Princípios SOLID de OO usando .NET

Ivan Paulovich
Arquiteto de Softwares

Banco Olé Consignado – Julho/2017

Ivan Paulovich

Microsoft Most Valuable Professional

De 2012 à 2014

100LOOP

Microsoft Most Valuable Professional
Reconnect

bhs

.net RAPTORS

Agenda

- Um pouco de história
- O que é modelagem de software?
- Por que investir em modelagem?
- Sintomas de problemas de modelagem
- O que é uma boa modelagem?
- Princípios de OO na modelagem de classes
- Um breve resumo de modelagem em OO
- Dúvidas?

Um pouco de história

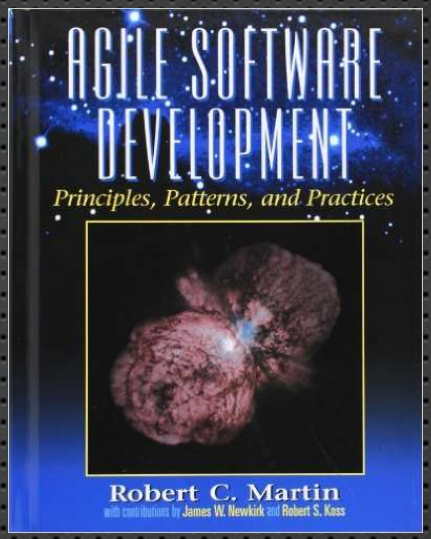
1995
Princípios de OO

"Existem muitas dependências,
dependências em todos os lugares"

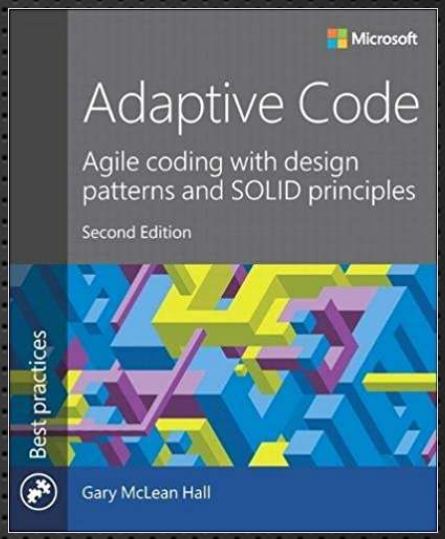
Robert C. Martin (Uncle Bob)
Manifesto Ágil

Um pouco de história

1995
Princípios de OO



2002
PPP



2015 e 2017
Adaptive Code (.NET)

Robert C. Martin (Uncle Bob)
Manifesto Ágil

Gary McLean Hall

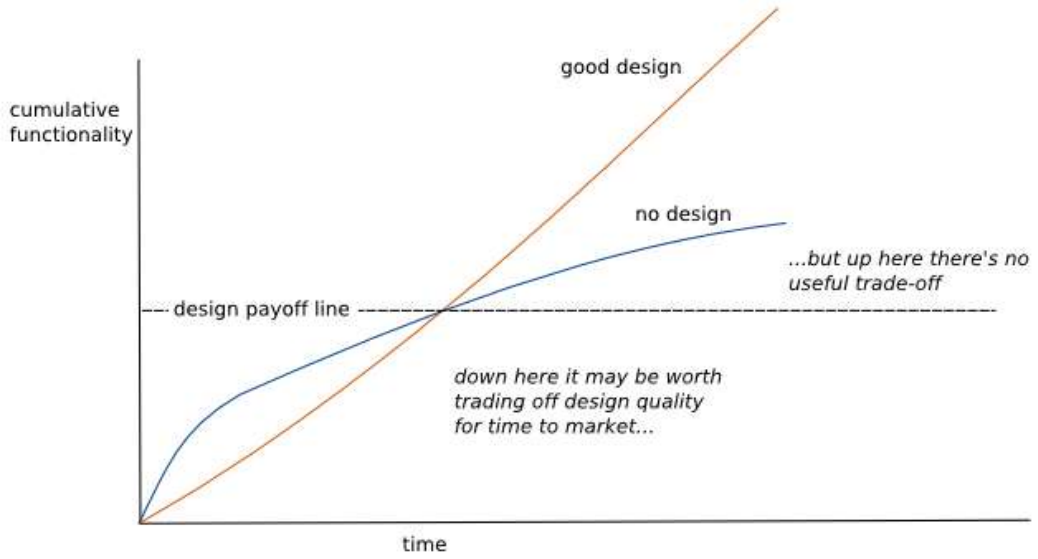
O que é modelagem de software?

- Diagramas UML representam parte da modelagem
- O processo de modelagem de software inclui escrever código, testar e refatorar
- O código fonte é a modelagem
- A modelagem do software é feita pelo programador

Por que investir em modelagem?

- Para entregar rápido
- Para gerenciar mudanças mais facilmente
- Para enfrentar a complexidade

Por que investir em modelagem?



https://martinfowler.com/bliki/DesignStaminaHypothesis.html

Sintomas de problemas de modelagem

- Rigidez
  Difícil alterar a modelagem
- Fragilidade
  Modelagem fácil de quebrar
- Imobilidade
  Modelagem difícil de reusar
- Viscosidade
  Modelagem torna difícil fazer a coisa certa
- Código Intestável
  Código que não é testável contém defeitos

O que é uma boa modelagem?

- Alta coesão
- Baixo acomplamento

Princípios de OO na modelagem de classes

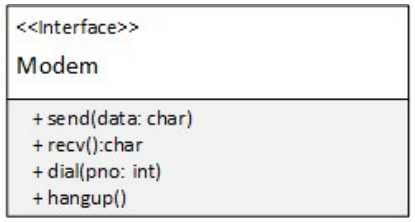
- **S**ingle Responsibility Principle
- **O**pen Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Single Responsibility Principle

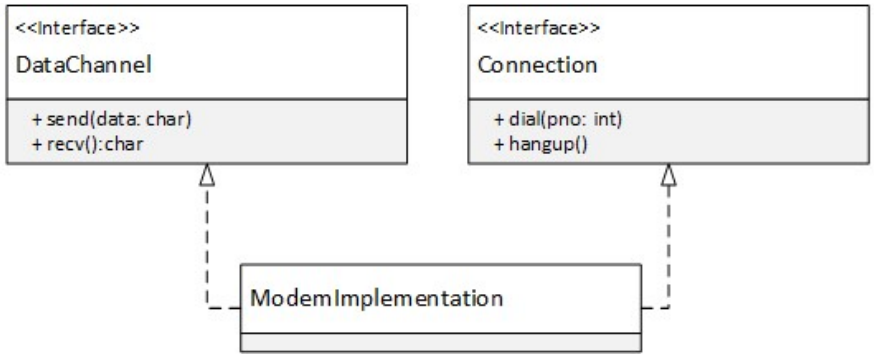
- "Uma classe deve ter uma, e somente uma razão para mudar." – Uncle Bob
- Benefício da alta coesão
- Não é fácil identificar responsabilidades diferentes.

Single Responsibility Principle

Versão A

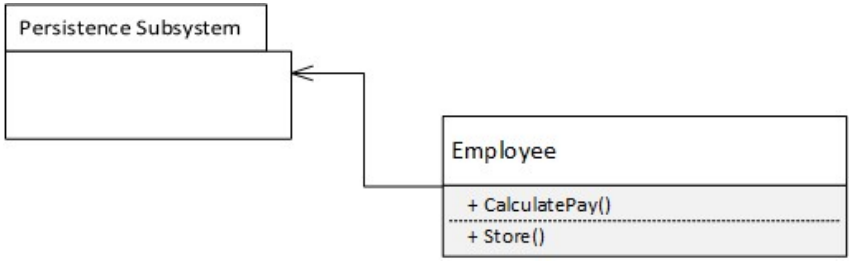


Versão B



- Geralmente a versão B é um código melhor

Single Responsibility Principle



- Violação do SRP.
- Regras de negócio e Persistência quase sempre não devem se misturar.
- Regras de negócio mudam frequentemente.

Single Responsibility Principle

- O princípio mais simples e a mais difícil de aplicar corretamente.
- Identificar e separar uma responsabilidade da outra é uma tarefa muito relevante na modelagem.

Open Closed Principle

- Você deve poder estender o comportamento de uma classe, sem modificá-la.

Open Closed Principle

Módulos aderentes ao OCP:

1. Abertos para extensão

   Podemos fazer a classe ter novos e diferentes comportamentos conforme os requisitos são alterados

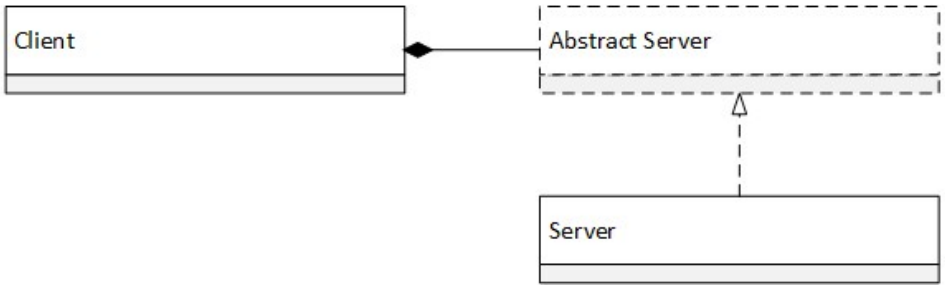
2. Fechados para modificação

   O código fonte da classe é inviolável.
   Ninguém é permitido realizar modificações.

Open Closed Principle

Cliente Fechado para Extensão

Cliente Aberto para Extensão



Liskov Substitution Principle

- As classes derivadas devem ser substituíveis por suas classes base.

```
foreach (var funcionario in Funcionarios)
{
    if(funcionario is Manager)
    {
        _printer.PrintManager(funcionario as Manager);
    }
    else
    {
        _printer.PrintEmployee(funcionario);
    }
}
```

Interface Segregation Principle

- Clientes não devem ser forçados a depender de interfaces que eles não usam
- Interfaces pequenas específicas para o cliente

```
public interface IRepository<T>
{
    T GetById(int id);
    IEnumerable<T> List();
    void Create(T item);
    void Update(T item);
    void Delete(T item);
}
```

Interface Segregation Principle

```
public interface IReadRepository<T>
{
    T GetById(int id);
    IEnumerable<T> List();
}

public interface IWriteRepository<T>
{
    void Create(T item);
    void Update(T item);
    void Delete(T item);
}

public interface IRepository<T> :
        IReadRepository<T>, IWriteRepository<T>
{

}
```

Dependency Inversion Principle

- Depender de abstrações, e não de implementações concretas.

```
public void CheckOut()
{
    MailMessage myMessage = new MailMessage(
        "loja@minhaloja.com",
        "usuario@minhaloja.com",
        "Seu pedido foi recebido.",
        "Obrigado!");

    SmtpClient smptClient = new SmtpClient("localhost");
    smtpClient.Send(myMessage);
}
```

Dependency Inversion Principle

```
public interface ISendEmail
{
    void SendMail(string to, string from,
        string subject, string body);
}

public class LiveSmtpMailer : ISendEmail
{
    void SendMail(string to, string from,
        string subject, string body)
    {
        MailMessage myMessage = new MailMessage(from, to,
                subject, body);
        SmtpClient smptClient = new SmtpClient("localhost");
        smtpClient.Send(myMessage);
    }
}
```

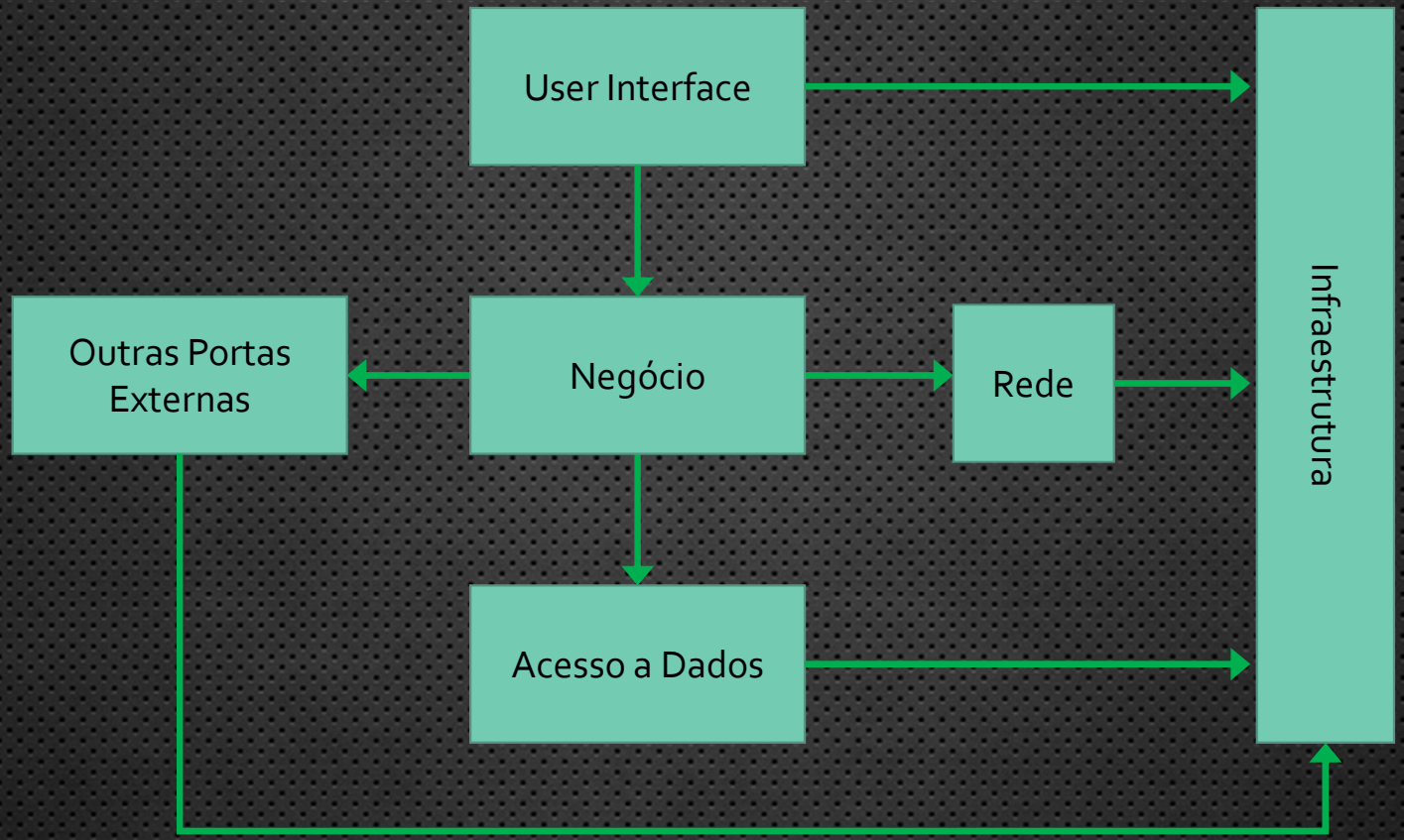
Dependency Inversion Principle

```
public class Cart
{
    private ISendEmail _emailProvider;

    public Cart(ISendEmail emailProvider)
    {
        _emailProvider = emailProvider;
    }

    public void Checkout()
    {
        this.emailProvider.SendMail(
            "loja@minhaloja.com",
            "usuario@minhaloja.com",
            "Seu pedido foi recebido.",
            "Obrigado!");
    }
}
```

Um breve resumo de modelagem em OO



Dúvidas?

Referências

- The Principles of OOD
  http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
- Adaptive Code - Agile Coding with design patterns and SOLID principles, Gary Mc Hall
- Solid Principles of OO Design
  https://www.slideshare.net/confiz/solid-principles-of-oo-design-29397774
- Refactoring Applications using SOLID Principles
  https://www.slideshare.net/ardalis/refactoring-applications-using-solid-principles