


# **ENGENHARIA DE SOFTWARE**

## **MEDIÇÃO E QUALIDADE DE SW**

How do Programmers learn AOP?

Péricles Alves, Alcemir Santos,  
Eduardo Figueiredo e Fabiano Ferrari

Aluno: Adriano Lages dos Santos



*“Toda descoberta da ciência é potencialmente subversiva; por vezes a ciência deve ser tratada como um inimigo possível”.*

Aldous Huxley

1894 – 1963

Escritor Britânico

# Agenda

- Introdução.
- Objetivos do estudo.
- Classificação de erros recorrentes em AOP.
- Dados coletados e Análises.
- Conclusões e trabalhos futuros.

# Introdução

- Programação Orientada a Aspectos (AOP).
  - Interesses transversais - > Aspectos.
- Técnica em amadurecimento.
- Requer experiência dos programadores.
- Vários estudos [1, 2, 6, 8, 19] são conduzidos para encontrar taxonomias de falhas e padrões de bug em AOP.

# Introdução - AOP

- Join Points - Representam pontos bem definidos em uma execução de um programa. Joinpoints típicos em AspectJ podem ser por exemplo chamadas de métodos, acessos a membros de uma classe entre outras ações de execução. Join points podem conter outros Join points.
- Pointcut é uma construção de linguagem que junta um conjunto de Join Points baseando-se em um critério pré-definido.

# Introdução - AOP

- Advice é o trecho de código que é executado antes `before()`, depois `after()` e simultaneamente `around()` a um Joinpoint.

```
import javax.swing.JOptionPane;

public aspect Aspecto {

    pointcut metodo(): execution(public static void main(..));

    before() : metodo() {
        JOptionPane.showMessageDialog(null, "Antes do Main!");
    }

    after() : metodo() {
        JOptionPane.showMessageDialog(null, "Depois do Main!");
    }
}
```

Ponto de atuação

Os advices são formados pelo ponto de atuação e pelo seu corpo, neste exemplo temos dois, o **before** e o **after**

O Aspecto é formado por todo o conteúdo.



# Objetivos do Estudo

- Investigar tipos de erros cometidos em AOP.
- Objetivo principal é identificar e classificar as categorias mais recorrentes de erros cometidos pelos programadores que estão aprendendo esta técnica de programação.

# Hipóteses de Pesquisa

- H0 = A experiência dos programadores que estão aprendendo AOP impacta nos erros que eles cometem.
- H1 = A experiência dos programadores que estão aprendendo AOP não impacta nos erros que eles cometem.



# Avaliação das Hipóteses

- Categorias de Erros de AOP.
- Três critérios de experiência dos programadores: Experiência específica em OOP, experiência de trabalho e idade.
- 47 estudantes de graduação e profissionais juniores participaram do experimento.
- Estudo realizado em três rodadas.

# Preparação para coleta dos dados

- Treinamento de 2 horas para os participantes sobre aspect J.
- Primeira rodada – Os participantes participaram em pares (graduação e Juniores).
- Segunda rodada – Os participantes (graduação e Juniores) trabalharam individualmente.
- Terceira rodada – Os participantes trabalharam em pares (somente alunos de graduação).

# Preparação para coleta dos dados

- Primeira e segunda rodada – Aspectizar o interesse “Logging” na aplicação OO ATM.
- Terceira rodada – Aspectizar o interesse “ErrorMessages” na aplicação Xadrez “Chess”.
- Interesse dos pesquisadores em observar a aspectização de interesses feitas por programadores trabalhando em pares.

# Preparação para coleta dos dados

- Todos os participantes preencheram um questionário.
  - Idade, experiência profissional e nível de conhecimento em tecnologia OOP.
- Quanto tempo trabalharam com desenvolvimento de software.
  - (i) Nunca trabalhou, (ii) trabalhou menos que seis meses, (iii) trabalhou entre seis meses e um ano, (iv) trabalhou entre um e três anos e (v) trabalhou mais que três anos.

# Preparação para coleta dos dados

- Nível de conhecimento em tecnologia OOP.
  - (i) Nunca programou em OOP, (ii) programou algumas vezes, (iii) programou várias vezes e (iv) muito familiarizado com OOP.

# Aplicações Alvo

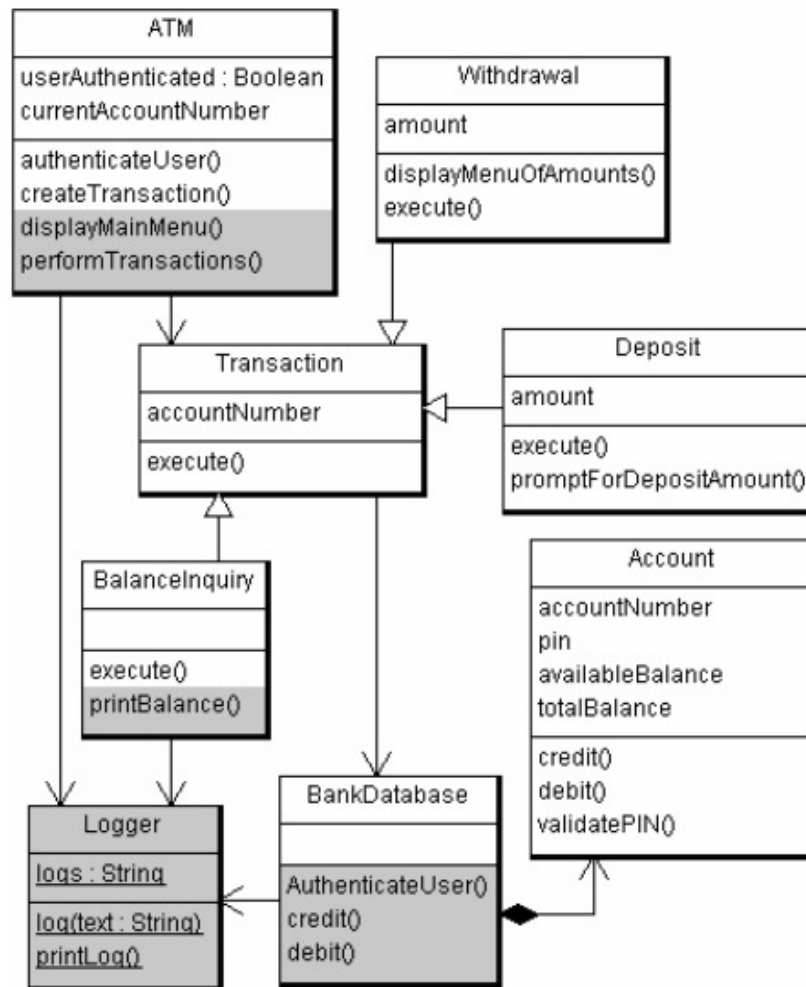


Figure 1. A simplified class diagram of the OO ATM application.

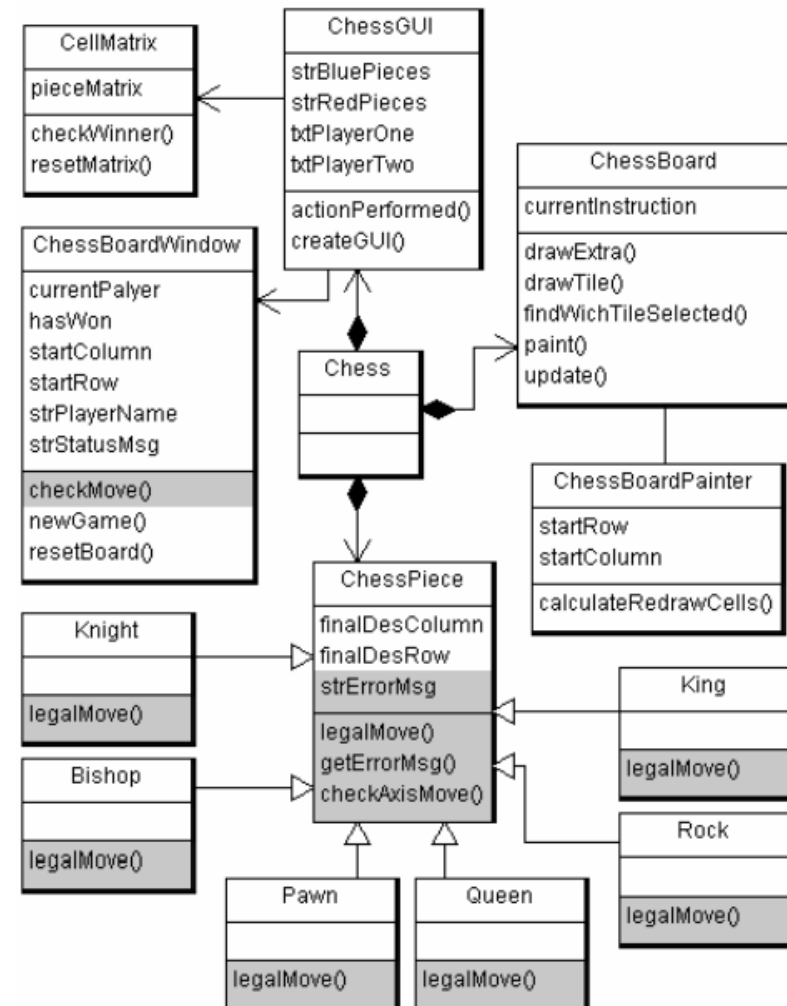


Figure 2. A simplified class diagram of the OO Chess application.



# Aplicações Alvo

- As aplicações foram escolhidas para permitir que o experimento fosse completado em uma aula de uma hora e 30 minutos.

TABLE I. SIZE METRICS OF THE APPLICATIONS AND CONCERNS

	<i>Application Size</i>		<i>Concern Size</i>	
	<i>NOM</i>	<i>LOC</i>	<i>CDC</i>	<i>LOCC</i>
ATM / Logging	12	606	4	19
Chess / ErrorMessage	13	1011	8	36

- NOM – Número de Módulos, LOC – Linhas de código, CDC – Espalhamento do interesse sobre os componentes, LOCC – Linhas de código do interesse.

# Classificação dos erros recorrentes em AOP

- Erros relacionados a “Advices”
  - A.L – Lógica de Implementação incorreta.
  - A.I – Tipo de Advice incorreto.
- Erros relacionados a compilação
  - C.E – Erro de Compilação ou warning.
- Outros erros
  - R.D – Código do interesse duplicado no aspecto e no código base.
  - R.I – Refatoração Incompleta.

# Dados coletados e Análises

- Número geral de erros

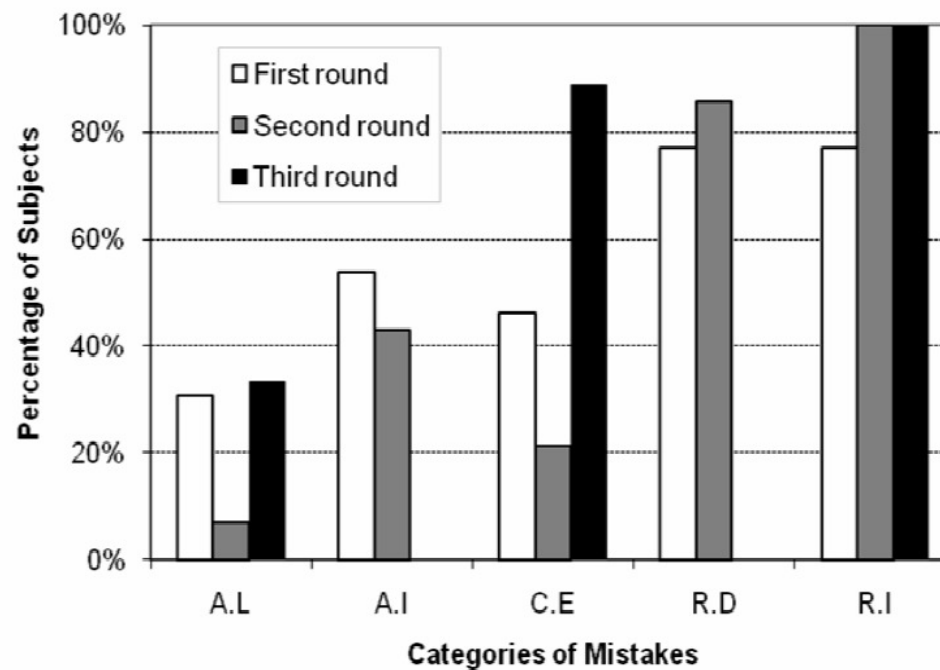


Figure 3. Percentage of subjects that made mistakes per category.

# Dados coletados e Análises

- Todos os participantes da segunda e terceira rodada cometeram erro de refatoração incompleta.
- Os alunos de graduação que trabalham em pares cometeram mais erros relacionados com advices e erros de compilação (AI, AL e CE) do que os participantes já graduados trabalhando individualmente.
- Este resultado sugere que graduandos não podem compreender facilmente como funciona um Advice em AspectJ.
- Os profissionais juniores tiveram um desempenho pior do que os participantes da graduação em relação a erros relacionados a refatoração. Possivelmente devido a primeira rodada em que juniores e alunos de graduação trabalharam em pares.

# Dados coletados e Análises

- Nível de experiência em OOP.
  - Indivíduos divididos em dois grupos: (i) (12) indivíduos com pouca ou nenhuma experiência e (ii) (24) indivíduos com moderada a alta experiência.

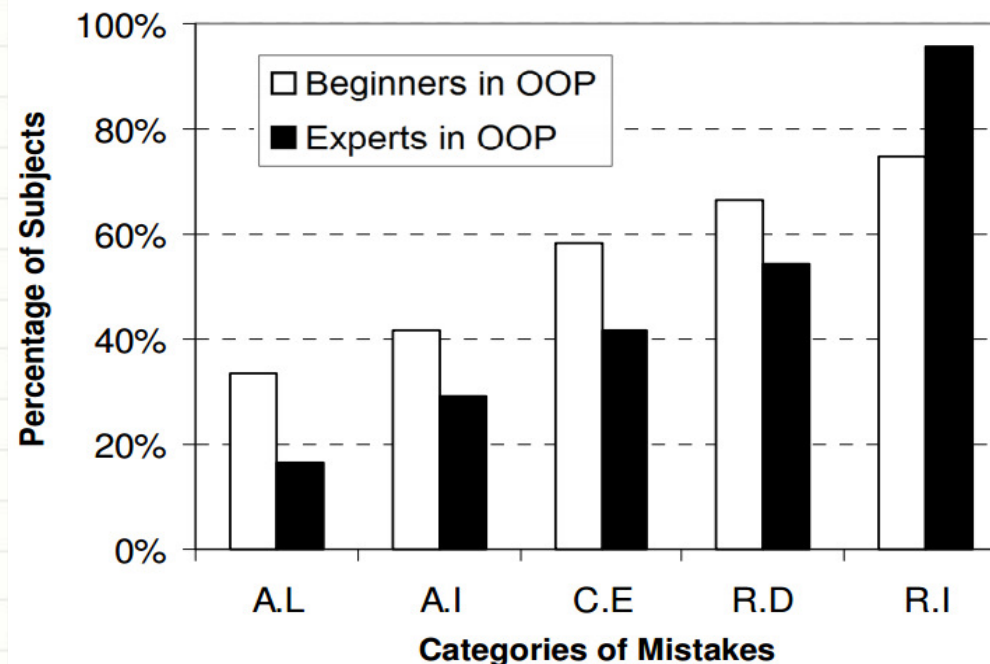


Figure 4. OOP experience and mistakes made by subjects.

# Dados coletados e Análises

- Como esperado, quanto mais experientes em OOP os indivíduos são, menos erros eles cometem. Isto é facilmente observado na Figura 4, uma vez que indivíduos experientes cometem menos erros do que os iniciantes em todas as categorias, exceto Refatoração Incompleta (RI). A explicação para este resultado pode ser o fato de que os experientes em OOP tendem a usar muito as abstrações OOP que eles estão confortáveis. Ou seja, eles tendem a não fatorar a aspectos vários pedaços de código em favor do uso de seu melhor conhecimento em OOP.



# Dados coletados e Análises

- Tipos de erro por experiência profissional:
  - (i) (21) Menos de um ano de experiência.
  - (ii) (15) Mais de um ano de experiência.

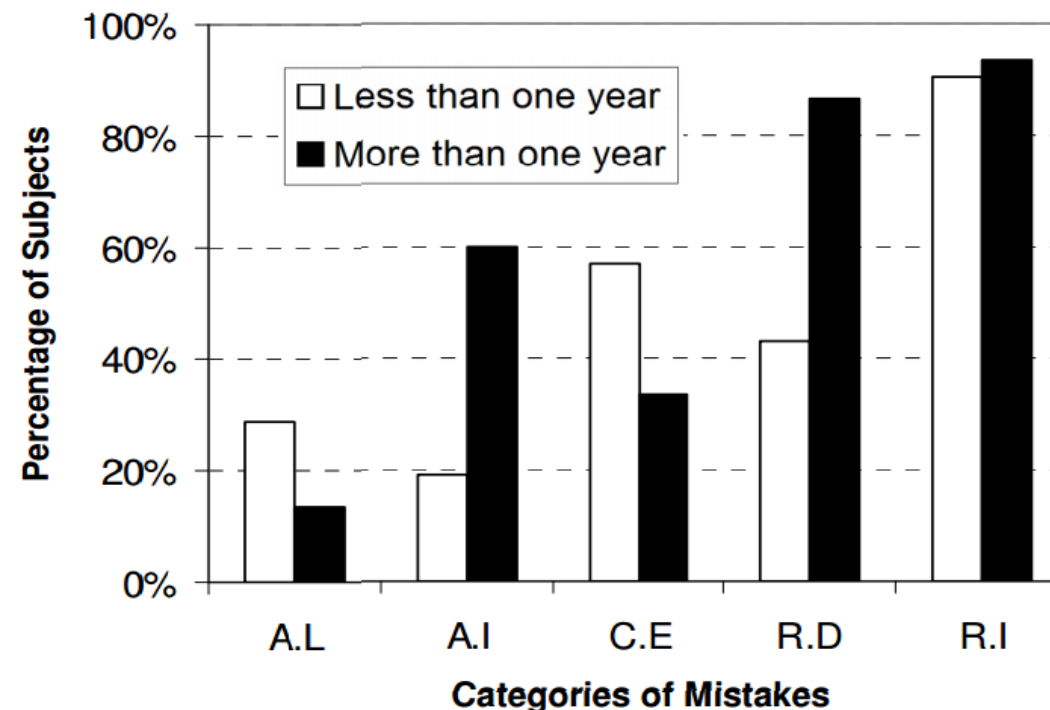


Figure 5. Work experience and mistakes made by subjects.

# Dados coletados e Análises

- Curiosamente, os resultados apresentados na Figura 5 não correspondem ao senso comum de que os desenvolvedores mais experientes cometem menos erros.
- Estes resultados sugerem que os desenvolvedores experientes cometem mais erros do que os iniciantes em pelo menos duas categorias: Incorrect Advice (AI) e Duplicated Crosscutting Code (RD).
- Para investigar mais este achado, verificamos que todos os indivíduos da terceira replicação foram classificados neste grupo (ou seja, menos de um ano de experiência). Além disso, esses indivíduos não têm a oportunidade de fazer esses dois tipos de erros (AI e RD), devido à natureza do interesse e da aplicação.

# Dados coletados e Análises

- Tipos de erro por faixa etária:

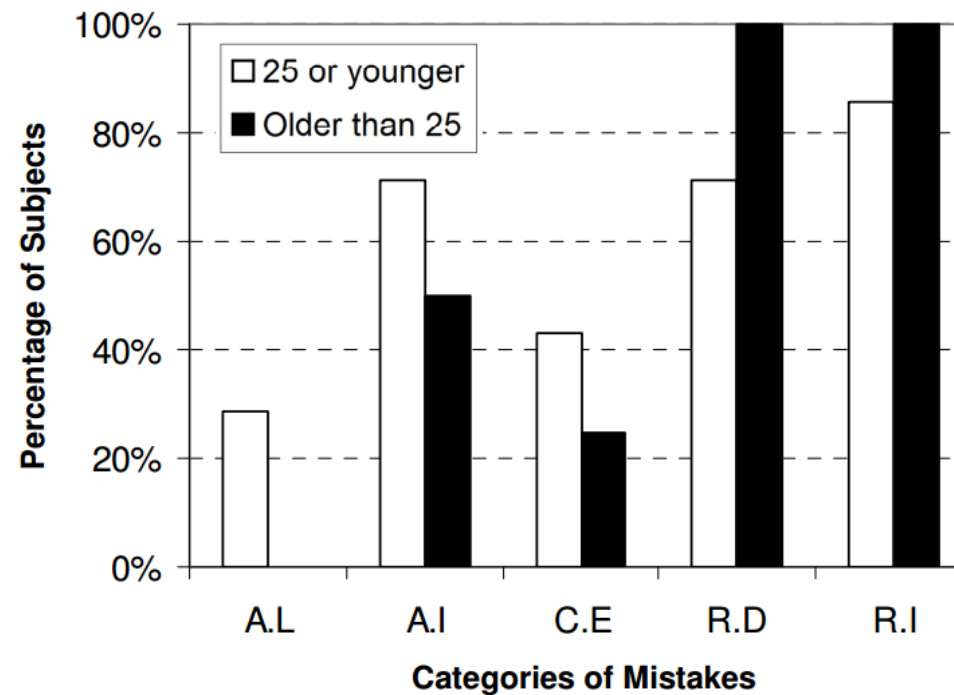


Figure 6. Age groups of developers with more tha 1 year work experience.

# Dados coletados e Análises

- Gráfico não conclusivo segundo os autores, pois somente 15 indivíduos do grupo 2 foram avaliados.
- Os dados deste gráfico indicam que os desenvolvedores mais jovens cometem mais erros do que os mais velhos nas três primeiras categorias (AL, AI, e CE). No entanto, os desenvolvedores mais velhos cometem mais erros de (RD e RI).
- Pelo fato de que esta observação é baseada apenas em 15 indivíduos, ela não é conclusiva. Outros estudos, como experimentos controlados, devem ser realizados para confirmar ou refutar nossos resultados preliminares.

# Conclusões

- Com um corpo crescente de conhecimento com relação a erros comuns em AOP, pesquisadores e profissionais podem empregar esforços para desenvolver ferramentas que automaticamente verificar a ocorrência de refatoração imprecisa ou analisar partes específicas do código com o objetivo de localizar falhas.
- Quando os programadores iniciantes são levados em conta, entender como eles cometem erros também pode melhorar a qualidade do ensino de programação em geral [17, 18], especialmente em AOP.

# Conclusões

- resultados indicam erros de AOP recorrentes feitos por programadores com experiências específicas e corroboram a nossa hipótese de investigação.
- Por exemplo, a programação em pares pode ajudar os programadores a evitar alguns erros, tais como refatoração incompleta, mas não ajuda a evitar os erros relacionados a Advices.
- Os resultados deste estudo podem ser utilizados para vários fins. Por exemplo, eles podem ajudar no desenvolvimento e aprimoramento de novas linguagens e ferramentas AOP.



# Conclusões

- Pesquisas futuras podem contar com as nossas configurações de estudo para realizar novas repetições do experimento, a fim de ampliar nosso conjunto de dados.
- Além disso, com base nas categorias identificadas de erros, uma ferramenta de suporte pode ser desenvolvida para aconselhar aos programadores sobre erros típicos de AOP.

# Referências Bibliográficas

- [1] R. T. Alexander, J. M. Bieman, and A. A. Andrews, "Towards the systematic testing of aspect-oriented programs," Dept. of Comp, Science, Colorado State Univ., Report CS-04-105, 2004.
- [2] J. Baekken and R. Alexander. "A Candidate Fault Model for AspectJ Pointcuts". In proceedings of the 17th Int'l Symposium on Software Reliability Engineering (ISSRE), pp. 169-178. Raleigh, USA, 2006.
- [6] R. Coelho, A. Rashid, A. Garcia, F. Ferrari, N. Cacho, U. Kulesza, A. Staa and C. Lucena. "Assessing the Impact of Aspects on Exception Flows: An Exploratory Study". In proceedings of the 22nd European conference on Object-Oriented Programming (ECOOP), pp. 207-234. Paphos, Cyprus, July 2008.
- [8] F. Ferrari, J. Maldonado and A. Rashid. "Mutation Testing for AspectOriented Programs". In proceedings of the 1st International Conference on Software Testing, Verification, and Validation (ICST), pp. 52-61. Lillehammer, Norway, April 2008.
- [17] E. Soloway and K. Ehrlich. "Empirical Studies of Programming Knowledge". IEEE Trans. on Softw. Eng. (TSE), 5, pp. 595-609. 1984.
- [18] J. Spohrer and E. Soloway. "Novice mistakes: are the folk wisdoms correct?". Communications of the ACM. vol. 29, issue 7. 1986.
- [19] S. Zhang and J. Zhao. "On Identifying Bug Patterns in Aspect-Oriented Programs". In proc. of the 31st Int'l Computer Software and Applications Conference (COMPSAC), 431-438. Beijing, China, 2007.



**Obrigado!**