

Conceitos básicos de modelagem de sistemas

A modelagem de sistemas orientados a objetos possibilita a representação dos requisitos e das soluções de análise e projeto de sistemas de qualquer porte e finalidade. A linguagem de modelagem unificada (UML) é a ferramenta utilizada para criar modelos, na forma de diagramas padronizados, que auxiliam a equipe de desenvolvimento a compreender o que o sistema deve fazer em um primeiro momento e como o sistema deve fazer em um momento posterior.

Prof. Marcelo Vasques de Oliveira Prof. Sylvio Jorge

1. Itens iniciais

Preparação

Antes de começar, instale um programa de modelagem UML em seu computador, como o Astah Free Student License. Use seu e-mail institucional para ativar a licença, preenchendo o formulário e aguardando a liberação da licença neste mesmo e-mail. Após receber a licença, siga as instruções para instalar o produto em seu computador. Você também pode encontrar outras ferramentas livres para modelagem de sistemas em UML fazendo buscas na internet.

Objetivos

- Reconhecer a importância dos modelos na exposição de requisitos e soluções sistêmicas.
- Distinguir os conceitos e pilares de análise e de projeto orientados a objetos.
- Descrever as visões, a síntese geral e os diagramas da UML.

Introdução

Vamos explorar os fundamentos da modelagem de sistemas para entender a realidade do negócio associada ao sistema e apresentar soluções que atendam às necessidades reais dos usuários. Começaremos examinando o conceito de modelo e mostrando como ele é aplicável no desenvolvimento de sistemas. É importante compreender que há diversos modelos, apresentados na forma de templates e diagramas, que devem ser aplicados em diferentes fases do processo de desenvolvimento do sistema.

Focaremos o desenvolvimento de sistemas orientado a objetos, compreendendo suas bases conceituais e seus pilares de sustentação. Dentro desse contexto, conheceremos a linguagem unificada de modelagem (UML, do inglês *unified modeling language*) e suas visões integradas de modelos, que oferecem diferentes perspectivas para abordar as diversas necessidades de modelagem, incluindo exposição de requisitos, análise de soluções e projeto do sistema em desenvolvimento. A UML é uma linguagem de modelagem independente de tecnologia, adaptável a diferentes processos e metodologias de desenvolvimento de sistemas orientados a objetos.

Para começar, assista ao vídeo!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

1. Modelos na exposição de requisitos e soluções sistêmicas

Utilizando modelos

O entendimento teórico sobre o conceito de modelos e sua aplicabilidade é fundamental em diversas disciplinas. 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. Ao compreender a natureza e a função dos modelos, os indivíduos podem empregar essas ferramentas de forma mais eficaz, contribuindo para o avanço do conhecimento em campos tão diversos quanto a ciência, a engenharia, a economia e a medicina.

Vamos entender neste vídeo a importância do uso de modelos, representados por meio de diagramas, no processo de desenvolvimento de sistemas computacionais. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O que são modelos e para que eles servem

Uma família decide adquirir um imóvel na planta para moradia. Eles vão a um lançamento imobiliário a convite de um corretor conhecido. Ao chegarem, encontram um terreno vazio e um stand de vendas. Imediatamente surge a dúvida: como escolher o bloco e o apartamento de forma a garantir privacidade, considerando a vizinhança?

O corretor inicia seu trabalho e leva a todos para conhecerem a maquete do empreendimento – que nada mais é do que a representação do empreendimento em bloco único – e a infraestrutura do parque aquático.

A maquete é simplesmente uma representação em miniatura do condomínio real, ou seja, um modelo do empreendimento real. O empreendimento real será construído à imagem e semelhança da maquete. Assim, podemos estabelecer a primeira finalidade de um modelo: antecipar a existência de uma realidade para avaliar sua estrutura e comportamento.

No exemplo a seguir, encontramos maquetes de um empreendimento imobiliário, mostrando a perspectiva externa, dando a visão clara da vizinhança, do posicionamento do imóvel, da piscina e área de lazer e da entrada.

Analisando a maquete e o posicionamento do empreendimento no terreno, os integrantes da família verificam o bloco e a posição do imóvel que mais lhes interessam, avaliando a vizinhança e respectivas distâncias entre eles.



Maquete de um empreendimento.

A família seleciona três unidades e pergunta sobre a disposição dos cômodos. O corretor, então, apresenta a planta baixa de cada unidade selecionada. Veja!



Planta de uma unidade de um empreendimento.

Em seguida, eles se sentam à mesa com o corretor para escolher a unidade no bloco selecionado. O corretor então apresenta a tabela de preços de cada unidade do bloco selecionado, informando a metragem, o valor da unidade e as condições de pagamento.

A planta ilustrativa da unidade é um segundo exemplo de modelo usado no mercado imobiliário, que possibilita ao comprador avaliar o posicionamento e a dimensão de cada cômodo.

A família decide-se pelo imóvel, fecha o negócio e recebe um pen drive contendo outras plantas da unidade: elétrica, hidráulica, dentre outras. Cada uma dessas plantas representa um modelo sob uma diferente perspectiva.

Outro exemplo de modelo muito comum atualmente são os **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. Os setores automobilístico e o de desenvolvimento de sistemas usam com eficácia protótipos como modelos.



Protótipo automotivo.

Um modelo é uma representação abstrata e simplificada da realidade, que pode variar conforme a perspectiva de observação. A principal finalidade de um modelo é antecipar a existência de uma realidade específica, permitindo avaliar sua estrutura e comportamento de maneira eficaz. Portanto, diferentes realidades podem exigir a criação de múltiplos modelos para atender às diversas necessidades de análise e compreensão.

Modelos se aplicam ao contexto de desenvolvimento de sistemas?

Na construção ou desenvolvimento de sistemas computacionais, assim como na construção imobiliária, há uma graduação da complexidade no processo de construção, que depende de alguns fatores, sendo o tamanho (do sistema ou do empreendimento) um deles.

Os modelos, 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.

Modelos são capazes de revelar as propriedades essenciais de um sistema, ajudando no processo de abstração (concentração nos pontos de interesse) e permitindo que foquem o que é relevante.

Entre os benefícios do uso de modelos no desenvolvimento de sistemas computacionais, além de prever o comportamento do sistema e gerenciar sua complexidade, destacam-se estes aqui. Veja!

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.

Agora, vamos conhecer mais a seguir sobre modelagem. Acompanhe!

Modelagem de sistemas

Assim como exemplificamos no mercado imobiliário, no contexto de desenvolvimento de sistemas computacionais podem ser usados diferentes modelos de um mesmo sistema, em que cada um apresenta uma perspectiva (uma visão).

Por exemplo, um sistema armazena e manipula dados por meio de funcionalidades e possui determinados controles. Podemos, então, considerar três diferentes perspectivas e construir modelos que representem cada uma delas, tais como:

Os dados

As funcionalidades

Os controles

Outra perspectiva seria a visão externa, a de um usuário, que enxerga as funcionalidades necessárias, mas desconhece o que ocorre internamente. Essa seria mais uma perspectiva e mais um modelo que ajudaria nessa compreensão.

Confira algumas formas de abordar os sistemas computacionais por meio de visões.

1 Externa

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

2

Comportamental

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

3

Estrutural

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

4

Interacional

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

A construção dos diferentes modelos para um sistema compreende o que denominamos **modelagem de sistemas**, em que:

- Os modelos são abstratos, deixando de lado os detalhes e concentrando-se nos aspectos de interesse que são relevantes. A esse processo chamamos de **abstração**.
- Cada modelo apresenta o sistema sob uma diferente visão ou perspectiva da realidade.
- Os modelos são descritos em notações gráficas, que denominamos diagramas.

Com base no que acabamos de estudar, observe!

“

Modelagem de sistema de software consiste na utilização de notações gráficas e textuais com o objetivo de construir modelos que representam as partes essenciais de um sistema, considerando-se várias perspectivas diferentes e complementares.

—
(Bezerra, 2015)

Como veremos mais adiante, a UML é uma linguagem unificada de modelagem que permite a construção de um conjunto de modelos, na forma de diagramas, sob diferentes visões ou perspectivas que, em conjunto, integram a solução de modelagem do sistema quando desenvolvido usando o paradigma orientado a objetos.

Atividade 1

Questão 1

Sobre os conceitos de modelos e modelagem, analise as assertivas a seguir.

I - Os modelos são representações da realidade.

II - Os modelos são construídos depois que os produtos são desenvolvidos.

III - A maquete de um empreendimento imobiliário mostra uma visão macro, do ponto de vista externo, de como será o empreendimento.

IV - Podemos ter apenas um modelo para cada produto.

V - Um protótipo de um novo avião pode ser considerado um modelo.

Com base em sua análise das assertivas, assinale a alternativa correta:

A Estão corretas apenas as assertivas I, II, III e V.

B Estão corretas apenas as assertivas I, III e V.

C Estão corretas apenas as assertivas I e V.

D Está correta apenas a assertiva III.

E Está correta apenas a assertiva V.



A alternativa B está correta.

Vamos analisar cada assertiva:

São verdadeiras as opções: I, pois os modelos representam a realidade, antes de sua existência; III, já que uma maquete mostra o exterior do empreendimento e não esclarece nada sobre o interior das unidades; V, porque um protótipo é uma representação da realidade.

Estão incorretas as opções: II, já que os modelos são construídos antes dos produtos (realidade) para que possamos compreender melhor a sua complexidade e seu comportamento futuro; IV, pois podemos ter mais de um modelo para cada produto.

Desenvolvimento de sistemas computacionais

Conhecer a teoria do desenvolvimento de sistemas computacionais é fundamental para a revolução digital contemporânea. Ao entender os princípios fundamentais da criação de software, os profissionais são capacitados a projetar, implementar e otimizar sistemas computacionais complexos e eficientes.

Nesse contexto, o processo de desenvolvimento de sistemas em fases é uma abordagem estruturada para criar software. Começa com a análise dos requisitos do sistema, na qual as necessidades dos usuários são identificadas e documentadas. Em seguida, vem a fase de projeto, em que a arquitetura do sistema é planejada e as especificações técnicas são elaboradas. Após o projeto, entra-se na fase de implementação, quando o código é escrito e o sistema começa a ganhar vida. 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.

Compreenderemos neste vídeo as fases comuns e mais relevantes do processo de desenvolvimento de sistemas computacionais. Acompanhe!

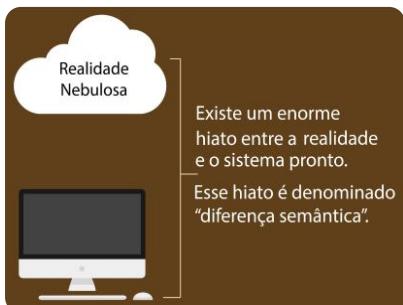


Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O processo de desenvolvimento de sistemas em fases

O desenvolvimento de sistemas computacionais é um processo que envolve pessoas e a necessidade de compreensão de uma realidade muitas vezes complexa e obscura, principalmente no início do desenvolvimento, quando o nível de abstração é alto e pouco se conhece da realidade. Podemos visualizar isso na imagem a seguir.



Conforme as fases em que o processo de desenvolvimento é partitionado 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. O principal recurso no desenvolvimento de um sistema são pessoas, profissionais capacitados.

Realidade X Sistema.

Primeiramente, precisamos entender as necessidades das pessoas que utilizarão o sistema e o que elas precisam que o sistema faça para desempenharem suas funções. Esse entendimento também requer que os profissionais responsáveis pela construção do sistema confirmem essa compreensão.

Mas como entender e confirmar a compreensão em uma linguagem ambígua como a nossa, tanto a falada como a escrita?

Se criarmos o sistema diretamente na linguagem de programação com base em uma compreensão suposta da realidade, certamente enfrentaremos problemas de interpretação inadequada. O sistema

provavelmente não atenderá às necessidades dos usuários e acabará sendo abandonado. Para representar a realidade de maneira adequada e entender o contexto do sistema a ser desenvolvido, precisamos traduzir a realidade em modelos.

Com base no que estudamos até agora, vamos aprofundar nosso entendimento sobre as fases mais importantes.

Fases comuns e mais relevantes do processo de desenvolvimento

Os modelos nos ajudam a entender a realidade e discutir essa compreensão, reduzindo a complexidade e o nível de abstração.



Atenção

Os modelos são representações simplificadas da realidade, focando os elementos de interesse em um dado momento. Eles permitem abstrair o que não é relevante e concentrar naquilo que é essencial para o desenvolvimento do sistema.

Desde a década de 1960, muitos processos, métodos e diversas técnicas de desenvolvimento de sistemas foram criados e postos em prática, visando à construção de sistemas computacionais robustos, eficientes e de fácil manutenção.

Para gerenciar melhor a complexidade, os processos e metodologias de desenvolvimento de sistemas geralmente são divididos em fases. Embora cada processo ou metodologia possa ter sua própria divisão, de maneira geral, podemos identificar as seguintes fases (com algumas exceções). Confira!

Identificação dos requisitos

São as necessidades que os usuários têm e que devem estar contidos nas funcionalidades e propriedades do sistema a ser construído.

Análise

Envolve compreender o que o sistema deve fazer para atender às necessidades de seus usuários.

Projeto

Envolve a adequação dos requisitos à forma como serão implementados, utilizando a tecnologia adequada. Define-se a arquitetura e os componentes do sistema, bem como toda a infraestrutura do ambiente computacional necessária para sua construção, incluindo redes de computadores, banco de dados, linguagem de programação e outros elementos.

Implementação

Refere-se à identificação dos programas necessários e sua codificação na linguagem de programação selecionada na fase de projeto, bem como o banco de dados que será usado.

Atividade 2

Questão 1

A modelagem de sistemas computacionais envolve a representação visual e conceitual de elementos, estrutura e processos de um sistema de software. Por que é tão importante traduzir a realidade em modelos durante as fases comuns e mais relevantes do processo de desenvolvimento de sistemas computacionais?

- A Para dificultar o nível de abstração da realidade.
- B Para facilitar a interpretação inadequada.
- C Para aumentar a complexidade do sistema.
- D Para confirmar a compreensão das necessidades dos usuários.
- E Para representar adequadamente a realidade e entender o contexto do sistema a ser construído.



A alternativa E está correta.

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.

Modelos como elementos de comunicação

Os modelos facilitam a compreensão e a representação da realidade, tanto para os membros da equipe de desenvolvimento quanto para os usuários finais. Eles servem como ferramentas eficazes para traduzir os requisitos e as necessidades dos usuários em uma linguagem comprehensível para os desenvolvedores, garantindo que o sistema atenda adequadamente às expectativas e às demandas. Além disso, os modelos auxiliam na validação e na verificação dos requisitos, contribuindo para o sucesso do projeto de desenvolvimento de software.

Compreenderemos neste vídeo a utilização de modelos como elementos de comunicação no processo de desenvolvimento de sistemas computacionais. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Entendimento e validação dos modelos com os usuários

A partir dos modelos, compreendemos e nos certificamos do correto entendimento da realidade dos usuários.

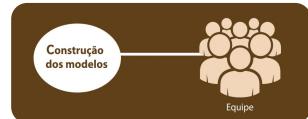
Momento 1

A equipe de desenvolvimento se reúne com os usuários e, usando técnicas de levantamento de dados, comprehende 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.



Os modelos são fundamentais para garantir o correto entendimento da realidade dos usuários. A equipe coleta dados, constrói os modelos com base nessas informações e, finalmente, valida-os com os usuários para assegurar que representam adequadamente a realidade.

Entendimento do sistema por membros da equipe de desenvolvimento

Outra finalidade dos modelos no desenvolvimento de sistemas é orientar membros da equipe em relação às suas tarefas no processo.



Exemplo

O programador deve construir os programas, mas não tem livre acesso aos usuários e nem precisa, pois os modelos servem como elementos de comunicação com todos da equipe. Os projetistas do software devem compreender a realidade dos requisitos para construir os modelos de projeto, por isso precisam ler os modelos das fases de requisitos e de análise. Os programadores consultam os modelos de seu interesse e conversam com os membros que fizeram o levantamento de dados e a modelagem para compreenderem melhor o contexto e desenvolverem com mais eficiência os programas necessários.

Outro exemplo é o projetista de interface, que precisa compreender o funcionamento de um determinado recurso para criar a interface adequada. Nesse caso, ele consulta o modelo que descreve a interação do

usuário com o sistema para a funcionalidade em questão. Esse processo ocorre em todas as fases do projeto, já que sempre haverá desenvolvedores recebendo tarefas e consultando os modelos relevantes.

A imagem a seguir ilustra a consulta de diferentes pessoas da equipe aos modelos desenvolvidos.



Atividade 3

Questão 1

No desenvolvimento de sistemas computacionais, os modelos são representações abstratas de elementos, processos ou comportamentos do sistema. Qual das opções indica uma das finalidades dos modelos no desenvolvimento de sistemas?

- A Aumentar a complexidade das tarefas da equipe.
- B Facilitar o acesso livre dos programadores aos usuários.
- C Servir como elementos de comunicação com todos os membros da equipe.
- D Substituir a interação direta entre os programadores e os usuários.
- E Limitar o acesso dos projetistas de interface aos modelos.



A alternativa C está correta.

Os modelos no desenvolvimento de sistemas atuam como ferramentas de comunicação, garantindo que todos os membros da equipe, desde programadores a designers e stakeholders, estejam alinhados quanto às expectativas e aos requisitos do sistema. Eles fornecem uma representação visual ou conceitual do sistema proposto, permitindo a discussão de ideias, a identificação de potenciais problemas e soluções antes da implementação, e facilitando o entendimento comum dos objetivos do projeto.

Orientação a objetos

Compreender os princípios fundamentais desse paradigma de desenvolvimento permite aos profissionais de tecnologia criarem sistemas modulares, flexíveis e de fácil manutenção. Além disso, 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. O conhecimento teórico nessa área é essencial para impulsionar a excelência na engenharia de software.

Entenda neste vídeo o paradigma orientado a objetos, que visualiza um sistema de software como uma coleção de agentes interconectados chamados objetos.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Paradigma orientado a objetos

Vamos analisar agora os conceitos, pilares, princípios e as orientações do paradigma orientado a objetos. Veremos também as visões, os objetivos e as entregas dos momentos de análise, projeto e implementação (em camadas), independentemente de processo ou metodologia de desenvolvimento de software.



Atenção

Ao longo do texto, utilizaremos a sigla OO para o termo orientação a objetos.

Com o aumento do tamanho do código e da complexidade dos programas, o paradigma estruturado, que antecedeu o paradigma orientado a objetos, começou a apresentar limitações nos sistemas sob o ponto de vista da dificuldade de manutenção e reúso de programas e rotinas padronizadas.

Vamos, inicialmente, definir o termo paradigma como a maneira de abordar um problema.

A orientação a objetos surge como solução para esses problemas, possibilitando uma maior organização, reutilização e extensibilidade de código por meio de propriedades como abstração, encapsulamento, herança e polimorfismo. Isso resulta em programas mais fáceis de serem escritos e mantidos.

O principal foco do paradigma orientado a objetos é permitir o desenvolvimento de sistemas de forma mais rápida e confiável.

Um dos pioneiros do paradigma orientado a objetos, Alan Kay, imaginou um sistema como um conjunto de agentes autônomos, os objetos, que interagem entre si. Ele estabeleceu os princípios centrais da orientação a objetos. Vejamos!

- 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.

A seguir, observe a definição sobre o paradigma da orientação a objetos.

“

O paradigma da orientação a objetos visualiza um sistema de software como uma coleção de agentes interconectados chamados objetos. Cada um deles é responsável por realizar tarefas específicas e, para cumprir com algumas das tarefas sob sua responsabilidade, um objeto pode ter que interagir com outros. É pela interação entre eles que uma tarefa computacional é executada. Um sistema de software orientado a objetos consiste, portanto, em objetos colaborando para realizar as funcionalidades desse sistema. É graças à cooperação entre os objetos que a computação do sistema se desenvolve.

—
(Bezerra, 2015)

Conceitos fundamentais da orientação a objetos

A orientação a objetos enfatiza a identificação, a representação e a organização dos objetos necessários ao funcionamento de um sistema. Tem por base os conceitos de objetos, classes, operação, mensagem e estado, e está calcada em quatro pilares fundamentais: abstração, encapsulamento, herança e polimorfismo, discutidos na sequência.

Objetos e classes

Um objeto pode referenciar qualquer coisa do mundo real, por exemplo, um aluno, uma disciplina, um curso, um professor, entre outros, considerando um sistema acadêmico como contexto. Ou seja, um objeto é qualquer coisa do mundo real, de interesse no contexto em estudo.

Quando analisamos os objetos pertinentes a um contexto, não estamos preocupados com um objeto específico, como o aluno José Carlos Aragão, e sim com todos os alunos envolvidos no estudo. Surge, então, o conceito de classe que, conceitualmente, reúne um conjunto de objetos com as mesmas propriedades. Ou seja, estamos interessados em todos os alunos e não apenas em José Carlos Aragão. Aplicando o princípio da abstração (que será detalhado posteriormente), uma classe reúne objetos com características ou propriedades semelhantes, que consistem em seus dados (atributos) e procedimentos (métodos) que implementam os serviços que essa classe oferece.



Programadores.

A classe Aluno agrupa “José Carlos Aragão” e os demais alunos envolvidos. Um objeto é um **elemento específico** de uma classe, ou ainda uma **instância** de uma classe.

As classes são, portanto, abstrações que definem uma estrutura que encapsula dados (chamados de atributos) e um conjunto de operações possíveis de serem usados, chamados de métodos. Por exemplo, a classe Aluno encapsula um conjunto de dados que identifique os alunos – matrícula, nome, endereço (rua, número, complemento, estado e CEP), CPF e identidade – e um conjunto de métodos: Incluir Aluno, Matricular Aluno, Cancelar Matrícula, entre outros.

Resumindo, podemos definir classe e objeto da seguinte forma. Observe!

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.

Confira a representação de uma classe em três compartimentos: o nome da classe (Aluno), seus atributos (Matrícula... Identidade) e métodos (Incluir Aluno... Cancelar Matrícula).



Classe Aluno

Agora, veja a representação de dois objetos da classe Aluno.

Aluno 1: Aluno

- Matricula : int = 8623871-1
- Nome : string = "José Carlos Aragão"
- rua : string = "Rua da Passagem"
- Numero : int = 160
- Complemento : string = "ap 703"
- Cep : string = "22776-056"
- CPF : string = "949.567.9990-21"
- Identidade : string = "07376534-3"

Objeto – Aluno 1 da classe Aluno.

Aluno 2: Aluno

- Matricula : int = 8623833-4
- Nome : string = "Marcelo Vasques de Oliveira"
- rua : string = "Rua dos Jacarandas"
- Numero : int = 1000
- Complemento : string = "ap 302 bloco 1"
- Cep : string = "22776-050"
- CPF : string = "949.455.9990-21"
- Identidade : string = "073093934-3"

Objeto – Aluno 2 da classe Aluno.

Operação, mensagem e estado

Um sistema orientado a objetos consiste na cooperação entre seus objetos. Cada um tem uma responsabilidade no sistema, correspondendo à parte das funcionalidades que lhes são atribuídas. Em outras palavras, uma tarefa computacional é realizada pela interação entre seus objetos, cada um executa parte da tarefa. Acompanhe!

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.

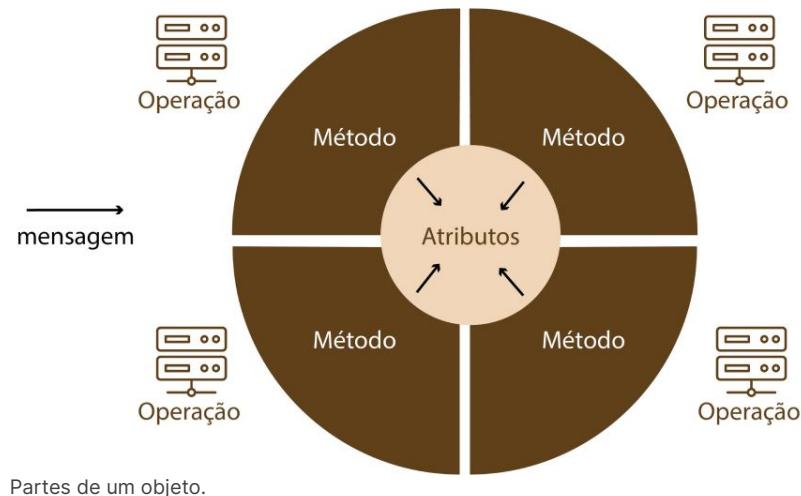
Um sistema orientado a objetos funciona pela cooperação entre objetos, cada um responsável por parte das funcionalidades. Objetos realizam ações chamadas operações em resposta a estímulos chamados mensagens. O estado de um objeto é determinado pelos valores de seus atributos, que podem ser alterados pelas mensagens recebidas.

Objeto

Imagine, por exemplo, que o objeto Notas_Aluno contenha as notas do aluno em determinada disciplina. Em dado momento, é recebida uma mensagem informando uma nova nota a ser armazenada. O estado desse objeto foi alterado, pois um de seus atributos recebeu a nova nota.

Agora, suponha que determinado objeto enviou uma mensagem a Notas_Aluno solicitando que seja exibida a média atual; nesse caso, não houve alteração de estado, pois as notas foram consultadas, e a média foi calculada (não fica armazenada) e exibida.

Observe a seguir como se comporta as partes de um objeto ao receber a mensagem.



Atividade 1

Questão 1

Sobre os conceitos e pilares do paradigma orientado a objeto, analise as assertivas a seguir.

I - Se uma disciplina é uma classe, "modelagem de sistemas com UML" é um objeto dessa classe.

II - Em um sistema orientado a objeto, quando um objeto precisa de um serviço prestado por outro, ele deve enviar uma mensagem a esse objeto.

III - Como consequência do princípio do encapsulamento, temos que os atributos de um objeto devem estar acessíveis diretamente por todos os demais objetos.

IV - É possível que apliquemos o polimorfismo sem antes aplicar o mecanismo da herança.

V - A orientação a objetos visa também facilitar o reúso de software.

Com base em sua análise das assertivas, assinale a alternativa correta:

A Estão corretas apenas as assertivas I, III e V.

B Estão corretas apenas as assertivas I, II e V.

C Estão corretas apenas as assertivas I e V.

D Estão corretas apenas as assertivas I, II, III e V.

E Estão corretas apenas as assertivas II e V.



A alternativa B está correta.

Estão corretas as afirmativas: I, pois objeto é uma instância específica de uma classe. Classe é o molde de objetos afins (mesmos atributos e métodos); II, pois esse é um dos conceitos fundamentais de orientação a objeto; V, já que o aumento do reúso é um dos propósitos da existência do paradigma OO.

Estão incorretas as afirmativas: III, porque o encapsulamento diz que os atributos devem ser protegidos de acesso direto por demais objetos; IV, visto que somente pode haver polimorfismo se antes houve a aplicação do princípio da herança.

Principais características do paradigma orientado a objeto

Compreender conceitos como encapsulamento, herança, polimorfismo e abstração permite aos desenvolvedores criarem soluções mais modulares, flexíveis e fáceis de manter. Essas características capacitam os profissionais de tecnologia a desenvolverem sistemas que melhor representam e modelam objetos do mundo real, resultando em um código mais legível, reutilizável e escalável. Em resumo, o

conhecimento teórico dessas características é essencial para impulsionar a qualidade e a eficácia no desenvolvimento de software orientado a objetos.

Entenda neste vídeo as principais características do paradigma orientado a objeto.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Os pilares da orientação a objetos

Entre as principais características do paradigma orientado a objeto (OO), destacamos:

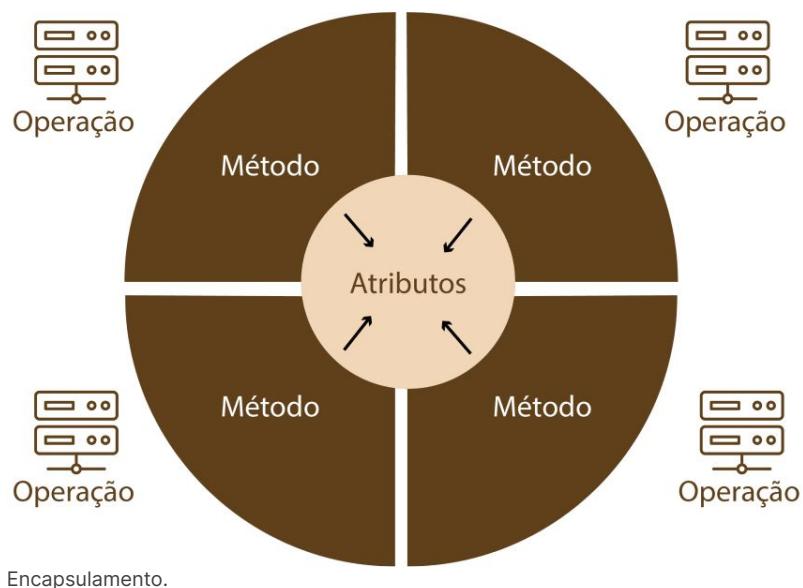
Abstração

É um processo mental que permeia toda a orientação a objetos, como um princípio básico que serve de base aos demais princípios. A abstração permite que, ao estudar algo, ponhamos nossa concentração nos aspectos relevantes e deixemos de lado os menos importantes. Permite, portanto, gerenciar a complexidade de um objeto para que possamos nos ater às suas propriedades essenciais. E os aspectos essenciais de um objeto dependem, claro, do contexto no qual os analisamos, podendo variar. Ou seja, uma propriedade de um objeto pode ser relevante em um contexto e não ser em outro.

Encapsulamento

O objeto esconde (encapsula) seus dados (atributos) do acesso indevido por outros objetos e somente permite que eles sejam acessados por operações implementadas pelos seus próprios métodos (funcionalidades que implementam o comportamento do objeto). Com isso, o encapsulamento protege os dados do objeto do uso arbitrário ou não intencional.

O encapsulamento é uma técnica para minimizar a interdependência entre as classes, pois apenas os métodos da respectiva classe podem alterar seus dados (atributos), facilitando a identificação de erros e a alteração dos programas. Em outras palavras, garante que apenas os métodos da própria classe possam alterar o estado de um objeto, como podemos ver na imagem.



Herança

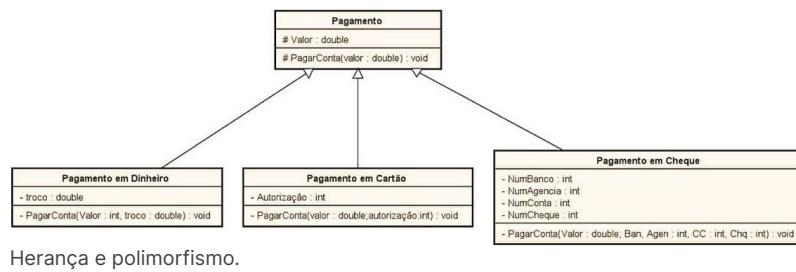
Mecanismo para derivar novas classes a partir da definição de classes existentes, com base em um processo de refinamento. Uma classe derivada ou descendente herda os dados (atributos) e o comportamento (métodos) da classe base ou ancestral ou ascendente. A implementação da herança garante a reutilização de código, que, além de economizar tempo e dinheiro, propicia mais segurança ao processo de desenvolvimento, visto que as funcionalidades da classe base podem ter sido usadas e testadas.

Polimorfismo

A palavra deriva do grego e significa “muitas formas”. A partir do momento em que uma classe herda atributos e métodos de uma (herança simples) ou mais (herança múltipla) classes base, ela tem o poder de alterar o comportamento de cada um desses procedimentos (métodos). Isso amplia o poder do reaproveitamento de código promovido pela herança, permitindo que se aproveite alguns métodos e se altere (redefina) outros. Dessa forma, um método com mesmo nome em classes distintas pode ter diferentes comportamentos.

Exemplificando herança e polimorfismo

Observe o exemplo da imagem a seguir, na qual identificamos uma herança: Pagamento em Dinheiro, Pagamento em CC (cartão de crédito) e Pagamento em Cheque herdam da classe Pagamento, o atributo Valor e o método Pagar.



Herança e polimorfismo.

Observe que, em cada classe filha (Pagamento em Dinheiro, Pagamento em Cartão e Pagamento em Cheque), o método PagarConta é escrito de forma diferente, com distintos parâmetros e códigos internos, conforme exige a respectiva forma de pagamento. Essa possibilidade ocorre pelo princípio do polimorfismo.

Orientação a objetos como elemento de reusabilidade e extensibilidade

A orientação a objetos minimiza a gestão da complexidade na medida em que permite uma organização mais adequada de seus componentes, além de seu reaproveitamento. Um dos principais motivos para a baixa produtividade na construção de sistemas computacionais é a dificuldade de reutilização de código.



Atenção

As hierarquias de classes (herança) são estruturas que permitem o seu reaproveitamento entre aplicações que, se bem projetadas, podem ser reutilizadas em vários sistemas, otimizando tempo e dinheiro.

Além disso, tais estruturas podem ser estendidas (usando o princípio do polimorfismo) sem corromper o que já existe. Assim, podemos concluir que a orientação a objetos traz em si alguns benefícios. Veja!

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.

Atividade 2

Questão 1

O paradigma orientado a objetos é um modelo de programação que organiza o software em objetos, que são instâncias de classes. Qual das seguintes características do paradigma orientado a objetos permite a reutilização de código entre aplicações?

A Encapsulamento.

B Abstração.

C Polimorfismo.

D Modularidade.

E Herança.



A alternativa E está correta.

A herança é um princípio fundamental do paradigma orientado a objetos que permite a reutilização de código. Por meio da herança, uma nova classe pode herdar atributos e métodos de uma classe existente, facilitando a reutilização e a extensibilidade do código sem a necessidade de reescrevê-lo. Esse mecanismo não só economiza tempo e recursos no desenvolvimento de software, mas também promove a consistência e a confiabilidade ao permitir que novas funcionalidades sejam construídas sobre bases já testadas e comprovadas.

Análise de sistemas orientada a objetos

É fundamental no desenvolvimento de software moderno, oferecendo uma abordagem coerente e eficiente para capturar os requisitos do sistema e modelar soluções complexas. Ao entender seus conceitos e práticas, desenvolvedores e analistas podem criar sistemas mais robustos, flexíveis e reutilizáveis, que se alinham melhor às necessidades dos usuários finais. Esse conhecimento teórico não apenas melhora a qualidade do design do sistema, mas também facilita a comunicação dentro das equipes de desenvolvimento, estabelecendo uma linguagem comum que apoia uma implementação eficaz e uma manutenção simplificada ao longo do ciclo de vida do software.

Entenda neste vídeo as etapas de levantamento e análise de requisitos na análise de sistemas orientada a objetos.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Análise de sistemas

Antes de entrarmos no universo da análise sob a perspectiva da orientação a objetos, vamos fazer breves comentários sobre a atividade de análise no contexto do desenvolvimento de software. De maneira simples, a atividade de análise busca identificar o que os usuários e outros interessados (stakeholders) necessitam que o sistema realize. A análise de sistemas envolve investigar os problemas e requisitos (necessidades dos usuários) de um contexto específico, com o objetivo de construir um sistema automatizado.

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.

Além desses, outros aspectos podem ser adicionados, dependendo da especificidade do sistema em construção.

A atividade de análise, por ser muito abrangente, costuma ser dividida da seguinte forma. Acompanhe!

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.

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.

A atividade de análise não considera nenhum recurso tecnológico que possa ser utilizado pelo sistema em construção. A ideia é construir uma estratégia de solução sem considerar como (com que tecnologia) essa estratégia será construída.



Atenção

A preocupação da atividade de análise é identificar O QUE o sistema deve fazer para atender às necessidades de seus usuários.

A finalidade da atividade de análise é construir a melhor solução, que possa ser implementada em qualquer tecnologia (plataforma operacional, linguagem de programação e banco de dados), de acordo com as disponíveis no mercado, naquele momento.

Na orientação a objetos, o foco está na identificação, compreensão e definição dos objetos de software e em como eles irão colaborar para atender satisfatoriamente aos requisitos dos usuários.

Levantamento de requisitos

Requisito pode ser definido como uma condição ou capacidade que deve ser alcançada por um sistema para satisfazer uma necessidade.

Nessa fase, o objetivo é dialogar com as pessoas ligadas ao sistema (patrocinadores, gestores, usuários atuais e futuros, entre outros) para compreender suas necessidades e expectativas individuais. O foco principal está na compreensão do problema.



Atenção

As necessidades e preferências dos usuários são tecnicamente denominadas requisitos. Os desenvolvedores e outras partes interessadas se reúnem para identificar os requisitos do sistema, levando em conta o contexto específico e o domínio do problema ao qual o sistema se aplica.

Para que os desenvolvedores possam identificar as necessidades dos usuários do sistema, é usado um conjunto de técnicas de levantamento de dados, desde as tradicionais entrevistas, reuniões, observação do ambiente do usuário e questionários até técnicas mais sofisticadas como brainstorm.

O produto gerado por essa fase é o **documento de requisitos**, que contém todos os requisitos necessários ao sistema. Confira sua classificação!

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.

Para exemplificar melhor, considere um sistema financeiro que precise das seguintes funcionalidades:

- Cadastramento dos pagamentos
- Quitação dos pagamentos
- Cadastramento das receitas
- Quitação das receitas
- Emissão das faturas

Cada uma das cinco funcionalidades anteriores representa um requisito funcional do sistema. Imagine, então, que o sistema precise ser *touch screen*. Tal característica da funcionalidade **Emissão das faturas** é um requisito não funcional de usabilidade (relacionada com uma interface de qualidade). Outro requisito não funcional de segurança seria a necessidade de um backup diário da base de dados.

A correta identificação e seu registro no documento de requisitos são cruciais para a precisão e a qualidade do processo de desenvolvimento do sistema. Um requisito faltante ou outro mal definido pode ser fatal para seu sucesso. Esse documento seguirá como base da comunicação entre os desenvolvedores e os usuários, devendo ser validado por estes, uma vez que servirá de norte para as atividades subsequentes do desenvolvimento do sistema. É, portanto, fundamental a participação ativa e efetiva dos usuários do sistema na fase de levantamento de requisitos.



Programadora em conexão com usuário do sistema.

No documento de requisitos estará definido, também, o escopo do sistema. Portanto, o principal objetivo da fase de levantamento de requisitos é compreender profundamente o sistema antes de iniciar sua construção.

Análise de requisitos

Nessa fase, os requisitos funcionais e não funcionais são traduzidos para os modelos que a equipe planeja utilizar.

A principal atividade é desdobrar os requisitos funcionais em funcionalidades do sistema, uma vez que não necessariamente há uma relação de 1 para 1, ou seja, um requisito funcional pode demandar mais de uma funcionalidade e uma funcionalidade pode agregar mais de um requisito funcional.

A análise de requisitos tem, minimamente, duas perspectivas ou visões. Confira!

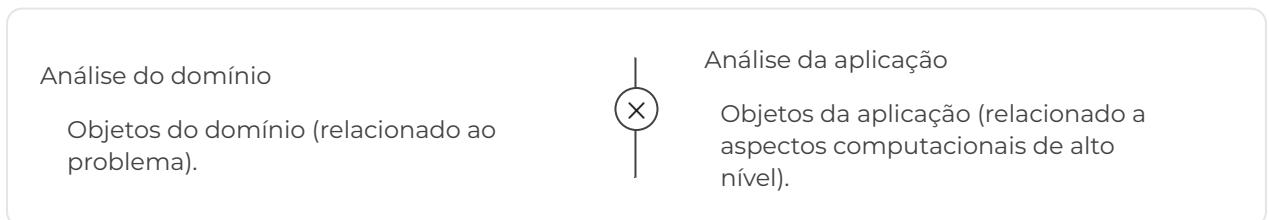
Análise do domínio (ou do negócio)

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.

Entenda melhor com o esquema a seguir.



Os principais diagramas UML utilizados nas fases de análise incluem os diagramas de casos de uso e de classes. Em alguns casos, também são úteis os diagramas de interação e de estados.



Curiosidade

Análise pode ser traduzida em "faça a coisa certa".

Atividade 3

Questão 1

Na fase de análise de um desenvolvimento de sistema orientado a objetos, são identificados e modelados os objetos do mundo real relevantes para o sistema, suas características e como eles interagem. Qual das seguintes afirmações descreve uma atividade da fase de análise em um desenvolvimento de sistema orientado a objetos?

- A Estabelecer os requisitos funcionais e não funcionais do sistema, identificando o que o sistema deve fazer.
- B Codificar e implementar as funcionalidades do sistema conforme especificado nos requisitos.
- C Testar o sistema para garantir que todos os requisitos identificados foram adequadamente implementados.
- D Decidir como o sistema funcionará para atender aos requisitos, definindo a arquitetura do sistema e os recursos computacionais necessários.
- E Treinar os usuários finais sobre como utilizar o sistema uma vez que esteja implementado e em funcionamento.



A alternativa A está correta.

Na análise de sistemas, os requisitos podem ser classificados em diferentes tipos. Os requisitos funcionais descrevem as funcionalidades específicas que o sistema deve executar, como processamento de dados ou interação com o usuário. Por outro lado, os requisitos não funcionais definem atributos do sistema, como desempenho, segurança e usabilidade. Compreender e documentar adequadamente esses diferentes tipos de requisitos é essencial para garantir que o sistema atenda às expectativas e necessidades dos usuários finais.

Projeto (desenho) de sistemas orientado a objetos

É essencial no desenvolvimento de software, pois estabelece a estrutura sobre a qual os sistemas são construídos. Com um sólido conhecimento teórico nessa área, é possível projetar sistemas mais eficientes, mantendo a coesão e minimizando o acoplamento entre os objetos. Isso não apenas facilita a implementação,

como também a manutenção e a expansão futuras do sistema. Dominar os princípios do design orientado a objetos promove a criação de software de alta qualidade, otimizando recursos e maximizando a adaptabilidade às mudanças de requisitos, fundamentais em um ambiente tecnológico que evolui rapidamente.

Entenda neste vídeo a etapa de projeto (desenho) de sistemas orientado a objetos.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

A atividade de projeto denota uma solução, voltada a atender aos requisitos identificados na fase de análise, considerando os recursos tecnológicos necessários para implementar os objetos do domínio e os objetos da aplicação.

O objetivo é decidir **como o sistema funcionará** para atender aos requisitos. Em geral, o projeto enfoca a definição da arquitetura do sistema, determinando suas partes e a relação entre elas, o padrão de interface gráfica, o ambiente computacional (sistema operacional e computacional), a linguagem de programação e gerenciamento do banco de dados.

O projeto, consequentemente, é uma extensão da análise e resulta em uma descrição dos recursos computacionais e de como o sistema deve operar no cotidiano. Em algumas situações, podem existir restrições tecnológicas, como a necessidade de desenvolver na linguagem X devido à predominância dos sistemas da organização nessa linguagem (se ainda estiver em uso no mercado), ou usar o sistema gerenciador de banco de dados Y, devido à base corporativa da organização estar nele. Veja como a fase de projeto pode ser dividida!

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.

Portanto, podemos concluir que o projeto pode ser traduzido em “faça certo a coisa”.

Atividade 4

Questão 1

Sobre as atividades de análise e projeto no contexto da orientação a objetos, examine as assertivas a seguir.

I - Na fase de análise de requisitos é feita a transposição dos registros dos requisitos funcionais e não funcionais para os respectivos modelos a serem usados.

II - Em um sistema acadêmico escolar, um requisito funcional é "o sistema deve emitir o boletim bimestral do aluno".

III - Um requisito não funcional de um sistema de autoatendimento em estacionamentos é "o sistema deve ser touch screen".

IV - No projeto detalhado, é feita a distribuição das classes de análise em subsistemas com seus componentes.

Com base nas assertivas, assinale a alternativa correta:

A Estão corretas apenas as assertivas I e III.

B Estão corretas apenas as assertivas I, II e III.

C Estão corretas apenas as assertivas I e II.

D Está correta apenas a assertiva I.

E Estão corretas apenas as assertivas II e IV.



A alternativa B está correta.

Estão corretas as afirmativas: I, já que antes da análise dos requisitos temos a listagem de todos, porém, alguns têm relação e podem ser agrupados em uma funcionalidade, por exemplo; II, pois essa é uma funcionalidade do sistema; III, já que essa é uma necessidade que vai guiar toda a interface desse sistema. A afirmativa IV está incorreta, pois é no projeto de arquitetura que essa distribuição das classes é realizada.

Desenvolvimento de sistemas em camadas

Compreender o desenvolvimento em camadas é essencial para estudantes de engenharia de software, fornecendo uma compreensão sobre a organização modular dos sistemas de software. Isso facilita a manutenção, escalabilidade e colaboração entre equipes. Além disso, ajuda na compreensão de padrões de projeto e arquiteturas modernas, preparando você para desafios no desenvolvimento de sistemas robustos.

Compreenda neste vídeo o desenvolvimento em camadas, que ajuda a separar o código em partes menores, tornando-o mais simples de gerenciar e manter.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vamos começar discutindo a arquitetura de projeto de software em camadas de maneira geral, sem nos prender a um modelo específico. O objetivo principal é dividir o código em camadas para simplificar sua manutenção, reduzindo sua complexidade. As camadas ajudam a separar responsabilidades e gerenciar dependências.

No início da computação, os sistemas eram **monolíticos**, ou seja, todo o código ficava confinado em uma única camada, manutenção no código, havia onde misturavam-se comandos de processamento, de construção e manipulação de interface, bem como de acesso e persistência de dados em SGBD (Sistemas de Gestão de Base de Dados). Tudo junto e misturado!

Quando era preciso fazer manutenção no código, havia dificuldade no entendimento e na separação do que de fato precisava ser alterado, sem contar a interferência em uma parte do código quando se alterava outra, em princípio sem relação entre si.



Programador analisando dados.

À medida que os sistemas cresceram e se tornaram complexos, a manutenção ficou mais difícil e a divisão em camadas foi uma das soluções encontradas para o projeto de arquitetura de um software.

Inicialmente, a rede cliente-servidor dividiu o software em duas camadas: a camada de código que roda no cliente (camada de interface com usuário) e a camada servidor (camadas de lógica do negócio e persistência dos dados). Posteriormente, com o advento da web, separou-se em três e depois em quatro camadas. Atualmente, pode-se criar tantas camadas quantas sejam necessárias, em função do tipo de aplicação.

Com base no que acabamos de estudar, podemos concluir que as 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.

No caso da orientação a objetos, as classes são organizadas em módulos maiores, as chamadas camadas. Uma camada somente pode usar serviço (de outras classes) da camada imediatamente inferior.

Aqui estão as vantagens e desvantagens do desenvolvimento de software em camadas. Confira!

Vantagens

Confira!

- Torna o código mais organizado e legível.
- Permite o desenvolvimento, o teste e a manutenção das camadas isoladamente.
- Permite melhor reuso 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

Confira!

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

Um modelo comumente utilizado nos últimos anos é o de três camadas, veja!

Apresentação

Compreende as classes do sistema que permitem a interação com o usuário, as chamadas classes de fronteira.

Negócio

Compreende as classes responsáveis pelos serviços e pelas regras do negócio, ou seja, reúne as classes de controle e negócio.

Dados

Responsável pelo armazenamento e pela recuperação dos dados persistentes do sistema, ou seja, as classes de persistência de dados.

Atividade 5

Questão 1

No desenvolvimento de sistemas de software em camadas, a aplicação é dividida em camadas distintas, cada uma responsável por uma função específica. Qual das seguintes afirmações melhor descreve uma característica importante do desenvolvimento de sistemas em camadas?

A Torna o código mais desorganizado e fácil de entender.

B Permite o desenvolvimento, o teste, mas dificulta a manutenção das camadas isoladamente.

C Diminui a flexibilidade do sistema para substituir tecnologias.

D Facilita o reúso do código ou dos objetos.

E Aumenta a complexidade do sistema e reduz o desempenho do software.



A alternativa D está correta.

O desenvolvimento em camadas torna mais fácil o processo de reúso do código ou dos objetos, pois permite que cada camada seja desenvolvida, testada e mantida independentemente das outras. Isso possibilita uma melhor organização do código e uma maior facilidade na implementação de mudanças ou substituição de tecnologias em determinada camada, sem afetar as demais partes do sistema.

O que é UML, afinal?

É uma linguagem visual padronizada usada para modelar sistemas de software, fornecendo uma representação gráfica e detalhada das estruturas, dos comportamentos e das interações de um sistema. Dominar os conceitos teóricos da UML capacita você a comunicar efetivamente suas ideias de design, colaborar de forma mais eficaz com suas equipes de desenvolvimento, entender modelos existentes e interpretar documentação técnica. Além disso, o conhecimento da UML é altamente valorizado na indústria de TI, oferecendo uma vantagem competitiva no mercado de trabalho.

Neste vídeo, vamos explorar a UML como uma linguagem padronizada que disponibiliza uma variedade de diagramas, cada um oferecendo perspectivas distintas, para a modelagem de sistemas orientados a objetos. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

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.

No final da década de 1990, as linguagens de programação orientadas a objeto já eram uma realidade, e cada profissional que desenvolvia atividade de análise e projeto de sistemas criava seus próprios modelos, baseados em suas necessidades e realidades. Ou seja, não havia consenso no mercado sobre os modelos a serem usados para modelagem de sistemas desenvolvidos sob a tecnologia de orientação a objetos.

Três competentes profissionais despontavam com seus modelos naquele momento:

- Ivar Jacobson, idealizador do método OOSE (Object-Oriented Software Engineering).
- James Rumbaugh, criador do método OMT (Object Modeling Technique).
- Grady Booch, criador do método que leva seu nome.

A UML foi adotada pela OMG (*Object Management Group*) em 1997, consolidando-se a partir da fusão dos três métodos mencionados, tornando-se uma linguagem de modelagem padrão para sistemas desenvolvidos sob o paradigma orientado a objetos.

“

UML tornou-se o padrão para modelagem gráfica, não apenas para objetos e, de fato, faz sentido essa afirmativa, pois a UML pode ser a linguagem de modelagem para diagramar sistemas concorrentes e distribuídos.

(Fowler, 2005.)

Desde sua versão inicial, a UML sofreu mudanças substanciais e atualmente encontra-se em sua versão 2.5.1, de dezembro de 2017.

A UML é uma linguagem padrão para construção de projetos de sistemas, voltada para a visualização, a especificação, a construção e a documentação de artefatos de um sistema. Foi projetada para ser **independente do método ou processo de desenvolvimento** utilizado.

A UML é uma linguagem de modelagem, não é um método de desenvolvimento nem tampouco uma metodologia ou um processo de desenvolvimento de sistemas, uma vez que não determina a ordem e nem como os diagramas devem ser usados. Simplesmente disponibiliza os diagramas, sob as várias visões necessárias ao desenvolvimento de sistemas, e cada empresa (ou equipe de desenvolvimento) a utiliza da forma como lhe convenha, ou seja, adequando a UML ao seu processo ou metodologia de desenvolvimento de sistemas. A UML é uma linguagem que tem o seguinte objetivo. Vejamos!

1

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.

2

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.

3

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.

A UML é uma linguagem de modelagem padronizada em que podemos observar o seguinte:

- 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.

Para Fowler (2005), a UML se tornou não somente a notação gráfica dominante no mundo orientado a objetos, como também uma técnica popular nos círculos não orientado a objetos.

Atividade 1

Questão 1

A linguagem unificada de modelagem (UML) é uma linguagem visual padronizada para representar modelos de sistemas de software. Ela oferece uma variedade de diagramas, como diagramas de classes, de sequência e de atividades. Qual é o principal objetivo da UML?

- A Padronizar as linguagens de programação orientadas a objeto.
- B Determinar a ordem dos diagramas a serem utilizados no desenvolvimento de sistemas.
- C Especificar os métodos de desenvolvimento de sistemas.
- D Disponibilizar diagramas sob diferentes visões ou perspectivas.
- E Ser independente de tecnologia, adequando-se a todo método, metodologia ou processo de desenvolvimento.



A alternativa E está correta.

A UML é uma linguagem de modelagem padronizada que não determina a ordem dos diagramas a serem utilizados nem os métodos de desenvolvimento de sistemas. Ela tem como principal objetivo disponibilizar diagramas sob diferentes visões ou perspectivas e ser independente de tecnologia, adaptando-se a qualquer método, metodologia ou processo de desenvolvimento.

Visões da UML

A UML oferece uma variedade de diagramas que representam perspectivas específicas de um sistema, como estrutura, comportamento e interação. Compreender essas visões permite que você analise e comunique efetivamente os requisitos, a arquitetura e o design de um sistema, promovendo uma modelagem mais precisa e ampla. Além disso, o conhecimento teórico sobre as visões da UML permite a colaboração de forma mais eficiente em equipes de desenvolvimento e a tomada de decisões fundamentadas durante todo o ciclo de vida do software.

Assista ao vídeo e entenda as visões da UML, um sistema visto sob diferentes perspectivas.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Assim como vimos nos exemplos do empreendimento imobiliário, as plantas das unidades, a planta elétrica, a planta hidráulica, entre outras, cada modelo tinha uma perspectiva de compreensão da mesma realidade (unidades residenciais em um empreendimento). No mundo de sistemas computacionais, o mesmo acontece em relação à modelagem de sistemas com UML. 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.

A UML implementa diagramas que atuam nas cinco visões descritas, permitindo ampla modelagem sobre todos os aspectos (visões) relevantes do sistema.

Nem todas as perspectivas ou visões são úteis em todo desenvolvimento, pois dependem das características, tipo e complexidade do sistema. A imagem a seguir mostra a ideia central da visão de casos de uso, influenciando todas as demais.



Visões da UML.

Atividade 2

Questão 1

Associe as duas colunas relacionando as visões da UML (coluna A) com sua respectiva finalidade ou característica.

Coluna A – visão UML	Coluna B – finalidade ou característica
1) Casos de uso	a) Enfatiza os pacotes, as classes, as interfaces, os subsistemas (pacotes) e as colaborações
2) Lógica ou projeto	b) Permite olhar o sistema sob o ponto de vista externo, descrevendo seu comportamento por interações usuário-sistema
3) Processo	c) Enfatiza a distribuição física do sistema em suas partes e as respectivas conexões entre elas
4) Implementação	d) Enfatiza aspectos físicos mais peculiares, como concorrência, sincronismo entre sistemas e desempenho
5) Implantação	e) Compreendem os componentes, os subsistemas e os arquivos que compõem fisicamente o sistema

Tabela: coluna A (visão UML) e coluna B (finalidade ou característica).
Alberto Tavares da Silva.

Com base nas assertivas, assinale a alternativa que associa corretamente as duas colunas:

A 1-a, 2-b, 3-d, 4-e, 5-c

B 1-b, 2-a, 3-d, 4-e, 5-c

C 1-c, 2-b, 3-e, 4-d, 5-a

D 1-e, 2-b, 3-c, 4-d, 5-a

E 1-a, 2-b, 3-e, 4-d, 5-c



A alternativa B está correta.

Vamos analisar cada assertiva:

1. Visão de casos de uso: visão externa, mostra os usuários e as funcionalidades com as quais interage; portanto, letra B (permite olhar do sistema sob o ponto de vista externo, descrevendo seu comportamento por interações usuário-sistema).

2. Visão lógica: permite visualizar o sistema sob o ponto de vista de sua estrutura interna e seu comportamento; portanto, letra A (enfatiza os pacotes, as classes, as interfaces, os subsistemas – pacotes – e as colaborações).

3. Visão de processo: concentra-se na divisão do sistema em processos e processadores; portanto, Letra D (enfatiza aspectos físicos mais peculiares, como concorrência, sincronismo entre sistemas e desempenho).

4. Visão de implementação: retrata a visão física do sistema em seus elementos; portanto, letra E (compreendem os componentes, subsistemas e arquivos que compõem fisicamente o sistema).

5. Visão de implantação: concentra-se em mostrar como o sistema vai ser distribuído entre os nós (elementos computacionais); portanto, letra C (enfatiza a distribuição física do sistema em suas partes e as respectivas conexões entre elas).

UML e integração com processos de desenvolvimento

Como uma linguagem padronizada de modelagem, a UML pode ser adaptada a uma variedade de metodologias e processos de desenvolvimento, como Ágil, Cascata, Iterativo e RUP. O conhecimento teórico nesse aspecto torna você apto a selecionar e aplicar adequadamente os diagramas UML em cada fase do ciclo de vida do software, garantindo uma comunicação eficaz entre equipes e uma implementação bem-sucedida dos sistemas.

Assista ao vídeo e entenda a integração da UML com os processos de desenvolvimento.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

A UML é independente de metodologia e processo de desenvolvimento. Isso é positivo para os fabricantes de software que implementam UML, pois não limita seu mercado.

Assim, cabe a cada empresa ou equipe de desenvolvimento a integração da UML com sua metodologia de desenvolvimento de sistemas computacionais, permitindo uma modelagem eficiente.

A UML pode ser usada de diversas formas, desde esboços manuais para interação com usuários ou equipe, passando pelo uso de ferramentas de diagramação UML até sofisticadas ferramentas CASE, que integram os modelos e geram código em linguagens específicas. A UML então pode ser adaptada em qualquer processo. Conheça alguns deles!

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.

Atividade 3

Questão 1

O processo de desenvolvimento de software é uma sequência de atividades planejadas e estruturadas para criar um produto de software. Inclui etapas como análise, design, implementação, teste e manutenção, visando atender aos requisitos do cliente. Qual dos seguintes processos de desenvolvimento é caracterizado por permitir modificações sempre que necessário e enfatizar o desenvolvimento de código?

A Cascata.

B Iterativo.

C Ágil

D RUP (*Rational Unified Process*).

E Estruturado.



A alternativa C está correta.

O processo de desenvolvimento ágil compartilha um conjunto de valores e princípios definidos pelo Manifesto Ágil. Ele permite modificações sempre que necessário e enfatiza o desenvolvimento de código, priorizando a comunicação com o usuário e a equipe de desenvolvimento.

Visão geral dos diagramas UML

Esses diagramas fornecem uma linguagem visual unificada para modelar sistemas complexos, permitindo uma comunicação clara e eficiente entre desenvolvedores, analistas e stakeholders. Compreender a variedade de diagramas UML disponíveis, e saber quando e como aplicá-los em diferentes estágios do desenvolvimento de software, é fundamental para garantir precisão, consistência e compreensão do sistema em desenvolvimento. Portanto, o conhecimento teórico sobre os diagramas UML oferece uma base sólida para análise, projeto e documentação desses sistemas.

Confira neste vídeo uma visão geral dos diagramas UML.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Aqui vamos analisar a versão mais recente da UML, a 2.5.1, que traz 14 diagramas com as divisões a seguir.

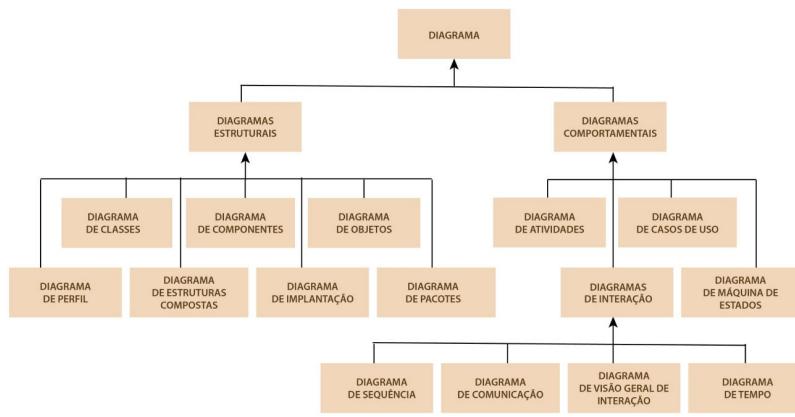
Diagramas estruturais

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.

Observe esses diagramas!



Fique agora com uma breve descrição de cada diagrama da UML. Vamos lá!

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).

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.

Atividade 4

Questão 1

Qual diagrama da UML é usado para descrever a estrutura e as conexões entre os componentes de um sistema, sendo uma parte importante da modelagem de sistemas complexos e podendo ser usado em conjunto com outros diagramas para fornecer uma visão abrangente do sistema?

A Diagrama de perfil.

B Diagrama de classes.

C Diagrama de estruturas compostas.

D Diagrama de componentes.

E Diagrama de implantação.



A alternativa D está correta.

O diagrama de componentes é especificamente destinado a apresentar a estrutura e as conexões entre os componentes de um sistema. Ele fornece uma visão geral das partes que compõem o sistema e como elas se relacionam entre si, contribuindo para uma compreensão clara da arquitetura geral do sistema.

Diagramas UML e sua utilização nas fases

Esses diagramas servem como uma linguagem padrão para visualizar, especificar, construir e documentar os artefatos de um sistema. Desde a análise até a implementação e manutenção, os diagramas UML ajudam a comunicar conceitos complexos de forma clara e concisa, facilitando a compreensão entre os membros da equipe e garantindo uma representação precisa dos requisitos e da arquitetura do sistema em todas as etapas do processo de desenvolvimento.

Comprenda neste vídeo o uso dos diagramas UML e sua utilização nas fases de análise e projeto do processo de desenvolvimento de software.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Dificilmente usamos todos os diagramas da UML em um único sistema. 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**. Todos os mencionados são relevantes, mas os em negrito são os essenciais.

Já descrevemos aqui as atividades das fases de análise (levantamento e análise de requisitos) e projeto independentemente do processo de desenvolvimento por entender que ambas estarão presentes em qualquer processo que usemos. Agora, partindo desse mesmo pressuposto, vamos descrever os diagramas UML que podem ser usados em cada uma dessas três fases. Citaremos apenas os diagramas mais usados.

Diagramas UML na atividade de análise

Aqui estão os itens essenciais para a atividade de análise. Veja!

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.

O diagrama conceitual de classes destaca as classes, incluindo seus principais atributos (inicialmente sem métodos), e os principais relacionamentos entre elas, geralmente representando apenas associações, que são os relacionamentos mais básicos entre classes.

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

Aqui estão os itens essenciais para a 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.

O diagrama de classes de projeto é derivado do diagrama conceitual de classes, adicionando novos atributos, todos os métodos necessários e identificando os relacionamentos corretos entre as classes (não apenas associações). Ele também inclui as multiplicidades e outros elementos relevantes da UML.

Atividade 5

Questão 1

Assinale o diagrama que não é usado (no sentido de elaborado) na fase de projeto, independentemente do processo de desenvolvimento adotado:

A Diagrama de sequência.

B Diagrama de atividades.

C Diagrama de casos de uso.

D Diagrama de implantação.

E Diagrama de componentes.



A alternativa C está correta.

Dos diagramas apresentados, o único cuja elaboração acontece exclusivamente na fase de análise é o diagrama de casos de uso. Os demais são típicos da fase de projeto, embora sequência e atividades possam também ser usados na fase de análise.

Considerações finais

- Modelagem de sistemas.
- Processo de desenvolvimento de sistemas.
- Conceitos fundamentais da orientação a objetos.
- Análise e projeto de sistemas orientados a objetos.
- UML e integração com processos de desenvolvimento.
- Diagramas UML.

Podcast

Agora, o especialista Marcelo Vasques de Oliveira encerra fazendo um resumo de todo o conteúdo.



Conteúdo interativo

Acesse a versão digital para ouvir o áudio.

Explore +

Confira as indicações que separamos!

- Visão geral do desenvolvimento em camadas: trabalhando com desenvolvimento em camadas utilizando C#.v
- Desenvolvimento em três camadas: conceitos.
- Modelagem de sistemas através da UML: uma visão geral.
- **UML (Unified Modeling Language) e Unified Software Development Process (Unified Process).**

Pesquise na internet:

- O diagrama da arquitetura em camadas, publicado no cogle.it, sob forma de infográfico de domínio público.

Referências

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 3. ed. Rio de Janeiro: Elsevier, 2015.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. **UML**: guia do usuário. 2. ed. Rio de Janeiro: Elsevier, 2005.

FOWLER, M. **UML essencial**: um breve guia para a linguagem-padrão de modelagem de objetos. Trad. João Tortello. 3. ed. Porto Alegre: Bookman, 2005.

LARMAN, C. **Utilizando UML e padrões**: uma introdução à análise e ao projeto orientados a objetos e ao processo unificado. 3. ed. Porto Alegre: Artmed, 2007.