Avoiding code pitfalls in Aspect-Oriented Programming

Adriano Santos, Péricles Alves, Eduardo Figueiredo, Fabiano Ferrari

18º Simpósio Brasileiro de Linguagens de Programação
Maceió, 2014

Apresentação: Aline Brito

AOP - Programação Orientada a Aspecto

- Técnica que visa melhorar a modularidade do software.
- **Aspecto**: Unidade modular que encapsula interesses transversais.
- **Interesse**: Qualquer consideração que possa ter impacto na concepção e manutenção de módulos de programa.

Objetivo

Investigar os tipos de erros cometidos por estudantes e profissionais iniciantes na Programação Orientada a Aspecto.

Questões de Pesquisa

RQ1. Quais os tipos de erros cometidos por programadores júnior?

RQ2. Quais são as armadilhas no código-fonte levam a esses erros?

RQ3. Quais habilidades um programador deve ter para cometer menos erros na AOP?

Características do Experimento

Number of subjects in each round.

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	

Number of subjects in each round.

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	

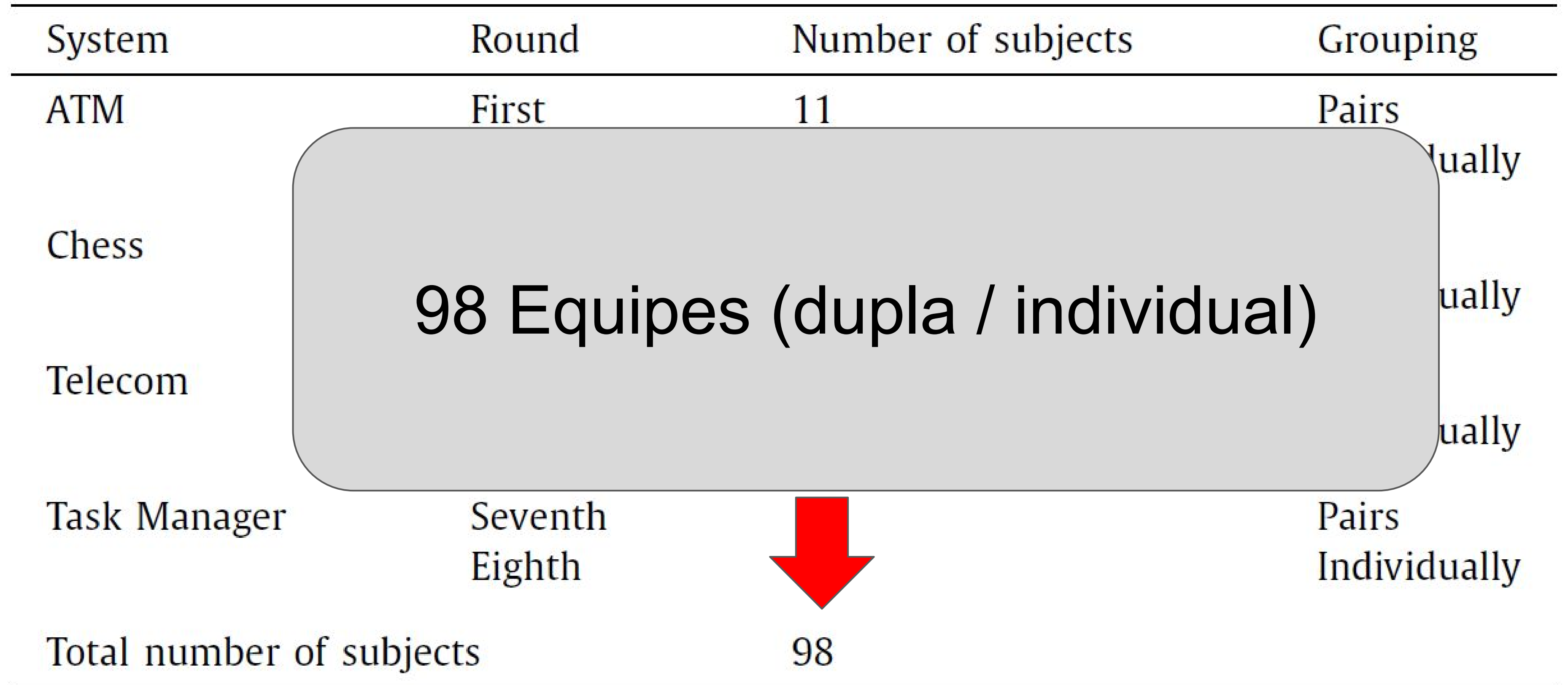
Number of subjects in each round.

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	

Number of subjects in each round.

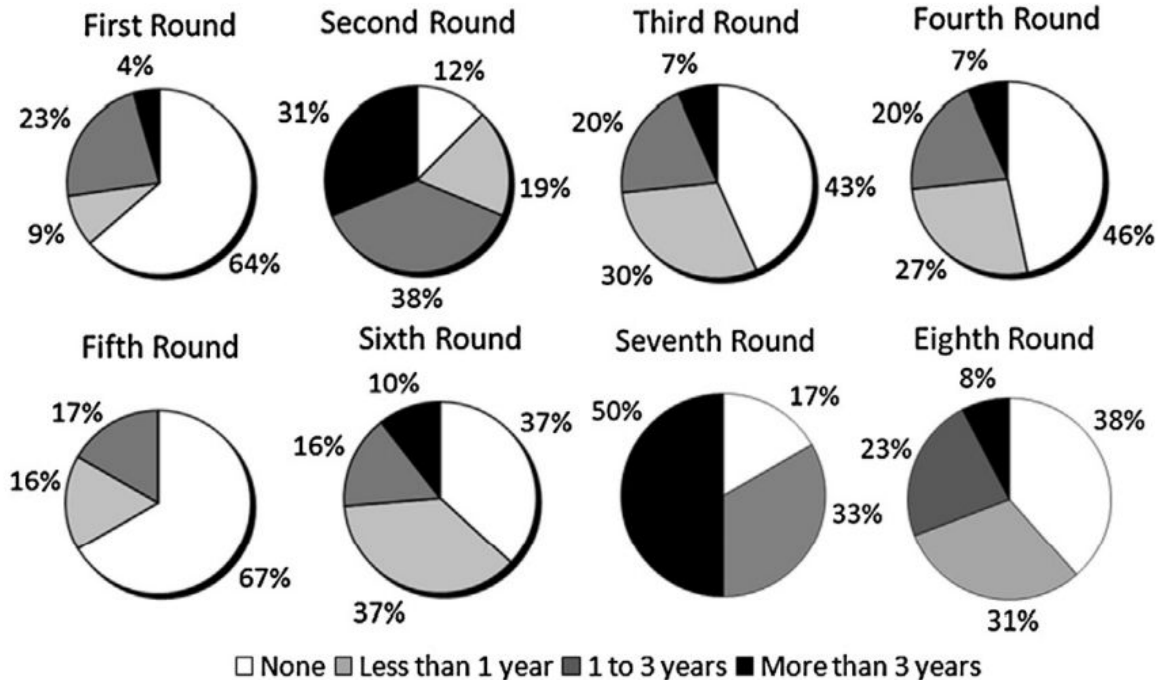
System	Round	Number of subjects	Grouping
ATM	First	11	Pairs
	Second	16	Individually
Chess	Third	15	Pairs
	Fourth	15	Individually
Telecom	Fifth	3	Pairs
	Sixth	19	Individually
Task Manager	Seventh	6	Pairs
	Eighth	13	Individually
Total number of subjects		98	

Number of subjects in each round.

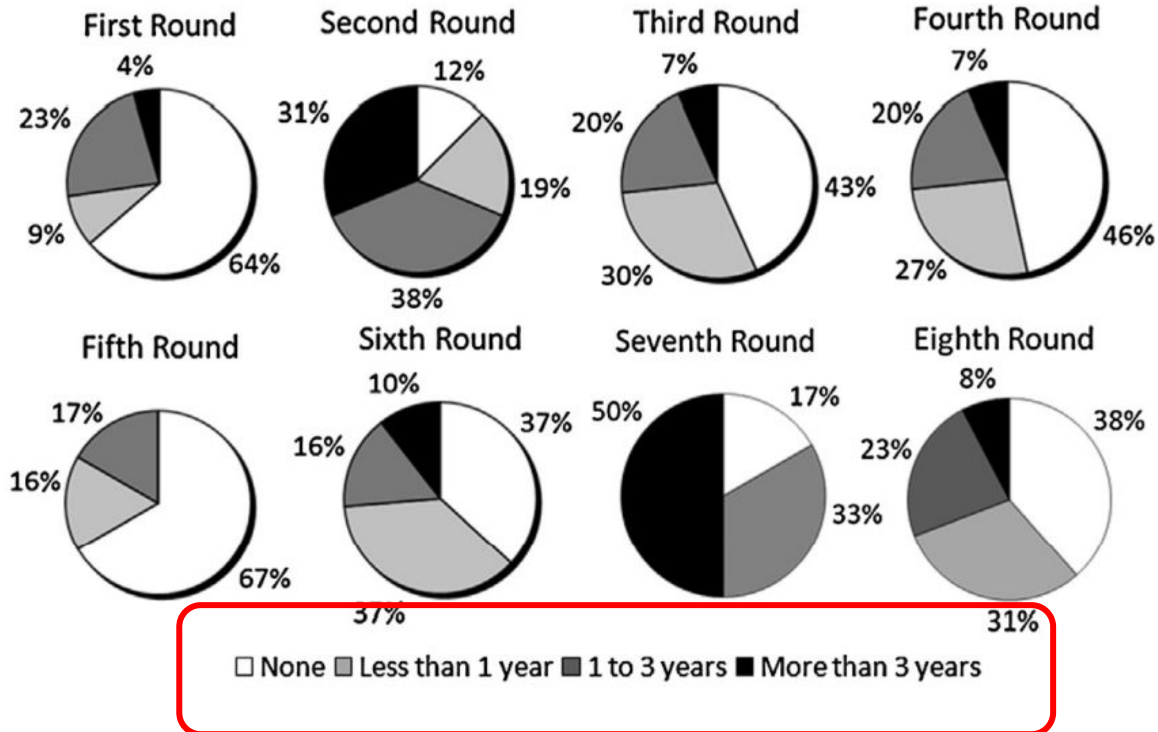


System	Round	Number of subjects	Grouping
ATM	First	11	Pairs
	[illegible]	[illegible]	[illegible]ually
Chess	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]ually
Telecom	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]ually
Task Manager	Seventh	[illegible]	Pairs
	Eighth	[illegible]	Individually
Total number of subjects		98	

Background dos participantes (POO)



Background dos participantes (POO)

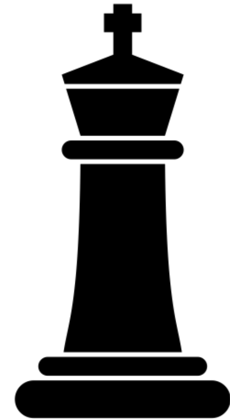


Aplicação ATM

- Simula funcionalidade de uma máquina de dinheiro (ver saldo, depositar, sacar).
- 19 LOC
- 4 módulos
- Interesse: Logging

Aplicação Chess

- Jogo de xadrez com interface gráfica.
- 36 LOC
- 8 classes
- Interesse: ErrorMessages

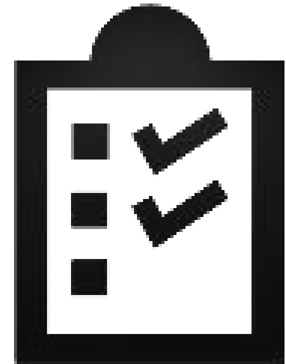


Aplicação Telecom

- Sistema de gerenciamento de conexões para chamadas telefônicas. Simula uma chamada entre dois ou mais clientes.
- 32 LOC
- 4 módulos
- Interesse: Timing

Aplicativo Task Manager

- Aplicativo para gerenciamento de tarefas.
- 22 LOC
- 3 módulos
- Interesse: Observer design pattern



Etapas

- Refatorações usando a linguagem de programação AspectJ.
- Treinamento: 2 horas
  - Conceitos básicos sobre AOP e AspectJ.
- Cada participante foi convidado a refatorar para aspecto um interesse de uma aplicação, usando o plug-in Eclipse AJDT.
- Recebeu o aplicativo e a descrição do interesse.

Análise dos Resultados

Etapa I - Classificação dos Erros Recorrentes

Lógica de Incorreta

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

(a) Before refactoring

```
public aspect Logging {

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

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

(b) Faulty code after refactoring

Lógica de Incorreta

```
public class ATM {
  private st
  void perfo
    ...
    while (!
      ...
      switch
        ...
        case LOG:
          Logger.printLog();
          break;
      }
    }
  }
  ...
}
```

(a) Before refactoring

```
                       nsactions() :
                       ansactions());

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

(b) Faulty code after refactoring

Mensagens de log são exibidas sempre.

Tipo de Advice Incorreto

```
public class BankDatabase {

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

```
  public aspect Logging {

    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);
    }
    ...
  }
```

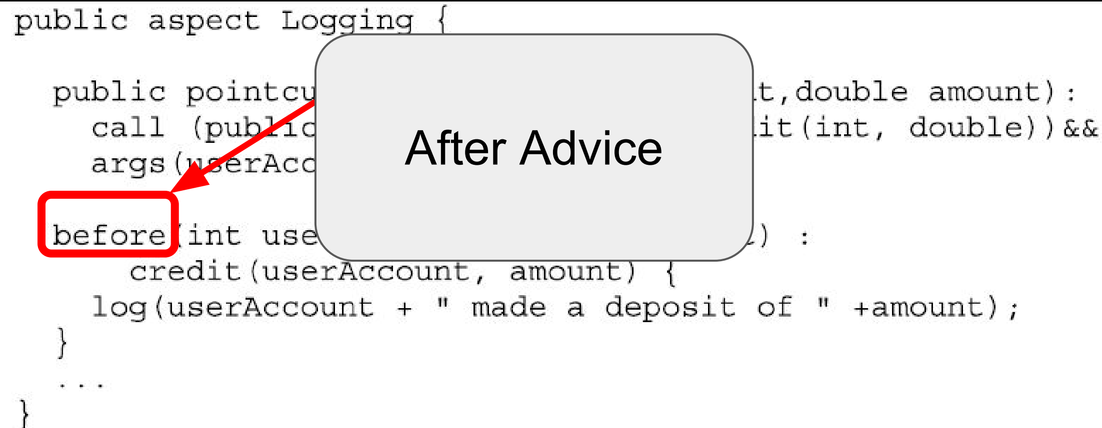
Tipo de Advice Incorreto

```java
public class BankDatabase {

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

```java
public aspect Logging {

  public pointcut credit(int userAccoun
    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);
  }
  ...
}
```

Tipo de Advice Incorreto

```
public class BankDatabase {

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



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

Refatorações em Excesso

```
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

Refatorações em Excesso

```
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();
  }
  ...
}
```

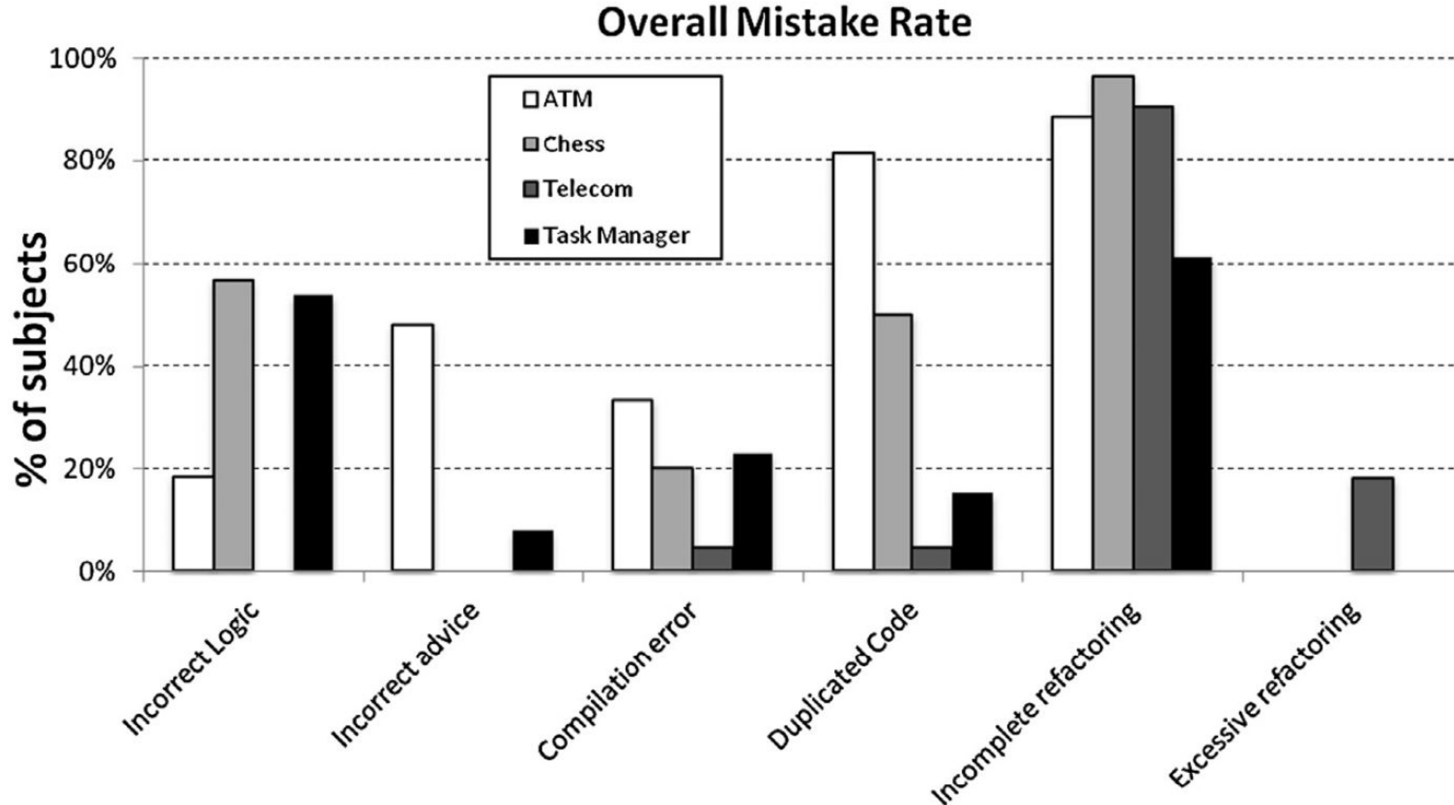
(b) After refactoring

Não pertence ao interesse

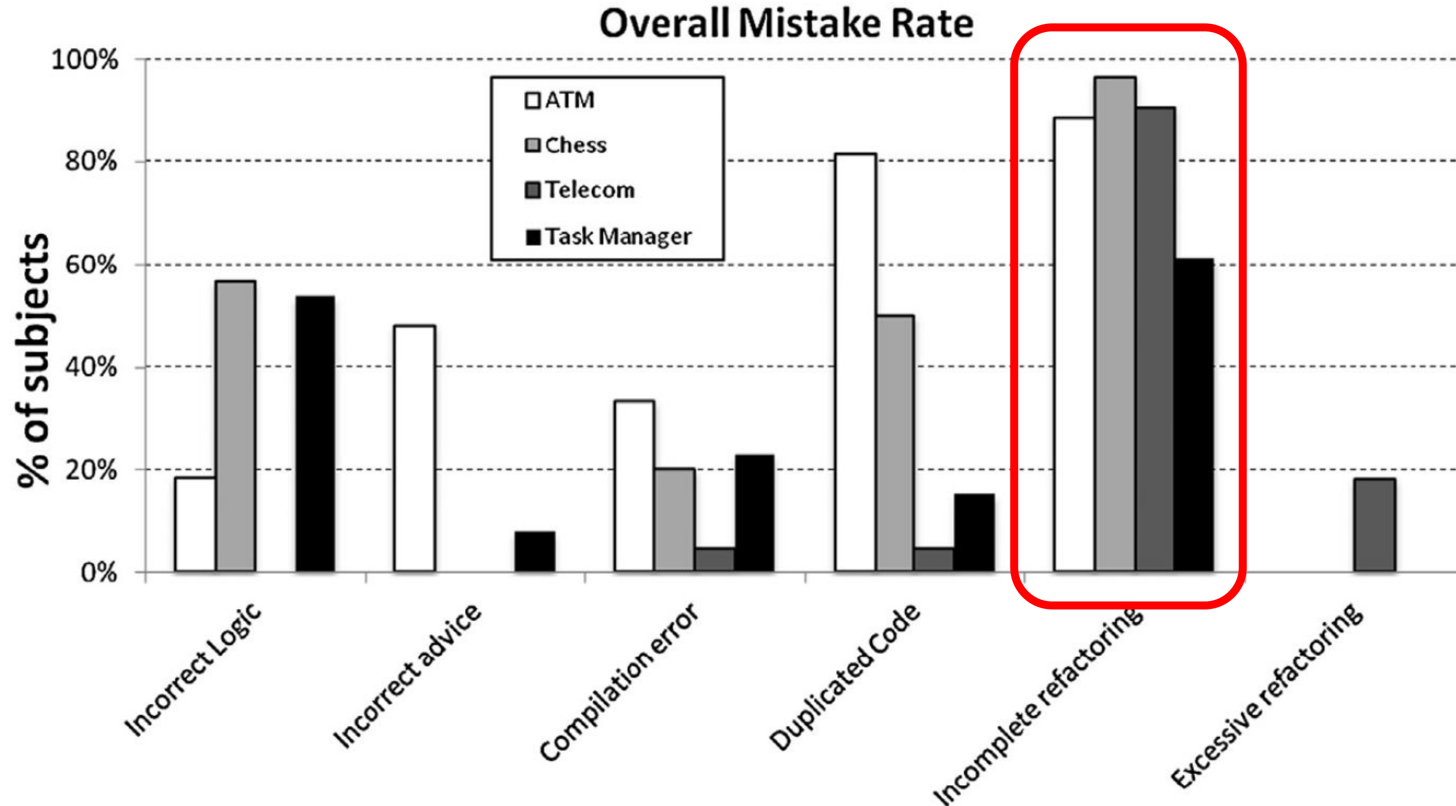
Análise dos Resultados

Etapa II - Erros por Nível de Conhecimento e Aplicação

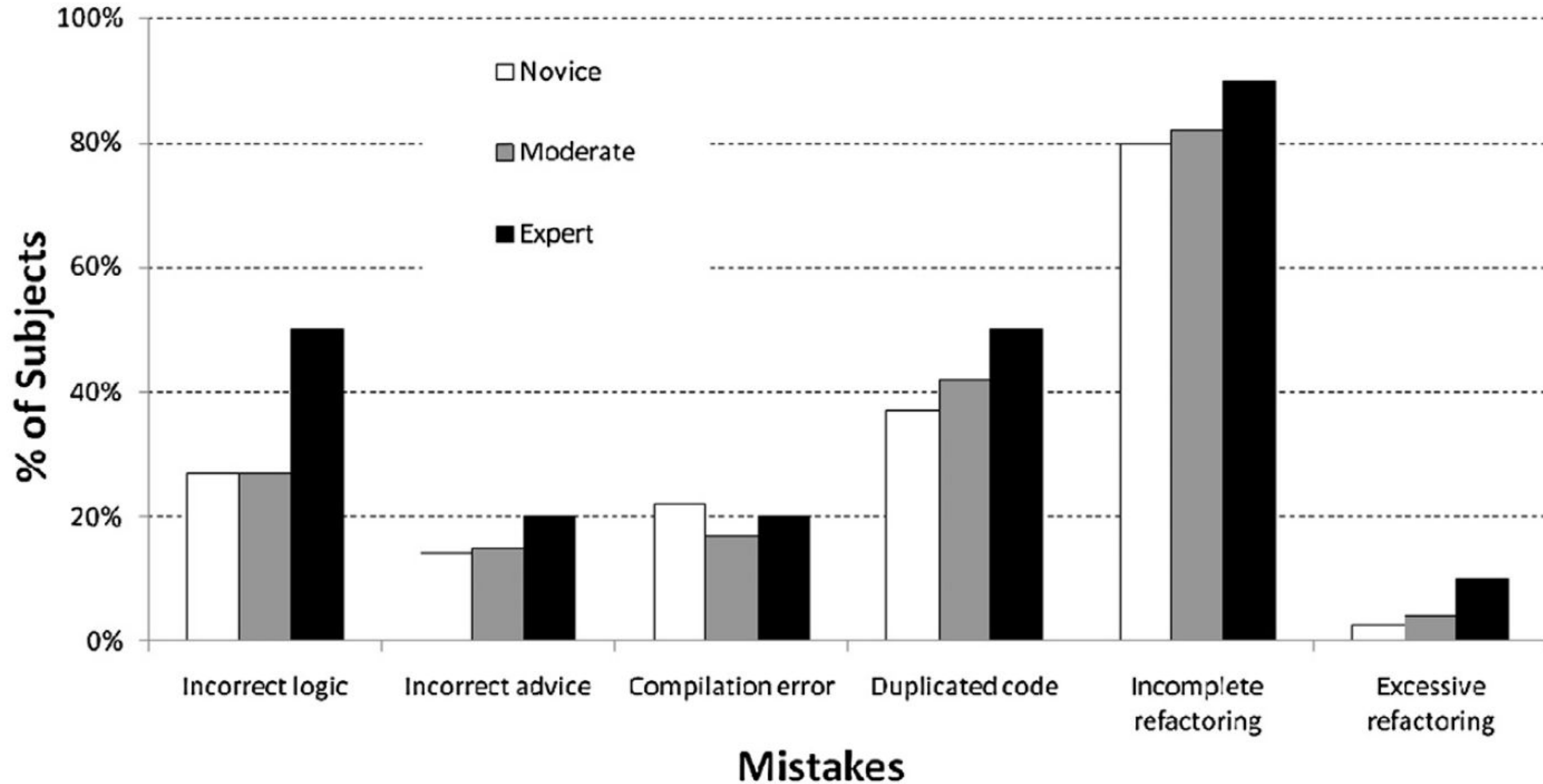
Erros por Aplicação



Erros por Aplicação

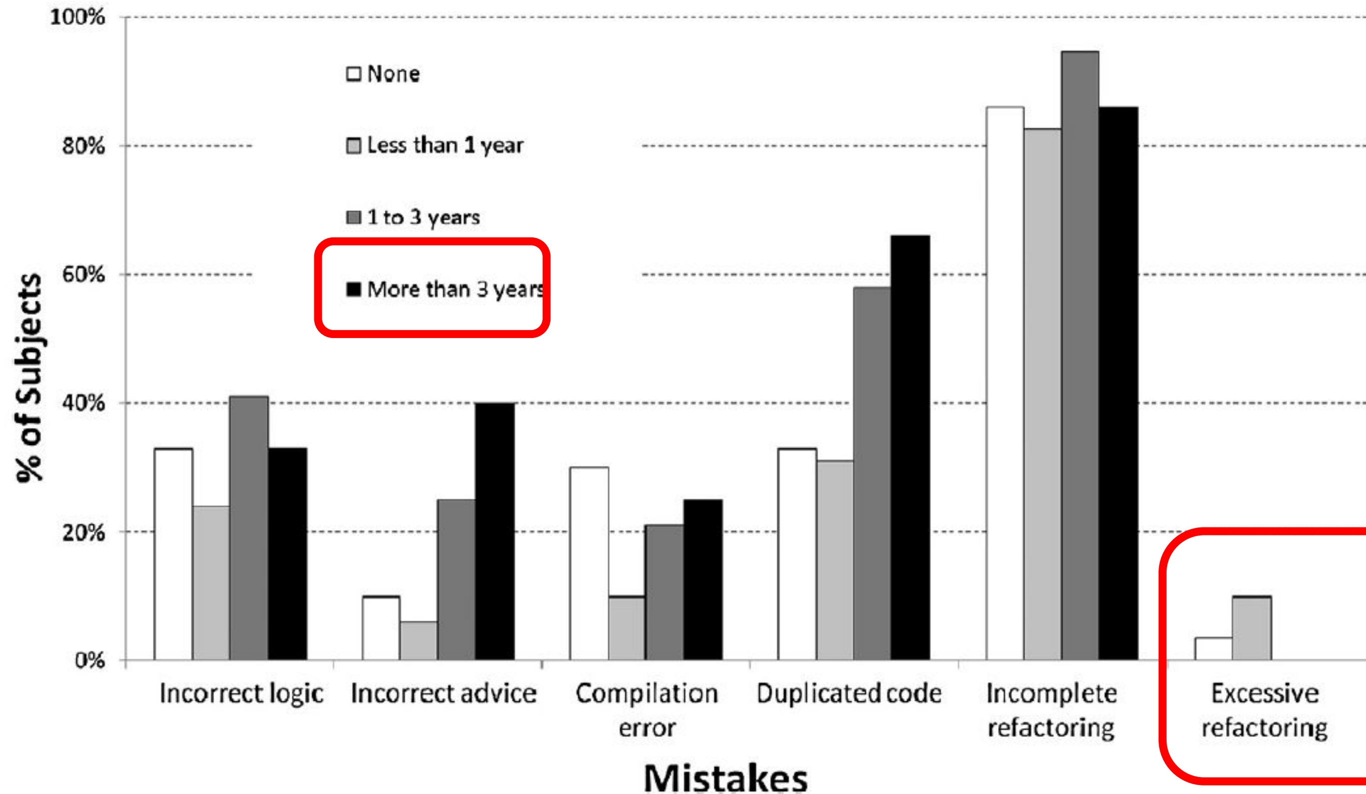


Erros por Nível de Experiência em POO

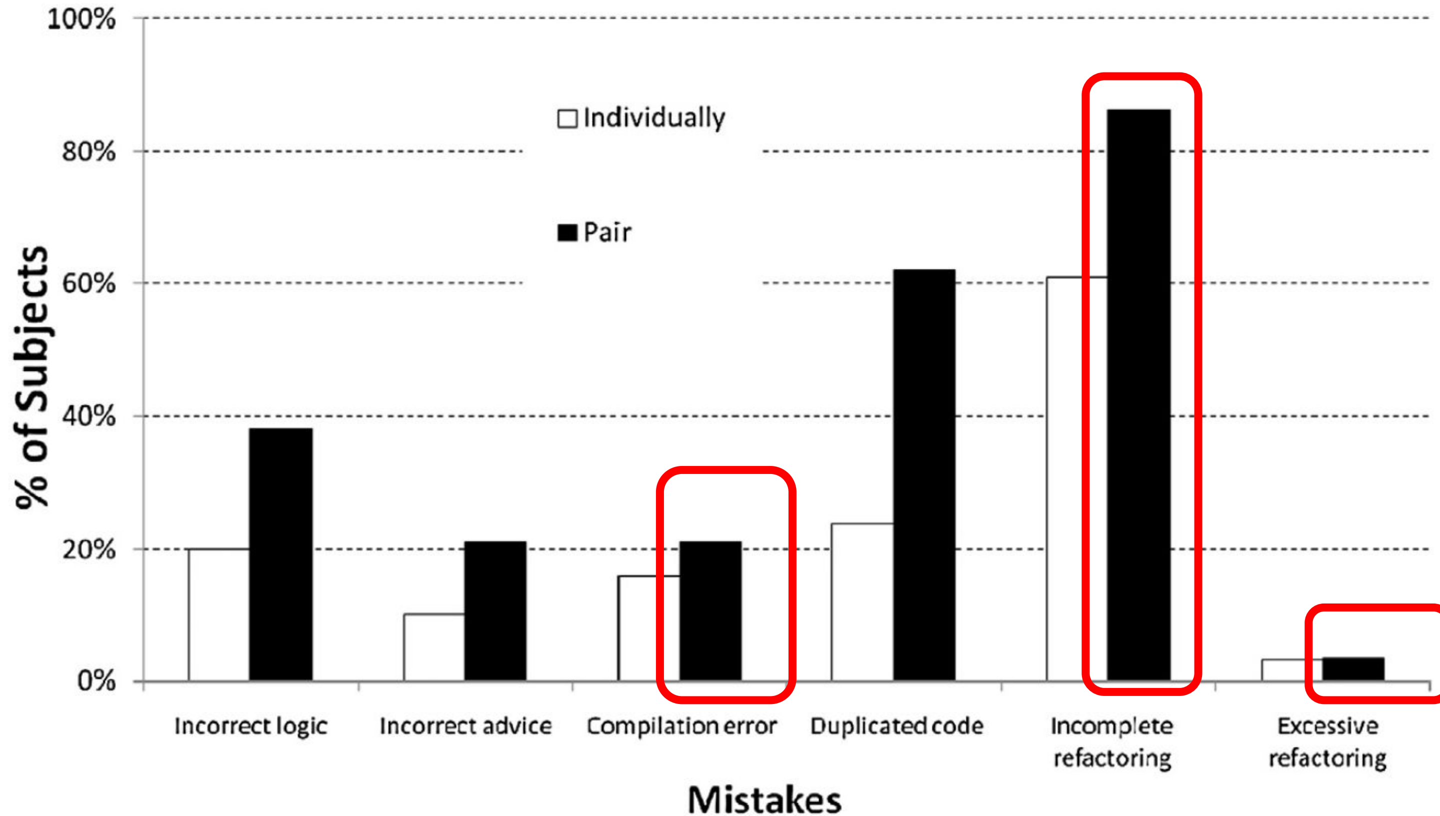


Erros por Nível de Experiência em POO

Erros por Nível de Experiência Profissional



Programação em Pares



Análise dos Resultados

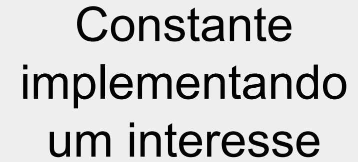
Etapa III - Armadilhas AOP

Constantes Primitivas

```
public class ATM {

    private static final int LOG= 0;
    ...
}
```

Constantes Primitivas

```
public class ATM {

    private static final int LOG= 0;
    ...
}
```



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

Atributos Non-Dedicated

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

(a) Attribute of a dedicated type

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

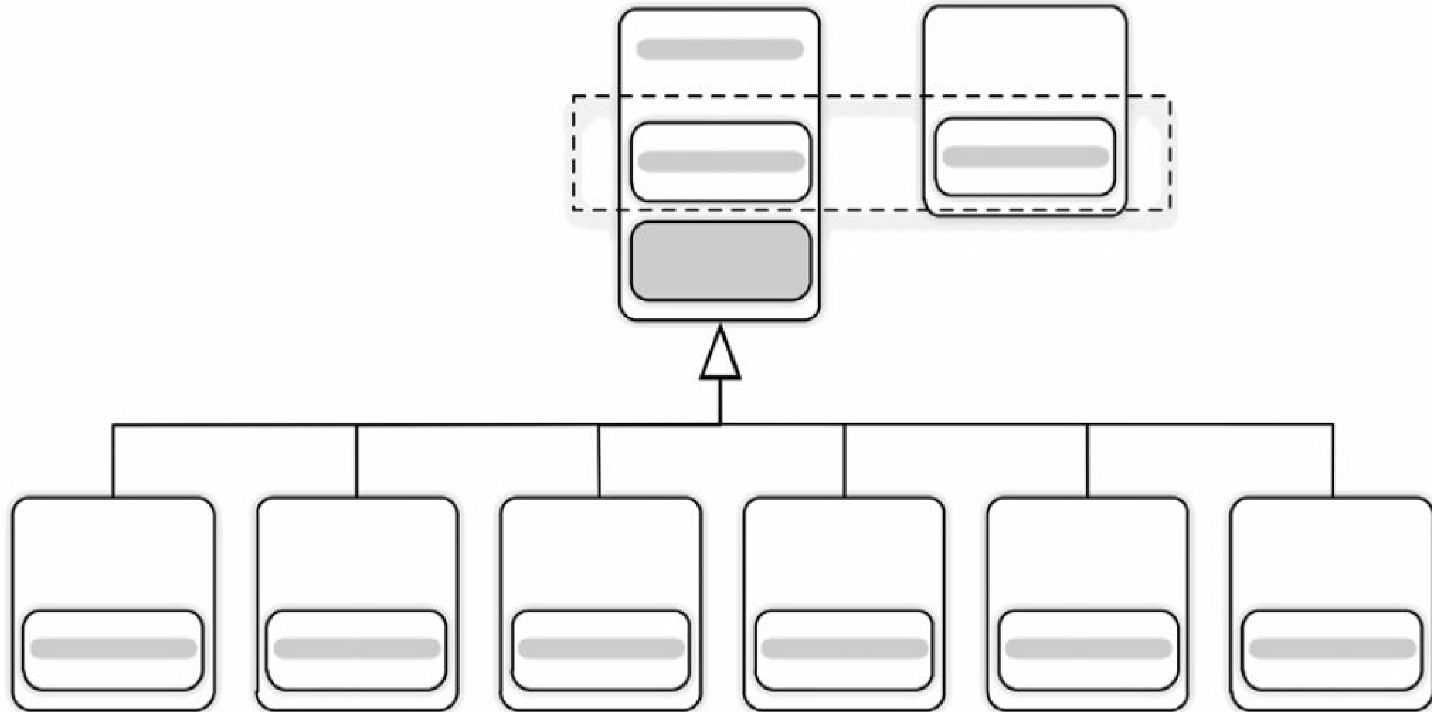
(b) Attribute of a non-dedicated type

Totalmente dedicado ao interesse.

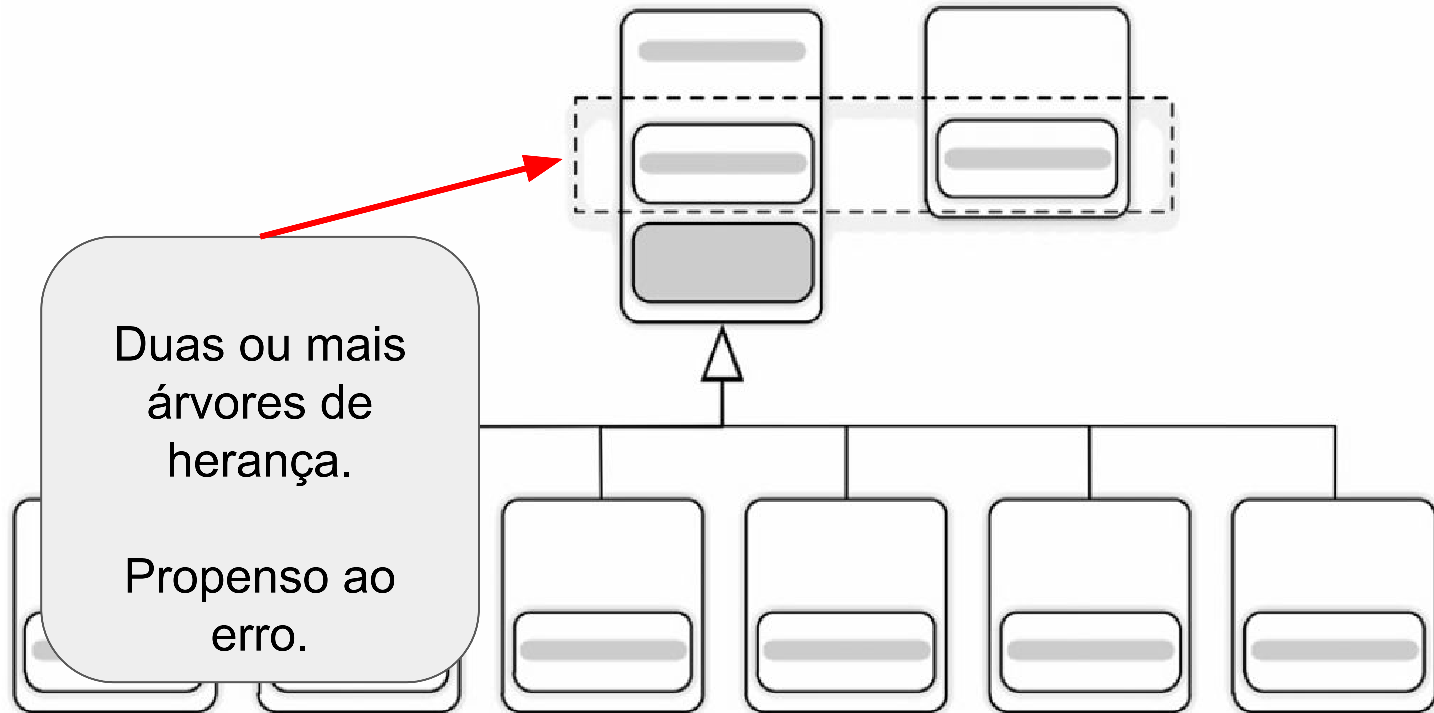
Sem erros.

Tipo primitivo

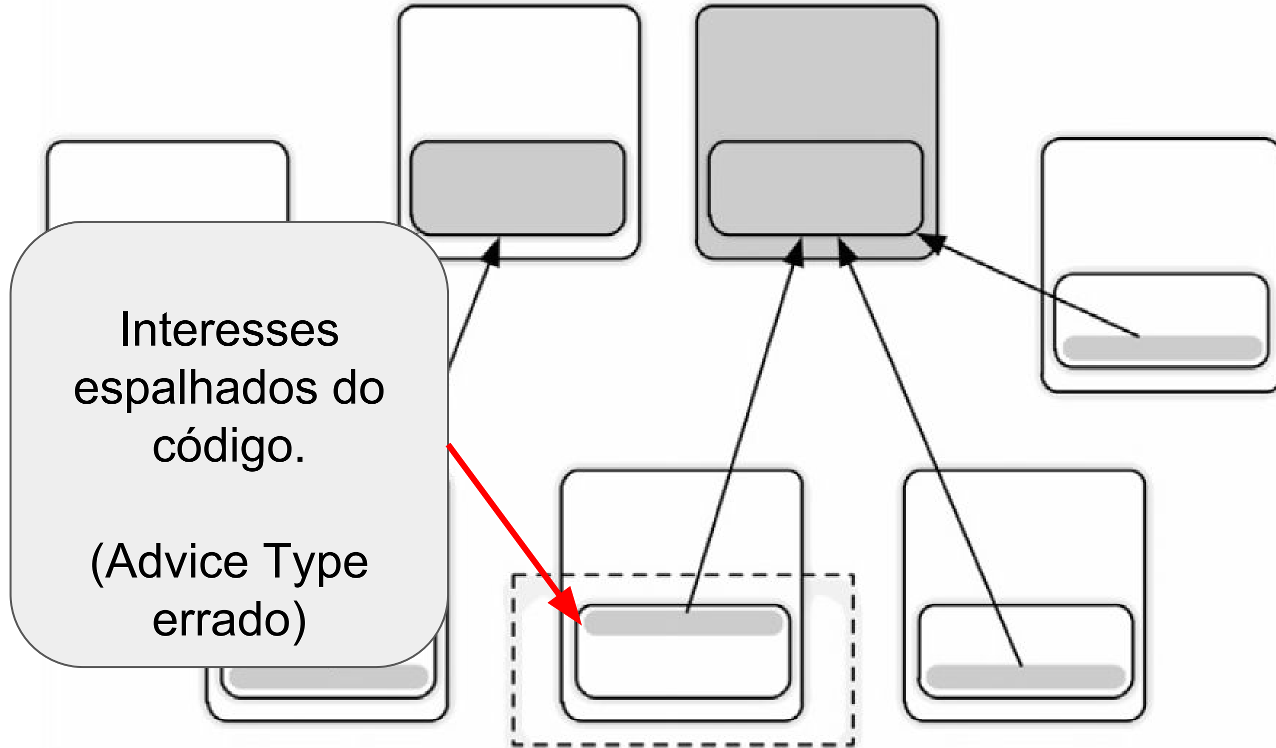
Árvores de Herança Disjuntas



Árvores de Herança Disjuntas



Localização



Variáveis Locais

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

Variáveis Locais

```
public class ATM {
  private void performTransactions() {
    ...
    while(!userExited) {
      int mainMenuSelection = displayMainMenu();
      switch(mainMenuSelection) {
        ...
        cas
          L
          b
        ...
      }
    }
  }
  ...
}
```

Interesse dentro de um método que acessa variável local.

Código mais difícil de refatorar.

Ferramenta ConcernReCS2

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

Ameaças à Validade

- Uso da linguagem AOP (AspectJ). Existem outras técnicas capazes de realizar os mesmos conceitos.
- Interesses selecionados.
- Dimensão do estudo.

Considerações Finais

- Documentação dos erros recorrentes.
- Programação em pares pode ajudar os programadores a evitar alguns erros. Exemplo: Refatoração Incompleta.
- Resultados podem ajudar a aperfeiçoar novas ferramentas.
- Ferramenta ConcernReCS para detecção de armadilhas no código.
- Erros de compilação são mais comuns entre os programadores mais experientes.