.NET - POO - O princípio da responsabilidade única - SRP

Há algum tempo atrás publiquei uma série de artigos sobre os conceitos básicos da programação orientada a objetos - POO (Object Oriented Programming-OOP) se você ainda tem alguma dúvida sobre estes conceitos sugiro a leitura dos meus artigos:

- VB .NET - Primeiros passos - Conceitos - VI - Conceitos OOP para você : sobrecarga , sobreposição , classes abstratas e interfaces.
- VB .NET - Primeiros passos - Conceitos - V - Orientação a objeto no VB .NET : classes , propriedades , métodos ; instanciando um objeto ; construtor , herança.
- VB .NET - Primeiros passos - Conceitos - IV - Escrevendo sua aplicação VB .NET com os conceitos apresentados. Calculando o valor futuro:passo a passo.
- VB .NET - Primeiros passos - Conceitos - III - Os conceitos básicos para começar a escrever código VB .NET ; Escrevendo suas próprias rotinas e funções e escopo de variáveis
- VB .NET - Primeiros passos - Conceitos - II - Os conceitos básicos para começar a escrever código VB .NET ; operações e funções básicas
- VB .NET - Primeiros passos - Conceitos - I- Os conceitos básicos para começar a escrever código VB .NET (ideal para iniciantes e iniciados) ; variáveis , constantes , vetores.

Com este artigo pretendo iniciar uma nova série abordando com mais detalhes os principais conceitos do mundo da programação orientada a objetos. Na verdade alguns dos conceitos surgiram há mais de 10 anos, mas somente hoje com a popularidade da programação orientada a objetos se tornaram mais conhecidos.

Não faz muito tempo que temos os conceitos da OOP aplicados ao desenho de sistemas, e, por experiência própria , sei que uma coisa é entender os conceitos e outra é colocá-los em prática no dia a dia com bom senso, pois de nada adianta engessar um sistema forçando a aplicação das técnicas OOP de forma artificial somente para dizer que o sistema é 'orientado a objetos'.

Antes de começar a falar sobre o assunto principal do artigo vou relacionar alguns dos maiores problemas encontrados em um sistema de software e que poderiam ser evitados :

1. **Rigidez** - O sistema foi feito de forma a ser difícil de mudar, qualquer alteração provoca uma cascata de operações por todo o sistema;
2. **Fragilidade** - O sistema foi feito de forma que qualquer mudança feita o

desestabiliza;

3. **Imobilidade** - O código foi feito de forma a ser difícil de ser reusado; nenhuma parte do sistema pode ser aproveitada em outro sistema;
4. **Complexidade** - O código foi feito com a aplicação de diversos padrões de projeto a um problemas simples;
5. **Repetição de código desnecessária** - O mesmo trecho de código esta esparramado por todo o sistema;

Estarei então apresentando em uma série de artigos com os conceitos sobre os mais importantes princípios para desenho de software com o objetivo evitar as más práticas na construção de software. Abaixo temos os princípios que irei abordar:

- *Principio da Responsabilidade Única;*
- *Princípio de Aberto-Fechado;*
- *Princípio da Substituição de Liskov;*
- *Princípio da inversão da dependência;*
- *Princípio da Segregação da Interface;*

Apresentando o Single Responsability Principle- SRP (Coesão)

Vou começar este artigo falando de um conceito fundamental conhecido como SRP - Single Responsability Principle ou princípio da responsabilidade única também conhecido como Coesão(Cohesion).

Nota: Este princípio foi introduzido por Tom DeMarco em 1979 no seu livro Structured Analysis and Systems Specification, Yourdon Press Computing Series.

O princípio da responsabilidade única é um princípio fundamental no desenho de software que reza o seguinte :

"Deve existir um e somente UM MOTIVO para que uma classe mude"

Portanto uma classe deve ser implementada tendo apenas um único objetivo.

Quando uma classe possui mais que um motivo para ser alterada é por que provavelmente ela esta fazendo mais coisas do que devia, ou seja, ela esta tendo mais de um objetivo.

Podemos então inferir as seguintes premissas a partir da definição da responsabilidade única:

- Baseado no princípio da coesão funcional, uma classe deve ter uma única responsabilidade;
- Se uma classe possuir mais de uma responsabilidade, deve-se considerar sua decomposição em duas ou mais classes;
- Cada responsabilidade é um “eixo de mudança” e as fontes de mudança devem ser isoladas;

Este conceito é fácil de entender mas difícil de ser posto em prática.

Existem muitas referências e exemplos na literatura sobre o **SRP** e eu vou procurar mostrar alguns para clarear o entendimento.

Vejamos o clássico exemplo da classe Retângulo que possui dois métodos conforme mostrado a seguir:

Métodos:

Area() - Calcula a área do Retângulo;

Desenhar() - Desenha o Retângulo;

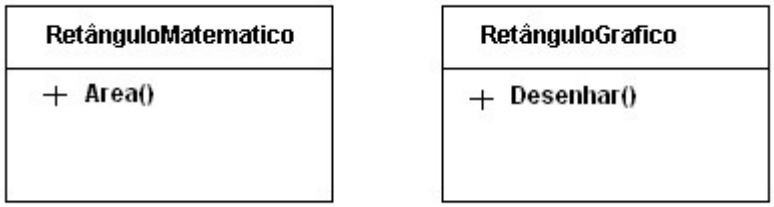
Este desenho viola o princípio da responsabilidade única - **SRP** pois a classe Retângulo possui duas responsabilidades definidas:

1. **Calcular a área do retângulo usando um modelo matemático;**
2. **Desenhar o retângulo usando uma interface gráfica;**

Qualquer alteração no modelo matemático implicará na modificação da classe o que pode afetar a interação com a interface gráfica e qualquer alteração na interface gráfica também implicará em alterações na classe podendo afetar a utilização do modelo matemático usado para calcular a área do retângulo.

Quando você perceber que isto esta ocorrendo analise bem a sua classe e divida-a em varias classes diferentes de forma que cada classe tenha apenas um motivo para ser alterada.

No caso da classe Retângulo um melhor desenho será separar as duas responsabilidades em duas classes diferentes:



Desta forma qualquer alteração feita em uma das classes tem apenas um único motivo para ser feita e não afeta a outra classe.

Podemos evocar o SRP para justificar o desenvolvimento de camadas visto que o objetivo de criar camadas é separar a apresentação do negócio estamos indiretamente aplicando o SRP de forma a termos somente um motivo para que cada camada seja alterada.

Assim, em uma classe **Clientes** que é usada para obter os dados dos clientes devemos apenas usar os métodos criados na camada de acesso a dados para obter conexão e selecionar os dados, desta forma qualquer alteração relacionado com acesso a dados será feita na camada de acesso a dados sem afetar a classe cliente que apenas usa os métodos.

Parece que não tem muita importância, mas modelar corretamente conceitos do modelo de negócios para o software torna qualquer mudança nos requisitos mais fáceis de controlar e implementar pois você consegue ir da lógica do negócio ao código com facilidade visualizando melhor o impacto de uma nova funcionalidade.

Referências:

- SRP: The Single Responsibility Principle
- Structured Analysis and Systems Specification, Yourdon Press Computing Series, 1979
- Padrões de Projeto - *Macoratti*.net
- Design Patterns - o *padrão* Factory - *Macoratti*.net
- Pílula de Arquitetura - Princípios SOLID - *Macoratti*.net
- OOP - O princípio de substituição de Liskok (*LSP*) - *Macoratti*.net
- Padrões de Projeto - O modelo MVC - Model View Controller
- O padrão Singleton
- VB.NET - Permitindo uma única instância da sua aplicação
- Conceitos sobre projetos - Decomposição
- Usando o padrão Strategy
- Seção Padrões de Projeto do site Macoratti.net
- Super DVD .NET - A sua porta de entrada na plataforma .NET
- Super DVD Vídeo Aulas - Vídeo Aula sobre VB .NET, ASP .NET e C#
- OOP - O princípio *Open-Closed* - *Macoratti*.net

José Carlos Macoratti