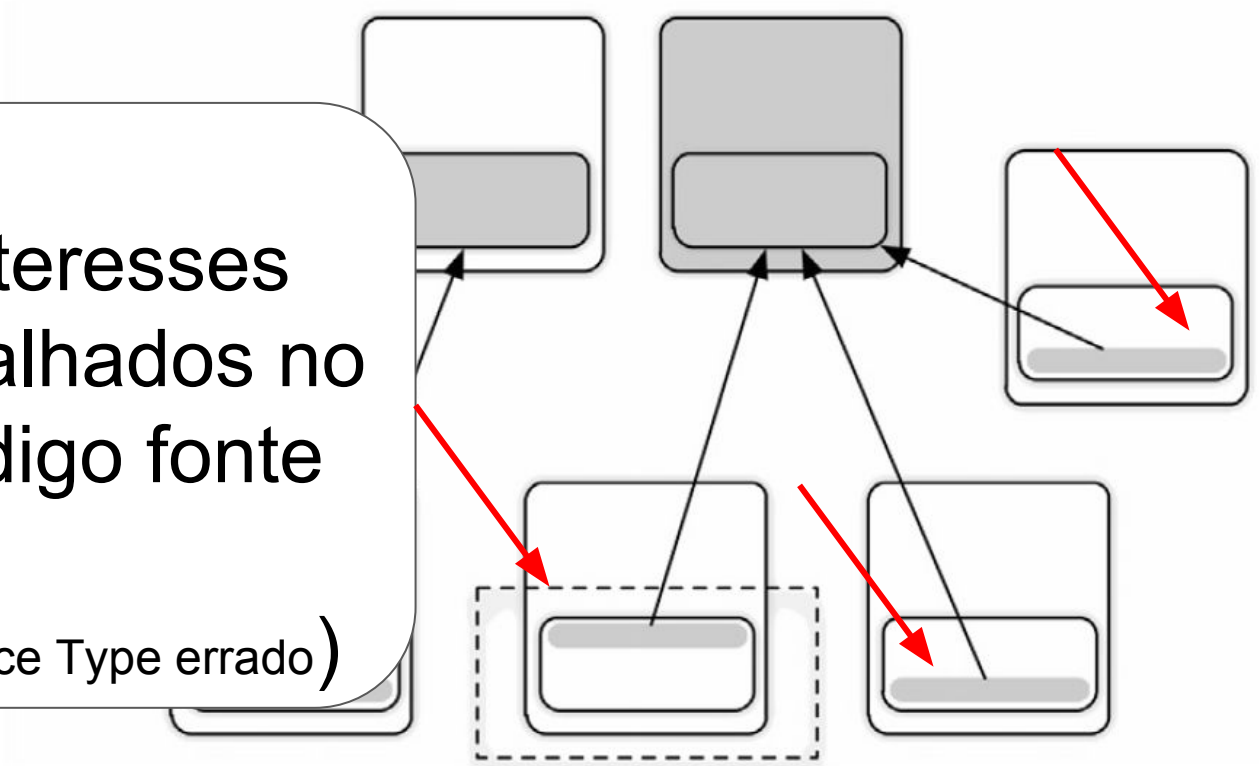
Localização

Partes do código fonte do interesse **espalhadas** no sistema podem levar a erros

Localização



Localização



Interesses espalhados no código fonte

(Advice Type errado)

Variáveis locais

Interesse implementado em método que acessa uma **variável local** pode **dificultar a refatoração**

Variáveis Locais

```
public class ATM {
  private void performTransactions() {
    ...
    while(!userExited) {
      int mainMenuSelection = displayMainMenu();
      switch(mainMenuSelection) {
        ...
        case Log:
          Logger.printLog();
          break;
        ...
      }
    }
  }
  ...
}
```

Ferramenta ConcernReCS2

Ferramenta ConcernReCS2

- Plug-in Eclipse
- Encontra falhas na AOP
- Extende *ConcernMapper*, ferramenta para mapear métodos e atributos
- Apresenta armadilhas de código
  - Interesse, linha, arquivo, percentual de erro

Limitações

Limitações

- Uso da linguagem AOP (AspectJ)
- Sistemas e interesses selecionados
- Dimensão do estudo

Considerações Finais

Considerações Finais

- Documentação dos erros recorrentes

- Programação em pares pode ajudar os programadores a evitar alguns erros (ex., refatoração incompleta)

Considerações Finais

- Resultados podem ajudar a aperfeiçoar novas ferramentas

- Ferramenta *ConcernReCS* para detecção de armadilhas no código

Considerações Finais

- Alguns erros são mais comuns entre os programadores mais experientes em POO