

Teste de Software

# Introdução a TDD: Projeto e Qualidade de Projeto



**PUC Minas**

Instituto de Ciências Exatas  
e Informática

Prof. Lesandro Ponciano

Departamento de Engenharia de Software  
e Sistemas de Informação (DES)

# **Questões Motivadoras**

---

- Como saber se um projeto tem qualidade?
- Quais as propriedades de um projeto de qualidade?
- O que é TDD?
- O que acontece com a qualidade do projeto de software e do código ao longo do tempo?
- O que são “maus cheiros” em projeto de software?

# Objetivos da Aula

---

- Analisar atributos de qualidade de projeto
- Analisar aspectos que determinam a qualidade de um projeto
- Contextualizar TDD
- Discutir mal cheiro em projetos (*bad smells*)

# Qualidade de Projeto

---

- Para se obter bons projetos, é necessário considerar alguns aspectos intimamente relacionados com a qualidade dos projetos
  - 1) Níveis de Abstração
  - 2) Arquitetura
  - 3) Padrões
  - 4) Separação por Interesse
  - 5) Modularidade
  - 6) Encapsulamento da Informação
  - 7) Independência Funcional
  - 8) Refinamento Gradual

# 1 - Níveis de Abstração

---

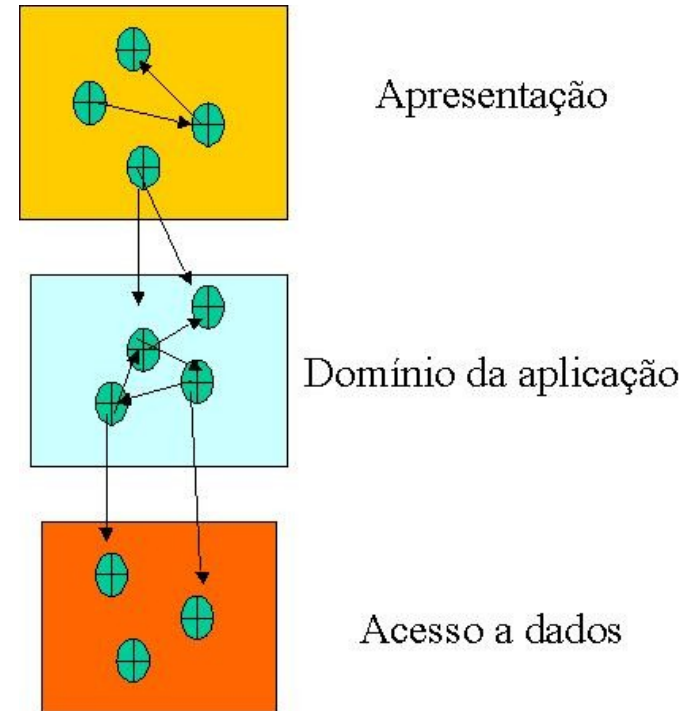
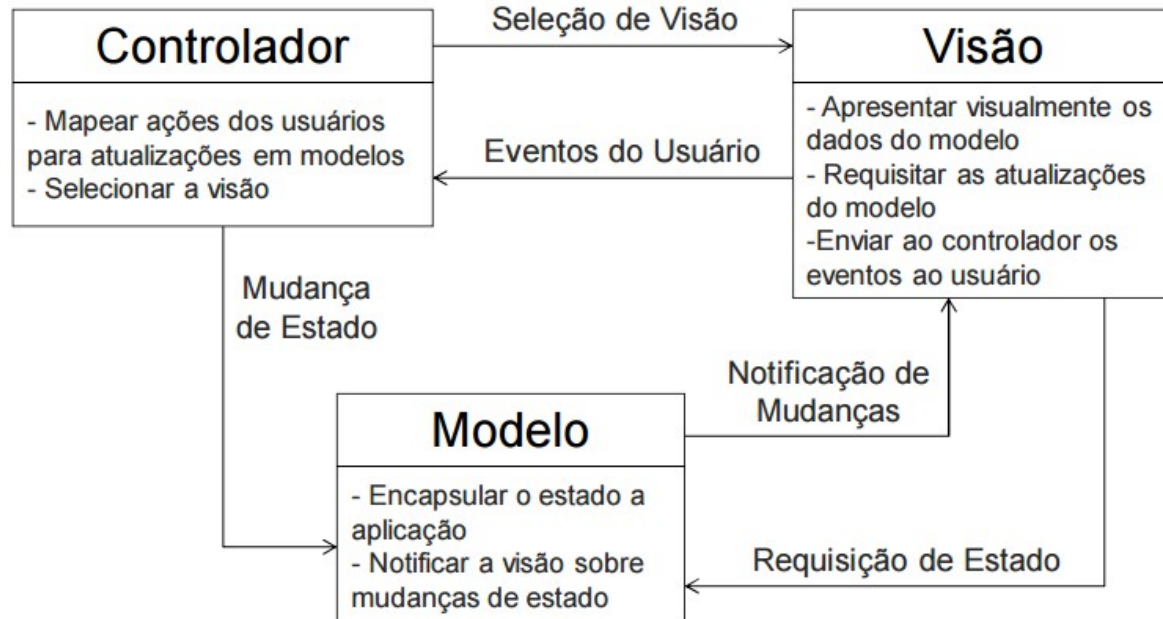
- No nível mais alto, a solução é expressa em termos mais abrangentes usando linguagem do domínio do problema
- Em níveis mais baixos, uma descrição mais detalhada da solução é fornecida
- No nível mais baixo, tem-se a solução técnica do software que pode ser implementada

## 2 - Arquitetura

---

- Consiste da estrutura ou organização de componentes de programas (módulos), a maneira como eles interagem e as estruturas de dados usadas
- A partir da arquitetura, as demais atividades do projeto podem ser realizadas
- Um bom projeto tem uma boa arquitetura: “*Não deixe que a arquitetura aconteça ao acaso.*”

# Exemplos de Arquitetura



## 3 - Padrões de Projeto

---

- É parte de um conhecimento consolidado e que transmite a essência de uma solução comprovada para um problema recorrente
- Estrutura de projeto que resolve uma particular categoria de problemas de projeto
  - Ex.: Abstract Factory, Singleton, Builder, Prototype



## 4 - Separação por Interesses

- Também chamado 'separação por afinidades'
- A complexidade de dois problemas combinados é maior que a soma da complexidade dos dois problemas isolados
  - É necessário dividir para conquistar
- Por separação por interesse em blocos menores, e portanto, mais administráveis, um problema toma menos tempo para ser resolvido

# 5 - Modularidade

---

- É a manifestação mais comum da separação por interesse
  - Software dividido em componentes chamados módulos
  - Módulos integrados para satisfazer os requisitos
- Modularização permite que
  - incrementos possam ser definidos e entregues
  - mudanças fiquem mais fáceis de serem acomodadas
  - testes e depuração possam ser conduzidos de forma mais eficaz
  - manutenção possa ser feita sem efeitos colaterais severos

## 6 - Encapsulamento de Informações

- Módulos devem ser projetados de modo que
  - as informações (algoritmos e dados) contidas em um módulo sejam inacessíveis a outros módulos que não precisam da informação
  - sejam disponibilizadas apenas as informações que interessam aos outros módulos
- Os detalhes de um módulo não precisam ser conhecidos por usuários do módulo

## 7 - Independência Funcional

- Módulos com uma função "única" e com "aversão" à interação excessiva com outros módulos
- É medida usando-se dois critérios qualitativos
  - **Coesão**: Força funcional relativa de um módulo; a força que mantém unidos os elementos de um módulo
  - **Acoplamento**: Grau de interdependência entre os módulos

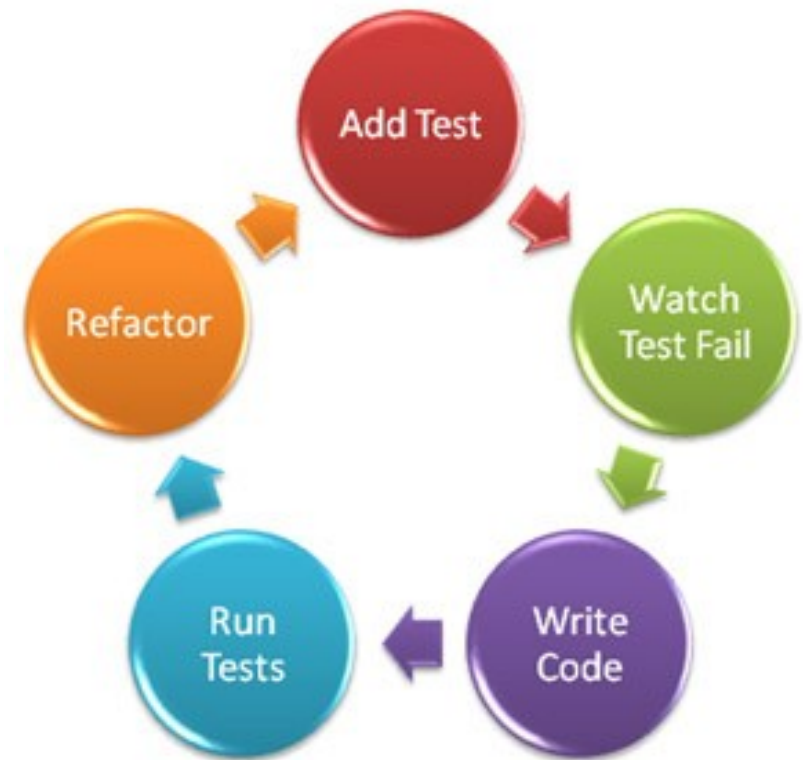
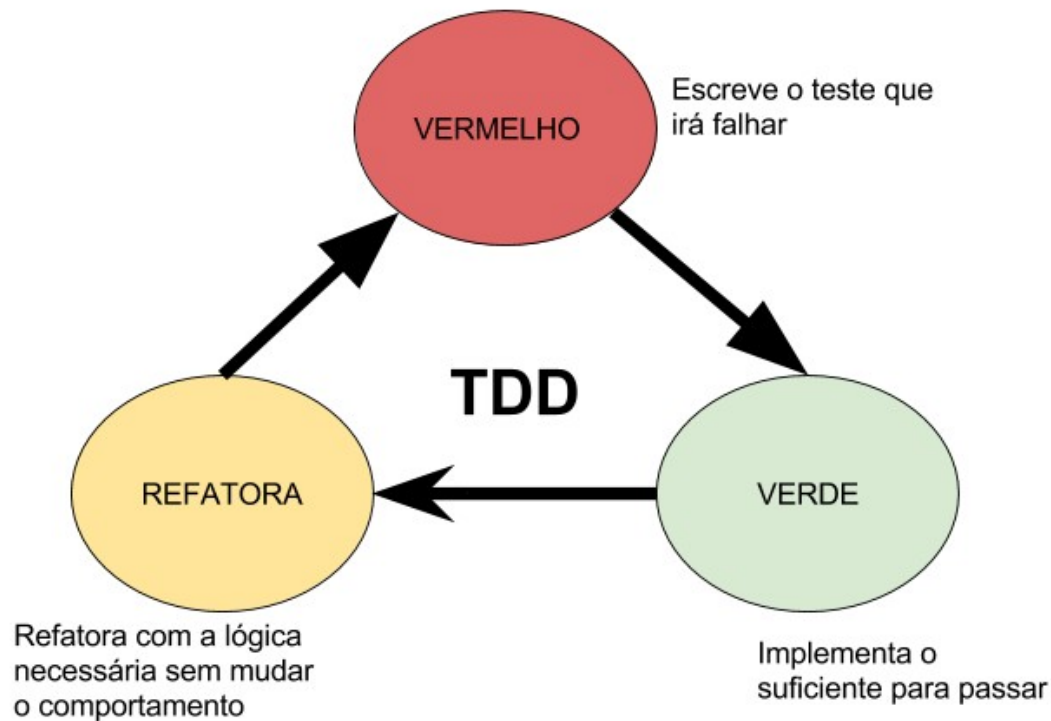
## 8 - Refinamento Gradual

---

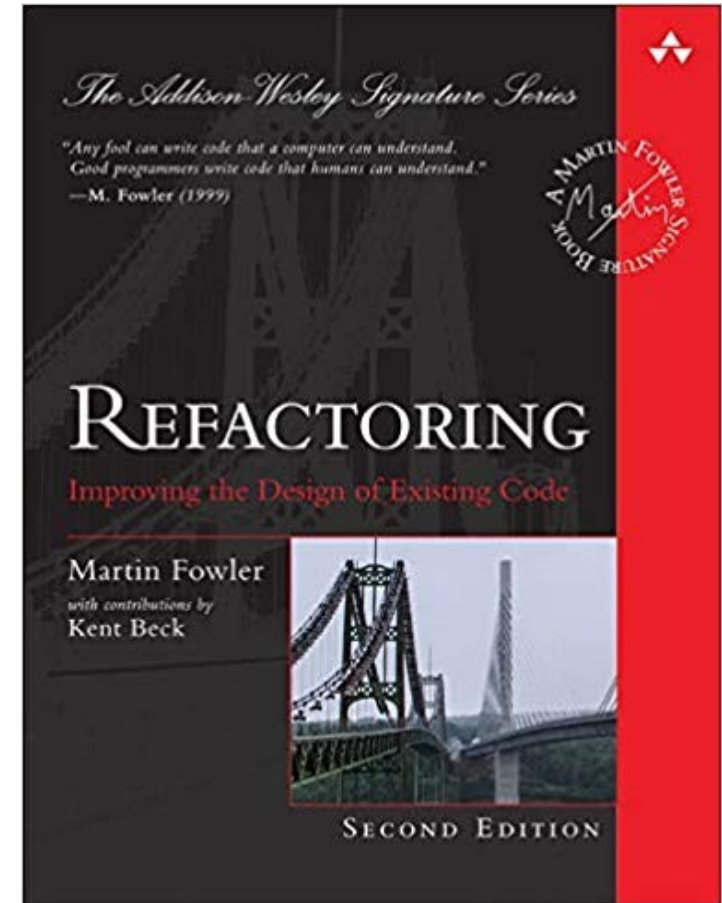
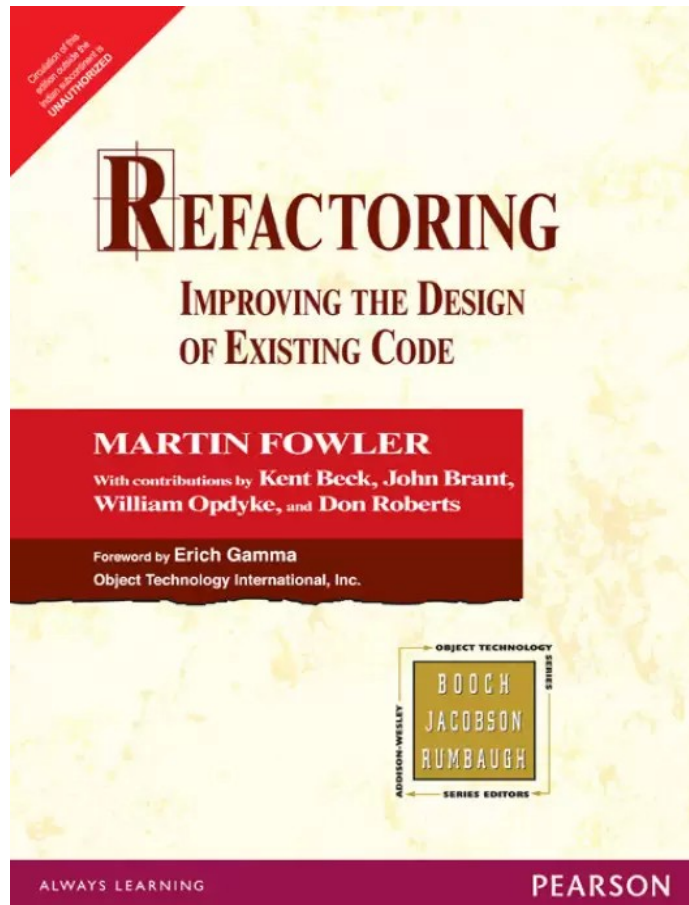
- Estratégia de projeto *top-down*
- Começamos com o enunciado da função (ou descrição das informações definidas) em um alto nível de abstração
- Em seguida, elaboramos uma declaração original, fornecendo cada vez mais detalhes à medida que ocorre cada refinamento (elaboração) sucessivo

# Desenvolvimento Dirigido por Testes

## O projeto em Métodos Ágeis



# O Projeto Apodrece



# O Projeto Apodrece

---

- Os requisitos mudam ao longo do tempo e, por consequência disso, o projeto do software muda ao longo do tempo
- Mudanças fazem o projeto e o software apodrecerem
- “*Feridas horríveis, cheias de pus e furúnculos, se acumulam*” no projeto
- Há muito mau cheiro (*bad smells*)



# Maus Cheiros do Projeto

---

- Você sabe que um software está apodrecendo quando ele começa a exalar um dos seguintes odores
  - Rigidez
  - Fragilidade
  - Imobilidade
  - Viscosidade
  - Complexidade desnecessária
  - Repetição desnecessária
  - Opacidade

# Rigidez

---

- Tendência do software ser de difícil alteração
- Uma única alteração, provoca uma sucessão de alterações subsequentes em módulos dependentes
  - Quanto mais módulos precisam ser alterados, mais rígido o projeto está
- Dificuldade de realizar mudanças
  - Difícil estimar o impacto delas
  - Forte impacto nos prazos, custos
  - Resposta: "Foi muito mais difícil do que eu esperava!"

# Fragilidade

---

- Tendência de, ao se fazer uma única alteração, o software apresentar defeitos em muitos lugares
  - Lugares que não estão conceitualmente relacionadas com a área alterada
- Corrigir os defeitos leva a mais defeitos
  - Sensação de um "cachorro correndo atrás do rabo"
- Módulos que quanto mais você corrige, pior ficam

# Imobilidade

---

- O projeto possui partes que poderiam ser úteis em outros sistemas, mas separar isso do sistema geraria mais riscos e trabalhos
- Dificulta o reuso de software

# Viscosidade

---

- Projeto de software é difícil de preservar
- É mais fácil fazer a coisa errada, mas mais difícil fazer a coisa certa!
  - Alterações que não preservam o projeto se tornam fáceis de serem feitas
  - Alterações que preservam o projeto se tornam difíceis de serem feitas

# Complexidade Desnecessária

- Projeto contém elementos que não são úteis no momento
- Antecipar mudanças de requisitos e colocar no projeto elementos que antecipam mudanças em potencial
  - Parece bom a primeira vista, evitar alterações futuras
  - É ruim, pois as mudanças podem nunca ocorrer e o código fica “pesado”

# Repetição Desnecessária

---

- Quando um mesmo código aparece inúmeras vezes, de forma ligeiramente diferente, está faltando uma abstração
  - Encontrar as repetições
  - Definir uma abstração
- Repetições torna complexo alterar o código
  - Ex.: correção de erro em uma repetição precisa ser feita nas diversas outras e a correção pode nem sempre ser a mesma

# Opacidade

---

- Dificuldade de se compreender um módulo
  - Código escrito de maneira opaca e enrolada
- Muitas vezes, quando olhamos um código antigo feito por nós mesmos, percebemos o quanto ele é opaco
- Ajuda um pouco
  - Se colocar no lugar do leitor
  - Revisão por terceiros



# **Atividades de Fixação**

---

- 1) Como saber se um projeto tem qualidade?
- 2) Quais as propriedades de um projeto de qualidade?
- 3) O que acontece com a qualidade do projeto de software e do código ao longo do tempo?
- 4) Dê exemplos de maus cheiros em projeto de software.

# Referências

---

SOMMERVILLE, Ian. Engenharia de Software - 9a edição. Pearson ISBN 9788579361081. (Capítulo 2 e Capítulo 5)

PRESSMAN, Roger. Engenharia de software. 8. Porto Alegre ISBN 9788580555349.

BECK, Kent. TDD desenvolvimento guiado por testes. Bookman Editora, 2009.

MARTIN, Robert; MARTIN, Micah. Princípios, padrões e práticas ágeis em C# (Seção II, Capítulo 7)