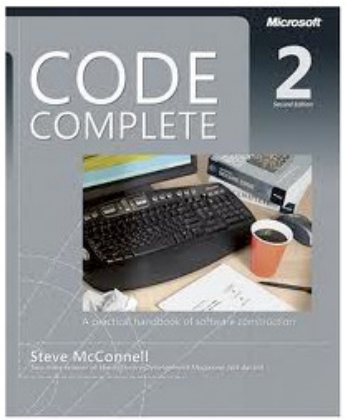
Qualidade de um Design OO

Universidade de Brasília

Rodrigo Bonifácio

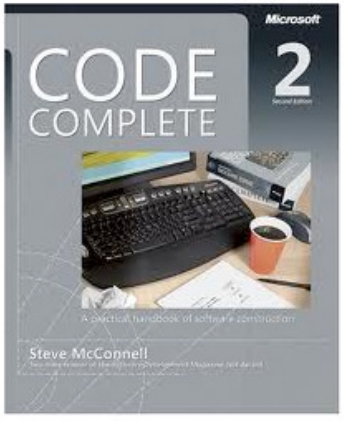
3 de dezembro de 2015

Motivação



Reality: code evolves substantially during its initial development. Many of the changes seen during initial coding are at least as dramatic as changes seen during maintenance. Coding, debugging, and unit testing consume between 30 to 65 percent of the effort on a typical project, depending on the project's size. (See Chapter 27, "How Program Size Affects Construction," for details.) If coding and unit testing were straightforward processes, they would consume no more than 20–30 percent of the total effort on a project. Even on well-managed projects, however, requirements change by about one to four percent per month (Jones 2000). Requirements changes invariably cause corresponding code changes—sometimes substantial code changes.

Motivação



Reality: code evolves substantially during its initial development. Many of the changes seen during initial coding are at least as dramatic as changes seen during maintenance. Coding, debugging, and unit testing consume between 30 to 65 percent of the effort on a typical project, depending on the project's size. (See Chapter 27, "How Program Size Affects Construction," for details.) If coding and unit testing were straightforward processes, they would consume no more than 20–30 percent of the total effort on a project. Even on well-managed projects, however, requirements change by about one to four percent per month (Jones 2000). Requirements changes invariably cause corresponding code changes—sometimes substantial code changes.

- Software deve ser facilmente mantido
- *Agilidade x Up front investment*

Refactoring

A change made to the internal structure of the software to make easier to understand and cheaper to modify without changing its observable behavior. (Martin Fowler)

Oportunidades

- “Bad smells”
- Métricas de qualidade de design

Bad smells

- Código duplicado
- Métodos longos
- Condições referenciando estado de outras classes
- Classes com muitas responsabilidades
- Muitos argumentos na assintatura de um método
- Mudanças não localizadas
- Variáveis temporárias
- ...

Muitos associados a modularidade

Muitos associados a modularidade, mas limitações de testabilidade também indicam que o design não está bom.

Exemplo

Aplicação simples para manter informações sobre aluguéis de filmes.

Exemplo

Aplicação simples para manter informações sobre aluguéis de filmes. Iniciamos com três classes, e um método de interesse: a impressão de um extrato de locação.

(pequeno) Desafio

Dependendo do tipo de tarifação (`priceCode`), os cálculos do valor do aluguel em relação ao número de dias e do total de pontos de fidelidade do cliente mudam.

Implementação do método `statment()`

```
public String statement() {
  double totalAmount = 0;
  int frequentRentPoints = 0;
  StringBuffer result = new StringBuffer("Rental records...");
  for (Rental rental : rentals) {
    double thisAmount = 0;
    switch (rental.getMovie().getPrice()) {
    case REGULAR:
      thisAmount += 2;
      if(rental.getDaysRented() > 2) thisAmount += //penality
      break;
    case NEW_RELEASE:
      thisAmount += rental.getDaysRented() * 3;
      break;
    case CLASSIC:
      thisAmount += 1.5;
      if(rental.getDaysRented() > 3) thisAmount += //penality
      break;
    }
    frequentRentPoints++;
    if(rental.getMovie().getPrice().equals(...)) {
      frequentRentPoints++;
   }
   result.append("\t - ....");
   totalAmount += thisAmount;
  }
  result.append("Total amount is " + totalAmount + "\n");
  result.append("You earned " + frequentRentPoints);
  return result.toString();
}
```

Testes

```
public class CustomerTest extends TestCase {
  private Movie classic1;
  private Movie classic2;
  private Rental rental1;
  private Rental rental2;
  private Customer customer;
  protected void setUp() throws Exception {
    classic1 = new Movie("A clock work orange", Movie.PriceCode.CLASSIC);
    classic2 = new Movie("Pulp fiction", Movie.PriceCode.CLASSIC);
    rental1 = new Rental(classic1, 2);
    rental2 = new Rental(classic2, 4);
    customer = new Customer("rbonifacio");
    customer.addRental(rental1);
    customer.addRental(rental2);
  }
  public void testStatementNotNull() {
    assertNotNull(customer.statement());
    System.out.println(customer.statement());
  }
  public void testStatementString() {
    String stmt = customer.statement();

    assertTrue(stmt.contains("- A clock work orange (2 day(s)): 1.5"));
    assertTrue(stmt.contains("- Pulp fiction (4 day(s)): 3.0"));
    assertTrue(stmt.contains("Total amount is 4.5"));
    assertTrue(stmt.contains("You earned 2 points"));
  }
}
```

Só nos sentimos confortáveis em executar atividades de refactoring quando temos uma boa suite de testes

Análise

- A classe `Customer` apresenta boa coesão?
- O método `statement()` pode ser facilmente testado?
- Uma nova formatação (HTML) leva a duplicação de código?
- Um novo tipo de tarifação pode ser implementado modularmente?

Então devemos refatorar o design

Passos (1/2)

- Extract Method: novo método para o cálculo do valor a ser pago para cada item, simplificando o método `statement()`
- Move Method: o método resultante em (1) deveria ser implementado na classe `Rental`, visando uma melhor distribuição das responsabilidades.
- Replace Temp with Query: variáveis temporárias perdem a utilidade com os novos métodos, devemos substitui-las por chamadas a métodos.

Passos (1/2)

- Extract Method: novo método para o cálculo do valor a ser pago para cada item, simplificando o método `statement()`
- Move Method: o método resultante em (1) deveria ser implementado na classe `Rental`, visando uma melhor distribuição das responsabilidades.
- Replace Temp with Query: variáveis temporárias perdem a utilidade com os novos métodos, devemos substitui-las por chamadas a métodos.

Passos (1/2)

- Extract Method: novo método para o cálculo do valor a ser pago para cada item, simplificando o método `statement()`
- Move Method: o método resultante em (1) deveria ser implementado na classe `Rental`, visando uma melhor distribuição das responsabilidades.
- Replace Temp with Query: variáveis temporárias perdem a utilidade com os novos métodos, devemos substitui-las por chamadas a métodos.

Passos (2/2)

- Move Method: o cálculo dos pontos de uma locação podem estar na classe `Rental`, favorecendo a coesão da classe `Customer`.
- Replace Temp with Query: as demais variáveis temporárias podem ser substituidas por queries, sendo necessária novas aplicações do *Extract Method*
- Replace Conditional Logic with Polimorfism: o padrão de projeto *Strategy* pode ser usado para tornar extensível o cálculo do valor da locação com base no tipo do filme.

Passos (2/2)

- Move Method: o cálculo dos pontos de uma locação podem estar na classe `Rental`, favorecendo a coesão da classe `Customer`.
- Replace Temp with Query: as demais variáveis temporárias podem ser substituidas por queries, sendo necessária novas aplicações do *Extract Method*
- Replace Conditional Logic with Polimorfism: o padrão de projeto *Strategy* pode ser usado para tornar extensível o cálculo do valor da locação com base no tipo do filme.

Passos (2/2)

- Move Method: o cálculo dos pontos de uma locação podem estar na classe `Rental`, favorecendo a coesão da classe `Customer`.
- Replace Temp with Query: as demais variáveis temporárias podem ser substituidas por queries, sendo necessária novas aplicações do *Extract Method*
- Replace Conditional Logic with Polimorfism: o padrão de projeto *Strategy* pode ser usado para tornar extensível o cálculo do valor da locação com base no tipo do filme.

Catálogo de refatoração

- Renomeção de atributos, métodos ou classes
- Extração de bloco de código para métodos
- Mover método de uma classe para outra
- Substituir variáveis temporárias por métodos de acesso
- Substituir condições por polimorfismo
- ...

Métricas OO

Chidamber e Kemerer

- Weighted Methods per Class (WMC)
- Coupling Between Objects (CBO)
- Depth of the Inheritance Tree (DIT)
- Number of Children (NOC)
- Response for a Class (RFC)
- Lack of Cohesion of Methods (LCOM)

Qualidade de um Design OO

Universidade de Brasília

Rodrigo Bonifácio

3 de dezembro de 2015