Resumo geral P2 Eng Software (para estudo).

PEDRO LUCAS DOS REIS SILVA

Resumo do Capítulo 5: Modelagem de Sistemas

1. Introdução à Modelagem de Sistemas A modelagem de sistemas envolve o desenvolvimento de modelos abstratos que oferecem diferentes perspectivas do sistema, utilizando notações gráficas como a UML (Unified Modeling Language). Esse processo é fundamental para entender a funcionalidade do sistema e comunicar-se eficazmente com os clientes.

2. Modelos de Sistemas Existentes e Planejados

- **Modelos de Sistemas Existentes:** Utilizados durante a engenharia de requisitos para esclarecer o que o sistema atual faz, facilitando discussões sobre melhorias e novos requisitos.
- **Modelos de Novos Sistemas:** Ajudam a explicar os requisitos propostos e documentar o sistema para implementação. Em processos dirigidos a modelos, é possível gerar implementações a partir desses modelos.

3. Perspectivas de Sistemas

- **Perspectiva Externa:** Modela o contexto ou ambiente do sistema.
- **Perspectiva de Interação:** Modela interações entre o sistema e seu ambiente ou entre componentes do sistema.
- **Perspectiva Estrutural:** Modela a organização do sistema e a estrutura dos dados processados.
- **Perspectiva Comportamental:** Modela o comportamento dinâmico do sistema e suas respostas a eventos.

4. Tipos de Diagramas UML

- **Diagramas de Atividades:** Mostram atividades envolvidas em processos ou no processamento de dados.
- **Diagramas de Casos de Uso:** Mostram interações entre o sistema e seu ambiente.
- **Diagramas de Sequência:** Mostram interações entre atores e o sistema.
- **Diagramas de Classe:** Mostram classes de objeto e associações entre elas.
- **Diagramas de Estado:** Mostram reações do sistema a eventos.

5. Uso de Modelos Gráficos

- **Discussão:** Facilita a discussão sobre sistemas existentes ou propostos.
- **Documentação:** Representação precisa de sistemas existentes para documentação.
- **Descrição Detalhada:** Usados para gerar implementações precisas e completas.

6. Modelos de Contexto Modelos de contexto ilustram o ambiente operacional do sistema, mostrando limites e relações com outros sistemas, influenciados por interesses sociais e organizacionais.

7. Modelos de Interação Essenciais para identificar requisitos do usuário, modelar problemas de comunicação e entender o desempenho do sistema. Utilizam diagramas de casos de uso e de sequência.

8. Modelos Estruturais Mostram a organização dos componentes do sistema, seja de forma estática (estrutura do projeto) ou dinâmica (organização durante a execução). Incluem diagramas de classe para definir a estrutura estática de classes e associações.

9. Modelos Comportamentais Descrevem o comportamento dinâmico do sistema em resposta a estímulos, que podem ser dados ou eventos. Incluem diagramas de atividades e de estado para modelar o processamento de dados e as respostas a eventos.

10. Engenharia Dirigida a Modelos (MDE) Abordagem onde modelos, e não programas, são os principais produtos do desenvolvimento. Aumenta o nível de abstração, reduzindo a necessidade de detalhes de linguagem de programação e plataformas de execução.

11. Arquitetura Dirigida a Modelos (MDA) Precursora da MDE, usa UML para descrever sistemas em diferentes níveis de abstração, gerando programas funcionais a partir de modelos independentes de plataforma (PIM).

12. Tipos de Modelos

- **Modelo Independente de Computação (CIM):** Abstrações de domínio importantes usadas no sistema.
- **Modelo Independente de Plataforma (PIM):** Operação do sistema sem referência à implementação.
- **Modelos Específicos de Plataforma (PSM):** Transformações do PIM para cada plataforma específica.

13. UML Executável Subconjunto da UML para permitir a transformação automática de modelos em códigos, focando em modelos de domínio, classe e estado.

14. Métodos Ágeis e MDA MDA pode auxiliar no desenvolvimento ágil, embora haja contradições com o manifesto ágil. Transformações automáticas completas poderiam integrar MDA ao desenvolvimento ágil.

15. Pontos Importantes

- Modelos comportamentais descrevem o comportamento dinâmico de sistemas em execução.
- Diagramas de atividades e de estado modelam processamento de dados e respostas a eventos.
- MDE representa sistemas como conjuntos de modelos transformáveis em códigos executáveis.

Resumo do Capítulo 6: Projeto de Arquitetura

1. Decisões de Projeto de Arquitetura

- **Arquiteturas de Software:** Processo para identificar subsistemas e o framework para controle e comunicação.
- **Projeto de Arquitetura:** Fase inicial para identificar componentes principais e suas comunicações, conectando especificação e design.

2. Abstração sobre a Arquitetura

- **Escala Pequena:** Foca na decomposição de programas individuais.
- **Escala Grande:** Envolve sistemas complexos distribuídos por diferentes computadores e organizações.

3. Vantagens da Arquitetura Explícita

- Facilita a comunicação entre stakeholders.
- Permite análise do sistema para verificar requisitos não-funcionais.
- Suporta reuso em larga escala e desenvolvimento de linhas de produtos.

4. Representações de Arquiteturas

- **Diagramas de Blocos:** Simples e úteis para comunicação, mas limitados em detalhes semânticos.
- **Diagramas de Caixa e Linha:** Abstratos, úteis para planejamento e comunicação, mas carecem de detalhes sobre relacionamentos.

5. Uso de Modelos de Arquitetura

- **Discussão de Projeto:** Visões de alto nível facilitam comunicação com stakeholders e planejamento.
- **Documentação:** Modelos completos mostram componentes, interfaces e conexões.

6. Decisões de Projeto de Arquitetura

- Considerar arquiteturas genéricas, distribuição do sistema, estilos apropriados, estruturação do sistema, decomposição em módulos, estratégias de controle, avaliação e documentação da arquitetura.

7. Reuso de Arquitetura

- Sistemas no mesmo domínio frequentemente compartilham arquiteturas similares.
- Linhas de produtos utilizam uma arquitetura central com variantes para diferentes requisitos.

8. Características de Arquitetura e de Sistema

- **Desempenho:** Minimizar comunicações e usar componentes de alta granularidade.
- **Proteção e Segurança:** Arquitetura em camadas para proteger ativos críticos.

- **Disponibilidade:** Componentes redundantes e tolerância a falhas.
- **Manutenibilidade:** Componentes autocontidos e de baixa granularidade.

9. Visões de Arquitetura

- Visões múltiplas (lógica, processo, desenvolvimento, física e cenários) são necessárias para projetar e documentar a arquitetura.

10. Padrões de Arquitetura

- **MVC (Modelo-Visão-Controlador):** Divide a aplicação em três componentes principais.
- **Arquitetura em Camadas:** Organiza o sistema em camadas de serviços, suportando desenvolvimento incremental.

11. Arquitetura de Repositório

- **Repositório Compartilhado:** Subsistemas compartilham dados em um banco de dados central.
- **Repositório Independente:** Cada subsistema tem seu próprio banco de dados e transmite dados explicitamente.

12. Arquitetura Cliente-Servidor

- **Modelo Cliente-Servidor:** Servidores prestam serviços específicos e clientes solicitam esses serviços através de uma rede.

13. Arquitetura de Duto e Filtro

- **Transformações Funcionais:** Processam entradas para produzir saídas, comumente usados em sistemas de processamento de dados.

14. Arquiteturas de Aplicações

- **Arquiteturas Genéricas:** Projetadas para atender necessidades organizacionais comuns, adaptáveis para requisitos específicos.

15. Sistemas de Processamento de Transações

- **Sistemas Interativos:** Processam pedidos de usuários para informações ou atualizações de banco de dados.

16. Sistemas de Informação Baseados na Web

- Implementados como arquiteturas cliente-servidor multicamadas com interfaces web, lógica de aplicação e gerenciamento de banco de dados.

17. Sistemas de Processamento de Linguagem

- **Processamento de Linguagem:** Incluem um interpretador para executar instruções em uma linguagem formal.

18. Componentes do Compilador

- **Análise Léxica e Sintática:** Convertem e verificam a linguagem de entrada.
- **Árvore de Sintaxe:** Representa o programa para geração de código de máquina abstrata.

19. Pontos Importantes

- Modelos genéricos ajudam a entender, comparar, validar e reutilizar arquiteturas de sistemas de aplicação.

Exemplos de Elementos do Diagrama de Classes (exceto Classes, Atributos e Métodos)

Associações

As Associações representam relacionamentos entre duas ou mais classes. Elas são ilustradas como linhas que conectam as classes.

Exemplo: Relacionamento entre `Professor` e `Curso`.

plaintext
Copiar código

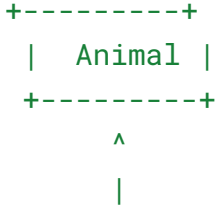
- Neste exemplo, cada `Professor` pode estar associado a vários `Cursos` e cada `Curso` pode ter vários `Professores`.

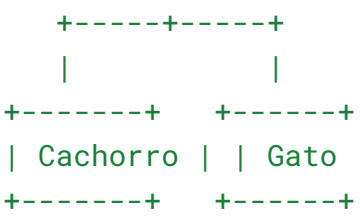
Generalização (Herança)

Generalização é uma relação onde uma classe (subclasse) herda atributos e métodos de outra classe (superclasse). Representada por uma linha sólida com um triângulo no final apontando para a superclasse.

Exemplo: Relacionamento de herança entre `Animal`, `Cachorro` e `Gato`.

plaintext
Copiar código





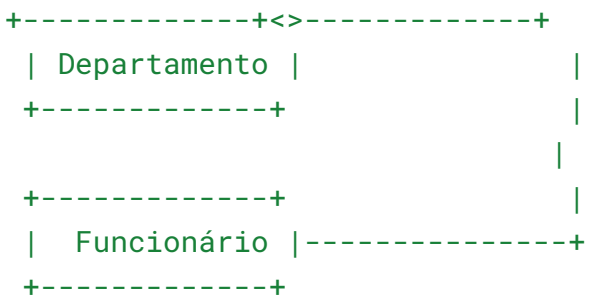
- `Cachorro` e `Gato` herdam de `Animal`.

Agregação

Agregação é um tipo de associação que representa uma relação "todo/parte", onde a parte pode existir independentemente do todo. Representada por uma linha com um losango aberto na extremidade do todo.

Exemplo: Relacionamento entre `Departamento` e `Funcionário`.

plaintext
Copiar código



- `Departamento` agrega `Funcionário`, mas `Funcionário` pode existir independentemente.

Composição

Composição é uma forma forte de agregação que indica uma relação "todo/parte" onde a parte não pode existir independentemente do todo. Representada por uma linha com um losango preenchido na extremidade do todo.

Exemplo: Relacionamento entre `Casa` e `Quarto`.

plaintext
Copiar código

```
|  Quarto |-------------+
+--------+
```

- `Quarto` não pode existir independentemente de `Casa`.

Multiplicidade

Multiplicidade indica quantos objetos de uma classe podem estar associados a um único objeto de outra classe. Representada por números próximos às extremidades das linhas de associação.

Exemplo: Relacionamento entre `Pedido` e `Produto`.

plaintext
Copiar código

```
+---------+1       *+----------+
|  Pedido |---------|  Produto  |
+---------+         +----------+
```

- Um `Pedido` pode conter vários `Produtos` e cada `Produto` pode estar em vários `Pedidos`.

Conclusão

Os elementos adicionais do Diagrama de Classes, como Associações, Generalização, Agregação, Composição e Multiplicidade, são essenciais para representar a complexidade e a estrutura dos sistemas de software. Esses relacionamentos ajudam a detalhar como as diferentes classes interagem e se relacionam entre si, proporcionando uma visão clara e compreensível do sistema modelado.

Perguntas Teóricas sobre Outros Diagramas da UML

1. **Diagrama de Casos de Uso (Use Case Diagram)**
   - **Pergunta**: Qual é o objetivo principal de um Diagrama de Casos de Uso?
   - **Resposta**: O objetivo principal é representar as interações entre os atores externos (usuários ou outros sistemas) e o sistema, mostrando os casos de uso que descrevem as funcionalidades do sistema.
2. **Diagrama de Sequência (Sequence Diagram)**
   - **Pergunta**: Como um Diagrama de Sequência ajuda na modelagem de sistemas?
   - **Resposta**: Ele ajuda a descrever a interação entre os objetos em um determinado cenário de tempo, mostrando a ordem das mensagens trocadas para realizar uma funcionalidade específica.
3. **Diagrama de Atividades (Activity Diagram)**
   - **Pergunta**: Quando é apropriado usar um Diagrama de Atividades?

    - **Resposta**: É apropriado quando se deseja modelar fluxos de trabalho ou processos, detalhando o fluxo de controle e de dados entre as atividades do sistema.
4. **Diagrama de Estados (State Diagram)**
    - **Pergunta**: O que representa um Diagrama de Estados?
    - **Resposta**: Representa os estados pelos quais um objeto passa durante seu ciclo de vida em resposta a eventos, mostrando as transições de estado, eventos que causam essas transições e ações resultantes.
5. **Diagrama de Componentes (Component Diagram)**
    - **Pergunta**: Qual é o propósito de um Diagrama de Componentes?
    - **Resposta**: O propósito é modelar a arquitetura física de um sistema, mostrando a organização e dependência entre componentes de software.
6. **Diagrama de Implantação (Deployment Diagram)**
    - **Pergunta**: O que é representado em um Diagrama de Implantação?
    - **Resposta**: Representa a configuração física de hardware e software no sistema, mostrando onde os componentes de software serão executados no hardware.

Padrões de Projeto

Definição e Importância

Os padrões de projeto são soluções recorrentes para problemas comuns no desenvolvimento de software, permitindo a reutilização de soluções bem-sucedidas e a promoção de boas práticas de programação. Eles suportam reuso orientado a objeto em um alto nível de abstração e fornecem um framework que guia as restrições de projetos orientados a objetos.

Elementos de um Padrão

Cada padrão de projeto consiste em:

- **Nome:** Identificação do padrão.
- **Problema:** Descrição das condições em que o padrão pode ser aplicado.
- **Solução:** Descrição abstrata de como resolver o problema usando classes e objetos disponíveis.
- **Consequências:** Custos e benefícios de aplicar o padrão, impacto na flexibilidade, extensibilidade, portabilidade e eficiência do sistema.

Classificações de Padrões

Padrões de projeto podem ser classificados de várias formas:

- **Por Propósito:**
    1. Criação de classes e objetos.
    2. Alteração da estrutura de um programa.
    3. Controle do comportamento do programa.
- **Por Escopo:** Classe ou objeto.

Padrões GoF (Gang of Four)

Os padrões de projeto descritos pelos autores Gamma, Helm, Johnson e Vlissides (conhecidos como Gang of Four - GoF) são amplamente utilizados e se dividem em três categorias principais:

- **Padrões de Criação:** Lidam com a criação de objetos.
  - **Exemplos:** Singleton, Factory Method, Abstract Factory, Builder, Prototype.
- **Padrões Estruturais:** Lidam com a composição de classes ou objetos.
  - **Exemplos:** Adapter, Composite, Proxy, Flyweight, Facade, Bridge, Decorator.
- **Padrões Comportamentais:** Lidam com a comunicação entre objetos.
  - **Exemplos:** Template Method, Observer, Strategy, Command, State, Visitor, Chain of Responsibility, Mediator, Iterator, Memento, Interpreter.

Padrões de Projeto

Definição e Importância

Os padrões de projeto são soluções recorrentes para problemas comuns no desenvolvimento de software, permitindo a reutilização de soluções bem-sucedidas e a promoção de boas práticas de programação. Eles suportam reuso orientado a objeto em um alto nível de abstração e fornecem um framework que guia as restrições de projetos orientados a objetos.

Elementos de um Padrão

Cada padrão de projeto consiste em:

- **Nome:** Identificação do padrão.
- **Problema:** Descrição das condições em que o padrão pode ser aplicado.
- **Solução:** Descrição abstrata de como resolver o problema usando classes e objetos disponíveis.
- **Consequências:** Custos e benefícios de aplicar o padrão, impacto na flexibilidade, extensibilidade, portabilidade e eficiência do sistema.

Classificações de Padrões

Padrões de projeto podem ser classificados de várias formas:

- **Por Propósito:**
  1. Criação de classes e objetos.
  2. Alteração da estrutura de um programa.
  3. Controle do comportamento do programa.
- **Por Escopo:** Classe ou objeto.

Padrões GoF (Gang of Four)

Os padrões de projeto descritos pelos autores Gamma, Helm, Johnson e Vlissides (conhecidos como Gang of Four - GoF) são amplamente utilizados e se dividem em três categorias principais:

- **Padrões de Criação:** Lidam com a criação de objetos.
    - **Exemplos:** Singleton, Factory Method, Abstract Factory, Builder, Prototype.
- **Padrões Estruturais:** Lidam com a composição de classes ou objetos.
    - **Exemplos:** Adapter, Composite, Proxy, Flyweight, Facade, Bridge, Decorator.
- **Padrões Comportamentais:** Lidam com a comunicação entre objetos.
    - **Exemplos:** Template Method, Observer, Strategy, Command, State, Visitor, Chain of Responsibility, Mediator, Iterator, Memento, Interpreter.

Padrões de Projeto

Os padrões de projeto são soluções gerais e reutilizáveis para problemas comuns no design de software. Eles promovem a reutilização de soluções comprovadas, facilitam a comunicação entre desenvolvedores e ajudam na criação de software mais robusto e flexível.

Elementos de um Padrão

Cada padrão de projeto consiste em quatro elementos principais:

- **Nome:** Identificação do padrão.
- **Problema:** Descrição do problema que o padrão resolve e as condições de aplicação.
- **Solução:** Abordagem abstrata para resolver o problema, incluindo a estrutura de classes e objetos.
- **Consequências:** Impactos do uso do padrão, incluindo benefícios e possíveis desvantagens.

Classificação dos Padrões GoF

Os padrões de projeto da Gang of Four (GoF) são classificados em três categorias principais:

1. **Padrões de Criação:** Lidam com a criação de objetos.
    - **Singleton:** Garante que uma classe tenha apenas uma instância e fornece um ponto global de acesso a ela.
    - **Factory Method:** Define uma interface para criar um objeto, mas permite que subclasses decidam qual classe instanciar.
    - **Abstract Factory:** Fornece uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.
    - **Builder:** Separa a construção de um objeto complexo da sua representação, permitindo a criação passo a passo.
    - **Prototype:** Permite a criação de novos objetos a partir de protótipos existentes, clonando-os.
2. **Padrões Estruturais:** Lidam com a composição de classes ou objetos.
    - **Adapter:** Permite que interfaces incompatíveis trabalhem juntas, convertendo a interface de uma classe em outra esperada pelos clientes.
    - **Bridge:** Desacopla uma abstração da sua implementação, permitindo que as duas variem independentemente.

    - **Composite:** Composição de objetos em estruturas de árvore para representar hierarquias parte-todo.
    - **Decorator:** Adiciona responsabilidades a objetos dinamicamente.
    - **Facade:** Fornece uma interface simplificada para um subsistema complexo.
    - **Flyweight:** Usa compartilhamento para suportar grandes quantidades de objetos de forma eficiente.
    - **Proxy:** Fornece um substituto ou marcador para outro objeto para controlar o acesso a ele.
3. **Padrões Comportamentais:** Lidam com a comunicação entre objetos.
    - **Chain of Responsibility:** Passa uma solicitação ao longo de uma cadeia de handlers.
    - **Command:** Encapsula uma solicitação como um objeto, permitindo parametrizar clientes com diferentes solicitações, filas ou logs de solicitações.
    - **Interpreter:** Interpreta sentenças de uma linguagem.
    - **Iterator:** Fornece uma maneira de acessar os elementos de um objeto agregado sequencialmente sem expor sua representação subjacente.
    - **Mediator:** Define um objeto que encapsula a forma como um conjunto de objetos interage.
    - **Memento:** Captura e externaliza o estado interno de um objeto sem violar a encapsulação, para que o objeto possa ser restaurado a esse estado mais tarde.
    - **Observer:** Define uma dependência de um-para-muitos entre objetos, de modo que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente.
    - **State:** Permite que um objeto altere seu comportamento quando seu estado interno muda.
    - **Strategy:** Define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis.
    - **Template Method:** Define o esqueleto de um algoritmo, deferindo alguns passos para subclasses.
    - **Visitor:** Representa uma operação a ser realizada nos elementos de uma estrutura de objeto.

Aplicação e Benefícios

Os padrões de projeto são aplicados para resolver problemas recorrentes no desenvolvimento de software. Eles ajudam a criar um código mais flexível, reutilizável e fácil de manter, além de facilitar a comunicação entre desenvolvedores por meio de um vocabulário comum de soluções.