Tema 2

Módulo 1

Os modelos são representações simplificadas da realidade, permitindo aos pesquisadores e profissionais analisarem, entenderem e preverem o comportamento de sistemas complexos. Eles servem como ferramentas poderosas para explorar cenários, testar hipóteses e tomar decisões embasadas.

**primeira finalidade de um modelo**: antecipar a existência de uma realidade para avaliar sua estrutura e comportamento.

**protótipos**, usados para aumentar a chance de sucesso dos produtos. A partir de um protótipo inicial, outros modelos podem ser demandados e aprimoramentos podem ser desenvolvidos.

**além da finalidade inicial,** funcionam também como instrumento de gerenciamento da complexidade, considerando a limitação humana em lidar com ela. Os sistemas grandes e complexos carecem de ser modelados para sua melhor compreensão em sua totalidade.

**Benefícios do uso de modelos no desenvolvimento de sistemas computacionais:**

**Comunicação entre as pessoas envolvidas:**

O modelo serve como elemento de comunicação ou difusão de informações entre as pessoas envolvidas em sua construção.

**Redução nos custos do desenvolvimento:**

A construção de modelos é bem mais barata que a construção do sistema em si. A descoberta de erros e falhas em modelos é bem menos onerosa e contribui para a redução dos custos finais do sistema computacional. Isso também vale para as eventuais necessidades de ajustes e melhorias no sistema.

**Facilidade para alterações do sistema:**

A análise de melhorias, seja na fase de construção ou de manutenção, tende a ser mais efetiva quando elaborada sobre os modelos construídos, aumentando a assertividade e diminuindo seus custos. Daí a relevância e a necessidade de manter os modelos sempre atualizados.

**Documentação do sistema:**

Os modelos servem de consulta e orientação a toda a equipe na construção e na manutenção do sistema, incluindo pessoas que sejam integradas após o início do desenvolvimento do sistema. Servem ainda para documentar as decisões tomadas.

**Delimitação do escopo do sistema:**

A modelagem ajuda na delimitação do escopo, ou seja, abrangência do sistema, definindo o que será ou não tratado pelo sistema.

**Algumas formas de abordar os sistemas computacionais por meio de visões:**

**Externa:**

Modela-se o ambiente em que o sistema está inserido, mostrando sua relação com os usuários e demais sistemas com que interage.

**Comportamental:**

Modela-se o comportamento dinâmico do sistema e como ele reage aos eventos que o afetam.

**Estrutural:**

Modela-se sua estrutura organizacional ou os dados que o sistema processa.

**Interacional:**

Modela-se as interações de seus componentes ou ainda do sistema e seu ambiente externo.

Módulo 1.1

**O processo de desenvolvimento de sistemas em fases é uma abordagem estruturada para criar software.**

1. Começa com a análise dos requisitos do sistema, na qual as necessidades dos usuários são identificadas e documentadas.
2. Em seguida, vem a fase de projeto, em que a arquitetura do sistema é planejada e as especificações técnicas são elaboradas.
3. Após o projeto, entra-se na fase de implementação, quando o código é escrito e o sistema começa a ganhar vida.
4. Finalmente, ocorre a fase de teste, na qual o software é submetido a uma série de testes para garantir sua funcionalidade e qualidade antes da implantação.

**O processo de desenvolvimento é particionado se sucedem,** o conhecimento sobre o sistema aumenta, diminuindo, consequentemente, o nível de abstração da realidade. Essa complexidade aumenta à medida que o tamanho do sistema cresce, requerendo um maior planejamento dos recursos a serem usados.

Traduzir a realidade em modelos no desenvolvimento de sistemas é essencial para assegurar que os requisitos e necessidades dos usuários sejam compreendidos e atendidos de maneira eficaz. Essa prática permite uma visualização clara do funcionamento do sistema, facilitando a identificação de potenciais problemas e a implementação de soluções adequadas.

Módulo 1.2

Entendimento e validação dos modelos com os usuários:

**Momento 1**

A equipe de desenvolvimento se reúne com os usuários e, usando técnicas de levantamento de dados, compreende a realidade e as necessidades dos usuários, visando implementá-las no sistema. Os dados levantados são registrados e usados na construção dos modelos. Esse momento acontece com maior intensidade na fase de requisitos, mas também está presente nas fases de análise e de projeto.

**Momento 2**

A equipe de desenvolvimento constrói os modelos que julga pertinentes para que se possa compreender e destacar os aspectos relevantes da realidade. Esse momento acontece nas fases de requisitos, análise e projeto.

**Momento 3**

A equipe de desenvolvimento se reúne com os usuários, apresentando e discutindo os modelos construídos, visando validá-los e responder à pergunta base: os modelos que construímos representam de fato a realidade dos usuários? Em caso positivo, prossegue-se no desenvolvimento; caso contrário, os modelos são ajustados e confirmados novamente com os usuários, até que estejam adequados. Esse momento acontece nas fases de requisitos, análise e projeto.

Outra finalidade dos modelos no desenvolvimento de sistemas é orientar membros da equipe em relação às suas tarefas no processo. Aqui todos os membros podem consultar em caso de dúvidas para aplicar na construção do projeto.

Módulo 2

A adoção de uma abordagem orientada a objetos promove a reutilização de código, a modularidade e a escalabilidade dos sistemas, contribuindo significativamente para a qualidade e a eficácia dos projetos de software.

Alan Kay, estabeleceu os princípios centrais da orientação a objetos:

* Qualquer coisa do mundo real é um objeto.
* Objetos realizam tarefas requisitando serviços a outros objetos.
* Os objetos similares são agrupados em classes e cada objeto pertence a uma classe.
* A classe determina o comportamento possível a um objeto.
* Classes são organizadas em hierarquias.

Classe

É abstração das características de um grupo de coisas do mundo real.

Objeto

É um elemento específico de uma classe ou uma instância de uma classe.

**Operação:** É o nome dado a cada ação (função) que o objeto sabe realizar. Mas um objeto não realiza nenhuma ação sem uma motivação, sem um estímulo.

**Mensagem:** Estímulo que chega a um objeto e solicita que ele realize uma de suas operações. Uma operação pode ser implementada através de pelo menos um método. Em outras palavras, cada objeto presta um serviço. Quando um objeto precisa de um serviço da responsabilidade de outro, ele precisa enviar uma mensagem a ele. Cada mensagem ativa uma das operações do objeto.

**Estado:** Chama-se estado do objeto o conjunto de valores de seus atributos em dado momento. Uma mensagem enviada a um objeto pode (ou não) alterar o seu estado, na medida em que pode alterar um ou mais valores de seus atributos.

Módulo 2.1

-Esse módulo fala muito sobre os pilares de Orientação a Obj, como Herança, Abstração, Polimorfismo e Encapsulamento.

Reusabilidade:

O uso de componentes já escritos pode ser a base para outros softwares (através da herança).

Extensibilidade:

Novos componentes podem ser desenvolvidos a partir de outros, já desenvolvidos, sem afetar o comportamento do componente de origem (mediante o princípio do polimorfismo) e permitindo que esse comportamento seja alterado, estendido para um novo contexto.

Módulo 2.2

O foco da atividade de análise é estudar e entender o funcionamento de um sistema sob pelo menos alguns pontos de vista:

* Da estrutura que sustenta o sistema (os dados).
* Dos procedimentos e processos intervenientes no sistema.
* Da dinâmica de funcionamento do fluxo de informações e dados.

**Levantamento de requisitos:**

Também chamado de investigação dos requisitos, consiste em entender a realidade, identificar a abrangência do sistema e capturar as necessidades dos usuários. Isso é feito usando técnicas específicas para o levantamento de requisitos.

**Requisitos funcionais**: Declaram as funcionalidades necessárias ao sistema.

**Requisitos não funcionais:** Apresentam algumas características associadas a uma, algumas ou todas as funcionalidades, e referem-se a aspectos de qualidade, confiabilidade, desempenho, portabilidade, segurança e usabilidade do sistema.

**Análise dos requisitos:**

Consiste no estudo e compreensão das necessidades, bem como do funcionamento e da dinâmica da organização. O objetivo é construir modelos que representem o sistema a ser desenvolvido em sua concepção lógica, sem considerar os recursos tecnológicos que o sustentarão.

**Análise do domínio (ou do negócio):**

Visa identificar ou modelar os objetos que serão usados na aplicação. Por exemplo, Pagamento é um objeto no contexto do sistema financeiro, usado para exemplificar os requisitos funcionais. Logo, Pagamento é um objeto do domínio, assim como Recebimento e Fatura.

**Análise da aplicação:**

Nessa etapa, identificamos objetos importantes para o sistema, mesmo que não sejam óbvios para os analistas de domínio. Esses objetos são específicos da aplicação, como uma interface de cadastramento de pagamentos em um sistema financeiro. O objetivo é apenas identificar esses objetos, sem detalhar sua implementação, que será feita na fase de projeto.

Análise do domínio

Objetos do domínio (relacionado ao problema).

X

Análise da aplicação

Objetos da aplicação (relacionado a aspectos computacionais de alto nível).

Módulo 2.3

**Projeto da arquitetura**: Ato de distribuir as classes de análise em subsistemas com seus componentes, bem como distribuir os componentes pelos recursos de hardware disponíveis. Entre os diagramas da UML, usamos aqui os diagramas de pacotes, componentes e implantação.

**Projeto detalhado**: Envolve atividades como modelagem das interações entre objetos, design da interface e do banco de dados, e considerações sobre aspectos computacionais avançados, como concorrência e distribuição de processamento e dados. Também inclui o design de algoritmos, se necessário, e a criação de diversos diagramas UML, como diagramas de interação, atividades e detalhamento dos diagramas de classes e de estados.

Módulo 2.4

Camadas em geral:

* Possuem alta coesão e baixo acoplamento, ou seja, concentram atividades afins (coesão) e são independentes umas das outras.
* Possuem propósito bem definido.
* A camada superior tem conhecimento apenas da imediatamente inferior, que fornece os serviços, por uma interface.

Vantagens:

* Torna o código mais organizado e legível.
* Permite o desenvolvimento, o teste e a manutenção das camadas isoladamente.
* Permite melhor reúso do código ou dos objetos.
* Pode substituir uma tecnologia que implemente uma camada, de forma simples, sem interferir nas demais. Por exemplo, para trocar o SGBD de SQL Server para PostgreSQL, basta alterar a camada de persistência. As demais permanecem como estavam.
* Disciplina as dependências entre as camadas.
* Mais adaptável a uma quantidade maior de usuários.

Desvantagens:

* Aumenta o número de classes do sistema.
* A adição de camadas torna o sistema mais complexo.
* Potencialmente, reduz o desempenho do software.

Modelo padrão adotado nos últimos tempos é o MVC.

Módulo 3

A linguagem unificada de modelagem (UML) oferece um conjunto de diagramas, sob diferentes visões, que permitem a modelagem de sistemas orientada a objetos, independentemente de tecnologia e de processos e metodologias de desenvolvimento de sistemas, cabendo seu uso em qualquer contexto de desenvolvimento orientado a objetos.

Foi projetada para ser **independente do método ou processo de desenvolvimento** utilizado.

Visualização:

A modelagem gráfica facilita a compreensão do sistema e das decisões tomadas durante as fases de análise e projeto, além de melhorar significativamente a comunicação entre os membros da equipe, permitindo que suas interpretações sejam feitas sem ambiguidades.

Especificação

Permite a construção de modelos precisos, não ambíguos e completos sob diferentes visões e atendendo às necessidades de modelagem das diferentes fases do processo de desenvolvimento de software, independentemente do processo ou modelo usado.

Construção

Os diagramas UML podem ser integrados às principais e mais populares linguagens de programação do mercado, como Java e C++. No entanto, para isso, é necessário utilizar uma ferramenta CASE (Computer-Aided Software Engineering) integrada que seja capaz de gerar código-fonte específico para essas linguagens a partir dos diagramas UML.

* É independente de tecnologia, adequando-se a todo método, metodologia ou processo de desenvolvimento.

* Não diz quais diagramas usar e nem em que ordem, pois a metodologia de desenvolvimento ditará essa ordem.

* Disponibiliza diagramas sob diferentes visões ou perspectivas.

Módulo 3.1

Os autores da UML entendem que um sistema deve ser visto sob cinco diferentes perspectivas ou visões, descritas a seguir.

Visão de casos de uso:

Oferece uma visão externa do sistema, do ponto de vista do usuário, descrevendo seu comportamento por meio de interações entre usuário e sistema. É uma perspectiva inicial em um projeto; a visão de caso de uso é estabelecida no estágio inicial do desenvolvimento e orienta todas as outras visões, ao capturar os requisitos funcionais que definem o sistema.

Visão de projeto (ou lógica):

Permite visualizar o sistema sob o ponto de vista de sua estrutura interna e seu comportamento, em resposta às funcionalidades externamente percebidas por seus usuários. Enfatiza os pacotes, as classes, as interfaces, os subsistemas (pacotes) e as colaborações.

Visão de implementação (ou de desenvolvimento):

Envolve o gerenciamento das versões do sistema, ou seja, suas implementações utilizáveis pelos usuários. Abrange os componentes, subsistemas e arquivos que compõem fisicamente o sistema.

Visão de implantação (ou física):

Enfatiza a distribuição física do sistema em suas partes (subsistemas e componentes) e as respectivas conexões entre elas. Enfatiza também a organização física dos computadores e as conexões entre eles (a rede de computadores).

Visão de processo:

Enfatiza aspectos físicos mais peculiares, como concorrência, sincronismo entre sistemas e desempenho (performance, confiabilidade, tolerância a falhas e outros aspectos) do sistema, considerando os processos e os processadores.

Módulo 3.2

**Cascata:**

Com ou sem retroalimentação, as fases ocorrem sequencialmente, iniciando a próxima quando a anterior é concluída. A desvantagem é que, sem retroalimentação, não há retorno à fase anterior. Por exemplo, se novos requisitos são identificados durante a fase de projeto, eles não podem ser considerados porque a fase de análise já congelou os requisitos identificados. Se a retroalimentação é permitida, esse problema pode ser minimizado, mas não totalmente resolvido. Um grande problema é que o usuário interage com a equipe de desenvolvimento apenas no início e no final do processo, quando o sistema é entregue.

**Iterativo:**

O sistema é dividido em subconjuntos de funcionalidades (com mínimo de dependência com os demais conjuntos), e as atividades de análise, projeto, implementação, teste e implantação são realizadas a cada subconjunto. Isso significa que a cada subconjunto haverá a implantação de uma parte do sistema, permitindo que ajustes das partes encerradas ocorram em paralelo com o novo subconjunto de funcionalidades que está sendo construído.

**Ágil:**

Compartilha um conjunto de valores e princípios definidos pelo Manifesto Ágil. O foco está na capacidade de adaptação contínua e no desenvolvimento de código. A modelagem existe, mas é menos abrangente, com ênfase na comunicação entre usuário e equipe de desenvolvimento. Os processos ágeis tendem a ser menos formais. Alguns dos métodos mais utilizados incluem Extreme Programming (XP), Scrum e FDD (Feature Driven Development).

RUP (Rational Unified Process):

É uma estrutura de processo, que vai usar casos de desenvolvimentos (processo a ser usado), a maioria deles iterativos. O RUP não se adapta a processo em cascata e, assim como os demais processos, é independente da UML.

Módulo 3.3

**Diagramas estruturais:**

São estruturas estáticas necessárias ao sistema, como os pacotes, as classes, os objetos, os componentes e a estrutura de nós (estruturas computacionais). São essenciais para o funcionamento do sistema. Também são chamadas de estruturas estáticas.

**Diagramas comportamentais:**

Evidenciam o comportamento (funcionamento) de parte de um sistema ou processo de negócio relacionado ao sistema, segundo determinada perspectiva. Estão relacionados às funcionalidades do sistema, aos estados de um objeto em seu ciclo de vida, às interações entre os objetos, entre outros aspectos. Também são chamados de diagramas dinâmicos.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

**Diagramas Estruturais**

**Diagrama de perfil:** Permite a definição de tipos padronizados, como estereótipos, restrições e valores rotulados. O foco é a adequação aos modelos UML para diferentes plataformas, por exemplo.

**Diagrama de classes:** Descreve, para cada classe, suas propriedades (atributos e métodos) e seus relacionamentos com as demais classes. Classe é a base estrutural dos sistemas orientados a objetos.

**Diagrama de estruturas compostas:** Possibilita a descrição de colaborações internas de classes, interfaces ou componentes, visando à especificação de uma funcionalidade.

**Diagrama de componentes:** Apresenta a estrutura e as conexões entre os componentes de um sistema.

**Diagrama de implantação:** Especifica o ambiente computacional sobre o qual o sistema vai operar, além de distribuir os artefatos (pacotes, classes e componentes) nos nós (elementos computacionais).

**Diagrama de objetos:** É um diagrama de classes instanciado, ou seja, mostra exemplos de instâncias de classes.

**Diagrama de pacotes:** Descreve estruturas hierárquicas que englobam outros elementos, como outros pacotes, casos de uso, classes. Usado para modularizar logicamente um sistema em suas partes relevantes (subsistemas).

**Diagramas Comportamentais**

**Diagrama de atividades:** Descreve o comportamento de procedimentos, processos de negócios e fluxos de trabalho, suportando processamento sequencial e paralelo.

**Diagrama de casos de uso:** Mostra como os usuários (atores) interagem com o sistema, do ponto de vista externo, evidenciando as funcionalidades com as quais interagem.

**Diagrama de estados:** Mostra como os eventos que afetam o sistema alteram o estado dos objetos ao longo de seus ciclos de vida.

**Diagrama de sequência:** Mostra como os objetos interagem, evidenciando a linha de tempo (a sequência das interações).

**Diagrama de comunicação:** Mostra como os objetos interagem, evidenciando as conexões e colaborações entre eles.

**Diagrama de visão geral de interação:** Um mix de diagrama de sequência e de atividades. Uma especialização do diagrama de atividades para interações.

**Diagrama de tempo:** Diagrama de interação, com foco nas restrições de temporização, por exemplo, para evidenciar as mudanças no estado de um objeto ao longo do tempo.

Módulo 3.4

Há um conjunto de 4 a 8 diagramas que são os mais usados: pacotes, **casos de uso, classes, sequência** (ou comunicação), **estados**, atividades, componentes e **implantação**. (Com ênfase nos em negrito)

Levantamento de requisitos:

* Esboço inicial do diagrama de pacotes para grandes sistemas, evidenciando a necessidade de particionamento.
* Esboço inicial do diagrama de casos de uso.
* Esboço inicial do diagrama conceitual de classes em alto nível (sem detalhes).

Análise de requisitos:

* Detalhamento e aprofundamento do diagrama de pacotes.
* Detalhamento e aprofundamento do diagrama de casos de uso.
* Detalhamento e aprofundamento do diagrama conceitual de classes.
* Diagrama de estados para os objetos mais complexos durante seu ciclo de vida.
* Diagrama de atividades para elucidar um processo ou fluxo de trabalho relevante ou ainda para elucidar um caso de uso mais complexo.

Os diagramas usados no levantamento de requisitos são, em geral, esboços para a própria pessoa ou para debater com colegas da equipe. Já na fase de análise de requisitos, alguns diagramas, em geral pacotes e casos de uso, são usados para conversas com usuários.

Diagramas UML na atividade de projeto:

* Adição de detalhes ao diagrama conceitual de classes, que passa a ser o **diagrama de classes** de projeto.

* **Diagrama de sequência ou diagrama de comunicação**, para cenários de uso mais relevantes, o que ajuda a identificar métodos das classes envolvidas na interação.

* Detalhamento dos **diagramas de estados** elaborados na atividade de análise, podendo modelar outros objetos percebidos.

* **Diagrama de componentes**, caso o software use algum já pronto ou a ser construído.

* **Diagrama de implantação**, evidenciando as unidades computacionais e alocando os componentes nelas.

* **Diagrama de atividades**, esclarecendo métodos de classes mais complexos ou que usem processamento paralelo.