**Estudo Arquitetura de Software (segunda unidade)**

1. **Revisão de Requisitos, código fonte e modelo**
   1. **Requisito** – Documentam “o que um sistema deve fazer, valendo-se de um nível de abstração próximo do problema e de seus usuários.
   2. **Código fonte-** é uma representação concreta, de baixo nível e executável do comportamento de um sistema.
   3. **Modelo** – São criados para ajudar no entendimento e análise de um sistema, e para cumprir sua missão em engenharia de software por serem mais detalhados do que requisitos e mas ainda menos complexos do que o codigo fonte.
2. **Modelo**
   1. **Conceito** -são representações gráficas de determinadas dimensões de um sistema de software, sendo utilizadas durante o levantamento de requisitos voltadas para a definição do problema que será resolvido pelo sistema. Quando se avança para atividades de projeto, o problema já deve estar devidamente entendido e as atenções se voltam para concepção de uma solução capaz de resolvê-lo, para depois usar uma linguagem de programação.
   2. **Modelos formais** – São modelos matemáticos, valendo-se de uma notação matemática baseada em lógica, teoria de conjuntos ou Redes de Petri, com isso, deriva especificações formais para sistemas de software. Assim, prova-se propriedades de um sistema mesmo antes da implementação.
3. **UML**
   1. **Conceito** – é uma notação gráfica para modelagem de software.
   2. **Ferramentas para desenhar** – CASE (Computer-Aided Software Engineering).
   3. **Como usar?** Definido por Martin Fowler existem três formas: com blueprint, como linguagem de programação e como esboço.
      1. **UML como blueprint** – Defende-se que após o levantamento de requisitos seja produzido um conjunto de modelos ou plantas técnicas que seriam criados pelos analistas de sistemas usando-se as ferramentas CASE e depois repassados para os programadores codificarem, sendo usados Processo Unificado (UP).
      2. **UML com linguagem de programação** – corresponde ao uso UML vislumbrado pela OMG, sendo conhecida como Desenvolvimento Dirigido por Modelos (MDD).
      3. **UML como esboço** – Nele usamos UML para construir diagramas leves e informais de partes de um sistema, vindo daí o nome esboço(sketch). Sendo assim, são desenvolvidos com em duas situações principais: Engenharia avante (quando os desenvolvedores usam os modelos de UML para discutir e analisar alternativas de design, antes que exista qualquer código, portanto, o objetivo é valida a proposta de tais classes antes de começar a codificar) e a engenharia reversa quando os desenvolvedores usam os modelos UML para analisar e discutir uma funcionalidade que já encontra implementada no código fonte. O importante é observar que nesse momento não se preocupa em usar somas de dinheiro com uso de ferramentas CASE e sim criar um diagrama em quadro e depois fotografá-los e apagá-los., podendo usar um subconjunto de diagramas UML. Por outro lado, evita-se o uso de uma notação informal e ad hoc, cuja semântica pode ser não clara para todos os desenvolvedores.

**4 Diagramas UML**

**4.1 Classificados:**

* **Diagramas Estáticos** – Modelam a estrutura e organização de um sistema incluindo informações sobre classes, atributos, métodos, pacotes etc. Ou seja, os diagramas estáticos lidam apenas com informações que estão disponíveis sendo uma visão que não muda a não ser que sejam realizadas mudanças nos modelos. Portanto, se quiser modelar a estrutura de um programa usar modelos estáticos. Os tipos de diagramas estáticos: Diagramas de Classe e Diagrama de pacotes.
* **Diagramas Dinâmicos** – Modelam eventos que ocorrem durante a execução de um sistema., ou seja, é comum ter execuções diferentes de um mesmo programa porque o usuário pode executar um programa com entradas diferentes. Portanto, se quiser modelar o comportamento do sistema usar modelos dinâmicos. Os tipos de diagramas dinâmicos são: Diagrama Sequência e Diagrama de Atividades.

**4.2 Diagramas Estáticos:**

**4.2.1 Diagramas de Classe** – São os diagramas mais usados, fornecendo informações sobre atributos, métodos e relacionamentos que existem entre as classes modeladas.

Tabela

Descrição gerada automaticamente

- = Privados

+ = Públicos

**4.2.1.1 Relacionamentos**- São de três tipos: Associação, Herança e dependência.

**4.2.1.1.1 Associação** – Quando uma classe A possui um atributo de um tipo B, dizemos que existe uma associação de A paara B sendo representado por meio de seta e na extremidade da seta, informa-se o nome do atributo A responsável pela associação.

Tela de celular

Descrição gerada automaticamente com confiança média

Diagrama, Linha do tempo

Descrição gerada automaticamente

Lembrando a associação em sua grande maioria deve ser bidirecional, ou seja, duas setas de pessoa para fone e fone para pessoa.

Diagrama

Descrição gerada automaticamente

O importante também quando estudamos associação são dois conceitos: composição (que é uma relação na qual a classe de destino não pode existir de forma independente da classe original) e agregação (quando as classes tem ciclos independentes).

**4.2.1.1.2 Herança** – Quando temos a classe pai e a classe filha, ou seja, a classe tem subclasse que podem adicionar novos membros.

Diagrama

Descrição gerada automaticamente

**4.2.1.3 Dependência -** Existe uma dependência de uma classe A para uma classe B representada por uma seta com linha tracejada de A para B, quando uma classe A usa a classe B porém essse uso não ocorre por meio de associação ou herança, e portanto, dependências ocorrem quando uma classe A declara um parâmetro ou variável local do tipo B ou quando um métodos de A lança uma exceção de tipo B.

Forma, Retângulo

Descrição gerada automaticamente

Diagrama

Descrição gerada automaticamente

**4.2.1.4 Diagramas de Pacotes –** São recomendados quando se pretende oferecer um modelo de mais alto nível do sistema, que mostre grupos de classe e as dependências entre eles, e por isso, um retângulo especial como mostrada figura abaixo.

Diagrama

Descrição gerada automaticamente

4.2.1.5 Diagramas de Sequência -