DOO
Design Orientado a Objeto

Estevão Cristino
Gabriel Pacheco
João Pedro Rodrigues
Pedro Henrique Coimbra

Goiânia, 20 de setembro de 2018

1) Introdução

- O que é DOO?

É a disciplina de definir os objetos e suas interações para resolver um problema que foi identificado e documentado durante a Análise Orientada a Objetos, ou seja, é um método de design que engloba o processo de decomposição orientada a objetos e uma notação para representar modelos lógicos e físicos, bem como estaduais e dinâmicos, do sistema em projeto.

2) Conceitos e Princípios

O Design Orientado a Objeto é permeado e guiado por diversos conceitos que definem como o mesmo será planejado e executado. Este conceitos possuem diferentes níveis, onde pode-se citar conceitos mais abertos e objetivos como “o que é um objeto”, que é um conceito mais concreto de fácil visualização. Também pode-se citar conceitos um pouco mais abstratos como “o que é herança” ou como “o que é coesão e acoplamento”, assim como também o DOO é permeado e guiado por princípios.

Onde, se executados de maneira correta se tem uma aplicação do DOO de forma concreta e otimizada, otimizando também a qualidade do produto de software criado.

Portanto neste tópico serão listados os principais conceitos com suas definições:

- Abstração: é a capacidade de transformar entidades reais no sistema de software, ou seja, representar entidades reais listadas do diagrama de caso de uso em forma de classes.

- Objeto: Objeto é a instanciação de uma classe, ou seja, a aplicação de uma classe.

- Encapsulamento e proteção da informação: é o ato de se esconder ideias do usuário. Uma classe possui dados e métodos reunidos, onde informações importantes não podem ser acessadas diretamente, logo o ato de esconder estes dados do usuário é o encapsulamento. Para tal os atributos da classe são construídos com modo *private.*

- Herança: O conceito em DOO é o mesmo conceito do que na vida real. Onde na vida real herança é quando um filho herda coisas de seus pais, e isto ocorre em software. Onde classes filhas podem herdar atributos e métodos de sua classe pai.

- Interface: é o contato da classe com o mundo exterior/usuário. É uma definição de dados e métodos disponíveis para uso para manipular um objeto.

- Polimorfismo: é a habilidade de mudar o comportamento, por exemplo de um método, de uma classe derivada herdando este método da classe base. Ou seja, classes derivadas de uma única classe base são capazes de invocar os métodos que, embora apresentem a mesma assinatura, comportam-se de maneira diferente para cada uma das classes derivadas.

Levando em conta os princípios de execução e boas práticas, o DOO apresenta cinco “pilares”, ou seja, cinco princípios que guiam o DOO para aplicação mais otimizada, onde sua maior preocupação é a coesão e acoplamento das classes no projeto de sistema de software.

Estes cinco princípios formam o acrônimo SOLID, onde estes princípios são:

- Single Responsibility Principle (princípio da responsabilidade única): tem como foco as classes de um sistema e suas responsabilidades. Este princípio define que cada classe deve possuir apenas uma única responsabilidade tendo assim coesão ao máximo; onde responsabilidade é uma tarefa ou ação que a classe deve realizar.

- Open Closed Principle (princípio do aberto/fechado): o foco deste princípio já está relacionado com a manutenção das classes do sistema, para isso a definição mais básica deste princípio está na frase “Classes devem ser abertas para extensão, mas fechadas para manutenção”. Logo a classe terá dados e métodos pertinentes privados, porém sua aplicação aberta está disponível para extensão permitindo assim “modificação” em métodos. Pode-se perceber que este princípio está intimamente ligado à criação de classes abstratas que se diferenciam através de um tipo comum.

- Liskov Substitution Principle (princípio da substituição de Liskov): O princípio da substituição é uma extensão do princípio aberto/fechado,onde é definido que uma subclasse pode substituir sua classe base. Essa substituição define o comportamento da classe filha, que deve refletir o escopo da classe base.

- Interface Segregation Principle (princípio da segregação de Interfaces): Já este princípio também tem uma correlação com outro princípio já listado, que é o princípio da responsabilidade única. O princípio da segregação de interfaces afirma que que uma interface não pode forçar uma classe a implementar métodos que não pertencem a ela, ou seja, uma interface deve respeitar a coesão da classe para não poluir a implementação.

- Dependency Inversion Principle (princípio da inversão de dependência): A peculiaridade deste princípio está em ser quase contraditório em relação ao anterior, ao ser guiado na execução por estes dois princípios deve-se tomar cuidado para que não ocorra discordância dos trabalhos. Pois este princípio afirma que toda classe de alto nível seja completamente independente de suas subclasses. Ou seja, as classes não podem depender dos detalhes de implementação de seus métodos e classes, e isto é resolvido no design voltado para interfaces, onde a dependência das classes ocorre por meio das interfaces.

3) Entradas para DOO

As entradas para o projeto orientado a objetos é fornecida pela saída da análise orientada a objetos. Um artefato de saída da análise não precisa ser completamente desenvolvido para servir como entrada de DOO, pois a análise e o projeto podem ocorrer em paralelo e, na prática, os resultados de uma atividade podem alimentar o outro em um ciclo de feedback curto por meio de um processo iterativo. Tanto a análise quanto o design podem ser executados de forma incremental, e os artefatos podem ser continuamente desenvolvidos em vez de serem completamente desenvolvidos em uma única vez.

Alguns artefatos de entrada típicos para DOO são:

- **Modelo Conceitual:** O resultado da análise orientada a objetos, captura conceitos no domínio do problema. O modelo conceitual é explicitamente escolhido para ser independente dos detalhes da implementação, como simultaneidade ou armazenamento de dados.

- **Casos de Uso:** Uma descrição de sequências de eventos que, em conjunto, levam a um sistema fazendo algo útil. Cada caso de uso fornece um ou mais cenários que transmitem como o sistema deve interagir com os usuários chamados agentes para alcançar uma meta ou função de negócios específica. Os atores de casos de uso podem ser usuários finais ou outros sistemas. Em muitas circunstâncias, os casos de uso são elaborados em diagramas de casos de uso. Os diagramas de casos de uso são usados para identificar o agente (usuários ou outros sistemas) e os processos que eles executam.

- **Diagrama de Sequência do Sistema:** Um diagrama de sequência do sistema (SSD) é uma figura que mostra, para um cenário específico de um caso de uso, os eventos que os atores externos geram, sua ordem e possíveis eventos entre sistemas.

4) Design Orientado a Objetos

- **Design do Sistema**

O design orientado a objetos de um sistema envolve a definição de um contexto seguido de um projeto da arquitetura do sistema. O contexto se define por possuir uma parte estática e dinâmica. A contexto estático do sistema é projetado apenas usando um simples diagrama de pacotes UML de todo o sistema, expandido para os subsistemas. O contexto dinâmico descreve como o sistema interage com seu ambiente, através dos casos de uso. A arquitetura do sistema é projetada de acordo com o contexto, princípios arquitetônicos e domínio, normalmente particionado em camadas, formando os subsistemas.

- Decomposição Orientada a Objetos

  A decomposição remete a divisão de um sistema maior em uma hierarquia de partes menores, em que cada uma destas é chamada de subsistema. Com a orientação a objetos, é possível identificar os objetos autônomos e a comunicação entre estes. Tal metodologia permite a redução de complexidade geral do sistema, divisão do trabalho a ser realizado e maior manutenibilidade.

- Identificar Concorrência

  A simultaneidade permite que mais de um objeto recebe eventos ao mesmo tempo e mais de uma atividade seja executada simultaneamente. Para habilitar a simultaneidade, cada elemento concorrente é atribuído a um thread separado de controle. Se a simultaneidade estiver no nível do objeto, dois objetos simultâneos serão atribuídos a dois encadeamentos diferentes de controle. Se duas operações de um único objeto forem de natureza concorrente, esse objeto será dividido entre segmentos diferentes. A simultaneidade está associada aos problemas de integridade de dados, impasse e inanição. Portanto, uma estratégia clara precisa ser feita sempre que a simultaneidade for necessária.

- Identificar Padrões

  Ao projetar aplicativos, algumas soluções comumente aceitas são adotadas para algumas categorias de problemas. Estes são os padrões de design. Um padrão pode ser definido como um conjunto

documentado de blocos de construção que podem ser usados em determinados tipos de problemas de desenvolvimento de aplicativos. Padrões normalmente usados são: Fachada, Observador, Proxy, dentre outros.

- Controlar Eventos

    Durante o projeto do sistema, os eventos que podem ocorrer nos objetos do sistema precisam ser identificados e tratados apropriadamente. Um evento é uma especificação de uma ocorrência significativa que possui uma localização no tempo e no espaço. Os 4 tipos de eventos a serem modelados são: sinal, chamada, tempo e alteração.

- Gerenciar Condições de Fronteira

    A fase de projeto do sistema precisa abordar a inicialização e a finalização do sistema como um todo, bem como de cada subsistema. Os diferentes aspectos documentados são: o início do sistema, isto é, a transição do sistema do estado não inicializado para o estado estacionário; O término do sistema, ou seja, o fechamento de todos os encadeamentos em execução, a limpeza de recursos e as mensagens a serem enviadas.

- **Design do Objeto**

Após a hierarquia de subsistemas ter sido desenvolvida, os objetos no sistema são identificados e seus detalhes são projetados. Aqui, o designer detalha a estratégia escolhida durante o projeto do sistema. A ênfase muda de conceitos de domínio de aplicativos para conceitos de computador. Os objetos identificados durante a análise são gravados para implementação com o objetivo de minimizar o tempo de execução, o consumo de memória e o custo geral.

    - Identificação do Objeto

        O primeiro passo para o design de objetos é a identificação destes. Os objetos identificados nas fases de análise orientadas a objeto são agrupados em classes e refinados, de modo que sejam adequados para a implementação real. Identifica-se as classes em cada subsistema; define-se as associações entre as classes; projeta-se associações hierárquicas entre classes e projeta-se suas agregações.

- Representação do Objeto

  Depois que as classes são identificadas, elas precisam ser representadas usando técnicas de modelagem de objetos. Este estágio envolve essencialmente a construção de diagramas UML, produzindo-se dois tipos de modelos: estáticos (descrevem a estrutura estática de um sistema usando diagramas de classes e objetos) e dinâmicos (descrevem a estrutura dinâmica de um sistema e mostrar a interação entre classes usando diagramas de interação e diagramas de gráfico de estado).

- Classificação das Operações

  Nesta etapa, a operação a ser executada em objetos é definida pela combinação dos três modelos desenvolvidos na fase OOA, a saber, modelo de objeto, modelo dinâmico e modelo funcional. Uma operação específica o que deve ser feito e não como deve ser feito. São realizadas as seguintes tarefas: diagrama de transição de estados de cada objeto, operações para os eventos são definidas e suboperações dentro das ações são identificadas.

- Design do Algoritmo

  As operações nos objetos são definidas usando algoritmos. Um algoritmo é um procedimento passo a passo que resolve o problema estabelecido em uma operação. Algoritmos se concentram em como isso deve ser feito. Pode haver mais de um algoritmo correspondente a uma determinada operação. Depois que os algoritmos alternativos são identificados, o algoritmo ideal é selecionado para o domínio de problema fornecido. As métricas para escolher o algoritmo ideal são: a complexidade computacional, a flexibilidade e a compreensibilidade.

- Design dos Relacionamentos

  A estratégia para implementar os relacionamentos precisa ser delimitada durante a fase de design do objeto. Os principais relacionamentos abordados são compostos de associações, agregações e heranças. Deve ser feito: identificação de associações unidirecionais ou bidirecionais, análise do caminho das associações, implementação das associações como um objeto distinto. Sobre as

heranças, deve ser feito: ajuste nas classes e associações, identificação de classes abstratas e assegurar-se que os comportamentos serão compartilhados quando necessário.

- **Otimização do Design**

Antes de se iniciar a implementação do design, este deve ser otimizado para que sua implementação seja mais eficiente. O foco da otimização é minimizar o custo.

Porém, não se deve otimizar em excesso, pois a otimização, apesar de diminuir custos na implementação, quando feita em excesso, pode ferir a facilidade na implementação, na manutenibilidade, e na extensibilidade do software, que são preocupações muito importantes para a engenharia de software. Um design perfeitamente otimizado é mais eficiente, porém menos compreensível e reusável. Então o projetista deverá ter em mente que se deve achar um meio termo.

A otimização pode ser alcançada utilizando-se das seguintes técnicas:

- Adição de Associações Redundantes:

  Durante a otimização do design, os modelos UML criados são analisados para verificar se a derivação de novas associações pode reduzir o custo de acesso, se sim, gerando associações redundantes. Apesar destas não deixarem o design mais informativo, o modelo em geral fica mais eficiente.

- Omissão de Associações Não-Utilizáveis:

  A existência de associações em excesso, por outro lado, pode deixar o sistema menos compreensível. Ou seja, o design será menos eficiente. Então, durante a otimização, todas as associações não utilizáveis deverão ser removidas.

- Otimização de Algoritmos:

  Assim que o design das classes estiver finalizado, as operações e algoritmos deverão ser otimizados da seguinte forma:
  - Rearranjando a ordem das tarefas computacionais;
  - Revertendo a ordem de execução dos laços definidos no modelo funcional.

- Salvar e Armazenar Atributos Derivados:

Atributos derivados são aqueles obtidos através de um cálculo baseado em outros atributos (atributos base). Realizar o cálculo destes atributos toda vez que for necessário pode ser um procedimento que gasta muito esforço desnecessário. Para evitar isso, esses valores calculados podem ser armazenados.

Contudo, armazenar os atributos derivados pode causar anomalias, como por exemplo, uma mudança em um valor base sem alterar nada no valor derivado. Para evitar isso, pode-se realizar os seguintes passos:

- Com cada atualização no valor dos atributos-base, o atributo derivado é recalculado;
- Todos os atributos derivados são recalculados periodicamente.

- **Documentação do Design**

Como toda área no desenvolvimento de software, as decisões de design deverão ser documentadas a fim de facilitar o entendimento do design de quem precisar.

- Áreas de Uso

  Uma boa documentação é indispensável para os seguintes casos:
  - Design de software que será desenvolvido por um grupo de desenvolvedores;
  - Estratégias de desenvolvimento de software interativas;
  - Desenvolvimento de versões subsequentes de um software;
  - Encontrar condições e áreas de teste;
  - Manutenção de software.

- Conteúdo

  Uma documentação de qualidade deverá ter essencialmente o seguinte conteúdo:
  - Arquitetura do sistema em alto nível - Diagrama de processos e diagramas de módulo;
  - Abstrações e mecanismos chave - Diagramas de classe e Diagramas de objeto;
  - Ilustrações do comportamento dos aspectos principais - Diagramas de comportamento
    .

- Características

Uma boa documentação deverá ser:

- Concisa, consistente e completa. Evitando ambiguidades;
- Facilmente relacionável às especificações de requisitos;
- Bem estruturada;
- Mais diagramática e menos descritiva.

5) Saídas do DOO

- Diagrama de Sequência: Um diagrama de seqüência mostra, como linhas verticais paralelas, diferentes processos ou objetos que vivem simultaneamente e, como setas horizontais, as mensagens trocadas entre eles, na ordem em que ocorrem. Isso permite a especificação de cenários simples de tempo de execução de maneira gráfica.
  Os diagramas de sequência geralmente são associados a realizações de casos de uso na Visão lógica do sistema em desenvolvimento. Os diagramas de seqüência às vezes são chamados de diagramas de eventos ou cenários de eventos .

- Diagrama de Classes: Um diagrama UML de estrutura estática que descreve a estrutura de um sistema, mostrando as classes do sistema, seus atributos e os relacionamentos entre as classes. As mensagens e classes identificadas através do desenvolvimento dos diagramas de seqüência podem servir como entrada para a geração automática do diagrama de classes global do sistema.

6) Conclusão

Por que DOO?

Sempre que iniciamos o desenvolvimento de um software, estamos buscando a resolução de algum problema ou de uma situação do mundo real. Por conta disso, a missão de representar essa solução de forma a ser entendida por um computador se torna complexa e onerosa. Porém, o mercado sempre busca representar estes cenários da forma mais detalhada e próxima da realidade possível. Sendo assim, faz-se necessária a utilização de métodos que facilitem esta representação. O Design Orientado a Objetos tem como base conceitos como abstração, encapsulamento, decomposição, generalização e composição, onde todos estes conceitos contribuem para uma melhor representação dos cenários da vida real.

7) Referências

Livro:
McLaughlin, Brett D. Pollice, Gary. West, David. Head First Object-Oriented Analysis and Design. Estados Unidos: O'Reilly Media, Inc, 2007.

Artigo:
Schmitz, Daniel Pace. Araújo, Marcos AP. Utilização dos princípios SOLID na aplicação de Padrões de Projeto. Engenharia de Software Magazine, Edição 50.

Links:
What is Object Oriented Design? (OOD). Disponível em: <http://www.selectbs.com/process-maturity/what-is-object-oriented-design>. Acesso em: 20 set. 2018.

OOAD - Object Oriented Design. Disponível em: <https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_design.htm>. Acesso em: 20 set. 2018.