

# Introdução a Testes Automatizados



**Cursos de Verão 2008 – IME/USP**

[www.agilcoop.org.br](http://www.agilcoop.org.br)

Paulo Cheque

# Testes Automatizados

- Teste de Software:
  - Executar o programa a ser testado com alguma entrada e conferir visualmente os resultados obtidos
- Teste Automatizado: **EXEMPLO**
  - Programa ou script executável que roda o programa a ser testado e faz verificações automáticas em cima dos efeitos colaterais obtidos.
- Testar **NÃO** é depurar
  - Testar é verificar a presença de erros
  - Depurar é seguir o fluxo para identificar um erro conhecido

# Testes Automatizados e os Princípios Ágeis

- **Software funcionando ...**
- **Adaptação a mudanças ...**
- **Colaboração com o cliente ...**
- **Indivíduos e Interações ...**

# Testes Automatizados e XP

- Entre as práticas mais importantes
  - A mais importante em muitos contextos
- Muitas das práticas dependem de testes automatizados:
  - Refatoração
  - Integração Contínua
  - Propriedade Coletiva do Código
  - Ainda: Design Simples, Releases Pequenos

# Roteiro

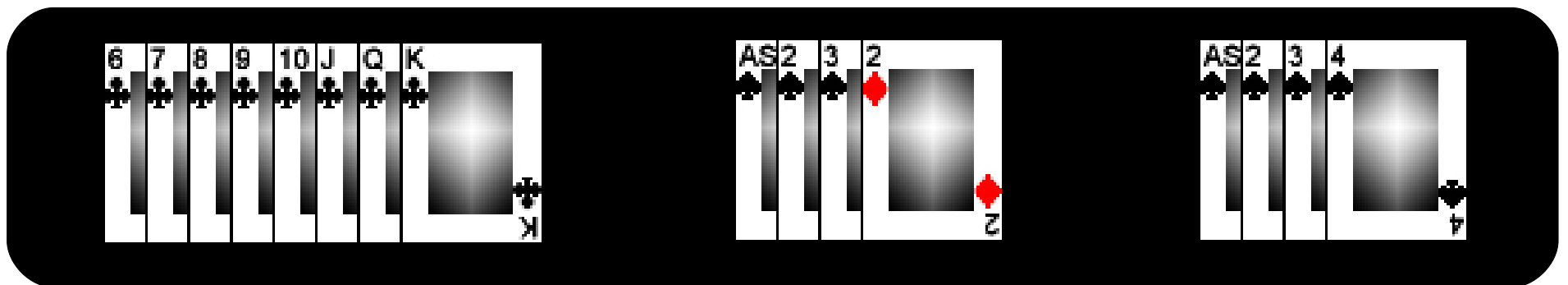
- 1) Motivação
- 2) Importância dos Testes Automatizados
- 3) Alguns Conceitos de Testes Automatizados
- 4) Alguns Tipos de Testes
- 5) Dicas para Escrever Bons Testes
- 6) Considerações Finais

# Exemplo de Teste Manual

## Jogo de cartas **Buraco**



Usuário só pode criar jogos válidos



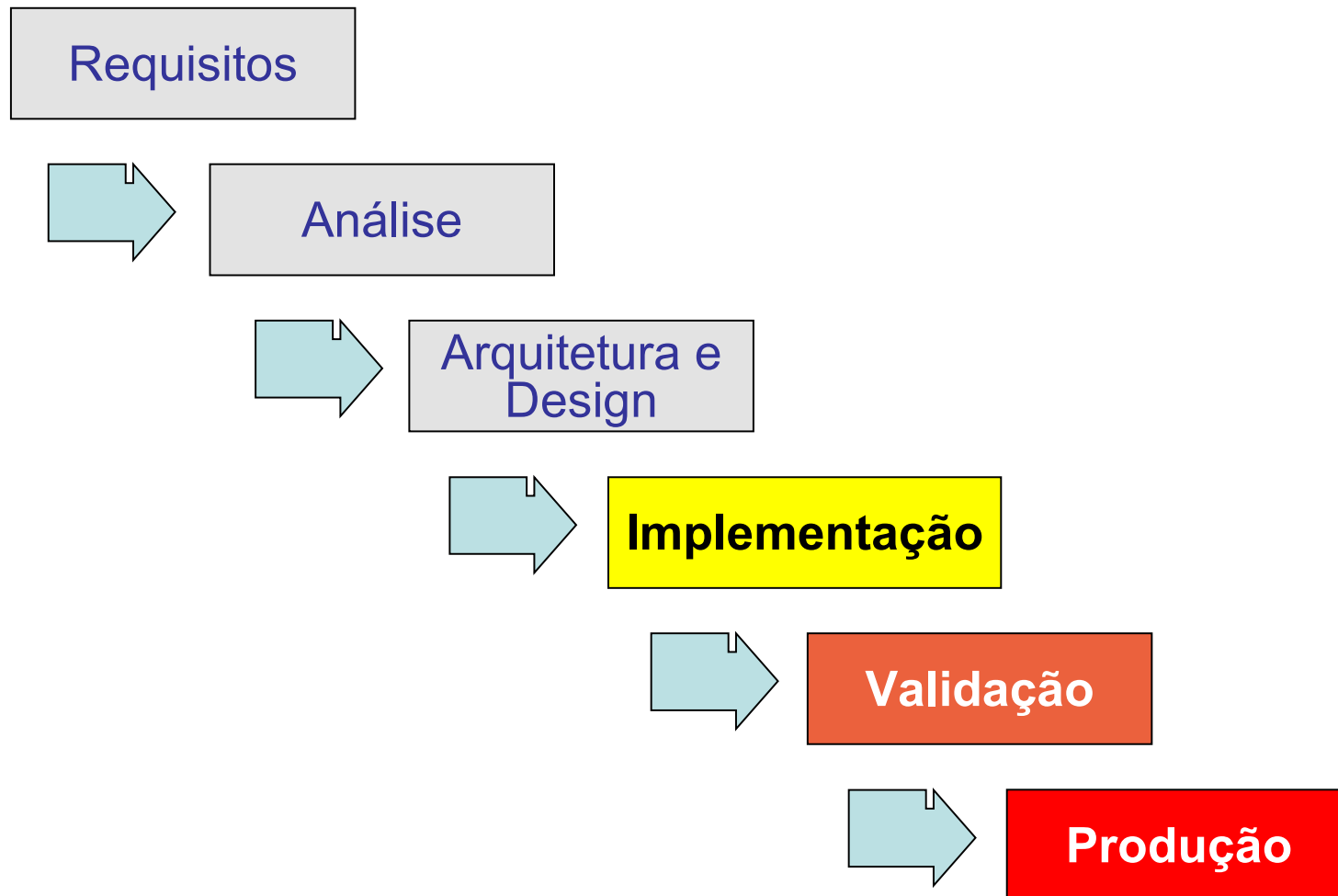
# Problemas

- Difícil repetir os testes
- Demorado
- Cansativo
- Executado poucas vezes
- Poucos casos
- Casos simples
- Muito tempo gasto com depuração
- Erros encontrados tardiamente:
  - Pressão, desconforto, estresse, perda de confiança
- Ciclo de adição de erros regressão



# Modelos “tradicionais”

Cascata, espiral...



# + Problemas

- Difícil repetir os testes
- Demorado
- Cansativo
- Executado poucas vezes
- Poucos casos
- Casos simples
- Muito tempo gasto com depuração
- Erros encontrados tardiamente:
  - Pressão, desconforto, estresse, perda de confiança
- Ciclo de adição de erros regressão

# + Problemas

- Difícil repetir os testes
- Demorado
- Cansativo
- Executado poucas vezes
- Poucos casos
- Casos simples
- Muito tempo gasto com depuração
- Erros encontrados tardiamente:
  - Pressão, desconforto, estresse, perda de confiança
- Ciclo de adição de erros regressão

# Solução

- Métodos Ágeis

*“Inspeccionar para prevenir defeitos é bom; Inspeccionar para encontrar defeitos é desperdício”*

-- Shigeo Shingo

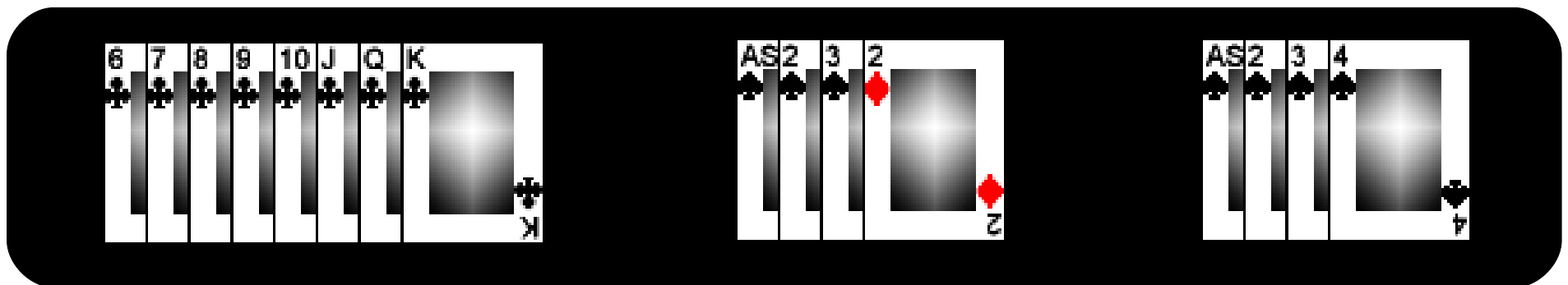
- Testes Automatizados

# Exemplo de Teste Automatizado

## Jogo de cartas **Buraco**



Usuário só pode criar jogos válidos



# Benefícios

- TODOS os testes podem ser executados a qualquer momento -> executado mais vezes
- Ajuda a encontrar erros mais cedo
- Aumenta a velocidade do desenvolvimento
- Melhora a qualidade do código, diminuindo o número de erros

Diminui o tempo gasto com depuração e correção  
Aumenta tempo gasto com a validação do software

# + Benefícios

- Segurança para mudanças no código:
  - Adição de novas funcionalidades
  - Correção
  - Refatoração
- Segurança para mudanças externas:
  - Infra-estrutura
  - Configurações de arcabouços
- Aumenta vida útil do software

# Roteiro

- 1) Motivação
- 2) Importância dos Testes Automatizados
- 3) Alguns Conceitos de Testes Automatizados**
- 4) Alguns Tipos de Testes
- 5) Dicas para Escrever Bons Testes
- 6) Considerações Finais

# Testes Codificados

- Códigos dos testes merecem a mesma atenção dos códigos do sistema (?!)
- Precisam de manutenção
- Testes podem conter erros
- O código **precisa** ser simples
- Deve ser 'legível' ↔ Documentação

# Set Up e Tear Down

- Os testes devem ser independentes:
  - Uns dos outros
    - Obs: padrão **Chained Tests**
  - Do número de vezes que é executado
  - De fatores externos

## EXEMPLO

- Set Up: Prepara o ambiente para a execução do teste
- Tear Down: Limpa / Remove o ambiente

# Conceitos Básicos

- Caixa-preta (Funcional)



- Caixa-branca (Estrutural)



- Caixa-cinza



# Roteiro

- 1) Motivação
- 2) Importância dos Testes Automatizados
- 3) Alguns Conceitos de Testes Automatizados
- 4) Alguns Tipos de Testes**
- 5) Dicas para Escrever Bons Testes
- 6) Considerações Finais

# Alguns Tipos de Testes

- Unidade

---

- Integração
- Interface de Usuário
- Aceitação

---

- Mutação

---

- Desempenho
- Estresse
- Segurança

# Testes de Unidade

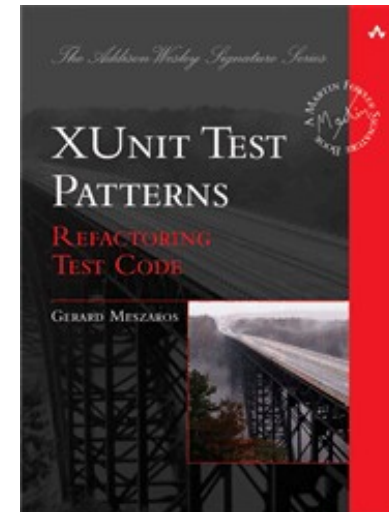
- Unidade: módulo/classe ou função/método

## EXEMPLO

- Teste básico e muito importante
  - Encontra muitos erros
- Não deve se preocupar com as responsabilidades de outras unidades
- Caixa-branca
  - Focando na funcionalidade

# Padrões de Testes de Unidade

- Data-Driven Tests
- Generated Value
- Testcase Class
- Testcase Class per Class
- Testcase Class per Feature
- Unfinished Test Assertion
- ...



# Teste de Integração

- Testa a integração entre unidades ou componentes
- Duas unidades podem funcionar perfeitamente individualmente, mas podem haver falhas quando elas se comunicam

# Testes de Interface de Usuário (GUI, WUI...)

- Simula eventos do usuário e verifica o estado da interface

**EXEMPLO** (FEST e FITNESS)

- Módulos distintos e isolados podem alterar o estado da interface
  - Estado pode ficar inconsistente
  - Sujeira na tela
- Integração ou Aceitação

# Testes de Aceitação

- Testa as funcionalidades do sistema a partir do ponto de vista do cliente

## EXEMPLO

- Caixa-preta
- **Data-Driven Test**
- Teste de alto nível
- “Prova” para o cliente de que o software funciona
- O cliente pode especificar os testes e os desenvolvedores os implementam

# Testes de Mutação

## Testes dos Testes

- *“Testes podem mostrar a presença de erros, não a sua ausência”*  
-- Dijkstra
- Infinitas possibilidades de erros?
- Hipótese: Programador competente + Efeito de acoplamento
- Mutante: Sistema sob teste com pequenas alterações
- Verificar o resultado dos testes e fazer uma análise dos mutantes que os testes não identificaram

# Testes de Desempenho

- Otimizar somente quando necessário
- Código claro traz mais benefícios do que milésimos de segundo de otimização
- Começar pelos gargalos da aplicação
- Utilizar *Profilers* para detectar os gargalos
  - Arcabouços mostram o tempo de duração dos testes

# Testes de Estresse

- Testar com grandes quantidades de dados gerados automaticamente
  - EXEMPLO (JMeter)
- Simulação de muitos usuários simultâneos
- Erros comuns:
  - Overflow
  - Falta de memória
- Alguns benefícios:
  - Encontrar vazamento de memória
  - Avaliar a capacidade de um ambiente

# Testes de Segurança

- Útil principalmente para aplicações Web
  - Aplicações expostas a usuários mal-intencionados
- Testar valores inesperados:
  - Null
  - Grande quantidade de dados
  - Tipos de dados não esperados  
ex: string em um campo que espera um inteiro
- Testes de estresse para lidar com ataques de negação de serviço (DoS)



# Roteiro

- 1) Motivação
- 2) Importância dos Testes Automatizados
- 3) Alguns Conceitos de Testes Automatizados
- 4) Alguns Tipos de Testes
- 5) Dicas para Escrever Bons Testes**
- 6) Considerações Finais

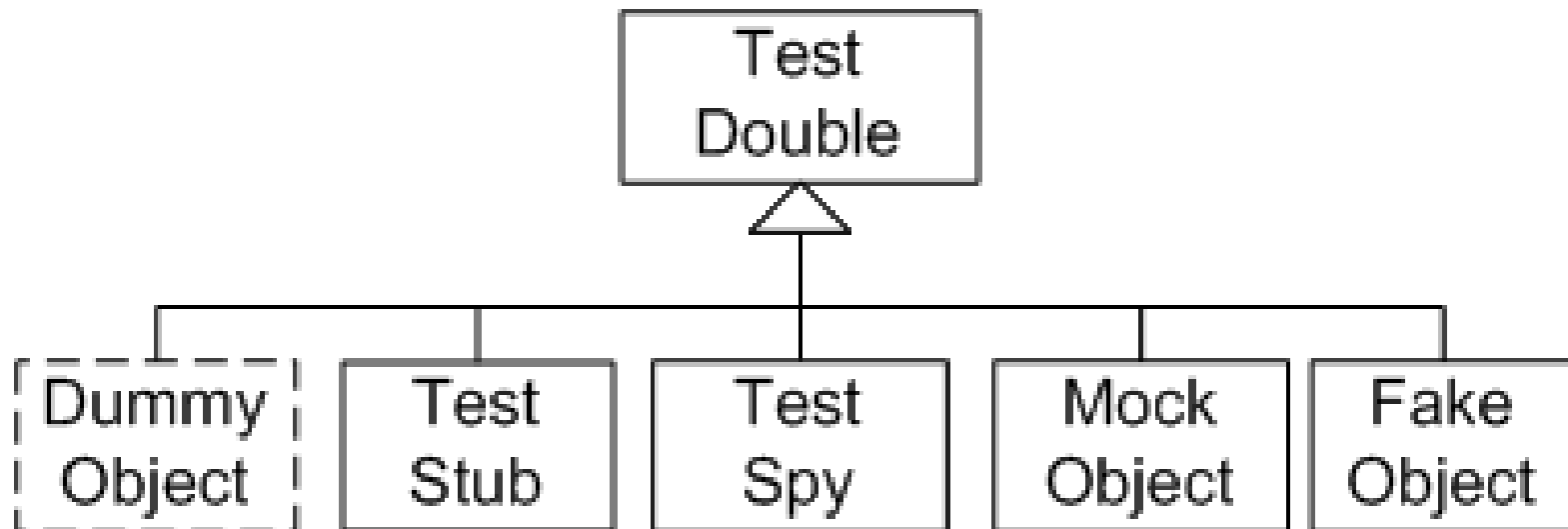
# Maus cheiros

- Código dos Testes:
  - Testes Obscuros
  - Código Complexo: condições
  - Difícil de testar
  - Replicação de código de testes
  - Lógica de testes em produção
- Cheiros de comportamento:
  - Testes frágeis
  - Intervenções manuais
  - Testes lentos



# Dublês

- Módulos com dependências:
  - não testadas
  - lentas
  - complexas
- => Crie um dublê (**Test Double**)



# Dicas Gerais

- Teste casos de sucesso
- Teste casos de erro:
  - Tratamento do erro
  - Exceção esperada

```
@Test
public final void testCPFValido() throws Exception {
    CPF cpf = new CPF("11122233344"); // Recebe 11 números
    assertEquals("111.222.333-44", cpf.formatar());
}
```

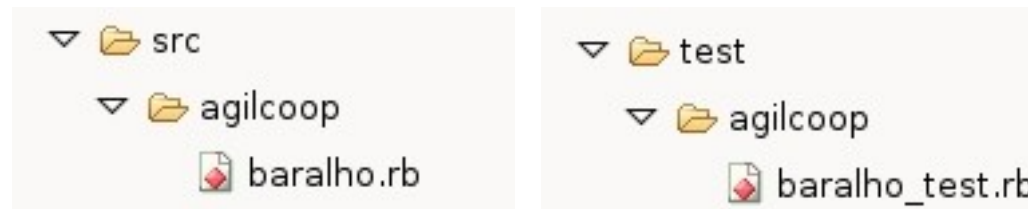
```
@Test(expected = CPFInvalidoException.class)
public final void testCPFInvalido() throws Exception {
    CPF cpf = new CPF("entradaInvalida");
    cpf.formatar();
}
```

# + Dicas gerais

- Particionamento de domínios
- Valores Limites:
  - Números: 0, -1, 1, muito pequenos, muito grandes
  - Strings: nula, vazia, pequenas, grandes
  - Coleções: nula, vazia, cheia
- Testes aleatórios

# Dicas p/ POO - Testando Métodos

- Protegidos: Organize os códigos dos testes seguindo a mesma hierarquia de pacotes do código do sistema (boa prática)



- Privados (Um alerta para o design): Utilize reflexão ou arcabouços próprios para isso
- Triviais (ex: get/set): Podem não ser testados

# + Dicas p/ POO - Testando Classes

- Internas: Testar a classe que usa é suficiente
- Abstratas: Instanciar uma subclasse e chamar os métodos da classe mãe
- Classes simples de armazenamento de dados não precisam ser testadas

# Testes com Banco de Dados

- DAOs, ORM, Queries, Stored Procedures, Triggers
- Set Up e Tear Down são importantes
- BDs de desenvolvimento/teste
  - Sujeira no BD:
    - **Transaction Rollback Teardown**
    - **Table Truncation Teardown**
  - Concorrência: **Database Sandbox**
- BDs em memória

# Sanity Tests

- Testes pouco precisos para módulos específicos (para erros “gigantes”)
- Exemplo 1:
  - $4 \times 4 = ?$
  - Sabemos que  $3 \times 3 = 9$  e  $5 \times 5 = 25$
  - Verifique que:  $9 < 4 \times 4 < 25$
- Exemplo 2:
  - $3 \times 153 = ?$  // (459)
  - Verifique que  $(4 + 5 + 9) \bmod 3 = 0$

# Smoke Tests

- Testes superficiais e abrangentes
- Realizados após a instalação do software
- Exemplo 1:
  - Acesse uma página Web e verifique que não aparece o número 404

# Roteiro

- 1) Motivação
- 2) Importância dos Testes Automatizados
- 3) Alguns Conceitos de Testes Automatizados
- 4) Alguns Tipos de Testes
- 5) Dicas para Escrever Bons Testes
- 6) Considerações Finais**

# Quando Escrever Testes

- Antes de escrever o código (Test First)
- Durante a implementação
- Antes de uma refatoração
- Quando um erro for encontrado: Escreva um teste que simule o erro e depois corrija-o
- Quando um teste de aceitação falha, pode ser um sinal que testes de unidade estão faltando
- Quando um usuário ou cliente encontra um erro, é um sinal que estão faltando testes de unidade e de aceitação

# Quando Executar os Testes

- A execução deve ser ágil
- Sempre que uma nova porção de código é criada
- À noite ou em uma máquina dedicada
- Assim que alterações são detectadas no repositório
- **Lembre-se:** É melhor escrever e executar testes incompletos do que não executar testes completos

# Para Finalizar...

- Testes automatizados é uma prática fundamental para garantir a qualidade do código
  - *“Qualquer funcionalidade que não possui testes automatizados simplesmente não existe”*  
-- Kent Beck
  - Capriche no código dos testes

# Algumas Ferramentas

- Testes de Unidade:
  - Família XUnit:
    - CxxTest: <http://cxxtest.sourceforge.net>
    - JUnit: <http://www.junit.org>
  - TestNG: <http://testng.org>
  - rSpec:
  - jBehave: <http://jbehave.org>

# + Algumas Ferramentas

- Mock Objects:
  - SevenMock (Java): <http://seven-mock.sourceforge.net>
  - EasyMock (Java): <http://www.easymock.org/>
  - JMock (Java): <http://www.jmock.org>
  - Rhino.Mocks (.NET): <http://www.ayende.com/projects/rhino-mocks.aspx>
  - SMock (Smalltalk): <http://www.macta.f2s.com/Thoughts/smock.html>
  - Mockpp (C++): <http://mockpp.sourceforge.net>

# + Algumas Ferramentas

- Testes de Interface Gráfica:
  - Fest: <http://fest.easystesting.org/swing>
  - Jemmy: <http://jemmy.netbeans.org>
  - Marathon: <http://www.marathontesting.com>
- Testes de Interface Web:
  - Selenium: <http://www.openqa.org/selenium>
  - Watir: <http://wtr.rubyforge.org>
- Testes de Aceitação:
  - Fit: <http://fit.c2.com>
  - rSpec: <https://rubyforge.org/projects/rspec>

# + Algumas Ferramentas

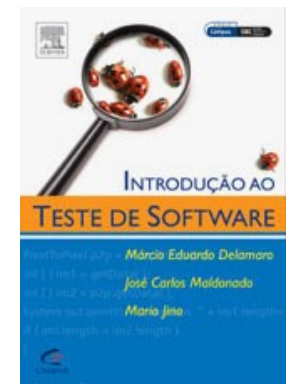
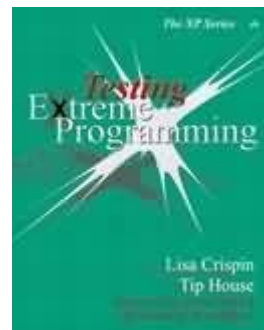
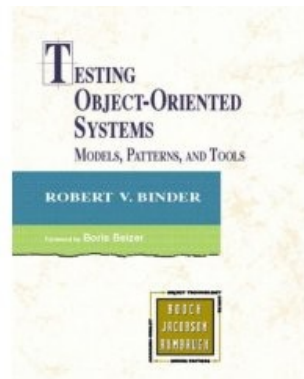
- Testes de Mutação:
  - Jester: <http://jester.sourceforge.net>
  - Heckle: <https://rubyforge.org/projects/seattlerb>
- Testes de Desempenho / Estresse:
  - JMeter: <http://jakarta.apache.org/jmeter>
- Outros:
  - JPDFUnit: <http://jpdfunit.sourceforge.net>

# Alguns links

- <http://www.testing.com>
- <http://www.opensourcetesting.org>
- <http://xunitpatterns.com>
- <http://www.mockobjects.com>
- <http://java-source.net/open-source/testing-tools>
- <http://www.junit.org>
- <http://www.agilcoop.org.br>

# Alguns livros

- Gerard Meszaros, “xUnit Test Patterns, Refactoring Test Code”, Addison-Wesley, 2007
- Robert V. Binder, “Testing Object-Oriented Systems”, Addison-Wesley Professional, 1999
- L. Crispin, T. House, “Testing Extreme Programming”, Addison-Wesley, 2005
- R. Mugridge, W. Cunningham, “Fit for Developing Software”, Prentice Hall, 2006
- M. Delamaro, J. Maldonado, M. Jino, “Introdução ao Teste de Software”, Campus, 2007



# ok ...

- Perguntas?
- Comentários?
- Críticas?
- Sugestões?

**AgilCoop** 

[agilcoop@agilcoop.org.br](mailto:agilcoop@agilcoop.org.br)