Benefícios da Programação Orientada a Objetos e Design Patterns (Padrões de Projetos)

Guilherme de Castro Gaspar[1], Ramon Lummertz[2]

[1]Acadêmico do Curso de Curso de Sistemas de Informação/Análise e Desenvolvimento de Sistemas – Universidade Luterana do Brasil (ULBRA) Campus Torres - RS

[2]Professor(a) dos Cursos de Computação – Universidade Luterana do Brasil (ULBRA) Campus Torres – RS

gui_castrogaspar@hotmail.com, ramonsl@gmail.com

Abstract. *In this article I will talk about the benefits of working with object-oriented programming, by focusing on the pillars of Object Oriented (Abstraction, Polymorphism, Inheritance and Encapsulation). In the second part of the article I will explain about the "Gang of Four" and also I will quote creation of standards, which is nothing more than a standard category GoF.*

Resumo. *Neste artigo falarei sobre os benefícios de se trabalhar com a programação orientada a objetos, dando enfoque nos pilares da Orientação a Objetos (Abstração, Polimorfismo, Herança e Encapsulamento). Na segunda parte do artigo explicarei sobre o "Gang of Four" e também citarei padrões de Criação, que nada mais é do que uma categoria do padrão GoF.*

1. Programação Orientada a Objetos – Introdução

Vamos começar falando um pouco sobre a programação orientada a objetos... A orientação a objetos parte do conceito que seu Software deve trabalhar com foco nos objetos da vida real, trazendo o mundo real para dentro do computador basicamente. Isso nasceu de cientistas e engenheiros que queria organizar seus programas para se familiarizar mais com o mundo real.

Já a indústria do software que ainda não conseguiu definição concreta para "orientado a objeto", sendo que esse termo provém de muita falta de clareza ainda, sendo assim, podendo permitir que qualquer pessoa possa dizer "basicamente" que seu programa é orientado a objetos, mas esse termo "orientado a objetos" abrange muito mais coisa.

E para o que serve esse tipo de programação? Em muitos livros os autores definem que a facilidade de lidar com um projeto de Software é o fator que se é mais abrangente a ponto de usar esse tipo de programação. Então, programação orientada a objetos serve para facilitar nossas vidas? São tantos os conceitos, trago aqui uma definição retirada de um livro de apresentação de Larry L. Constantine (um dos primeiros pensadores sobre programação orientada a objetos):

"Alguns reacionários diriam que a orientação a objeto não serve para nada; ela é meramente um culto religioso ou uma conspiração global com sede em algum local da Costa Oeste dos EUA. Alguns revolucionários dirão que a orientação a objeto é a primeira e única solução milagrosa para todas as nossas aflições de software. Ela não apenas criará janelas, mas também vai picar e ornamentar verduras, encerar o chão e pôr cobertura nas sobremesas." (MEILIR PAGE-JONES, 1997).

2. Os benefícios da Programação Orientada a Objetos

A orientação a objetos nasceu de uma forma para ajudar ainda mais no desenvolvimento do software, por ser um "jeito" de se programar, muitas características da orientação a objetos irão ajudar no decorrer do desenvolvimento, mas isso tudo vale da "análise" da "regra de negócio", escolher sempre o meio de desenvolvimento adequado... Então existem casos 100% onde não se é utilizado orientação a objetos e existem casos onde não se deve trabalhar com outra coisa além orientado a objeto.

Analisando cada necessidade de usuário, projetando o software, construindo o software, dando manutenção ao software, usando o software e gerenciando projetos do mesmo, em cada uma dessas partes do ciclo de vida de um software, a orientação a objetos entra mais profundamente, trazendo mais liberdade, robustez, reutilização e confiabilidade para o seu software, além de muitas outras características que POO aborda.

Daí nasce os benefícios, na orientação a objetos existem basicamente pilares, onde nascem o conceito de "orientado a objeto".

2.1. Pilares na orientação a objetos

Basicamente existem 4 pilares que a orientação a objetos aborda, que seriam Abstração, Encapsulamento, Polimorfismo e Herança, e esses pilares são conceitos que esse jeito de programar trouxe consigo e seriam os benefícios que abordam desenvolver "orientado a objetos".

2.1.1 Abstração

É o princípio da orientação a objetos, é utilizada para a definição de entidades do mundo real. Sendo onde são criadas as "classes" (uma forma de um objeto). Abstração parte do conceito de abstrair de tudo aquilo de fora (do mundo real), abstrair e jogar isso tudo dentro de um programa.

2.1.2 Encapsulamento

O encapsulamento em orientação a objeto tem uma finalidade similar à sub-rotinas (funções). Mas se falando em termos de software ele muito mais complexo.

Encapsulamento é um conceito quase tão antigo quanto o próprio software. Desde a década de 1940, os programadores começaram a observar que padrões similares de instruções apareciam várias vezes dentro de um mesmo programa. Algumas pessoas perceberam que esses padrões repetidos poderiam ser amontoados em um canto do programa e invocados através de um único nome a partir de vários pontos diferentes no programação principal. (WILKES ET AL, 1951)

2.1.3 Polimorfismo

O polimorfismo é a facilidade pela qual pode ser definido um único nome de método sobre muitas classes diferentes e pode assumir diferentes implementações em cada uma daquelas classes.

A palavra polimorfismo deriva de duas palavras gregas que significam, respectivamente, muitas e formas. Qualquer coisa que seja polimórfica tem, portanto, habilidade de assumir várias formas. Polimorfismo também é definido como uma propriedade pela qual uma variável pode apontar para (manter o identificador de) objetos de diferentes classes em instantes diferentes. (MEILIR PAGE-JONES, 1997)

2.1.4 Herança

A herança representa uma outra maneira importante pela qual a orientação a objeto se distancia das abordagens tradicionais de sistema. Ela, efetivamente, permite que se construa o software de maneira incremental do seguinte modo: primeiro, criam-se classes para atenderem ao caso mais direto (ou geral). Depois, para tratar dos casos especiais, adicionam-se classes mais especializadas que herdam da primeira classe. Essas novas classes serão habilitadas a usar todos os métodos e variáveis da classe original. (MEILIR PAGE-JONES, 1997)

A herança veio para distinguir uma quantidade grande de códigos que poderiam facilmente "herdar informações" de outras classes.

3. Padrões de Projeto (Padrões de Criação)

O que seria basicamente um padrão de projeto? Segundo Christopher Alexander cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira. (CHRISTOPHER ALEXANDER, 1997)

Em geral um padrão tem quatro elementos essenciais, o nome do padrão, o problema, a solução e as consequências. Neste artigo falarei especificamente sobre padrões de criação, que ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados.

Um padrão de criação de criação tem como principal objetivo abstrair o processo de criação de objetos, ou seja, a sua instanciação. Desta maneira o sistema não precisa necessariamente se preocupar com como o objeto é criado, como é composto, e qual sua representação real. Padrões de criação se subdividem em Abstract Factory, Builder, Factory Method, Prototype e Singleton.

3.1 Abstract Factory

A intenção desse padrão é fornecer uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. Também conhecido como "Kit".

E quando se deve usar esse padrão? (1) – Quando um sistema deve ser independente de como seus produtos são criados, compostos ou representados. (2) – Quando um sistema deve ser configurado como um produtos de uma família de múltiplos produtos. (GRADY BOOCH, 1994)

3.2 Builder

A intenção desse padrão é separar a construção de um objeto complexo da sua representação de modo que o mesmo processo de construção possa criar diferentes representações.

E quando se deve usar esse padrão? (1) – Quando o algoritmo para criação de um objeto complexo deve ser independente das partes que compõem o objeto e de como elas são montadas. (2) – Quando o processo de construção deve permitir diferentes representações para o objeto que é construído. (GRADY BOOCH, 1994)

3.3 Factory Method

A intenção desse padrão é definir uma interface para criar um objeto, mas deixar as subclasses decidirem que classe instanciar. O Factory Method permite adiar a instanciação para subclasses.

E quando se deve usar esse padrão? (1) – Quando uma classe não pode antecipar a classe de objetos que deve criar. (2) – Quando uma classe quer que suas subclasses especifiquem os objetos que criam. (GRADY BOOCH, 1994)

3.4 Prototype

A intenção desse padrão é especificar os tipos de objetos a serem criados usando uma instância protótipo e criar novos objetos pela cópia deste protótipo.

E quando se deve usar esse padrão? (1) – Quando um sistema dever ser independente de como os seus produtos são criados, compostos e representados. (2) – Quando as classes a instanciar são especificadas em tempo de execução, por exemplo, por carga dinâmica. (GRADY BOOCH, 1994)

3.5 Singleton

A intenção desse padrão é garantir que uma classe tenha somente uma instância e fornecer um ponto global de aceso para a mesma.

E quando se deve usar esse padrão? (1) – Quando haver apenas uma instância de uma classe, e essa instância deve dar acesso aos clientes através de um ponto bem conhecido. (2) – Quando a única instância tiver de ser extensível através de subclasses, possibilitando aos clientes usarem uma instância estendida sem alterar o seu código. (GRADY BOOCH, 1994)

4. Conclusão

Conclui-se que nem sempre o jeito que se programar orientado a objetos será o mais "eficaz", vale lembrar que o desenvolvimento de um software e o crescimento de uma empresa está na análise das melhores possibilidades, está na decisão de escolher o melhor ambiente em que se vai trabalhar.

A programação Orientada a objeto oferece um caminho cheio de facilidade e "abstrações" real do mundo de fora de um computador. Deixando assim o entendimento de um software muito mais limpo e claro.

O mesmo caso da "análise" da empresa para ver o melhor jeito de se programar, está também presente para escolher o melhor padrão de projeto possível. Existem diversos padrões que podem se sair melhores que outros, de acordo com a necessidade e "regra de negócio", sendo assim, padrões de projetos também interagem e podem trabalhar juntos.

Assim então, conclui-se que parte de cada um analisar e escolher a melhor opção para desenvolver um Software.

Referências

MEILIR, Pages Jones. Aprentação de Larry L. Constantine:o que todo programador deveria saber sobre projeto orientado a objeto. Makron Books, Brazil, 1997.

WILKES, J. The Science of Computing: Shaping a Discipline. Wilkes et al. New York, 1951.

CHRISTOPHER, Alexander. ISHIKAWA, Sara. SILVERSTEIN, Murray. A Pattern Language. Londres, 1977.

GAMMA, Erich. HELM, Richard. JOHNSON, Ralph. VLISSIDES, John. Padrões de Projetos: soluções reutilizáveis de Software Orientado a Objetos. Apresentação de GRADY BOOCH. Londres, 1994.