Projeto de Sistemas

Projeto de Sistemas

Thiago Salhab Alves

Dados Internacionais de Catalogação na Publicação (CIP)

Alves, Thiago Salhab
A474p Projeto de sistemas / Thiago Salhab Alves. – Londrina : Editora e Distribuidora Educacional S.A., 2018.
184 p.

ISBN 978-85-522-1168-6

1. Fases do projeto. 2. Arquitetura de software. 3. Banco de dados. 4. Programação orientada a objetos. I. Alves, Thiago Salhab. II. Título.

CDD 001.6

Thamiris Mantovani CRB-8/9491

2018
Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: http://www.kroton.com.br/

Sumário

Palavras do autor

Olá, alunos, sejam bem-vindos à disciplina de Projeto de Sistemas. Com ela iremos aprender conceitos, técnicas e práticas para a análise e projeto de sistemas, habilidades estas essenciais para a formação do profissional da área de análise e desenvolvimento de sistemas e sistemas de informação. Esta disciplina irá permitir que você aprenda a analisar projetos de sistemas, modele a análise para projeto por meio dos diagramas de casos de uso e do diagrama de classes e avalie e aplique ações de arquiteturas relacionadas ao projeto de sistemas. Temos um grande desafio pela frente, não é mesmo? Saiba que essas competências serão aplicadas nesta disciplina de forma prática, permitindo que você vivencie, realizando a análise de projetos de sistemas, em situações profissionais reais. Estão prontos para dominar essas importantes atividades de nossa área de atuação profissional? Então, vamos lá!

Na Unidade 1 você irá aprender as habilidades e funções do analista de sistemas, conhecendo o ciclo de vida de desenvolvimento de sistemas, por meio das etapas do planejamento, análise, projeto e implementação. Veremos, também, nesta unidade, a fase de análise e determinação dos requisitos, utilizando técnicas de elicitação (obtenção de dados junto aos usuários detentores das informações) e estratégias de requisitos. Iremos aprender sobre projeto de sistemas Orientado a Objetos, introduzindo a UML (*Unified Modeling Language*) e seus diagramas e aplicações.

Na Unidade 2 você irá entender a passagem da modelagem para análise de projeto por meio do diagrama de Casos de Uso, detalhando relacionamentos e estereótipos. Veremos a modelagem da documentação de Casos de Uso detalhados e aplicações em alto nível e também as aplicações dos Casos de Uso em sistemas complexos e teste dos Casos de Uso em projeto de sistemas.

Na Unidade 3 você irá aprender a passagem da modelagem da análise para o projeto utilizando o Diagrama de Classes, com o refinamento dos aspectos estáticos e estruturais das classes. Veremos como realizar relacionamentos e associações e modelagem do diagrama de classes refinado, por meio das especificações dos relacionamentos e associações das classes. Iremos aprender

também como realizar o mapeamento das classes para o modelo relacional, conhecendo a notação do esquema das tabelas em mapeamento de classes e regras do mapeamento das classes.

Na Unidade 4 você irá aprender sobre o projeto da arquitetura, conhecendo as categorias e estilos arquiteturais chamados *Layered-based Architecture, Model-View-Controler, Service Oriented Architecture e Component-based Development*. Iremos também aprender sobre frameworks e padrões de projeto *Composite, Observer, Strategy, Factory Method, Mediator e Façade*. Veremos inclusive os elementos de um Design de Arquitetura, criando um design de arquitetura do sistema.

Você viu quanta coisa iremos aprender juntos? Tenho certeza que será um desafio muito prazeroso e empolgante para você. Para que possa maximizar seu aprendizado, realize todas as atividades propostas. Bons estudos.

Análise e projeto de sistemas

Convite ao estudo

Prezados alunos, esta unidade de ensino tem por objetivo introduzir o processo de análise e projetos de sistemas. Quando temos algum problema do cotidiano para solucionar, a primeira coisa que fazemos não é analisá-lo e planejar a sequência de atividades para resolvê-lo? O processo de análise e projeto de um sistema é muito parecido.

Dois jovens irmãos herdaram de seus familiares uma padaria e confeitaria em sua cidade natal. Com o passar dos anos, a padaria foi se popularizando e aumentou, ainda mais, o número de clientes, tornando seu gerenciamento, antes totalmente manual ou parcialmente informatizado, absolutamente inviável de ser mantido. No início eram apenas quatro funcionários e hoje são mais de trinta. O sucesso está tão grande que os irmãos se viram obrigados a informatizar o local, trazendo mais controle e conforto aos seus clientes. Como os sócios não entendem muito de tecnologia, você foi contratado para auxiliá-los na implantação de um projeto de sistema para controlar as atividades da padaria e confeitaria. Quais habilidades você, como Analista de Sistemas, deve ter para realizar este projeto? Você acha que este projeto é viável? Sendo viável, como realizar o plano do projeto? Quais estratégias devem ser adotadas para elicitação e análise dos requisitos? Como se desenvolve um projeto Orientado a Objetos e quais ferramentas para modelagem podem ser utilizadas? Você irá trabalhar essas e outras questões nesta unidade de ensino 1.

Desta forma você irá conhecer as habilidades do Analista de Sistemas e suas funções, bem como o ciclo de vida de

desenvolvimento de um sistema, desde seu planejamento, pelos estudos de viabilidade, até sua implementação. Aprenderá a determinar os requisitos por meio das técnicas de elicitação de requisitos e projetar sistemas orientados a objetos usando a UML. Bom trabalho.

Seção 1.1

Introdução à análise e projeto de sistemas

Diálogo aberto

Prezados alunos, vocês já pararam para pensar como era realizado todo o gerenciamento de uma empresa, por exemplo, uma padaria, sem o uso de sistemas de informação? Como o estoque era controlado? Como as contas a pagar e receber eram administradas? Todo o processo era realizado manualmente, em geral utilizando algum tipo de planilha, manual ou eletrônica, e livros caixa. Era um processo bem trabalhoso e de difícil gerenciamento, não? Certamente, um sistema de informação acaba por facilitar as atividades diárias de uma empresa, permitindo maior controle e gerência.

Neste cenário, dois jovens irmãos herdaram de seus familiares uma padaria e confeitaria em sua cidade natal. Com o passar dos anos, a padaria se popularizou tanto que seu gerenciamento, antes totalmente manual ou parcialmente informatizado, tornou-se absolutamente inviável de ser mantido. O sucesso está tão grande que os irmãos se viram obrigados a informatizar o local. Como os sócios não entendem muito de tecnologia, você foi contratado para auxiliá-los na implantação de um projeto de sistema para controlar as atividades da padaria e confeitaria. Desta forma, quais habilidades e funções você, como Analista de Sistemas, deve ter para conduzir o projeto da padaria? É possível realizar um estudo de viabilidade da implantação do projeto de sistema? Como realizar este estudo de viabilidade? Como deve ser o ciclo de vida de desenvolvimento deste sistema e como elaborar o plano do projeto? Você deverá apresentar para os proprietários da padaria um estudo de viabilidade e ciclo de vida do projeto.

Para responder a esses questionamentos e poder auxiliar os irmãos na implantação de um projeto de sistema para controlar as atividades da padaria e confeitaria, você deve aprender as habilidades e funções do analista de sistemas, conhecendo o ciclo de vida de desenvolvimento de sistemas, por meio das etapas do planejamento; análise, projeto e implementação; identificação e início do projeto;

análise de viabilidade; plano do projeto e etapas de gerenciamento; e controle do projeto. Essas são etapas fundamentais para o processo de análise e projeto de um sistema, sendo de extrema importância para seu aprendizado e para as próximas etapas a serem realizadas.

Não pode faltar

Habilidades e funções do analista de sistemas

De acordo com Dennis, Wixom e Roth (2014), o analista de sistemas desempenha papel fundamental em projetos de desenvolvimento de sistemas, trabalhando muito próximo a todos os membros da equipe de projeto para que desenvolvam o sistema de maneira correta e eficiente. Os analistas devem saber aplicar as tecnologias para resolver problemas da organização, atuando como agentes de mudança, identificando melhorias que a instituição possa necessitar, projetando sistemas que implementem as modificações e motivem as pessoas a utilizá-los.

Os analistas de sistemas devem ter habilidades técnicas para entender o ambiente técnico existente na organização. Habilidades empresariais são necessárias para compreender como aplicar as tecnologias de informação (TI) a situações comerciais, garantindo que TI agregue valor real. São habilidades dos analistas de sistemas:

- Comunicação: ter clareza, pois frequentemente irão se comunicar pessoalmente com usuários, gerentes de empresas (que geralmente possuem pouca experiência com tecnologia) e programadores (em geral, possuem mais experiência técnica que os analistas).
- Apresentação: ser capazes de fazer apresentações para grupos grandes e pequenos, mostrando resultados de suas análises e relatórios;
- Gerência: gerenciar pessoas com quem trabalha e administrar pressão, riscos e situações incertas.
- Ética e honestidade: lidar de maneira justa, honesta e ética com outros membros da equipe de projeto, gerentes e usuários do sistema, pois, frequentemente, irão lidar com informações confidenciais ou que possam causar prejuízos à empresa.

- Confiança e credibilidade: como vão trabalhar com pessoas e informações, das mais variadas áreas e níveis de conhecimento, necessitam obter confiança e credibilidade.

Segundo Dennis, Wixom e Roth (2014), as organizações e a tecnologia tornam-se cada vez mais complexas, fazendo com que a maioria das organizações crie equipes de projeto que incorporem vários analistas, com funções diferentes, porém complementares. São funções dos analistas:

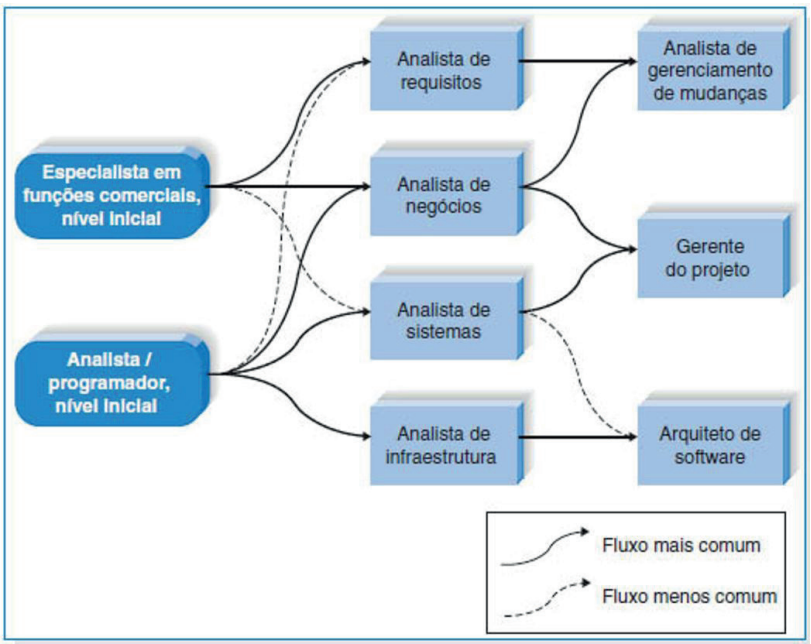
- Analista de sistemas: projetando sistemas de informações, desenvolvendo ideias e sugestões de como a TI pode fornecer e melhorar os processos da empresa. Deve possuir treinamento e experiência em análise, projeto e programação de computadores.
- Analista de negócios: concentra-se nas questões empresariais que envolvem o sistema, ajudando na identificação do valor comercial que o sistema agregará. Deve possuir treinamento e experiência comercial e sobre análise e projeto.
- Analista de requisitos: atua na elicitação dos requisitos relacionados ao sistema, precisando entender bem o negócio. Deve possuir boa habilidade de comunicação e ser especializado em conjunto de técnicas de elicitação dos requisitos.
- Analista de infraestrutura: atua nas questões técnicas que envolvem o modo com o sistema vai interagir com a infraestrutura técnica da empresa (hardware, software, redes e banco de dados). Assegura que o sistema de informação está em conformidade com padrões organizacionais e ajuda a identificar mudanças na infraestrutura para suporte ao sistema. Deve possuir treinamento e experiência em redes de computadores, administração de banco de dados, hardware e software.
- Analista de gerenciamento de mudanças: concentra-se nas questões de pessoal e gerenciamento relacionado à instalação do sistema, garantindo que a documentação e o suporte ao sistema estejam disponíveis para os usuários, fornecendo treinamento do novo sistema e estratégias para

superar a resistência às mudanças. Deve possuir treinamento e experiência em comportamento organizacional e gerenciamento de mudanças.

- Gerente do projeto: assegura que o projeto esteja concluído dentro do prazo e orçamento, garantindo que o sistema agregue valor à organização. Geralmente o gerente de projetos atuou por muitos anos como Analista de Sistemas e adquiriu competências para gerenciar o projeto.

A Figura 1.1 ilustra os vários caminhos que o profissional da área de análise e desenvolvimento de sistemas pode seguir.

Figura 1.1 | Carreiras para desenvolvedores de sistemas



Fonte: Dennis, Wixom e Roth (2014, [s.p.]).

Reflita

Uma das mais importantes habilidades do analista de sistema é a Comunicação. Temos pessoas que possuem uma facilidade muito

grande em se comunicar e pessoas que apresentam uma grande dificuldade na comunicação. Visto que essa é uma habilidade essencial para o analista, quais técnicas podem ser utilizadas para melhorá-la?

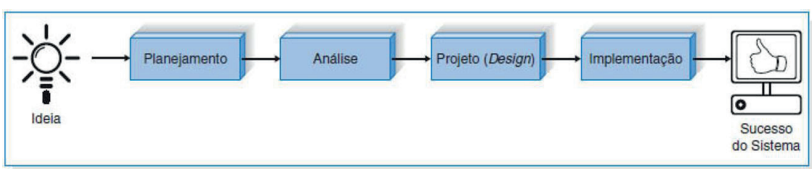
Ciclo de vida de desenvolvimento de sistemas

De acordo com Schach (2010), o ciclo de vida é uma descrição de etapas que devem ser seguidas quando se cria um produto de software. Por ser mais fácil realizar uma sequência de tarefas menores do que uma grande tarefa, o ciclo de vida de desenvolvimento de sistemas é subdividido em etapas menores, chamadas fases, que envolvem atividades de planejamento, análise, projeto (design) e implementação.

Segundo Dennis, Wixom e Roth (2014), o ciclo de vida para a criação de um sistema de informação assemelha-se muito com a construção de uma casa, sendo realizado em fases (etapas que devem ser seguidas quando se cria um produto de software). Primeiramente o proprietário descreve a visão da casa para o projetista. Em seguida, essa ideia é colocada na forma de desenhos e mostrada ao proprietário até ele concordar que as ilustrações refletem o que ele necessita. Na sequência, são desenvolvidos conjuntos de plantas detalhadas, apresentando informações específicas sobre a casa (posição dos quartos, localização dos aparelhos hidráulicos, tomadas etc.) e, por fim, a casa é erguida de acordo com essas plantas, podendo sofrer algumas alterações enquanto está sendo construída.

Construir um sistema de informações segue um conjunto de quatro fases fundamentais: planejamento, análise, projeto e implementação (DENNIS; WIXOM; ROTH, 2014). A Figura 1.2 apresenta este ciclo de vida.

Figura 1.2 | Ciclo de vida de desenvolvimento de sistemas



Fonte: Dennis, Wixom e Roth (2014, [s.p.]).

A fase de planejamento é o processo para entendimento de por que um sistema de informação deve ser construído e para determinar como a equipe de projeto procederá na sua construção. Nessa etapa realiza-se a iniciação do projeto, identificando o valor agregado do sistema para a empresa. É realizada uma análise de viabilidade para sua execução. São consideradas viabilidades técnicas (Podemos construí-lo?); viabilidade econômica (Ele agregará valor?); e viabilidade organizacional (Se o construirmos, ele será usado?).

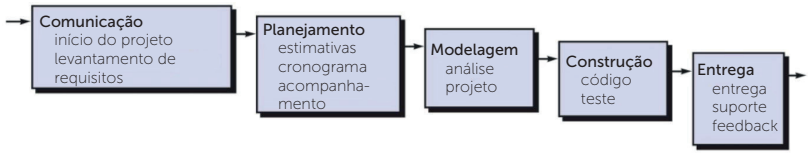
A fase de análise é o processo que foca em quem usará o sistema, o que o sistema fará, onde e quando ele será usado. Nessa fase é realizado o levantamento dos requisitos, por meio de entrevistas, questionários etc.) e a análise dessas informações. De posse da análise, do conceito de sistema e dos modelos disponíveis é gerado um documento denominado proposta de sistema, que é apresentado aos responsáveis pelo projeto, que decidirão se o projeto deve prosseguir.

A fase de projeto aborda como o sistema funcionará em termos do software, do hardware e da infraestrutura de rede que serão implementados, incluindo a interface do usuário, formulários, relatórios, programas, banco de dados e arquivos que serão necessários.

A fase de implementação foca na construção do sistema, sendo a fase que exige mais atenção, pois para a maior parte dos sistemas, é a parte mais cara do processo de desenvolvimento. A primeira etapa é a implementação do sistema, construindo e testando, a fim de garantir que funcionará do modo que foi concebido. A seguir, o sistema é instalado e um plano de treinamento deve ser elaborado. Por fim, um plano de suporte deve ser criado, identificando as alterações principais e secundárias que o sistema necessita.

De acordo com Pressman e Maxim (2016), o modelo cascata, também conhecido como ciclo de vida clássico, trabalha de forma semelhante ao apresentado anteriormente. Ele segue uma abordagem sequencial e sistemática, começando com a especificação dos requisitos do cliente, avançando pelas fases de planejamento, modelagem, construção e entrega, possuindo suporte contínuo do software construído. A Figura 1.3 apresenta o modelo cascata.

Figura 1.3 | Modelo cascata



Fonte: Pressman e Maxim (2016, p. 42).

Identificação e início do projeto

De acordo com Pressman e Maxim (2016), o projeto de software inclui um conjunto de princípios, conceitos e práticas, que levam ao desenvolvimento de um sistema ou produto de alta qualidade, sendo o guia de trabalho que devemos desempenhar. A atividade de projeto é crucial para uma engenharia de software bem-sucedida.

Segundo Dennis, Wixom e Roth (2014), um projeto é identificado quando alguém reconhece a necessidade da empresa ou do negócio de construir um sistema. As necessidades podem incluir um novo tipo de cliente ou algum ponto de insatisfação dentro da organização, tais como: perda de mercado, taxas inaceitáveis de defeitos em produtos, aumento da concorrência ou por meio da identificação e uso da TI como estratégia competitiva.

Quando a necessidade de projeto é reconhecida pela empresa, surge uma pessoa (ou grupo de pessoas) com interesse no sucesso do sistema. Esta pessoa ou grupo de pessoas são os patrocinadores (*sponsors*). O patrocinador tem por objetivo garantir que o projeto está na direção correta e é o principal contato com a equipe de projeto.

Uma vez determinado o patrocinador, a solicitação do sistema é documentada, descrevendo os motivos que levam a empresa a construir o sistema e o valor esperado que ele irá custar. Normalmente este documento é composto de cinco elementos (DENNIS; WIXOM; ROTH, 2014):

- Patrocinador do projeto: pessoa que inicia o projeto e serve como ponto principal de contrato no lado comercial. Exemplos: Gerente de TI, CIO, CEO etc.
- Necessidade de negócio: o motivo relacionado à atividade empresarial para o sistema ser iniciado. Exemplos: aumento das vendas; melhoria da fatia de mercado; diminuição dos defeitos nos produtos etc.

- Requisitos do negócio: capacidades empresarias que o sistema fornecerá. Exemplos: fornecer acesso on-line às informações; produzir relatórios de gestão; incluir suporte on-line aos clientes etc.
- Valor do negócio: vantagens que o sistema criará para a organização. Exemplos: aumento de 3% nas vendas; economia de custos; redução das horas trabalhadas etc.
- Restrições: questões importantes e pertinentes à implementação do sistema e que precisam ser conhecidas para aprovação. Exemplos: prazo final em 30 de maio de 2018; necessidade que o sistema esteja pronto a tempo para o natal, etc.

Exemplificando

Quadro 1.1 | Exemplo de solicitação para início do projeto

Solicitação de Sistema – Projeto de Download de Músicas Digitais
Patrocinador do projeto: Carly Edward, vice-presidente adjunto de marketing
Necessidade de negócio: iniciado para aumentar as vendas, criando a capacidade de comercializar downloads de músicas digitais aos clientes por intermédio de quiosques em nossas lojas e pela Internet, usando nosso site na Web.
Valor do negócio: esperamos que a Tune Source aumente as vendas por permitir que seus clientes comprem trilhas específicas de músicas digitais, além de pesquisar por novos clientes que estejam interessados em nosso arquivo exclusivo de músicas raras e difíceis de encontrar. Esperamos ganhar uma nova parcela de lucro em virtude de assinaturas de clientes em nossos novos serviços de downloads. Acreditamos em um aumento nas vendas casadas, quando os clientes que tiverem realizado o download de uma ou duas trilhas de um CD decidirem comprar o CD inteiro em uma loja ou por intermédio de nosso site na Web. Também esperamos uma nova parcela de lucro em consequência da venda de vales-presente de download de músicas. Estimativas conservadoras de valores tangíveis para a companhia incluem as seguintes: • US$757.500 em vendas de downloads de músicas isoladas. • US$950 mil em vendas de assinaturas de clientes. • US$205 mil em vendas adicionais de CD na loja ou no site na web. • US$153 mil em vendas de vales-presente de downloads de músicas.

Restrições:

- O departamento de marketing considera que esse é um sistema estratégico. A capacidade de oferecer downloads de músicas digitais é crítica para que a empresa permaneça competitiva em nossa fatia do mercado. Nosso arquivo de músicas raras e difíceis de encontrar é um ativo da empresa que atualmente está subutilizado.
- Muitos de nossos clientes leais estiveram solicitando essa oferta e precisamos fornecer esse serviço ou enfrentar a perda dos negócios com esses consumidores.
- Em face de estarem disponíveis aos clientes, em outros locais, várias opções de downloads de músicas, precisamos trazer esse sistema ao mercado com a maior brevidade possível.

Fonte: Dennis, Wixom e Roth (2014, [s.p.]).

Estudo de viabilidade

De acordo com Pressman e Maxim (2016), um dos pontos principais do projeto de software é determinar seu escopo, que descreve suas funções e características. As funções relatadas na definição do escopo são utilizadas para determinar estimativas de custo e cronograma de trabalho. Uma vez identificado o escopo, é essencial responder algumas perguntas: "Podemos criar o software que atenda a esse escopo?", "O projeto é viável?". Dessa forma, delimitar apenas o escopo não é suficiente, pois é necessário verificar se ele pode ser desenvolvido segundo as dimensões de tecnologia, orçamento, tempo e recursos disponíveis.

Segundo Dennis, Wixom e Roth (2014), a análise de viabilidade orienta a empresa a determinar se o projeto deve prosseguir, identificando os riscos associados a ele. Cada empresa possui seu próprio processo e formato para análise de viabilidade, mas geralmente incluem:

- Viabilidade técnica: determinar se a organização possui condições técnicas de desenvolver o projeto e a tecnologia a ser utilizada. Uma análise técnica de risco deve ser aplicada. É necessário levar em consideração a familiaridade com a aplicação e a tecnologia.
- Viabilidade econômica: realizar a análise custo versus benefício. Deve-se responder à pergunta: "Nós devemos

construir o sistema?". Levar em consideração os custos de desenvolvimento e custos operacionais.

- Viabilidade organizacional: verificar se o sistema construído será realmente utilizado, isto é, o quanto, no final das contas, o sistema será bem-aceito por seus usuários e incorporado nas operações em curso na empresa.

Pesquise mais

Para um maior conhecimento sobre o Ciclo de Vida no desenvolvimento de software, realize a leitura do seguinte artigo: RIBEIRO, Rodrigo de Carvalho et. al. **A importância do ciclo de vida no desenvolvimento de software**. Disponível em: <http://revistaconexao.aems.edu.br/wp-content/plugins/download-attachments/includes/download.php?id=993>. Acesso em: 27 mar. 2018.

Plano do projeto

De acordo com Dennis, Wixom e Roth (2014), uma vez que o projeto foi aprovado para iniciação, o mesmo deve ser planejado. Assim, o gerente do projeto irá seguir um conjunto de diretrizes de gerenciamento de projetos, determinando a melhor metodologia, desenvolvendo um plano de trabalho para o projeto, escolhendo os integrantes da equipe e estabelecendo mecanismos para coordenar e controlar o projeto.

Quando o gerente de projeto tiver uma ideia geral do tamanho e tempo aproximado do projeto, deve-se criar um plano de trabalho que registre e mantenha o controle de todas as tarefas que precisam ser realizadas durante o projeto. O plano de trabalho deve conter: as tarefas que devem ser realizadas; o tempo que cada uma delas consumirá; e a equipe responsável por cada tarefa.

De acordo com Padua Filho (2009), o gerente do projeto também deve garantir a qualidade do projeto. Para isso os gerentes precisam:

- Tomar providências necessárias, no nível do projeto, para a realização das atividades previstas no plano do projeto.
- Designar responsáveis pelas tarefas relacionadas com a qualidade interna do projeto.
- Procurar resolver completamente os problemas em relação ao projeto.

Desta forma, o gerente de projetos deve elaborar um conjunto planejado e sistemático de ações necessárias para estabelecer um nível adequado de confiança de que um item ou produto está em conformidade com seus requisitos técnicos.

Assimile

De acordo com Dennis, Wixom e Roth (2014), alguns procedimentos não são recomendados para garantir que a motivação do projeto seja a melhor possível:

- Atribuir prazos irreais: poucas pessoas vão trabalhar com afinco se perceberem que o prazo é impossível de cumprir;
- Ignorar esforços positivos: pessoas trabalharão mais se perceberem que seu trabalho está sendo reconhecido;
- Criar produto de baixa qualidade: pessoas ficam desmotivadas em trabalhar em um projeto de baixa qualidade;
- Dar aumento a todos do projeto: se todos recebem a mesma recompensa, as pessoas de maior qualidade vão acreditar que não vale a pena se esforçar;
- Tomar uma decisão importante sem envolver a equipe: o gerente do projeto, se necessitar tomar decisão que envolve e afeta os membros da equipe, deve envolve-los no processo de tomada de decisão;
- Manter condições precárias de trabalho: uma equipe de projeto necessita de um bom ambiente de trabalho ou ficarão desmotivados.

Com isso, prezado aluno, fechamos a nossa primeira seção. Aprendemos aqui sobre as habilidades e funções do analista de sistemas, conhecemos o ciclo de vida de desenvolvimento de sistemas, por meio das etapas: planejamento; análise, projeto e implementação; identificação e início do projeto; análise de viabilidade; plano do projeto e etapas de gerenciamento; e controle do projeto. Na próxima seção iremos aprender a determinar os requisitos utilizando técnicas de elicitação de requisitos. Até lá!

Sem medo de errar

Chegou a hora de resolver nossa situação problema. Antes disso, vamos relembrá-la: dois jovens irmãos herdaram de seus familiares uma padaria e confeitaria em sua cidade natal. Com o passar dos anos, a padaria popularizou-se tanto que seu gerenciamento, antes totalmente manual ou parcialmente informatizado, tornou-se absolutamente inviável de ser mantido. O sucesso está tão grande que os irmãos se viram obrigados a informatizar o local. Como os sócios não entendem muito de tecnologia, você foi contratado para auxiliá-los na implantação de um projeto de sistema para controlar as atividades da padaria e confeitaria. Desta forma, quais habilidades e funções você, como analista de sistemas, deve ter para conduzir o projeto da padaria? É possível realizar um estudo de viabilidade da implantação do projeto de sistema? Como realizar este estudo de viabilidade? Como deve ser o ciclo de vida de desenvolvimento deste sistema e como elaborar o plano do projeto?

Para resolver essa situação problema e responder essas questões, você deve, primeiramente, identificar quais são as habilidades e funções do analista de sistema.

São habilidades esperadas:

- Comunicação: o analista deve ter habilidades de comunicação, visto que irá se comunicar com clientes, geralmente pessoas leigas na área computacional.
- Gerência: o analista deve saber gerenciar pessoas com quem trabalha e administrar pressão, riscos e situações incertas.
- Ética e honestidade: o analista deve lidar de maneira justa, honesta e ética com outros membros da equipe de projeto, gerentes e usuários de sistema.
- Confiança e credibilidade: como o analista deve lidar com pessoas e informações, das mais variadas áreas e níveis de conhecimento, necessita obter confiança e credibilidade.

São funções desse profissional:

- Analista de Sistemas: projeta sistemas de informações, desenvolvendo ideias e sugestões de como a TI pode fornecer e melhorar os processos da empresa. Deve possuir treinamento e experiência em análise, projeto e programação

de computadores; deve atuar como analista de requisitos e também como gerente de projetos.

Em seguida, realize um estudo de viabilidade desse projeto e crie um plano e ciclo de vida do projeto, com atividades de planejamento, análise e projeto de sistemas.

Estudo de viabilidade:

Primeiramente, determine o escopo do projeto da padaria e confeitaria, que descreve suas funções e características. As funções enumeradas na definição do escopo são utilizadas para determinar estimativas de custo e cronograma de trabalho. Devem ser considerados, neste momento, a viabilidade técnica (temos condições técnicas de desenvolver o projeto e a tecnologia a ser utilizada?), viabilidade econômica (análise custo x benefício) e viabilidade organizacional.

Exemplo de estudo de viabilidade:

Objetivo e meta do projeto

- Criar um sistema automatizado para padaria e confeitaria.

Escopo

- O sistema deverá realizar toda a gestão da padaria, permitindo o cadastro de produtos, funcionários, fornecedores, vendas, compras, contas a pagar, contas a receber e emissão de relatórios (produtos, vendas, compras, contas a pagar e receber). Os funcionários da padaria poderão atualizar e consultar os seguintes elementos cadastrados: produtos, fornecedores, funcionários, compras e vendas. Apenas funcionários com perfil de administrador poderão acessar contas a pagar e receber e emissão de relatórios. O sistema será desenvolvido em linguagem orientada a objetos e banco de dados relacional.

Restrições

- O sistema deverá realizar rotina de *backup*. Apenas usuários cadastrados podem acessar o sistema.

Justificativas para o projeto

- O sistema irá proporcionar todo o controle e gestão da padaria, permitindo maior acesso às informações e à tomada de decisões.

Ciclo de vida do projeto

O ciclo de vida do projeto da padaria deve adotar as seguintes atividades:

- Planejamento: você deve planejar a duração e equipe que irá realizar o projeto. Uma equipe de quatro pessoas, sendo um analista, dois programadores e um teste, levará, aproximadamente, quatro meses para concluir o projeto gastando, em média, 20% do tempo para análise e projeto, 60% para desenvolvimento e 20% para teste.
- Análise: nesta etapa, os analistas devem realizar o levantamento de requisitos com os funcionários da padaria, determinando as exigências e restrições do sistema. Nesta fase, uma lista de requisitos funcionais e não funcionais é determinada.
- Projeto: nesta etapa, os analistas devem realizar o projeto Orientado a Objetos, utilizando para isso a UML (Linguagem de Modelagem Unificada) para modelar as funcionalidades do sistema (Diagrama de Casos de Uso) e Classes e Relacionamentos (Diagrama de Classes).
- Implementação: nesta etapa, os programadores irão seguir o projeto e implementar, em linguagem Orientada a Objetos, o sistema que foi analisado e projetado.

Ao realizar a apresentação para os proprietários da padaria mostre o estudo de viabilidade e de ciclo de vida do projeto realizado.

Avançando na prática

Projeto de sistema universitário

Descrição da situação-problema

Atuando em uma universidade, você foi contratado para solucionar problemas encontrados no processo de satisfação do aluno, no formato de uma avaliação institucional. Pensando na universidade, quais melhorias poderiam ser implementadas

para aumentar a satisfação do universitário matriculado com os processos da faculdade? Atualmente, os estudantes podem avaliar a instituição? Como isso é feito? Depois, pense em como a tecnologia pode ajudar a dar suporte à sua ideia. Você precisaria de uma tecnologia inteiramente nova? O sistema atual pode ser alterado?

Resolução da situação-problema

Para a resolução deste problema, você deve elaborar a solicitação de um sistema de avaliação institucional on-line, criando uma proposta de solicitação de sistema, contendo: o responsável, a necessidade do negócio, os requisitos do negócio e o valor potencial do projeto. Inclua quaisquer restrições ou questões que devam ser levadas em consideração.

Proposta do sistema: avaliação institucional on-line.

Responsável: nome do coordenador

Necessidade do negócio: este projeto foi iniciado para facilitar a avaliação institucional, totalmente on-line, utilizando o próprio portal da Faculdade, alterando o antigo formato de pesquisa, que era totalmente manual.

Valor potencial do projeto: espera-se que o sistema de avaliação institucional de forma on-line aumente a satisfação dos alunos, visto que atualmente o processo é realizado totalmente de forma manual e o estudante não recebe um retorno dos itens avaliados e propostas de melhoria. O coordenador de curso também poderá fazer todo o acompanhamento, relacionado ao seu curso, pelo sistema de avaliação institucional on-line.

Restrições: serviço de *backup* e integração total com o website da faculdade.

Faça valer a pena

1. As organizações e a tecnologia tornam-se cada vez mais complexas, fazendo com que a maioria das organizações crie equipes de projeto que incorporem vários analistas, com funções diferentes, porém, complementares. Uma equipe heterogênea, com analistas e programadores sêniores, júniores e estagiários, permite um bom desempenho e troca de conhecimentos.

Quais são as funções do analista de sistemas?

a) Projetar sistemas de informação; analisar requisitos; gerenciar projetos.
b) Projetar redes de computadores; analisar requisitos; gerenciar projetos.
c) Projetar infraestrutura técnica; projetar sistemas de informação; gerenciar projetos.
d) Projetar sistemas de informação; analisar requisitos; projetar redes de computadores.
e) Projetar infraestrutura técnica; analisar requisitos; gerenciar projetos.

2. O patrocinador de um projeto tem por objetivo garantir que o planejamento está na direção correta e é o principal contato com a equipe de projeto Uma vez determinado o patrocinador, a solicitação do sistema é documentada, descrevendo os motivos que levam a empresa a construir o sistema e o valor esperado que ele irá custar.

Normalmente esse documento é composto de cinco elementos. Quais são eles?

a) Patrocinador do projeto, necessidade de negócio, requisitos do negócio, valor do negócio e Riscos.
b) Patrocinador do projeto, necessidade de negócio, requisitos do negócio, valor do negócio e restrições.
c) Patrocinador do projeto, necessidade de negócio, requisitos do negócio, valor do negócio e prazos.
d) Patrocinador do projeto, necessidade de negócio, requisitos do negócio, valor do negócio e resultados.
e) Patrocinador do projeto, necessidade de negócio, requisitos do negócio, valor do negócio e cronograma.

3. A análise de viabilidade orienta a empresa a determinar se o projeto deve prosseguir, identificando os riscos associados a ele. Cada empresa possui seu próprio processo e formato para análise de viabilidade, mas geralmente incluem alguns processos em comum.

Quais análises de viabilidades são comumente realizadas?

a) Viabilidade técnica, viabilidade econômica e viabilidade educacional.
b) Viabilidade técnica, viabilidade social e viabilidade organizacional.
c) Viabilidade técnica, viabilidade econômica e viabilidade organizacional.
d) Viabilidade técnica, viabilidade social e viabilidade educacional.
e) Viabilidade técnica, viabilidade social e viabilidade estratégica.

Seção 1.2

Análise de sistemas

Diálogo aberto

Prezados alunos, agora que você já realizou o estudo das habilidades e funções do analista de sistemas e análise da viabilidade e ciclo de vida do projeto, iremos dar continuidade ao nosso projeto, iniciando a fase de elicitação e estudo de requisitos. Vocês já pararam para pensar como o processo de comunicação é importante para o levantamento de requisitos? De que maneira devemos conduzir a comunicação para um correto e completo processo de elicitação de requisitos? Como um levantamento errôneo de requisitos pode impactar nas demais atividades do projeto? Parecem atividades muito simples de ser realizadas, mas que devem demandar bastante atenção dos analistas de requisitos.

Pensando nessas questões, voltemos ao caso dos dois jovens irmãos que herdaram de seus familiares uma padaria e confeitaria em sua cidade natal. Com o passar dos anos, a padaria popularizou-se tanto que seu gerenciamento antes totalmente manual ou parcialmente informatizado tornou-se absolutamente inviável de ser mantido. O sucesso está tão grande, que os irmãos se viram obrigados a informatizar o local. Como os sócios não entendem muito de tecnologia, você foi contratado para auxiliá-los na implantação de um projeto de sistema para controlar as atividades da padaria e confeitaria. Desta forma, como determinar os requisitos do sistema? Quais técnicas podem ser utilizadas para a elicitação de requisitos? Quais os requisitos funcionais e não funcionais do sistema?

Para responder a esses questionamentos e poder auxiliar aos irmãos na implantação de um projeto de sistema para controlar as atividades da padaria e confeitaria, você deve aprender as técnicas de elicitação de requisitos; as estratégias para análise de requisitos; e como elicitar e analisar os requisitos. Essas são etapas fundamentais para o processo de análise e projeto de um sistema, sendo de extrema importância para seu aprendizado e para as próximas etapas a serem realizadas.

Não pode faltar

Determinação e análise dos requisitos

De acordo com Dennis, Wixom e Roth (2014), a determinação dos requisitos é utilizada para transformar a declaração de alto nível dos requisitos em uma lista mais detalhada e precisa do que o sistema deve fazer para fornecer valor necessário ao negócio.

Requisitos são declarações do que o sistema deve fazer e quais características ele deve possuir. No processo de projeto de sistemas são criados requisitos descrevendo: as necessidades do negócio (requisitos do negócio), o que os usuários devem fazer (requisitos dos usuários), o que o software deve fazer (requisitos funcionais) e as características que o sistema deve ter (requisitos não funcionais).

Segundo Pressman e Maxim (2016), entender os requisitos de um problema está entre as tarefas mais difíceis enfrentadas pelos analistas de sistemas e engenheiros de software. Quando pensamos isso, pela primeira vez, nos parece que entender claramente a engenharia de requisitos não parece assim tão difícil, afinal, o cliente não sabe o que é necessário? Os usuários não deveriam ter um bom entendimento das características e funções que serão vantajosas? Em muitos casos a resposta é não!

Determinar as maneiras pelas quais o sistema deve suportar as necessidades dos usuários leva às declarações dos requisitos funcionais do sistema. Um requisito funcional está diretamente ligado a um processo que o sistema precisa realizar como parte do suporte fornecido a uma tarefa do usuário e/ou à informação que o sistema deve fornecer quando o usuário estiver realizando uma tarefa. Podemos definir requisitos como as capacidades do produto ou como as coisas que um produto deve fazer para seus usuários.

Considere, por exemplo, o seguinte requisito do usuário: agendar reunião com um cliente. Os requisitos funcionais associados a esta tarefa incluem: determinar a disponibilidade do cliente; encontrar horários na agenda do usuário correspondentes à disponibilidade do cliente; selecionar o horário desejado; registrar a reunião e confirmar a reunião.

Os requisitos funcionais podem ser divididos em orientados por processos e orientados por informações. Os orientados por processo apresentam o que o sistema deve realizar, como por exemplo, o

sistema deve permitir que os clientes registrados revejam seu próprio histórico de pedidos nos últimos três anos e o sistema deve verificar a disponibilidade em estoque dos pedidos que chegam dos usuários. Os requisitos orientados por informações apresentam as informações que o sistema deve conter, como por exemplo, o sistema deve conservar o histórico dos pedidos dos clientes por três anos e o sistema deve incluir níveis de estoque em tempo real em todos os depósitos.

Reflita

O relato abaixo, de Barbara Wixom (DENNIS; WIXOM; ROTH; 2014, [s.p]) apresenta a problemática da não definição de requisitos de forma correta, impactando no orçamento final do projeto.

> Como um levantamento de requisitos incompleto pode impactar no projeto?
>
> Barbara Wixom

Uma vez, trabalhei em um projeto de consultoria no qual meu gerente criou uma definição de requisitos sem listar os requisitos não funcionais. O projeto foi, então, avaliado com base na definição de requisitos e vendido ao cliente por US$5 mil. Na cabeça do meu gerente, o sistema que construiríamos para o cliente seria um sistema autônomo muito simples, executado com tecnologia atual. Ele não deve ter consumido mais de uma semana para ser analisado, concebido e construído.

Infelizmente, o cliente tinha outras ideias. Ele desejava que o sistema fosse usado por muitas pessoas em três departamentos diferentes e queria a capacidade de que qualquer número de pessoas pudesse trabalhar simultaneamente no sistema. A tecnologia que eles tinham implementado era antiquada, mas, apesar disso, queriam que o sistema fosse executado de forma eficiente nos equipamentos existentes. Por não termos estabelecido de maneira adequada o escopo do projeto, incluindo nossas hipóteses sobre requisitos não funcionais na definição de requisitos, tivemos basicamente de fazer tudo o que queriam.

As capacidades desejadas pelo cliente levaram semanas para serem projetadas e programadas. O projeto acabou levando quatro meses e seu custo final foi de US$250 mil. Nossa empresa teve de pagar a conta de tudo, exceto o acordo a respeito dos US$5 mil. Essa foi, sem dúvida, a situação de projeto mais frustrante que já vivenciei.

Um requisito não funcional pode ser descrito como um atributo de qualidade, de desempenho, de segurança ou como uma restrição geral de um sistema. Embora o termo "não funcional" não seja muito descritivo, essa categoria de requisitos inclui importantes propriedades comportamentais que o sistema deve ter, tais como: desempenho e usabilidade. Os requisitos não funcionais descrevem várias características relativas ao sistema: operacionais, de desempenho, de segurança, culturais e políticas. Não explicam processos ou informações de negócio, mas são muito importantes para o entendimento de como deve ficar o sistema final.

Os requisitos não funcionais podem ser do tipo operacional, de desempenho, de segurança e cultural e político. Os requisitos não funcionais Operacionais descrevem o ambiente físico e o ambiente técnico no qual o sistema vai trabalhar, por exemplo, o sistema pode ser executado em dispositivos portáteis e deve ser capaz de se integrar com o sistema de estoque existente. Os requisitos não funcionais de Desempenho estão relacionados com a velocidade, capacidade e confiabilidade do sistema, por exemplo, qualquer interação entre o usuário e o sistema não deve ser superior a 2 segundos e o sistema deve suportar 300 usuários simultâneos das 9h às 11h da manhã e 150 usuários simultâneos em todos os outros horários. Os requisitos não funcionais de segurança identificam quem tem acesso autorizado ao sistema e sob quais circunstâncias, por exemplo, apenas os gerentes diretos podem ver os registros pessoais dos funcionários e o sistema inclui todos os dispositivos de segurança contra vírus, *worms* etc. Os requisitos não funcionais cultural e político impactam em fatores culturais, políticos e exigências legais que afetam o sistema, por exemplo, as informações pessoais estão protegidas segundo as prescrições do *Data Protection Act* e é permitido aos gerentes regionais autorizar o uso de interfaces personalizadas com o usuário dentro de suas unidades.

Exemplificando

Requisitos funcionais

1. Gerenciamento de veículos novos

1.1 O sistema permitirá aos gerentes ver o estoque atual de veículos novos.

1.2 O sistema permitirá que o gerente de carros novos emita pedidos de novos veículos.

1.3 O sistema registrará a adição de veículos novos ao estoque, quando eles forem recebidos dos fabricantes.

2. Gerenciamento das vendas de veículos novos

2.1 O sistema permitirá que os vendedores criem uma oferta aos clientes.

2.2 O sistema autorizará que os vendedores saibam se uma oferta está vigorando para um veículo específico.

3. Gerenciamento de veículos usados

3.1 O sistema registrará as informações sobre a negociação de um veículo por um cliente.

Requisitos não funcionais

1. Operacional

1.1 O sistema deve ser executado em um tablet PC a ser usado pelo vendedor.

1.2 O sistema deve ter interface com o sistema de gerenciamento de serviços.

2. Desempenho

2.1 O sistema deve fornecer suporte a uma equipe de vendas com 15 vendedores.

2.2 O sistema deve ser atualizado com ofertas vigentes de veículos a cada 15 minutos.

3. Segurança

3.1 Nenhum vendedor pode ter acesso aos contatos de qualquer outro vendedor.

3.2 Apenas o proprietário e o gerente de vendas podem aprovar as ofertas aos clientes.

4. Cultural e político

4.1 A política da companhia estabelece que todo equipamento computacional seja comprado da Dell.

4.3 O sistema estará de acordo com as leis estaduais específicas para negociações com automóveis.

Fonte: Dennis, Wixom e Roth (2014, [s.p]).

Técnicas de elicitação de requisitos

Iremos apresentar agora as cinco técnicas de elicitação de requisitos mais comumente usadas: entrevista, sessões JAD (desenvolvimento de aplicações conjuntas), questionários, análise de documentos e observação.

Entrevista

Dentre as técnicas de elicitação de requisitos, a entrevista é a mais usada (Dennis, Wixom e Roth, 2014). Nesse processo, a seleção dos entrevistados deve ser feita de maneira estratégica, escolhendo pessoas em diferentes níveis da empresa, que terão pontos de vistas diferentes sobre o sistema, incluindo tanto os gerentes que controlam os processos como o pessoal que os executa. Devem ser planejadas com antecedências as perguntas da entrevista, que são classificadas em três tipos: fechadas, abertas e exploratórias. As perguntas fechadas são as que exigem uma resposta específica. As perguntas abertas deixam espaço para serem elaboradas por parte do entrevistado e as perguntas exploratórias dão continuidade ao que acabou de ser discutido para que o entrevistador saiba mais sobre o assunto. O Quadro 1.2 apresenta alguns exemplos de perguntas na entrevista.

Quadro 1.2 | Exemplos de perguntas nas entrevistas

Tipos de perguntas	Exemplos
Perguntas fechadas	• Quantos pedidos são recebidos por dia por meio do telefone? • Quantos clientes emitem pedidos? • Que informações estão faltando no relatório mensal de vendas?
Perguntas abertas	• Quais são alguns dos problemas que você enfrenta diariamente? • Quais são as melhorias que você gostaria de ver no modo como as faturas são processadas?
Perguntas investigativas	• Por quê? • Você pode fornecer-me um exemplo? • Você pode explicar isso com um pouco mais de detalhes?

Fonte: Dennis, Wixom e Roth (2014, [s.p]).

JAD

De acordo com Dennis, Wixom e Roth (2014), JAD (Joint Application Development) é uma técnica de coleta de informações que permite à equipe de projeto, aos usuários e à gerência trabalharem em conjunto a fim de identificar requisitos para o sistema. É um processo estruturado, no qual 10 a 20 usuários reúnem-se sob a direção do facilitador treinado em técnicas de JAD. O facilitador é um especialista em técnicas de processos de grupo como análise de sistemas e técnicas de projeto. Um ou mais auxiliares ajudam o facilitador fazendo anotações para registrar as informações à medida que a seção JAD prossegue. O grupo de JAD reúne-se por várias horas, dias ou semanas, até que todos os assuntos tenham sido discutidos e as informações coletadas. As seções ocorrem em salas de reuniões, onde geralmente os participantes sentam na forma de U para que todos possam se ver com facilidade. Os participantes da JAD são: patrocinador (cliente), que estabelece as diretrizes e objetivos do projeto; usuários-chaves, que são aqueles que utilizarão o sistema, sendo responsáveis pelo conteúdo das sessões, engenheiro de sistemas (analista de requisitos), responsável por criar os documentos das sessões e registro das decisões e especificações produzidas;

executor (analista de sistemas), responsável pelo desenvolvimento do produto; condutor (líder de sessão), pessoa que irá conduzir a JAD e reunir o pessoal para as sessões; e observadores, que são outras pessoas interessadas no projeto.

Questionários

Segundo Dennis, Wixom e Roth (2014), questionário é um conjunto de perguntas para obter informações. São usadas quando há um grande número de pessoas de quem as informações e opiniões são necessárias e geralmente estão em diferentes localizações geográficas. Os questionários podem ser enviados de forma eletrônica, por e-mail ou pela web.

As perguntas de um questionário devem ser formuladas de maneira clara e não deixar margem a má interpretação. Alguns exemplos de perguntas:

- Problemas na rede são comuns?
- Com que frequência ocorrem problemas na rede: uma vez por hora, uma vez por dia ou uma vez por semana?

Análise de documentos

De acordo com Dennis, Wixom e Roth (2014), as equipes de projeto usam com frequência a análise de documentos para compreender o sistema atual existente. Há muitos documentos úteis na empresa: relatórios de papel, memorandos, manuais de políticas da empresa, manuais de treinamento de usuários, formulários, etc.

Observação

Segundo Dennis, Wixom e Roth (2014), a observação é o ato de verificar atentamente processos em execução, sendo uma ferramenta poderosa para obter uma noção sobre o sistema existente. Permite ao analista ver a realidade de uma situação, ao invés de escutar outras pessoas descrevê-la. A observação é uma boa maneira de verificar a validade de informações reunidas por outras fontes, como entrevistas e questionários. De modo geral, ela é usada para complementar as informações da entrevista.

Assimile

De acordo com Dennis, Wixom e Roth (2014), habilidades interpessoais são aquelas que permitem desenvolver afinidade com outras pessoas, e essas habilidades são muito importantes em entrevistas. As habilidades interpessoais, como a maioria das habilidades, podem ser aprendidas. Aqui estão algumas dicas:

- **Não se preocupe, seja feliz**. Pessoas felizes irradiam confiança e projetam seus sentimentos nas outras. Tente entrevistar alguém sorrindo e, depois, tente entrevistar outra pessoa estando carrancudo e observe o que ocorre!
- **Preste atenção**. Fique atento ao que a outra pessoa está dizendo (é mais difícil do que você pode imaginar). Observe como muitas vezes você se surpreenderá com seu pensamento voltado para outra coisa diferente da conversa em curso.
- **Resuma os pontos principais**. No final de cada tema ou ideia importante que alguém esteja explicando, você deve repetir para o interlocutor os pontos principais (por exemplo, "Deixe-me ter certeza de que compreendi. As questões principais são..."). Isso demonstra que você considera as informações importantes — e também o obriga a prestar atenção (você não pode repetir aquilo que não ouviu).
- **Seja sucinto**. Quando falar, seja sucinto. O objetivo da entrevista (e porque não dizer da vida) é aprender, não impressionar. Quanto mais você fala, menos tempo dá aos outros.
- **Seja honesto**. Responda a todas as perguntas de forma verdadeira e, se não souber a resposta, diga que não sabe.

Pesquise mais

Para um maior conhecimento sobre técnicas de elicitação de requisitos, realize a leitura do seguinte material de aula:

FIGUEIREDO, Eduardo. **Técnicas de elicitação de requisitos**. Disponível em: < http://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/eng-req-tecnicas_v01.pdf>. Acesso em: 31 mar. 2018.

Dessa forma, você aprendeu a determinar requisitos, conhecendo os requisitos funcionais e não funcionais e as principais técnicas para elicitação de requisitos, sendo tudo isso de fundamental importância para seu desenvolvimento enquanto analista e desenvolvedor de sistemas.

Sem medo de errar

Chegou a hora de resolver nossa situação-problema. Como os irmãos não entendem muito de tecnologia, você foi contratado para auxiliá-los na implantação de um projeto de sistema para controlar as atividades da padaria e confeitaria. Dessa forma, como determinar os requisitos do sistema? Quais técnicas podem ser utilizadas para a elicitação de requisitos? Quais os requisitos funcionais e não funcionais do sistema?

Para solucionar essa situação-problema e responder essas questões, devemos, primeiramente, focar na forma de obter os requisitos do sistema e quais são as técnicas usadas para elicitação de tais requisitos.

Para obtermos os requisitos do sistema utilizamos algumas das técnicas:

- Entrevistas: elaborar perguntas fechadas, abertas e investigativas, realizando uma seleção dos entrevistados, diversificando as pessoas a serem entrevistadas (diferentes hierarquias na organização) e preparando com antecedências as perguntas para a entrevista;
- JAD (*Joint Application Development*): é uma técnica de coleta de informações que permite à equipe de projeto, aos usuários e à gerência trabalharem em conjunto, a fim de identificar requisitos para o sistema. Pode-se realizar uma sessão de JAD para obter os requisitos do sistema.
- Questionários: podemos criar um conjunto de perguntas para obter informações.
- Análise de documentos: podemos analisar documentos, tais como: formulários, planilhas, relatórios e também verificar softwares antigos utilizados na empresa.
- Observação: podemos utilizar a técnica de observação para verificar processos em execução, facilitando assim seu entendimento.

Dando continuidade ao processo de elicitação de requisitos, devemos determinar os requisitos funcionais e não funcionais. Para isso, é necessário escolher uma ou mais técnicas de levantamento de requisitos. Pode-se realizar entrevistas, por exemplo.

Requisitos funcionais:

RF1: O sistema deve realizar o cadastro (inserção, alteração, exclusão e consultas) dos dados dos clientes.

RF2: O sistema deve realizar o cadastro (inserção, alteração, exclusão e consultas) dos dados dos fornecedores.

RF3: O sistema deve realizar o cadastro (inserção, alteração, exclusão e consultas) dos dados dos produtos.

RF4: O sistema deve realizar o cadastro (inserção, alteração, exclusão e consultas) dos dados dos funcionários.

RF5: O sistema deve permitir que um funcionário realize a venda de um ou mais produtos para um cliente.

RF6: O sistema deve permitir que um funcionário realize a entrada de novas quantidades de produtos no sistema.

RF7: O sistema deve emitir Nota Fiscal Eletrônica dos produtos vendidos ao cliente.

RF8: O sistema deve permitir a geração de relatório de vendas diárias ou por período dos produtos.

Requisitos não funcionais:

RNF1: O sistema deve possuir uma interface gráfica de fácil uso e intuitiva.

RNF2: O sistema deve realizar backup automático diário.

RNF3: O sistema só deve permitir o acesso de funcionários cadastrados.

RNF4: O sistema deve estar de acordo com as leis estaduais para emissão de notas fiscais.

Dessa forma você resolveu a situação-problema determinando os requisitos funcionais e não funcionais do sistema e conhecendo quais técnicas de elicitação de requisitos podem ser aplicadas.

Avançando na prática

Identificando os requisitos

Descrição da situação-problema

Um analista de sistemas está tendo dificuldades para determinar os requisitos funcionais e não funcionais para um sistema de vendas. Um dos erros mais comuns, e que está sendo cometido pelo analista, é confundir os requisitos funcionais com os requisitos não funcionais. Dessa forma, você recebeu a lista de requisitos a seguir para um sistema de vendas e deve auxiliar o analista de sistemas a determinar os requisitos funcionais e não funcionais:

Requisitos para o sistema proposto:

O sistema deve...

1. Ser acessível a usuários da web.
2. Incluir o logotipo e o esquema de cor-padrão da companhia.
3. Restringir o acesso às informações de rentabilidade.
4. Incluir informações sobre custo real e o previsto em orçamento.
5. Fornecer relatórios de gerenciamento.
6. Incluir informações sobre vendas que sejam atualizadas pelo menos uma vez por dia.
7. Apresentar tempo de resposta máximo de dois segundos para consultas predefinidas e tempo de resposta máximo de 10 minutos para consultas não programadas (ad hoc).
8. Incluir informações de todas as subsidiárias da companhia.
9. Imprimir os relatórios das subsidiárias em seu idioma principal.
10. Fornecer classificação mensal de desempenho do pessoal de vendas.

Quais são os requisitos funcionais do negócio?

Quais são os requisitos não funcionais do negócio? Que tipos de requisitos não funcionais são esses?

Resolução da situação-problema

Requisitos funcionais do problema:

RF1. O sistema deve incluir o logotipo e o esquema de cor-padrão da companhia.

RF2. O sistema deve incluir informações sobre custo real e o previsto em orçamento.

RF3. O sistema deve fornecer relatórios de gerenciamento.

RF4. O sistema deve incluir informações sobre vendas que sejam atualizadas pelo menos uma vez por dia.

RF5. O sistema deve incluir informações de todas as subsidiárias da companhia.

RF6. O sistema deve imprimir os relatórios das subsidiárias em seu idioma principal.

RF7. O sistema deve fornecer classificação mensal de desempenho do pessoal de vendas.

Requisitos não funcionais do problema:

RNF1: O sistema deve ser acessível a usuários da web.

RNF2: O sistema deve restringir acesso às informações de rentabilidade.

RNF3: O sistema deve apresentar tempo de resposta máximo de dois segundos para consultas predefinidas e tempo de resposta máximo de 10 minutos para consultas não programadas (*ad hoc*).

Faça valer a pena

1. A determinação dos requisitos é utilizada para transformar a declaração de alto nível dos requisitos em uma lista mais detalhada e precisa do que o sistema deve fazer para fornecer valor necessário ao negócio, sendo uma das mais importantes etapas do processo de desenvolvimento de software.

Para a técnica de elicitação de requisitos, duas categorias de requisitos são utilizadas para descrever o sistema, uma se preocupando com os requisitos de funcionalidade do sistema e a outra com as restrições do sistema. Assinale a alternativa que apresenta, corretamente, os dois tipos de classificação dos requisitos.

a) Requisitos funcionais e requisitos não funcionais.
b) Requisitos funcionais e requisitos não aplicáveis.
c) Requisitos obrigatórios e requisitos não obrigatórios.
d) Requisitos obrigatórios e requisitos não aplicáveis.
e) Requisitos funcionais e requisitos não relevantes.

2. Requisitos são declarações do que o sistema deve fazer e quais características ele deve possuir. No processo de projeto de sistemas, são criados requisitos descrevendo as necessidades do negócio (requisitos do negócio), o que os usuários devem fazer (requisitos dos usuários), o que o software deve fazer (requisitos funcionais) e características que o sistema deve ter (requisitos não funcionais).

Assinale a alternativa que apresenta, corretamente, as técnicas para levantamento de requisitos:

a) Entrevista, JAD, questionários, análise de documentos e observação.
b) Entrevista, JAD, questionários, análise de documentos e supervisão.
c) Entrevista, JAP, questionários, análise de documentos e observação.
d) Entrevista, JAP, questionários, análise de documentos e supervisão.
e) Entrevista, JAD, questionários, análise de requisitos e observação.

3. A determinação dos requisitos é utilizada para transformar a declaração de alto nível dos requisitos em uma lista mais detalhada e precisa do que o sistema deve fazer para fornecer valor necessário ao negócio, sendo uma das mais importantes etapas do processo de desenvolvimento de software.

Assinale a alternativa que apresenta, corretamente, os três tipos perguntas utilizadas em uma entrevista?
a) Diretas, indiretas e exploratórias.
b) Diretas, informais e exploratórias.
c) Fechadas, abertas e exploratórias.
d) Fechadas, abertas e informais.
e) Fechadas, abertas e indiretas.

Seção 1.3

Projeto de sistemas orientado a objetos

Diálogo aberto

Prezados alunos, agora que você já realizou a determinação dos requisitos, utilizando as principais técnicas de elicitação de requisitos e obtendo a lista de requisitos funcionais e não funcionais, daremos continuidade ao nosso projeto, iniciando o projeto orientado a objetos, pensando em termos de classes e objetos. Vocês já verificaram a quantidade de projetos que falham por falta de projeto? Já analisaram que o custo de reparo de projetos na fase de implementação ou testes chega a ser dez vezes mais caro do que na fase de projeto? Por que muitas organizações ignoram a fase de projeto? Essas são questões que devem ser avaliadas por vocês para uma boa condução do projeto.

Dando continuidade ao seu trabalho de projeto de sistema para controlar as atividades (controle de estoque, contas a pagar e receber) da padaria e confeitaria, quais são as características de um projeto orientado a objetos? Quais são as prováveis classes e atributos para este sistema? Como modelar projetos orientados a objetos? Como elaborar um diagrama de casos de uso inicial do projeto? Qual é o propósito da UML (*Unified Modeling Language*) e quais diagramas são utilizados para modelagem?

Nesta seção vamos estudar o levantamento das características de projetos orientados a objetos, em termos de classes e objetos, as principais técnicas para modelagem de projetos orientado a objetos e UML e os principais diagramas para modelagem. Para responder a essas perguntas, realize o levantamento das características de projetos orientados a objetos, em termos de classes e objetos. Em seguida, verifique quais técnicas são destinadas para modelar projetos orientados a objetos e, para finalizar, o que faz a UML e quais são os principais diagramas para modelagem. A partir da compreensão desses temas, você será capaz de responder as questões levantadas anteriormente.

Certamente esses assuntos serão essenciais para sua formação, enquanto analista e desenvolvedor de sistemas, a fim de conhecer o paradigma de programação orientado a objetos e o processo de modelagem com a UML. Bons estudos.

Não pode faltar

Projeto orientado a objetos

De acordo com Schach (2010), a análise orientada a objetos tem por objetivo auxiliar o desenvolvimento para programação orientada a objetos, utilizada em linguagens como Java e C#.

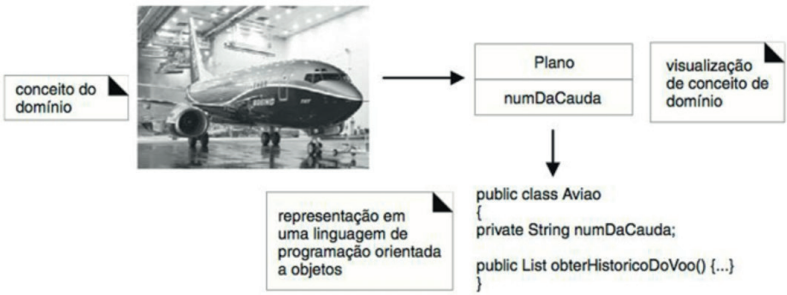
Um paradigma é um modelo a ser utilizado. O paradigma de desenvolvimento faz uso de métodos e técnicas que utilizam os mesmos princípios. No paradigma Orientado a Objetos, utiliza-se o conceito do mundo real que é composto por objetos, que combinam dados e funções, e os problemas são mapeados para objetos que estão relacionados e interagem entre si.

Como benefícios da orientação a objetos, podemos considerar: uma melhor comunicação analista e especialista, melhora do processo de análise, representação padronizada para análise e projeto e apoio a reutilização de código.

Segundo Larman (2007), na análise orientada a objetos busca-se determinar a descrição de objetos do domínio do problema, tal como em um sistema de informação da área da aviação, cujos conceitos incluem avião, voo e piloto.

No projeto orientado a objetos, o foco é a definição de objetos de software e como são utilizados em conformidade com os requisitos, por exemplo: um objeto avião pode ter um atributo (característica do objeto) *numDaCauda* e um método (funcionalidades do objeto) obterHistoricoDoVoo. Durante a implementação orientada a objetos, os objetos são implementados em uma linguagem orientada a objetos, como a linguagem Java, criando assim uma classe (representação de um agrupamento de objetos) Avião. A Figura 1.4 ilustra como a orientação a objetos trabalha com a representação de objetos.

Figura 1.4 | Orientação a objetos e representação de objetos



Fonte: Larman (2007, p. 35).

Na Figura 1.4 é apresentado o exemplo de um conceito do domínio, que é a referência a um avião. Esse conceito é visualizado através da representação de uma classe da UML (*Unified Modeling Language*). Por fim, o conceito de domínio é representado em uma linguagem de programação orientada a objetos, no caso a classe Aviao, com o atributo (característica) numDaCauda e o método (propriedade) obterHistoricoDoVoo().

Classes e objetos

De acordo com Schach (2010), classes são definidas com um agrupamento e descrição de um conjunto de objetos. Os objetos pertencem a uma determinada classe, isto é, são instâncias de classes, que descrevem as características e propriedades de um objeto. Objetos são instâncias de apenas uma classe. As classes são usadas como classificadoras de objetos do mundo real. As classes descrevem objetos de qualquer tipo de sistema.

Identificar as classes de um sistema não é tarefa fácil, por isso faz-se necessário um domínio do problema a que se destina o modelo de software. As classes são derivadas do domínio do problema e nomeadas de acordo com o que elas representam no sistema. Ao definir as classes de um sistema, algumas questões podem ajudar a identificá-las.

Quais informações serão armazenadas ou analisadas? Se há informações que devem ser guardadas e processadas, provavelmente elas são candidatas para serem uma classe. Há

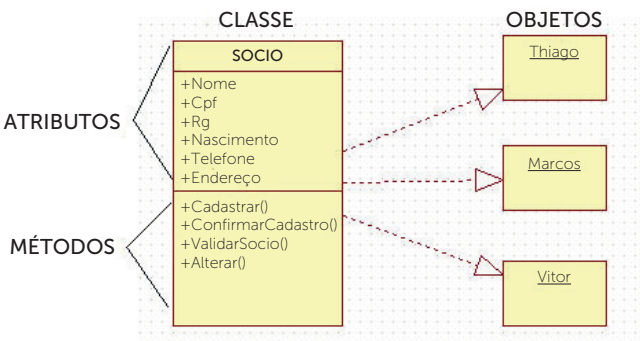
classes, bibliotecas ou componentes que vão ser utilizados pelo sistema? Caso a resposta seja positiva, essas classes, componentes e modelos podem conter classes.

Segundo Schach (2010), uma classe é constituída de dois grupos de elementos: atributos e métodos. Um atributo (*atribute*) de uma classe é uma característica pertencente a esta classe, utilizada para armazenar alguma informação que está associada a classe. Por exemplo, uma classe cliente possui como atributos Nome e Idade.

Enquanto os atributos armazenam dados de objetos, os métodos (*methods*) são utilizados para realizar operações sobre os atributos de uma classe. A Figura 1.5 ilustra o conceito de classe e objeto.

Objetos são entidades independentes e dois objetos, mesmo que possuam os mesmos atributos, não são os mesmos objetos. Na computação, podemos fazer uma analogia com a memória do computador: dois objetos, mesmo que possuam atributos iguais, estando em partes diferentes da memória, são diferentes.

Figura 1.5 | Classe e objeto



Fonte: <http://theclub.com.br/Restrito/Revistas/201208/ling1208.aspx>. Acesso em: 2 maio 2018.

Assimile

A orientação diminui a distância que existe entre o mundo real e o computacional.

O sistema é organizado em classes de objetos, que inicia com a análise de requisitos, seguida do projeto do software, finalizando com a implementação.

Desde a concepção de um sistema há relação direta entre atributos e operações.

Segundo Schach (2010), a maioria dos principais avanços de paradigma de orientação a objetos foi feita entre 1990 e 1995. Na década de 1960, quando os computadores começaram a ser amplamente usados em aplicações comerciais, o hardware era muito mais caro do que hoje em dia. Como resultado, a vasta maioria dos produtos de software daquela época representava as datas usando apenas os dois últimos dígitos de um ano; o 19 que vinha à frente ficava subentendido. O problema com esse esquema é que o ano 00 seria interpretado como 1900, e não 2000.

Quando o hardware se tornou mais barato nas décadas de 1970 e 1980, poucos gerentes viam algum motivo para gastar enormes quantias reescrevendo produtos de software existentes para incorporar datas com quatro dígitos. Afinal, quando o ano 2000 chegasse, seria um problema para outra pessoa resolver. Como consequência, sistemas legados não estavam preparados para a questão do ano 2000. Entretanto, à medida que a data fatal de 1º de janeiro de 2000 se aproximava, as empresas desenvolvedoras de software foram forçadas a trabalhar contra o relógio para corrigir seus produtos; não havia mais jeito de postergar o problema do ano 2000.

Reflita

Entre os problemas enfrentados pelos programadores de manutenção estavam a falta de documentação para muitos produtos de software legados, bem como produtos escritos em linguagens de programação que ficaram obsoletas. Quando era impossível modificar um produto de software existente, a única alternativa era reiniciar da estaca zero. Algumas empresas decidiram usar software de prateleira. Outros acharam que eram necessários novos produtos de software personalizados. Por razões óbvias, os gerentes queriam que esses fossem desenvolvidos por meio de tecnologia moderna, que já havia demonstrado ser eficaz em termos de custo, e isso significava usar o paradigma de orientação a objetos. O bug do milênio foi, portanto, um importante catalisador para a larga aceitação do paradigma de orientação a objetos. Quais seriam as vantagens e desvantagens de softwares de prateleira e sistemas personalizados?

Modelagem de projetos orientados a objetos

De acordo com Fowler (2005), a UML é um conjunto de representações gráficas que auxiliam na descrição e no projeto de sistemas de softwares construídos através da orientação a objetos.

A UML é a combinação de várias linguagens de modelagem orientada a objetos, que surgiram no final da década de 1980 e início da década de 1990. Ela incorporou a ideia de vários autores, tendo destaque: James Rumbaugh com o método de modelagem OMT (*Object-Modeling Techinique*); Ivar Jacobson com o método OOSE (*Object-Oriented Software Engineering*); e Grady Booch com o método de Booch Method (FOWLER, 2005).

Segundo Larman (2007), a UML começou com um esforço de Booch e Rumbaugh, em 1994, não apenas com o intuito de criar uma notação comum, mas de combinar seus dois métodos. Dessa forma surgiu o Método Unificado (*Unified Method*). Ivar Jacobson juntou-se a Booch e Rumbaugh na Rational Corporation e formaram um grupo, decidindo reduzir o escopo de seus esforços e focar em uma notação comum de diagramação, a UML. O OMG (*Object Management Group*), um grupo de padrões da indústria para tratar padrões relacionados a Orientação a Objetos, foi convencido de que um padrão aberto era necessário. Assim, o processo abriu-se e levou à UML 1.0 em 1997 e emergiu como notação de diagramação padrão para modelagem orientada a objetos e tem continuado a ser refinada em novas versões UML.

A UML é utilizada nas etapas de especificação dos requisitos, elaboração e documentação dos sistemas. A UML além de ser uma padronização de uma notação unificada, incorpora conceitos diferentes de outros métodos orientados a objetos. Foram unificadas visões de diversos tipos de sistemas e fases de desenvolvimento, permitindo que determinados projetos pudessem ser realizados, o que antes não era possível pelos métodos existentes.

UML e diagramas para modelagem

De acordo com Larman (2007), um diagrama é uma representação de um modelo. Dentre os diagramas da UML destacam-se:

- Visão estática: diagramas de casos de uso e classe.
- Visão dinâmica: diagramas de sequência, estados e atividades.

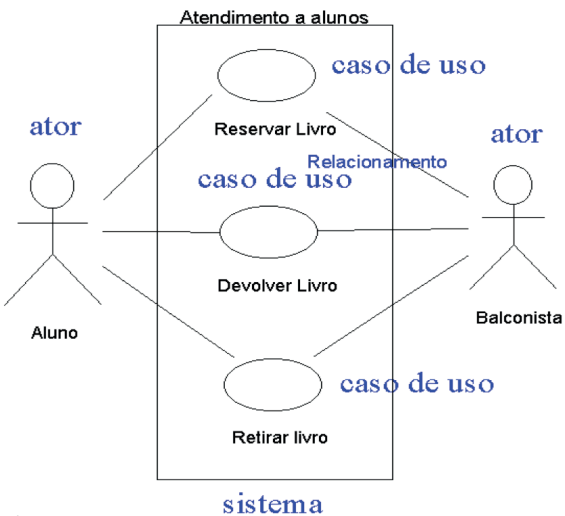
Exemplificando

A análise de requisitos tem por objetivo determinar o que os clientes necessitam de um produto de software. Algumas práticas são úteis neste processo:

- Casos de uso, que descreve as funcionalidades do sistema.
- Um diagrama de classes, representando as classes do sistema, obtido a partir da visão conceitual do sistema.
- Um diagrama de atividades pode mostrar o contexto dos casos de uso e também os detalhes sobre como um caso de uso complicado funciona.
- Um diagrama de estados que apresenta estados e eventos de um único objeto reativo.

O diagrama de casos de uso obtém as funcionalidades do sistema pela visão do usuário. Criado no início do desenvolvimento, tem por objetivo especificar a funcionalidade do sistema, os requisitos, avaliar a arquitetura do sistema e auxiliar a implementação e os casos de teste. A Figura 1.6 ilustra um diagrama de casos de uso do atendimento a alunos em uma biblioteca.

Figura 1.6 | Diagrama de casos de uso de sistema de biblioteca



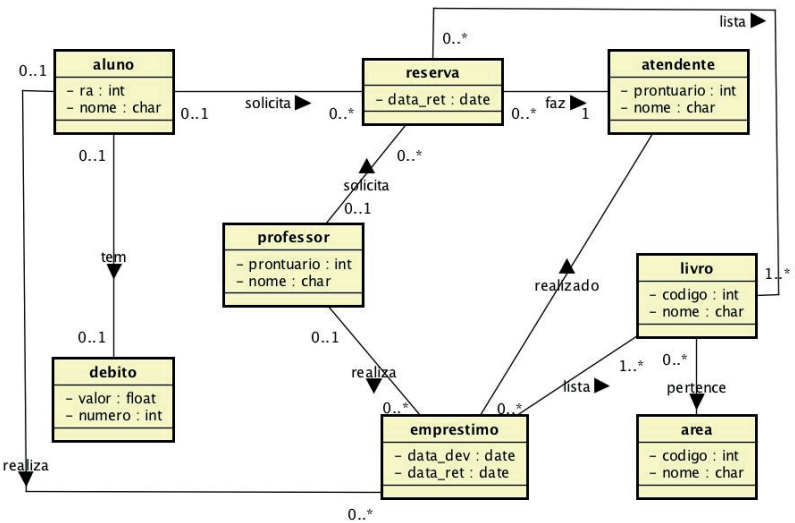
Fonte: elaborada pelo autor.

O diagrama de casos de uso é composto de alguns elementos principais:

- Ator: representado pelo boneco, o ator representa um usuário do sistema e também pode representar outro sistema. No exemplo da Figura 1.6, os atores são os alunos, que fazem reservas, devoluções e retiradas de livros, e os balconistas, que atendem a essas solicitações dos alunos.
- Casos de uso: representado pela elipse, o caso de uso representa uma ação realizada no sistema. No exemplo da Figura 1.6, os casos de uso são reservar livro, devolver livro e retirar livro.
- Relacionamento: os relacionamentos são representados por setas que ligam os atores a casos de uso e também liga casos de uso. No exemplo da Figura 1.6, os relacionamentos ocorrem entre o ator aluno e os casos de uso reservar livro, devolver livro e retirar livro. O ator balconista também se relaciona com esses casos de uso.

O diagrama de classes captura o vocabulário do sistema, tendo por objetivo nomear e modelar conceitos dentro do sistema, especificar colaborações e especificar esquemas lógicos de banco de dados. A Figura 1.7 ilustra o diagrama de classe para sistema de reserva de biblioteca.

Figura 1.7 | Diagrama de classe para sistema reserva de biblioteca



Fonte: elaborada pelo autor.

O diagrama de classes é composto de alguns elementos principais:

- Classes: representam uma sociedade de objetos com características e propriedades que os definem. Uma classe é representada por um retângulo dividido em três compartimentos: nome da classe, atributos (características dos objetos da classe) e métodos (ações realizadas pela classe). A Figura 1.7 ilustra as classes aluno (com atributos RA e nome), professor (com atributos prontuário e nome), débito (com atributos valor e número), reserva (com atributo data_retirada), empréstimo (com atributos data_retirada e data_devolução), atendente (com atributos prontuário e nome), livro (com atributos código e nome) e área (com atributos código e nome).
- Relacionamentos: as classes se relacionam entre si, representando o relacionamento dos objetos das classes. Um relacionamento é representado por uma linha que conecta as classes. Os relacionamentos apresentam rótulos, que são informações textuais para nomear o relacionamento e apresentam multiplicidades, que representam quantos objetos de uma classe se relacionam com objetos de outra classe. No exemplo da Figura 1.7, um aluno solicita zero ou muitas reservas (0..*), e uma reserva é solicitada por zero ou um aluno (0..1).

Pesquise mais

Para uma melhor compreensão da metodologia orientada a objetos e da linguagem de modelagem UML e seus diagramas, faça a leitura do artigo:

CASTRO, F. R.; DA CRUZ, F. M.; ODDONE, N. E. O paradigma da orientação a objetos, a linguagem unificada de modelagem (UML) e a organização e representação do conhecimento: um estudo de caso de um sistema para bibliotecas. **Inf. Inf.**, Londrina, v. 18, n. 1, p. 82, jan./abr. 2013. Disponível em: <http://www.uel.br/revistas/uel/index.php/informacao/article/view/9547/pdf>. Acesso em: 14 jun. 2018.

Os diagramas de casos de uso e diagramas de classes serão abordados em mais detalhes nas próximas seções. Além destes diagramas, a UML também possui o diagrama de sequência, que obtém os aspectos dinâmico (por tempo), tendo por objetivo modelar o fluxo de controle e ilustrar cenários típicos. O diagrama de estados captura o comportamento dinâmico (orientado por eventos). O diagrama de atividades obtém aspecto dinâmico (orientado por atividades) com objetivo de modelar fluxos de negócios e modelar operações.

Dessa forma, conhecemos sobre projeto orientado a objetos, classes e objetos e UML e os principais diagramas da UML, tais como diagrama de casos de uso, classes, sequência, atividades e estados. Esses conceitos são de fundamental importância para nosso estudo e para sua formação acadêmica e profissional.

Sem medo de errar

Chegou a hora de resolver nosso desafio. Como os irmãos não entendem muito de tecnologia, você foi contratado para auxiliá-los na implantação de um projeto de sistema para controlar as atividades da padaria e confeitaria. Quais são as características de um projeto orientado a objetos? Como modelar projetos orientados a objetos? Qual é o propósito da UML e quais diagramas são utilizados para modelagem?

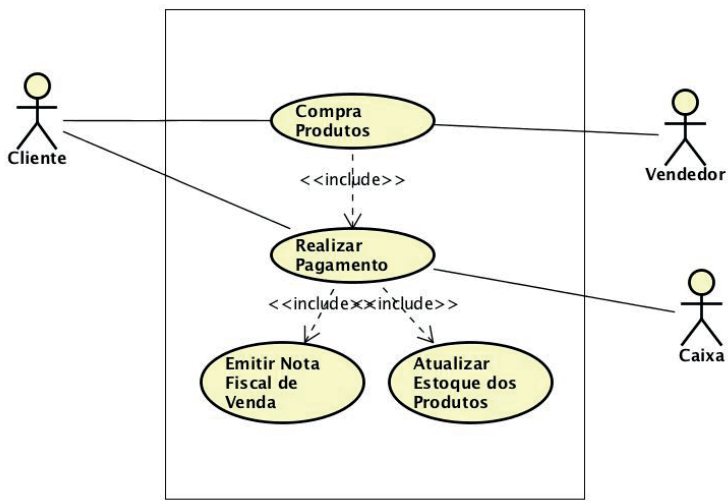
Para responder essas questões, devemos primeiramente conhecer as características de projetos orientados a objetos. Na análise orientada a objetos, o objetivo é obter objetos do domínio do problema, e no projeto orientado a objetos, obter objetos de software. Em projetos orientados a objetos, falamos de classes e objetos. Uma classe é a descrição de um tipo de objeto.

Para o projeto em questão, algumas classes e atributos que devem fazer parte são: classe Cliente (com atributos Código do Cliente, Nome, Endereço, RG, CPF, Telefone e Celular), Produto (Código do Produto, Nome, Data de Fabricação, Data de Validade, Peso), Fornecedor (Código do Fornecedor, Nome, Endereço, CNPJ, Telefone e Celular), Vendedor (Código do Vendedor, Nome, Endereço, RG, CPF, Telefone e Celular).

Em seguida, para modelar projetos orientados a objetos, deve-se conhecer a UML, que é uma família de notações gráficas que auxiliam na descrição e no projeto de sistemas de software construídos através da orientação a objetos. A UML incorpora a ideia de vários autores, dos quais se destacam: James Rumbaugh com o método de modelagem OMT (*Object-Modeling Techinique*); Ivar Jacobson com o método OOSE (*Object-Oriented Software Engineering*); e Grady Booch com o método de Booch Method.

Para o projeto em questão, você pode começar a elaborar o diagrama de casos de uso inicial, com os atores: cliente e vendedor, com o cenário, por exemplo, venda de produto e baixa no estoque. A Figura 1.8 ilustra esse diagrama de casos de uso inicial.

Figura 1.8 | Diagrama de caso de uso da compra de produtos



Fonte: elaborada pelo autor.

Para finalizar, deve-se verificar qual é o propósito da UML e quais diagramas são utilizados para modelagem. A UML é uma linguagem utilizada para especificação e documentação dos sistemas e combina: conceitos de modelagem de dados (diagramas entidade-relacionamento), modelagem de negócios (fluxo de trabalhos), modelagem de objetos e modelagem de componentes. Dentre os diagramas da UML destacam-se:

- Visão estática: diagramas de casos de uso e classe.
- Visão dinâmica: diagramas de sequência, estados e atividades.

Dessa forma, você conheceu as características de um projeto orientado a objetos, as formas de modelar projetos orientados a objetos e o propósito da UML e seus diagramas para modelagem, iniciando seus conhecimentos de projeto e modelagem orientados a objetos.

Avançando na prática

Utilizando a UML

Descrição da situação-problema

Um jovem analista de sistemas realizou o levantamento de requisitos de um sistema para uma loja de conveniência. De posse dos requisitos funcionais e não funcionais, o analista optou por realizar o desenvolvimento de um projeto orientado a objetos, porém está com dificuldades para saber quais diagramas da UML utilizar para determinadas necessidades. Dessa forma, faça a análise de quais diagramas da UML devem ser utilizados para modelar as seguintes necessidades:

1. Funcionalidades do sistema.
2. Classes e relacionamento das classes.
3. Comportamento dinâmico (por tempo).
4. Comportamento dinâmico (orientado por eventos), modelando o ciclo de vida do projeto e objetos reativos.
5. Comportamento dinâmico (por atividades), modelando fluxos de negócios e operações.

Resolução da situação-problema

Analisando as necessidades que precisam ser modeladas com os diagramas da UML, os seguintes diagramas da UML devem ser utilizados:

1. Funcionalidades do sistema: modelagem usando diagrama de casos de uso.

2. Classes e relacionamento das classes: modelagem usando diagrama de classes.
3. Comportamento dinâmico (por tempo): modelagem usando diagrama de sequência.
4. Comportamento dinâmico (orientado por eventos): modelagem usando diagrama de estados.
5. Comportamento dinâmico (por atividades): modelagem usando diagrama de atividades.

Faça valer a pena

1. No projeto orientado a objetos, há uma ênfase em definir objetos de software e sua conformidade aos requisitos. Como benefícios da orientação a objetos podemos considerar uma representação básica consistente para análise e projeto e apoio à reutilização de código.

Assinale a alternativa que apresenta, corretamente, os dois grupos de elementos que uma classe possui.

a) Atributos e relacionamentos.
b) Atributos e métodos.
c) Atributos e objetos.
d) Objetos e métodos.
e) Relacionamentos e métodos.

2. A UML (*Unified Modeling Language*) é um conjunto de representações gráficas que auxiliam na descrição e no projeto de sistemas de softwares construídos através da orientação a objetos, tendo por objetivo padronizar uma notação de representação.

Assinale a alternativa que apresenta, corretamente, as finalidades da UML:

a) Especificar, construir e documentar artefatos de sistemas.
b) Especificar, codificar e documentar artefatos de sistemas.
c) Codificar, construir e documentar artefatos de sistemas.
d) Especificar, testar e documentar artefatos de sistemas.
e) Codificar, testar e documentar artefatos de sistemas.

3. A UML (*Unified Modeling Language*) é um conjunto de representações gráficas que auxiliam na descrição e no projeto de sistemas de softwares

construídos através da orientação a objetos, tendo por objetivo padronizar uma notação de representação.

Considere a seguinte definição: captura a funcionalidade do sistema conforme a visão do usuário, sendo construído nos estágios iniciais do desenvolvimento, e tem por objetivo especificar o contexto de um sistema, capturar os requisitos de um sistema, validar a arquitetura do sistema e direcionar a implementação e gerar casos de teste.
Qual diagrama da UML apresenta as características descritas?

a) Diagrama de casos de uso.
b) Diagrama de classes.
c) Diagrama de sequência.
d) Diagrama de estados.
e) Diagrama de atividades.

Referências

DENNIS, A.; WIXOM, B. H.; ROTH, R. M. **Análise e projeto de sistemas**. 5. ed. Rio de Janeiro: LTC, 2014.

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

LARMAN, C. **Utilizando UML e padrões**: uma introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

PADUA FILHO, W. P. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. Rio de Janeiro: LTC, 2009.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

SCHACH, S. R. **Engenharia de software**: os paradigmas clássico e orientado a objetos. 7. ed. Porto Alegre: AMGH, 2010.

Modelagem da análise para projeto – diagrama de casos de uso

Convite ao estudo

Prezado aluno, já em reparou como os sistemas comerciais online ganharam espaço nos últimos anos? Muitos comércios físicos foram obrigados a migrar seu negócio para o mundo digital a fim de se manterem competitivos no mercado. Agora pense em um desses sistemas, vários usuários acessando produtos e serviços ao mesmo tempo e, muitas vezes, tentando adquirir o mesmo objeto simultaneamente. Cenário complexo, não acha? Os sistemas computacionais têm se tornado cada vez mais complexos, o que implica diretamente no trabalho do projetista desses sistemas, pois, caso exista alguma falha, os prejuízos podem ser irreversíveis. Na construção de uma solução em software, após o levantamento de requisitos, o próximo passo consiste em modelar o sistema, do ponto de vista de suas funcionalidades e, nesse momento, uma importante metodologia que pode ser empregada são os casos de uso e sua representação gráfica, assuntos centrais dessa unidade. Esse conhecimento lhe permitirá modelar sistemas de forma clara através dos diagramas de casos de uso, que podem ser construídos a partir de ferramentas específicas.

Uma loja voltada ao público de colecionadores está se instalando em sua cidade. Ela trabalha com a comercialização de colecionáveis, revistas em quadrinhos, acessórios de arte e decoração. Os proprietários possuem um conhecimento limitado em computação, mas sabem que é necessário adquirir um sistema para gerenciar o estabelecimento e controlar o estoque, as contas a pagar e receber, clientes e fornecedores. Você foi contratado para auxiliar os proprietários na etapa de modelagem da análise para o projeto do sistema. Como

realizar essa modelagem? Como modelar as funcionalidades do sistema? Os casos de uso e os diagramas de casos de uso são os modelos mais indicados nessa atividade? Existem outras alternativas para modelagem das funcionalidades do sistema? Como utilizar casos de uso e diagramas de casos de uso para modelagem do sistema? Você vai trabalhar esta e outras questões nesta Unidade de Ensino 2.

Para cumprir sua missão, você aprenderá o que são os casos de uso, bem como representá-los graficamente. Após a construção de um diagrama, é preciso documentá-lo e fazer a validação com os requisitos, sempre buscando otimizar os processos do sistema. Você aprenderá sobre a consistência do diagrama de casos de uso e como realizar a interação do diagrama de sequência nos casos de uso.

Bom trabalho!

Seção 2.1

Modelagem de análise de projetos

Diálogo aberto

Prezado aluno, já observou a construção de um prédio ou alguma outra grande obra de construção civil? Podemos fazer uma analogia entre a construção de um prédio e de um software. Primeiro, é preciso entender o que o cliente quer (levantamento de requisitos). Com essas informações em mãos, o arquiteto cria o projeto, que apresenta a "forma" da obra e que será usado tanto para validação com o cliente, como para orientar o processo de construção que será realizado por outras equipes. Assim como na construção civil, o projeto é uma parte primordial no desenvolvimento de um software, pois ele definirá o escopo do sistema, apresentando as funcionalidades que o sistema terá. Dentro desse contexto, uma das formas de estruturar esse conjunto de ações é através da modelagem dos casos de uso, que será abordada nessa seção.

Você foi contratado para modelar um sistema para uma loja que comercializa objetos colecionáveis. O sistema deverá fazer o gerenciamento do estabelecimento e o controle do estoque, das contas a pagar e a receber, bem como de clientes e fornecedores. Como deve ser feita a modelagem desse sistema utilizando os casos de uso? É mais indicado modelar todas as funcionalidades do sistema ou somente as mais importantes? Existem ferramentas para auxiliar esse processo? Existem outras formas de modelagem, além dos casos de uso? Crie um documento com pelo menos dois casos de uso do sistema a ser desenvolvido em forma textual, bem como sua representação gráfica.

Nessa seção você aprenderá o que é um caso de uso e como realizar a passagem da modelagem de análise para projeto usando o diagrama de casos de uso.

A modelagem do projeto usando casos de uso demanda seu empenho, então, mãos à obra!

Não pode faltar

A construção de um sistema computacional envolve vários profissionais para atender a todas as demandas e especificidades. Para que as diversas funcionalidades de um sistema cumpram suas tarefas da maneira correta e de forma integrada, é preciso que o projeto especifique todas essas funcionalidades e como elas devem ser acionadas. Uma das formas de se criar esse projeto é modelando as funcionalidades através da metodologia de casos de uso ou na sua nomenclatura original, *use cases*.

Na metodologia de casos de uso, o projeto será modelado de acordo com as atividades que ele realizará. Um caso de uso deve ser entendido como um conjunto de atividades que serão realizadas para produzir um resultado final. Para entender melhor essa definição, considere uma locadora de veículos, no contexto em que um usuário pode alugar ou devolver um veículo. Se ele for alugar, o sistema executará uma série de atividades pertinentes a essa ação; caso ela vá devolver, o sistema executará uma outra sequência de atividades. Ele pode ainda devolver com atraso, o que implicaria em outra série de atividades (por exemplo, calcular juros). Segundo Dennis, Wixom e Roth (2014), casos de uso são utilizados para documentar e mostrar uma tarefa requisitada pelo usuário no sistema.

Os casos de uso são muito utilizados na fase de análise de requisitos como forma de descobrir as exigências do usuário e os funcionais. Nesse sentido, eles podem ser usados para o melhor entendimento das necessidades do usuário. Segundo Fowler (2005), casos de uso são técnicas para captar os requisitos funcionais do sistema e são usados para descrever as interações entre usuários e o próprio sistema, apresentando descrição de como o sistema será utilizado.

Os casos de uso originaram como parte da orientação a objetos, mas hoje são empregados como metodologia independente do paradigma de desenvolvimento utilizado. Eles mostram como um sistema interage com o ambiente, apresentando atividades realizadas pelos usuários do sistema e as respostas desse sistema. Outra informação importante a respeito desse tipo de modelagem é que "o caso de uso descreve o que o sistema fará sob o ponto de vista do usuário". (DENNIS; WIXOM; ROTH, 2014, s. p.).

De acordo com Larman (2007), casos de uso são descrições textuais, muito usados para descobrir e registrar requisitos, influenciando muitos aspectos de um projeto, inclusive a análise e o desenvolvimento orientado a objetos. A criação de casos de uso é recomendada quando há a necessidade de compreender melhor a realização das interações exigidas entre o usuário e o sistema. Casos de uso são descrições textuais das ações de um ator em um determinado sistema. Esse ator pode ser tanto um usuário, como um outro sistema ou serviço.

Componentes dos casos de uso

Os casos de uso, de modo geral, são constituídos de alguns componentes essenciais:

- **Ator**: o ator representa algo ou alguém que usa o sistema, podendo ser uma pessoa, um sistema de computador ou uma organização. De acordo com Larman (2007), um ator é algo que possui comportamento, podendo ser o próprio sistema quando invoca serviços de outros sistemas. Há três tipos de atores: (i) ator principal: pessoas que usam diretamente o sistema, por exemplo, cliente e caixa em um sistema de venda; (ii) ator de suporte: oferece serviços para o sistema. Considere como exemplo, um serviço automático para autorizar pagamentos; (iii) ator de bastidor: não é um ator principal, mas tem interesse nos casos de uso, por exemplo um órgão para recolhimento de impostos.
- **Cenário**: sequência de passos que descreve uma interação entre usuário e sistema.

Exemplificando

Para exemplificar, considere o cenário de compra de equipamentos para TI em uma revendedora online. Esse cenário poderia ser descrito textualmente da seguinte forma: "o cliente procura no catálogo de produtos e seleciona as opções e quantidades desejadas. Após concluir a seleção, descreve o endereço de entrega, preenche os dados do pagamento e confirma. O sistema verifica os dados preenchidos e envia um e-mail confirmando a compra".

Esse cenário descreve uma opção do sistema, porém, certamente deve haver outras situações, por exemplo, o usuário escolheu como meio

de pagamento o cartão de crédito e o pagamento não foi autorizado, ou o usuário escolheu como meio de pagamento o boleto e passou a data de pagamento.

- **Caso de uso:** coleção de cenários relacionados, descrevendo um ator utilizando sistema como forma de atingir determinado objetivo. Você viu no exemplo da loja de equipamentos para TI online que podem existir diversos cenários para um mesmo objetivo que, no caso, seria comprar um equipamento. Esse conjunto de cenários utilizados para um único objetivo é o que caracteriza o caso de uso.

Os casos de uso podem ter formatos e formalidades diferentes:

- **Resumido:** resumo do cenário principal. Exemplo: processar venda.
- **Informal:** formato informal de parágrafos cobrindo vários cenários. Exemplo: tratar devoluções.
- **Completo:** contém todos os passos e detalhes.

Exemplificando

Considere o exemplo, descrito por Larman (2007), de casos de uso completo para processo de venda:

Escopo: aplicação de PDV.

Ator: caixa.

Interessados: caixa, vendedor, cliente, empresa, gerente, órgãos fiscais e serviço de autorização de pagamentos.

Garantia de Sucesso: venda salva, impostos corretamente calculados, contabilidade e estoque atualizados, comissões registradas, recibo gerado e autorizações de pagamento registradas.

Cenário Principal:

1) Cliente chega ao PDV (ponto de venda) com produtos e serviços a comprar.

2) Caixa começa nova venda.

3) Caixa insere código do item.

4) Sistema registra o item da venda e apresenta sua descrição, preço e total parcial da venda.

5) Sistema exibe total já com as taxas.

6) Caixa mostra total e solicita pagamento.

7) Cliente faz pagamento e sistema analisa pagamento.

8) Sistema armazena venda completada e transmite dados da venda e pagamentos para sistema externo de contabilidade e sistema de estoque.

9) Sistema apresenta recibo.

Cenário Alternativo:

1) Sistema apresenta falha. Caixa reinicia sistema, sistema restaura estado anterior.

2) Sistema informa que ID do item é inválido. Sistema envia mensagem de erro e não aceita entrada.

3) Cliente solicita ao caixa para remover um item da compra. Caixa insere o código do item a ser removido da venda e sistema remove, recalculando o total.

4) Cliente solicita cancelamento de venda e o caixa faz cancelamento no sistema.

Com o exemplo apresentado, você pode visualizar todos os componentes utilizados para a descrição de um caso de uso. Quanto maior o grau de detalhamento, maior o entendimento sobre a atividade descrita.

Considere algumas dicas, propostas por Larman (2007), para identificação dos casos de uso e atores:

- Casos de uso são definidos para satisfazer os objetivos dos atores. Determine a fronteira do sistema: é uma aplicação de software ou uma pessoa usando o sistema?
- Identifique os atores principais, que são aqueles que têm objetivos satisfeitos por meio do uso de serviços do sistema.
- Identifique os objetivos para cada ator principal.
- Defina casos de uso que satisfaçam objetivos dos usuários, nomeie-os de acordo com o objetivo.
- Algumas perguntas ajudam a identificar atores não tão óbvios:
    - Quem faz uso do sistema?
    - Quem faz administração de usuários e da segurança?
    - Quem é noticiado quando há erros ou falhas?
    - Além dos atores principais humanos, existe algum sistema externo de software ou robótica que solicita os serviços do sistema?

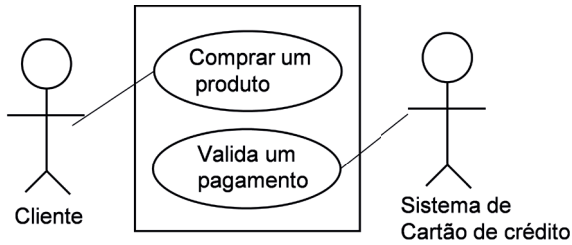
Assimile

Os casos de uso são utilizados para atender às necessidades dos atores. Inicie sempre se perguntando: o que o sistema precisa fazer? Quais funções devem ser realizadas? Dessa forma, você vai identificar com facilidade os casos de uso. Para identificar os atores, faça a seguinte indagação: quem realiza essa função?

Diagramas de casos de uso

Uma das formas mais utilizadas para representação dos casos de uso é através da *Unified Modeling Language* (UML). Dentre os 13 tipos de diagramas que essa família de notações gráficas possui, o de casos de uso é usado para modelar como os usuários interagem com um sistema, por isso ele faz parte do conjunto de diagramas de comportamento da UML. De acordo com Larman (2007), a UML possui elementos gráficos que permitem representar os nomes dos casos de uso, atores e seus relacionamentos. Veja na Figura 2.1 um diagrama de caso de uso com dois atores, um físico (o cliente) e um sistema (de cartão de crédito).

Figura 2.1 | Diagrama de caso de uso

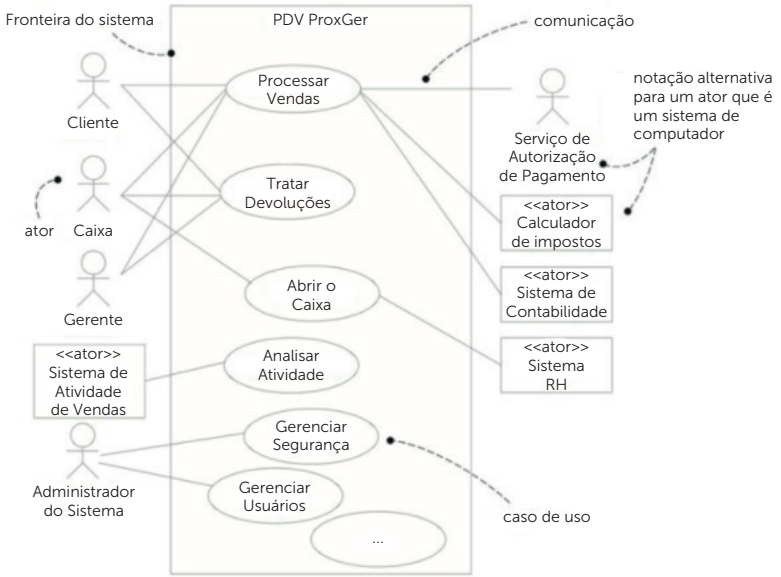


Fonte: elaborada pelo autor.

A Figura 2.2 ilustra um diagrama de casos de uso para um sistema de PDV, com os seguintes componentes:

- Ator: representado pelo boneco, o ator é um usuário do sistema e também pode representar outro sistema. No exemplo da Figura 2.2, os atores são: cliente, caixa, gerente, administrador do sistema e serviço de autorização de pagamento. Alternativamente, um ator pode ser representado por um retângulo, com o estereótipo <<ator>>, representando um ator que é um sistema de computador. São exemplos os sistemas de atividades de venda, calculador de impostos, sistema de contabilidade e sistema RH.
- Casos de uso: representado pela elipse, o caso de uso é uma ação realizada no sistema. No exemplo da Figura 2.2, os casos de uso são processar vendas, tratar devoluções, abrir o caixa, analisar atividade, gerenciar segurança e gerenciar usuários.
- Relacionamento: os relacionamentos são representados por setas que ligam os atores a casos de uso e também liga casos de uso. No exemplo da Figura 2.2, os relacionamentos ocorrem entre os atores cliente e casos de uso processar venda e tratar devoluções. O ator caixa se relaciona com processar vendas, tratar devoluções e abrir caixa. O ator gerente se relaciona com processar vendas e tratar devoluções. O ator serviço de autorização de pagamento se relaciona com processar vendas. O ator calcular impostos e sistema de contabilidade se relacionam com processar vendas. O ator sistema RH se relaciona com abrir caixa. O ator sistema de atividade de vendas se relaciona com analisar atividade e o ator administrador do sistema se relaciona com gerenciar segurança e gerenciar usuários.

Figura 2.2 | Diagrama de casos de uso para um sistema de PDV



Fonte: Larman (2007, p. 116).

Na Figura 2.2, os atores são representados por "bonecos", tais como, Cliente, Caixa, Gerente e Administrador do Sistema. Atores também podem ser representados alternativamente com retângulos, com estereótipo de <<ator>>, indicando que se trata de um sistema de computador. Os casos de uso são representados por elipses, tais como, Processar Vendas, Tratar Devoluções, Abrir o Caixa, Analisar Atividade, Gerenciar Segurança e Gerenciar Usuários. Os casos de uso representam ações e devem ser nomeados com verbos. Os relacionamentos entre atores e casos de uso e entre casos de uso são representados por uma linha que os une. O retângulo utilizado para "armazenar" os casos de uso representa o limite do sistema a ser desenvolvido.

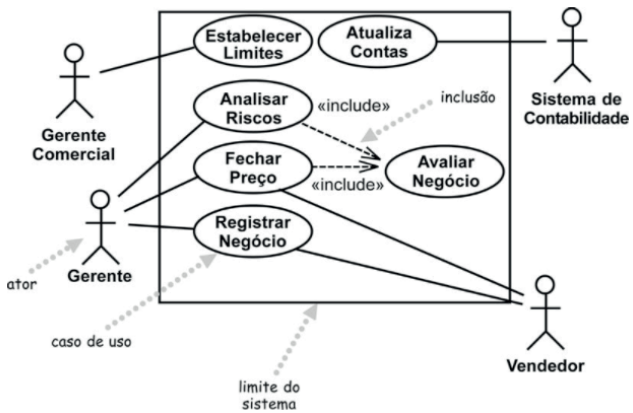
Um diagrama de caso de uso apresenta o contexto do sistema, isto é, a fronteira de um sistema, o que está fora dele e como é usado. Ele é utilizado como mecanismo de comunicação do comportamento do sistema e seus atores. Todos os atores e casos de uso obtidos na etapa de análise devem ser modelados para o diagrama de casos de uso com sua sintaxe, conforme estudamos anteriormente. É muito importante que casos de uso e diagramas de casos de uso estejam corretamente alinhados e representem as mesmas informações.

Reflita

Considerando casos de uso e diagramas de casos de uso, qual apresenta uma maior importância no trabalho de definição de casos de uso?

A Figura 2.3 ilustra outro exemplo de diagrama de casos de uso, utilizando relacionamentos do tipo inclusão <<include>>. Relacionamentos com estereótipos desse tipo indicam que o caso de uso de onde a seta parte inclui obrigatoriamente as ações do caso de uso de destino.

Figura 2.3 | Diagrama de casos de uso



Fonte: Fowler (2005, p. 107).

Na Figura 2.3 são ilustrados os atores Gerente Comercial, Gerente, Sistema de Contabilidade e Vendedor. Os casos de uso apresentam um caso particular de relacionamento do tipo <<include>>, que indica, por exemplo, que o Gerente, ao Analisar Riscos, deve obrigatoriamente incorporar o Avaliar o Negócio. O mesmo vale para Fechar Preço, ação que deve, obrigatoriamente, incorporar a atividade de Avaliar o Negócio. Neste diagrama de casos de uso são retratadas atividades de gestão comercial, retratando as atividades relacionadas no processo de um determinado negócio. O gerente comercial estabelece os limites (metas) e o gerente analisa riscos e fecha preços, que incluem avaliar o negócio. O gerente também registra o negócio que é acompanhado pelo vendedor.

Pesquise mais

Para um melhor conhecimento sobre casos de uso e diagramas de casos de uso, realize a leitura do material da Profa. Dra. Elisa Yumi Nakagawa, da USP (Universidade de São Paulo).

NAKAGAWA, E. Y. Casos de Uso e Diagrama de Casos de Uso. USP (Universidade de São Paulo), [s.d.]. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/3720765/course/section/857581/Aula02_CasosDeUso.pdf>. Acesso em: 20 jul. 2018.

Desta forma, você compreendeu o que são casos de uso e seus componentes e o uso do diagrama de casos de uso da UML.

Sem medo de errar

Você foi contratado pelos proprietários da loja de colecionáveis para fazer a modelagem da análise para o projeto do sistema. Para realizar essa tarefa usando casos de uso, você primeiramente deve pensar nas funcionalidades do sistema (casos de uso) e em como os usuários interagem com o ele (atores). Deve-se modelar as funcionalidades mais importantes do sistema, pois as menores podem ser contempladas somente com os requisitos.

Para poder auxiliar no processo de modelagem do sistema, você pode fazer uso de ferramentas CASE (*Computer-Aided Software Engineering*), tais como o Astah Community e o Microsoft Visio. Além de usar os casos de uso e diagramas de casos de uso, podemos detalhá-lo através do diagrama de sequência, que será visto ainda nesta unidade.

Podemos identificar alguns atores e casos de uso que devem fazer parte de nosso sistema. Como atores, podemos identificar Clientes, Vendedores e Caixa. Alguns casos de uso deste sistema são Comprar Colecionáveis, Comprar Revistas em Quadrinho e Comprar objetos de Arte e Decoração.

Veja o caso de uso 'Comprar Colecionáveis que envolve o ator Cliente na forma descritiva:

Caso de Uso para Comprar Colecionáveis:

Escopo: central de vendas.

Ator: cliente.

Interessado: proprietário loja e caixa.

Garantia de sucesso: venda armazenada no sistema.

Cenário principal:

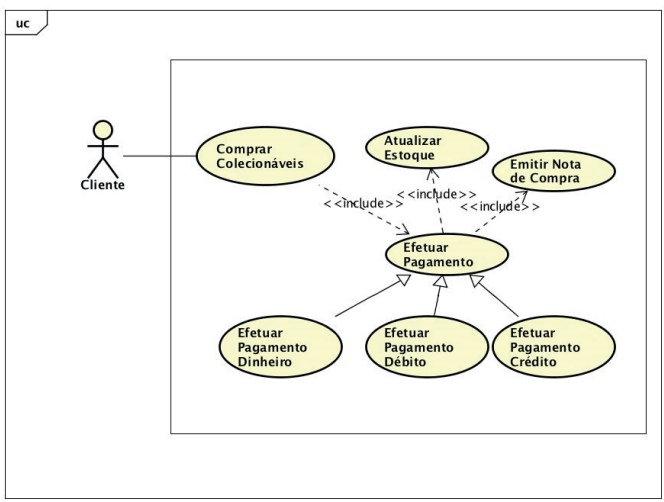
- Cliente escolhe o(s) colecionável(eis) que deseja comprar e efetua pagamento.
- Cliente pode efetuar pagamento em dinheiro, débito ou crédito.
- Ao realizar pagamento, o sistema atualiza o estoque do(s) colecionável(eis) vendido(s).
- Ao realizar pagamento, o sistema emite a nota de compra.

Cenário alternativo:

- Sistema não reconhece o código do colecionável e emite mensagem para nova tentativa de código.
- Sistema apresenta falha para imprimir a nota fiscal. Sistema deve emitir mensagem de aviso para verificar status da impressora.

Além da apresentação dos casos de uso na forma descritiva, uma representação bastante prática e de fácil entendimento para os clientes é a representação gráfica usando o diagrama de casos de uso da UML. Uma sugestão para o sistema está ilustrada na Figura 2.4.

Figura 2.4 | Diagrama de caso de uso para o sistema



Fonte: Captura de tela do software Astah, elaborada pelo autor.

Agora que você já tem uma visão do sistema, complemente o documento a ser entregue para o cliente com a descrição de outros casos de usos do sistema.

Avançando na prática

Criando casos de uso

Descrição da situação-problema

Atuando em um SAC de uma empresa de serviços de telefonia celular, você foi contratado para automatizar o processo de serviço de atendimento ao cliente no formato de registro de reclamações, dúvidas, sugestões e elogios dos serviços. Quais casos de uso seriam utilizados para a avaliação dos serviços da empresa? Pense em sua própria operadora de celular e elabore os casos de uso.

Resolução da situação-problema

Casos de uso SAC empresa telefônica celular:

Escopo: central de atendimento SAC.

Ator: cliente.

Interessados: coordenadores e diretores de serviços.

Garantia de sucesso: respostas armazenadas e avaliadas.

Cenário principal:

1) Cliente faz login no sistema informando CPF e senha cadastrados.
2) Cliente digita a reclamação ao serviço de telefonia celular no sistema SAC.
3) Cliente recebe e-mail de confirmação de registro da reclamação.
4) Cliente digita a dúvida ao serviço de telefonia celular no sistema SAC.
5) Cliente recebe e-mail de confirmação de registro da dúvida.
6) Cliente digita a sugestão ao serviço de telefonia celular no sistema SAC.
7) Cliente recebe e-mail de confirmação de registro da sugestão.
8) Cliente digita o elogio ao serviço de telefonia celular no sistema SAC.

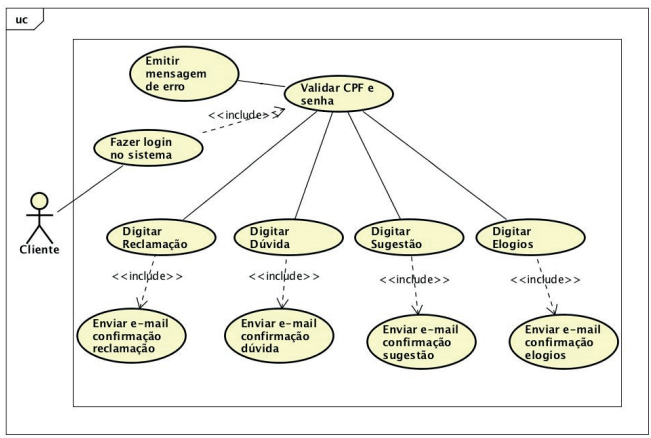
9) Cliente recebe e-mail de confirmação de registro do elogio.

Cenário alternativo:

1) Sistema não reconhece CPF e senha do cliente para Login. Sistema emite mensagem de erro e solicita nova tentativa.
2) Sistema apresenta instabilidade de conexão e não salva todas as informações digitadas. O sistema deve permitir que o cliente realize novo Login e continue a entrada das informações.

A Figura 2.5 apresenta a modelagem os casos de uso do cenário principal.

Figura 2.5 | Diagrama de casos de uso do sistema SAC



Fonte: captura de tela do software Astah Community, elaborada pelo autor.

Faça valer a pena

1. Casos de uso são utilizados para documentar e mostrar uma tarefa requisitada pelo usuário no sistema. São usados para melhor entendimento das etapas envolvidas para atender às necessidades do usuário.

Um ator é algo que possui comportamento, podendo ser o próprio sistema quando invoca serviços de outros sistemas. Há três tipos de atores. Quais são?

a) Ator principal, ator de suporte e ator de bastidor.
b) Ator principal, ator de suporte e ator secundário.
c) Ator principal, ator secundário e ator de bastidor.

d) Ator principal, ator secundário e ator de suporte.
e) Ator principal, ator secundário e ator de sistema.

2. Os casos de uso são frequentemente aplicados para melhor entendimento e detalhamento dos requisitos funcionais. São muito utilizados na fase de análise de requisitos como forma de descobrir os requisitos do usuário e os funcionais. Os casos de uso variam muito de forma de uma organização para outra em termos de conteúdo incluído, mas, de modo geral, são constituídos por alguns componentes essenciais.

Assinale a alternativa que apresente corretamente esses componentes essenciais.

a) Ator e casos de uso.
b) Ator, cenário e casos de uso.
c) Cenário e casos de uso.
d) Casos de uso e relacionamentos.
e) Ator, casos de uso e relacionamentos.

3. Os relacionamentos entre atores e casos de uso e entre casos de uso são representados por uma linha que os une. O retângulo utilizado para "armazenar" os casos de uso representa o limite do sistema a ser desenvolvido. Os relacionamentos podem utilizar estereótipos.

Relacionamentos podem ser estereotipados indicando que o caso de uso de onde a seta parte inclui obrigatoriamente as ações do caso de uso de destino. Qual o nome deste estereótipo?

a) <<include>>.
b) <<extend>>.
c) <<ator>>.
d) <<input>>.
e) <<extended>>.

Seção 2.2

Casos de uso do projeto de sistemas

Diálogo aberto

Prezado aluno, quando realizamos um bom projeto, como o de uma casa, ele vai muito além de paredes para sustentar o teto e lhe proteger da chuva e dos ventos. Ao projetar a casa, levamos em consideração a forma como essa casa vai ser utilizada. Se você gosta de cozinhar para sua família, certamente gostará de ter uma cozinha planejada para armazenar mantimentos e utensílios de modo prático. No caso de sua família ser bem numerosa, será preciso pensar na utilização dos banheiros, planejando adequadamente a posição correta deles. Pensar em como vamos utilizar a casa é um exemplo de análise baseada em caso de uso. Consideramos os vários modos de utilização da casa e os casos de uso orientam a arquitetura. Assim deve ser a construção de um software, bem planejado.

Você já parou para avaliar os benefícios que um diagrama de casos de uso detalhado proporciona à equipe de desenvolvimento, principalmente com relação ao entendimento e à validação dos requisitos, junto ao cliente? Um diagrama de casos de uso detalhado serve como meio de comunicação entre a equipe de desenvolvimento e o cliente, facilitando assim a exploração e melhor entendimento dos requisitos.

Pensando neste cenário, vamos retomar o problema da loja de colecionáveis. Os proprietários precisam de sua ajuda para projetar um sistema para gerenciar a loja e controlar o estoque, as contas a pagar e receber e seus clientes e fornecedores. Como os proprietários possuem conhecimento limitado para o desenvolvimento de um sistema desse porte, eles necessitam de sua ajuda. Anteriormente, você realizou a modelagem dos requisitos funcionais da loja através do diagrama de casos de uso e agora está na hora de efetuar a modelagem detalhada desses casos de uso. Como realizar a modelagem detalhada dos casos de uso? Como aplicar casos de uso de alto nível? Como ficaria o diagrama de casos de uso detalhado do projeto?

Para responder a esses questionamentos e poder auxiliar os proprietários da loja de colecionáveis, você vai aprender a realizar a

modelagem de casos de uso detalhados utilizando generalização de atores e casos de uso, colaboração de casos de uso e dependência entre casos de uso. Você realizará a modelagem detalhada dos diagramas de casos de uso do projeto. Essas são etapas fundamentais para o processo de análise e projeto de um sistema, sendo de extrema importância para seu aprendizado e para as próximas etapas a serem cumpridas.

Não pode faltar

Um detalhe importante sobre casos de uso está no fato de podermos criá-los sem especificar como serão implementados. Podemos, por exemplo, estabelecer como um sistema de caixa eletrônico deve funcionar, utilizando os casos de uso para definir como os usuários devem interagir com o sistema mesmo sem saber nada sobre o funcionamento interno do caixa eletrônico. A grande vantagem é permitir que os analistas se comuniquem com os usuários e demais desenvolvedores.

Os diagramas de casos de uso podem ser detalhados. Um ator representa um conjunto de papéis que usuários de casos de uso desempenham. Os atores são representados como figuras esquematizadas e podem ser detalhados utilizando a generalização de relacionamentos. Você pode definir grupos gerais de atores e especializá-los.

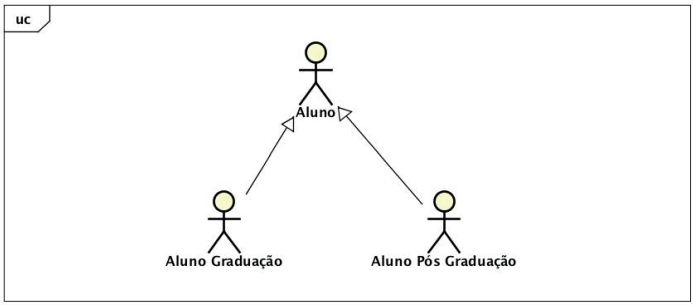
Reflita

O processo de desenvolvimento de sistemas tem se tornado cada vez mais difícil devido à complexidade das regras de negócio das corporações. Dessa forma, para obter sucesso no desenvolvimento de sistemas, devemos ter total compreensão do processo de negócio dos clientes, entendendo claramente suas necessidades, riscos para o desenvolvimento e como o software deverá se comportar no dia a dia. O que podemos utilizar para nos guiar no desenvolvimento de software, rastreando impactos nos requisitos?

Segundo Booch, Rumbaugh e Jacobson (2005), uma generalização é um relacionamento entre itens gerais e tipos mais específicos, sendo geralmente chamados de relacionamentos "é um tipo de". Uma generalização é representada graficamente com linhas sólidas e uma grande seta triangular apontada para o item

geral. A Figura 2.6 apresenta um exemplo de generalização de atores. O ator Aluno é considerado um ator geral e os atores Aluno Graduação e Aluno Pós-graduação são considerados específicos.

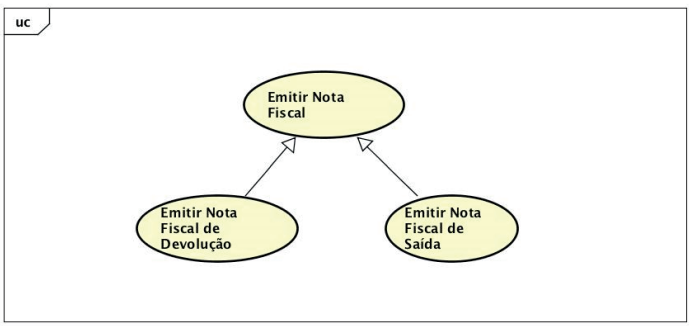
Figura 2.6 | Generalização de atores



Fonte: captura de tela do software Astah Community, elaborada pelo autor.

Assim como os atores podem ser generalizados, os casos de uso também podem ser organizados pela especificação de relacionamentos de generalização. A generalização significa que um caso de uso filho herda o comportamento do caso de uso pai. O caso de uso filho deverá acrescentar ou sobrescrever o comportamento do caso de uso pai. A Figura 2.7 apresenta um exemplo de generalização de casos de uso. O caso de uso pai (geral), Emitir Nota Fiscal, representa os elementos gerais para a emissão de uma nota fiscal. Os casos de uso filhos (específicos), Emitir Nota Fiscal de Devolução e Emitir Nota Fiscal de Saída, que são casos de uso que contêm os elementos específicos para a emissão de determinado tipo de nota fiscal.

Figura 2.7 | Generalização de casos de uso

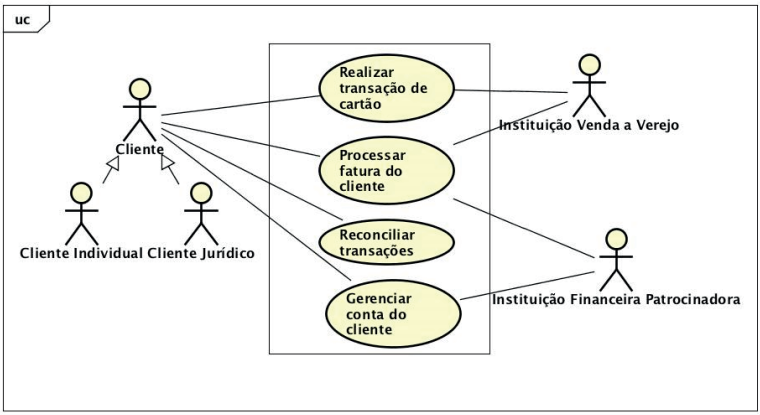


Fonte: captura de tela do software Astah Community, elaborada pelo autor.

Exemplificando

Considere o contexto de um sistema de validação de cartões de crédito. Há dois tipos de clientes: individual e jurídico. O cliente pode realizar transação de cartão, solicitar processamento da fatura, reconciliar transações e gerenciar conta. Ao realizar a transação do cartão e processar a fatura do cliente, ele é atendido pelo ator Instituição de Venda a Varejo. Ao solicitar Gerenciar conta do cliente, ele é atendido pelo ator Instituição Financeira Patrocinadora. A Figura 2.8 ilustra o diagrama de casos de uso do sistema de validação de cartão de crédito.

Figura 2.8 | Modelagem de sistema de validação de cartão de crédito

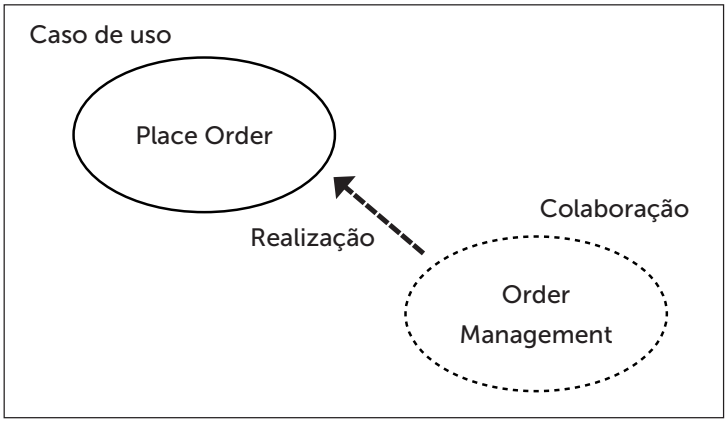


Fonte: adaptada de Booch Rumbaugh e Jacobson (2005, p. 246).

Aplicações de casos de uso em alto nível

Segundo Booch, Rumbaugh e Jacobson (2005), um caso de uso obtém o comportamento esperado de um sistema, sem especificar como esse comportamento é implementado. Podemos efetuar casos de uso através da criação de uma sociedade de classes e outros elementos que trabalham em conjunto para implementar o comportamento desse caso de uso. Essa sociedade de elementos é modelada na UML como uma colaboração. A Figura 2.9 ilustra casos de uso e colaboração.

Figura 2.9 | Casos de uso e colaborações



Fonte: Booch, Rumbaugh e Jacobson (2005, p. 234).

A Figura 2.9 ilustra o caso de uso Place Order (Fazer encomenda), realizado através da colaboração de Order Management (Gerenciar pedidos).

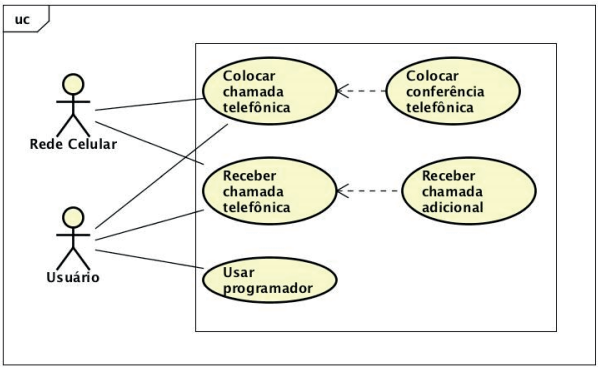
Pesquise mais

Para melhor desenvolvimento dos casos de uso detalhados, como generalização de atores, casos de uso e dependência de casos de uso, realize a leitura do material do prof. Anderson Cavalcanti.

CAVALCANTI, **A. Modelo de casos de uso e diagrama de casos de uso**. UFRN-CT-DCA (Universidade Federal do Rio Grande do Norte-Departamento de Engenharia de Computação e Automação), Natal, [s.d.]. Disponível em: <https://bit.ly/2mAExPR>. Acesso em: 23 jul. 2018.

Além da representação de generalização e colaboração, os casos de uso podem ser detalhados através de relacionamentos de dependência. Este relacionamento apresenta dependências de um caso de uso em relação a outro. O relacionamento de dependência é apresentado por uma linha tracejada entre os casos de uso, contendo uma seta apontando para o caso de uso dependente. Na Figura 2.10 o caso de uso Colocar conferência telefônica depende do Colocar chamada telefônica. Do mesmo modo, o caso de uso Receber chamada adicional depende do Receber chamada telefônica. Desta forma, representamos a dependência entre casos de uso.

Figura 2.10 | Dependência entre casos de uso



Fonte: adaptada de Booch, Rumbaugh e Jacobson (2005, p. 242).

Assimile

De acordo com Booch, Rumbaugh e Jacobson (2005), um diagrama de casos de uso bem-estruturado:

- Tem foco em comunicar um aspecto da visão estática de caso de uso do sistema.
- Contém apenas casos de uso e atores essenciais à compreensão desse aspecto.
- Fornece detalhes consistentes com seu nível de abstração.
- Não é tão minimalista, que informe mal o leitor sobre a semântica, que é importante.

Ao se definir um diagrama de caso de uso:

- Atribua um nome que comunique seu propósito.
- Distribua seus elementos para evitar cruzamento de linhas.
- Use notas e cores como indicações visuais e chame a atenção para características importantes do diagrama.

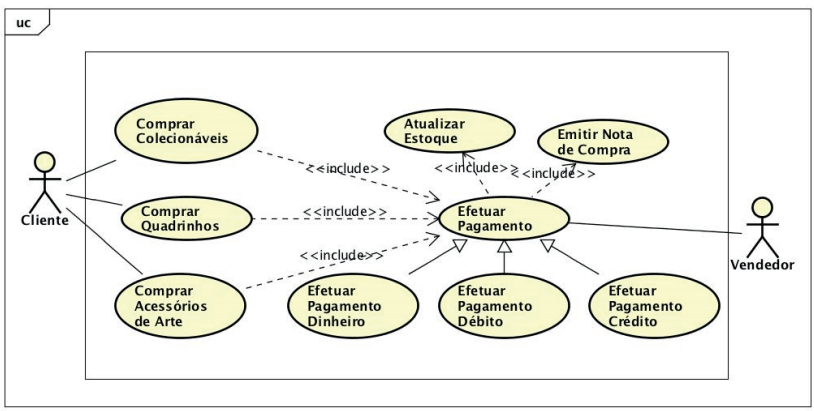
Assim, você aprendeu mais detalhes sobre o diagrama de casos de uso, como generalização de atores e casos de uso e colaborações e dependências entre casos de uso. Esses elementos serão essenciais para o refinamento e detalhamento de seus diagramas de casos de uso.

Sem medo de errar

Dando continuidade à modelagem do sistema para a loja de colecionáveis, após você realizar a modelagem dos requisitos funcionais da loja através do diagrama de casos de uso, agora está na hora de efetuar a modelagem detalhada desses casos de uso.

Vamos começar a resolver o problema analisando quantos atores estão envolvidos. Como trata-se de um sistema comercial, podemos considerar dois atores: cliente e vendedor. O cliente pode Comprar colecionáveis, Comprar quadrinhos e Comprar acessórios de arte. Ao realizar a compra, o cliente Efetua o pagamento, atendido pelo vendedor. O pagamento pode ser Efetuar pagamento em dinheiro, Efetuar pagamento débito e Efetuar pagamento crédito. Ao Efetuar pagamento, é realizada a atualização do estoque (Atualizar Estoque) e a emissão da nota (Emitir nota de Compra). Abaixo segue uma proposta de modelagem de casos de uso detalhados para a loja.

Figura 2.11 | Diagrama de casos de uso



Fonte: captura de tela do software Astah Community, elaborada pelo autor.

Dessa maneira, obtivemos a modelagem detalhada dos casos de uso, possibilitando uma melhor compreensão das funcionalidades do sistema, que vai auxiliar o analista na identificação dos requisitos do sistema.

Avançando na prática

Diagramas de casos de uso detalhados

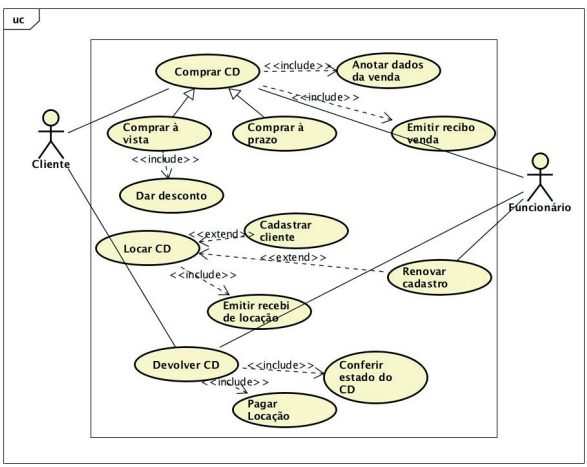
Descrição da situação-problema

Uma empresa de comercialização de CDs possui discos para venda e locação. O cliente pode comprar ou locar os discos. Para realizar a locação, é necessário que o cliente esteja cadastrado na empresa, ou seja, tenha sua ficha de cadastro, renovada a cada 6 meses. As vendas de CDs podem ser efetuadas à vista, com 10% de desconto, ou sem desconto através de cheque pré-datado, descontado 30 dias após a compra. As locações somente podem ser pagas à vista, no ato da devolução dos discos, que tem de acontecer 2 dias após a locação. O valor da locação de cada disco é de R$ 3,00. A loja possui um funcionário cuja função é atender os clientes durante a venda e locação dos discos. Suas principais tarefas são: receber e conferir o pagamento efetuado pelos clientes, emitir recibo de venda e locação (este último em duplicata), registrar o valor de cada venda e o nome dos discos vendidos e conferir o estado dos CDs devolvidos (caixa, disco e encarte).

Resolução da situação-problema

A modelagem do problema proposto é apresentada a seguir:

Figura 2.12 | Diagrama de casos de uso



Fonte: captura de tela do software Astah Community, elaborada pelo autor.

Faça valer a pena

1. Uma generalização é um relacionamento entre itens gerais e tipos mais específicos. Geralmente são chamadas de relacionamentos "é um tipo de". Uma generalização é representada graficamente com linhas sólidas e uma grande seta triangular apontado para o item geral.

Assinale a alternativa correta que apresenta os tipos de generalização que podem ser representadas em diagramas de casos de uso.

a) Apenas generalização de casos de uso.
b) Apenas generalização de atores.
c) Generalização de atores e generalização de casos de uso.
d) Generalização de colaboração.
e) Todas as alternativas estão corretas.

2. Um caso de uso obtém o comportamento esperado de um sistema, sem especificar como esse comportamento é implementado. Podemos efetuar casos de uso através da criação de uma sociedade de classes e outros elementos que trabalham em conjunto para implementar o comportamento desse caso de uso.

Assinale a alternativa correta que descreve como essa sociedade de elementos é modelada na UML.

a) Colaboração.
b) Dependência.
c) Associação.
d) Generalização.
e) Agregação.

3. Além da representação de generalização e colaboração, os casos de uso podem ser detalhados através de relacionamentos de dependência. O relacionamento de dependência é apresentado por uma linha tracejada que conecta os casos de uso relacionados.

Assinale a alternativa correta que apresenta os possíveis casos de dependência no diagrama de casos de uso.

a) Apenas dependência entre atores.
b) Apenas dependência entre casos de uso.
c) Dependência entre atores e casos de uso.
d) Apenas dependências entre colaborações.
e) Todas as alternativas estão corretas.

Seção 2.3

Diagrama de casos de uso do projeto de sistemas

Diálogo aberto

Prezado aluno, você já parou para pensar em como as lojas de móveis planejados trabalham? Podemos fazer uma analogia entre a loja de móveis planejados e a construção de um software. Primeiramente, o cliente se dirige até a loja de móveis planejados e passa ao projetista (analista de sistemas) todas as suas necessidades (requisitos). Com as informações em mãos, o projetista cria um projeto, geralmente utilizando computador, que apresenta um modelo (em duas ou três dimensões) dos móveis que devem ser construídos. Assim como no projeto dos móveis planejados, o projeto de sistemas é parte primordial no processo de desenvolvimento de sistemas, pois ele vai definir todas as necessidades do sistema. Dentro desse contexto, uma das formas de estruturar o conjunto de necessidades e funcionalidades de um sistema é utilizar casos de uso em alto nível e o diagrama de sequência para detalhamento de um caso de uso, que você aprenderá nessa seção.

Você foi contratado para modelar um sistema para uma loja que comercializa objetos colecionáveis. Agora que já temos os casos de uso detalhados e de alto nível, devemos representá-los por meio do diagrama de sequência, que vai detalhar os casos de uso para seu melhor entendimento. Como devemos proceder para representar casos de uso em diagramas de sequências? Quais casos de uso devem ser detalhados?

Crie um documento com pelo menos dois diagramas de sequência e com o detalhamento dos casos de uso para o sistema da loja de colecionáveis.

Nessa seção você aprenderá a criar casos de uso em alto nível, utilizando o diagrama de sequência para detalhamento dos casos de uso.

A modelagem do projeto usando os diagramas de sequência para seu detalhamento demanda seu empenho, então, mãos à obra!

Não pode faltar

Segundo Larman (2007), os casos de uso apresentam como os atores externos interagem com o software que estamos projetando. Durante a interação do usuário com o sistema, um ator gera eventos de sistema, geralmente solicitando alguma operação de sistema para tratar o evento.

De acordo com Sommervile (2011), um caso de uso pode ser detalhado por um diagrama de sequência, que vai apresentar a ordem em que ocorrem os eventos, as mensagens enviadas, os métodos chamados e como os objetos se interagem. O diagrama de sequência faz parte da modelagem dinâmica, buscando capturar o comportamento do sistema ao longo do tempo. O comportamento de sistemas orientados a objetos é observado pela comunicação entre objetos, ocorrendo a comunicação através da troca de mensagens.

Uma mensagem é uma maneira de comunicação entre dois objetos e é implementada como uma operação. Na fase de análise, uma mensagem pode ser vista como solicitação de serviço ou informação. Em programação orienta a objetos, a mensagem é a invocação de uma operação (função). Simbolicamente, conforme o Quadro 2.1, a mensagem é representada como uma seta, que pode assumir os seguintes significados:

Quadro 2.1 | Formas de representação de uma mensagem

Tipo	Símbolo	Explicação
Simples		Representa um fluxo de controle passado entre dois objetos sem revelar maiores detalhes sobre ele.
Síncrona		Um fluxo de controle tipicamente implementado sob forma de uma chamada de função, em que o chamador aguarda o fim da operação chamada para continuar sua execução.
Assíncrona		Representa um fluxo de controle para o qual não se espera nenhum tipo de retorno do objeto chamado.
Síncrona com retorno imediato		É uma combinação de mensagem simples e síncrona, em que o retorno é quase imediato à chamada da operação.

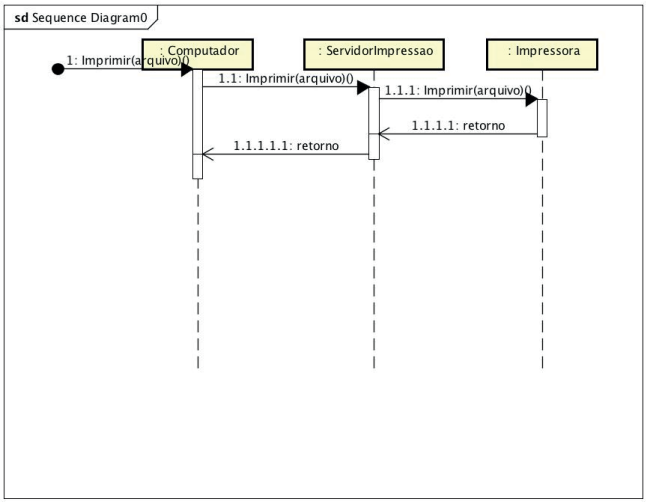
Fonte: elaborado pelo autor.

Diagrama de sequência

O diagrama de sequência apresenta interação entre objetos com foco na ordem de tempo das mensagens. Na fase de análise, os diagramas de sequência podem ser usados para fazer a modelagem do fluxo de controle de um caso de uso. Dois elementos são visualizados no diagrama de sequência: objetos e mensagens.

Imagine a situação de um usuário utilizando um computador para enviar arquivos a serem impressos para uma impressora. Quais objetos estão envolvidos e quais trocas de mensagens seriam realizadas? Podemos considerar os objetos Computador (de onde partirá a solicitação de impressão), Servidor de Impressão (que receberá a solicitação e direcionará para a impressora) e Impressora (que fará a impressão). O diagrama de sequência da Figura 2.13 apresenta os objetos e as trocas de mensagens para essa situação.

Figura 2.13 | Diagrama de sequência para impressão de arquivos



Fonte: captura de tela do software Astah Community, elaborada pelo autor.

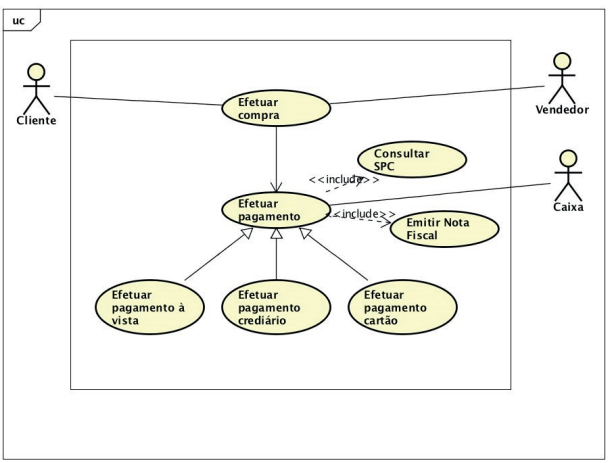
Cada um dos retângulos da Figura 2.13 representa um objeto modelado por uma classe, sendo os objetos Computador, ServidorImpressao e Impressora, que são ativados quando recebem uma mensagem. Na primeira mensagem que chega a Computador, Imprimir(arquivo) ativa sua linha de vida. Computador e ServidorImpressao trocam mensagem síncrona, aquela que faz o chamador aguardar a sua execução. O ServidorImpressao envia a mensagem de impressão do arquivo para Impressora. As mensagens retornadas são simples, sem apresentar nenhum detalhe.

Reflita

O diagrama de sequência é da UML, que representa a sequência de mensagens passadas entre objetos em um programa de computador. De que maneira o diagrama de sequência vai auxiliar o programador de sistemas na implementação de um sistema?

Para apresentar o detalhamento de um caso de uso pelo diagrama de sequência, considere a modelagem da Figura 2.14, que apresenta o diagrama de casos de uso para a seguinte situação: quando um cliente compra um produto em uma loja, ele vai até o caixa da loja para efetuar o pagamento, o que é possível fazer à vista, em crediário ou com cartão de crédito. Independentemente da forma de pagamento, a loja consulta o Serviço de Proteção ao Crédito e depois emite a Nota Fiscal de Venda.

Figura 2.14 | Diagrama de casos de uso para efetuar uma compra

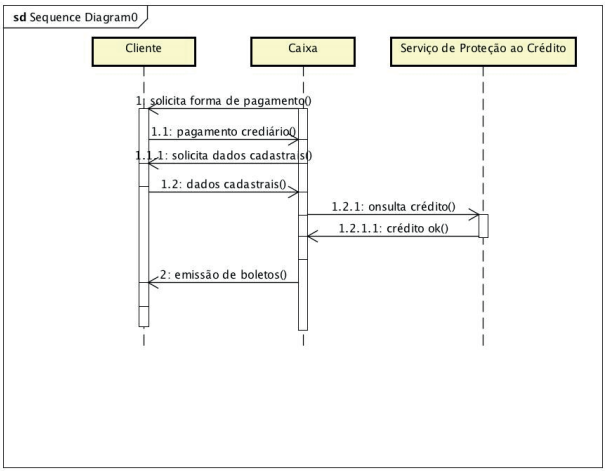


Fonte: captura de tela do software Astah Community, elaborada pelo autor.

Na Figura 2.15, é apresentado o diagrama de sequência do fluxo de controle para o caso de uso efetuar pagamento no crediário, que foi apresentado na Figura 2.14. Quando uma mensagem é recebida, tem-se uma ativação, iniciando uma atividade no objeto receptor. No exemplo da Figura 2.15, o primeiro objeto a ser ativado é o Cliente, através da mensagem "solicita forma de pagamento()" vinda do objeto Caixa. Quando um objeto está ativado, ou ele está

executando alguma atividade ou está esperando pelo retorno de outro objeto para o qual enviou uma mensagem.

Figura 2.15 | Diagrama de sequência para o caso de uso Efetuar Pagamento Crediário



Fonte: captura de tela do software Astah Community, elaborada pelo autor.

Assimile

O diagrama de sequência apresenta o detalhamento dos casos de uso, permitindo uma maior compreensão de como as funcionalidades do sistema são efetivamente aplicadas, isto é, qual sequência de operações é realizada para satisfazer determinada funcionalidade.

No diagrama de sequências, o objeto é representado por um retângulo com um identificador, como Cliente, Caixa e Serviço de Proteção ao Crédito. Cada objeto possui uma linha de vida, representada por uma linha vertical tracejada. Quando um objeto está ativo, é representado por um retângulo que sobrepõe a linha de vida do objeto.

O exemplo da Figura 2.15 ilustra a sequência de troca de mensagens entre os objetos Cliente, Caixa e Serviço de Proteção ao Crédito. O Caixa envia uma mensagem ao Cliente, solicitando a forma de pagamento. O Cliente informa que a forma de pagamento é crediário. O Caixa solicita dados cadastrais ao Cliente, que retorna os dados cadastrais. O Caixa envia uma mensagem para o objeto de Serviço de Proteção ao Crédito, sendo ativado e recebendo a solicitação da consulta de crédito. O Serviço de Proteção ao Crédito informa ao Caixa que o crédito está OK e o Caixa envia os boletos ao Cliente.

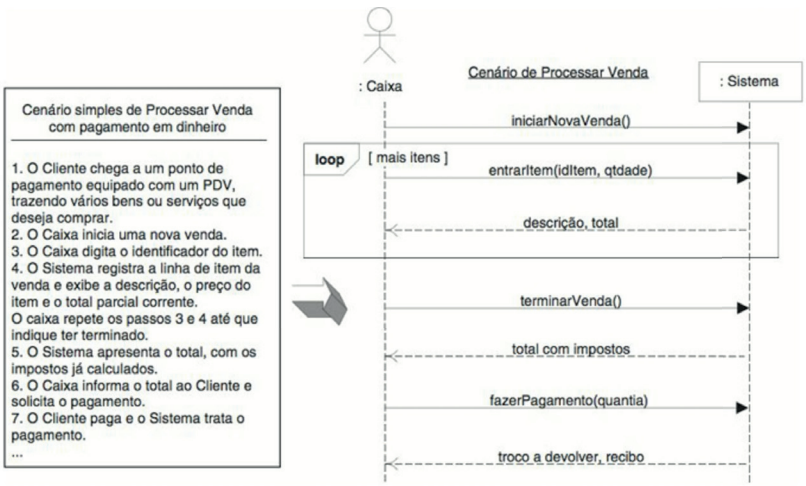
Pesquise mais

Para maior entendimento de diagrama de sequência do sistema e detalhamento de um caso de uso pelo diagrama de sequência, realize a leitura do material a seguir:

NAKAGAWA, E. Y. Diagramas de Sequência do Sistema. USP (Universidade de São Paulo), São Paulo, [s.d.]. Disponível em: <https://bit.ly/2mBVcTd>. Acesso em: 23 jul. 2018.

De acordo com Larman (2007), um diagrama de sequência é utilizado para ilustrar as interações de atores e operações iniciadas por eles. O diagrama de sequência do sistema mostra um cenário específico de um caso de uso, os eventos que os atores externos geram, sua ordem e os eventos entre sistemas. Dessa forma, o diagrama de sequência é gerado a partir da inspeção de um caso de uso. A Figura 2.16 ilustra o diagrama de sequência derivado do cenário de caso de uso Processar Venda com pagamento em dinheiro.

Figura 2.16 | Diagrama de sequência derivado de casos de uso



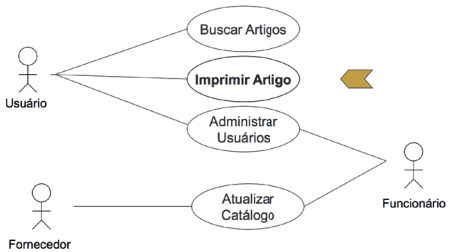
Fonte: Larman (2007, p. 199).

Exemplificando

Considere o diagrama de casos de uso abaixo, Figura 2.17, que ilustra o funcionamento do sistema de manipulação de artigos. O usuário, ao

utilizar o sistema, pode buscar e imprimir artigos e administrar usuários. O fornecedor atualiza o catálogo. O funcionário administra usuários e também atualiza catálogos.

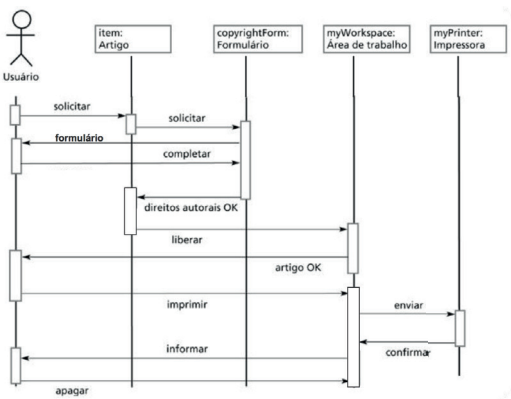
Figura 2.17 | Diagrama de casos de uso Manipulação de artigos



Fonte: elaborada pelo autor.

A Figura 2.18 ilustra o diagrama de sequência do caso de uso Imprimir artigo. Esse caso de uso é detalhado e utiliza a representação dos objetos Artigo, Formulário, Área de trabalho e Impressora, bem como a troca de mensagens entre os objetos.

Figura 2.18 | Diagrama de sequência do caso de uso Imprimir artigo



Fonte: elaborada pelo autor.

Na Figura 2.18, o usuário solicita um artigo, ativando o objeto Artigo. O objeto artigo envia uma mensagem para Formulário, solicitando o copyrightForm que pede ao usuário o preenchimento do formulário. O formulário é validado e o artigo é liberado na Área de trabalho.

O usuário envia a solicitação de impressão do artigo, atendida pela Impressora, que devolve a confirmação para a Área de trabalho.

Dessa forma, você aprendeu a aplicar o diagrama de sequência para detalhamento de um caso de uso, sendo fundamental para melhor desempenho das atividades de um caso de uso.

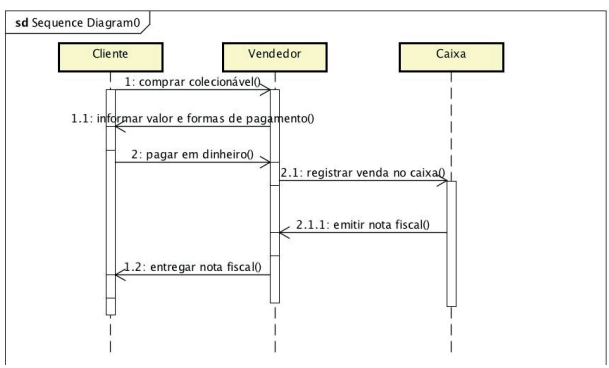
Sem medo de errar

Você foi contratado pelos proprietários da loja de colecionáveis para fazer a modelagem da análise para o projeto de sistema. Como devemos proceder para representar casos de uso em diagramas de sequências? Quais casos de uso devem ser detalhados?

Para poder representar casos de uso em diagramas de sequência, você deve inicialmente conhecer o diagrama de sequência, seus componentes e, em seguida, avaliar quais casos de uso devem ser detalhados no diagrama de sequência. Para o projeto em questão, alguns casos de uso podem ser detalhados, tais como, Comprar Colecionáveis, Comprar Quadrinhos e Comprar Acessórios. Para cada um desses casos de uso, detalhe também os casos de uso que estão relacionados, que são Efetuar Pagamento, Atualizar Estoque e Emitir Nota Fiscal de Compra.

Para o projeto, você deve elaborar os diagramas de sequência para os casos de uso escolhidos. A Figura 2.19 ilustra o detalhamento do caso de uso Comprar Colecionável. O processo inicia com o objeto Cliente enviando mensagem ao objeto Vendedor solicitando comprar colecionável. O Vendedor devolve ao Cliente o valor e as formas de pagamento. O Cliente informa que vai realizar o pagamento em dinheiro. O vendedor registra a venda no Caixa. O Caixa emite nota fiscal e o Vendedor entrega a nota fiscal ao Cliente.

Figura 2.19 | Diagrama de sequência detalhando caso de uso Comprar Colecionável



Fonte:captura de tela do software Astah Community, elaborada pelo autor.

Agora que você já elaborou o diagrama de sequência, detalhando o caso de uso Comprar Colecionável, realize a modelagem do diagrama de sequência para detalhar outro caso de uso.

Avançando na prática

Elaborando diagrama de sequência

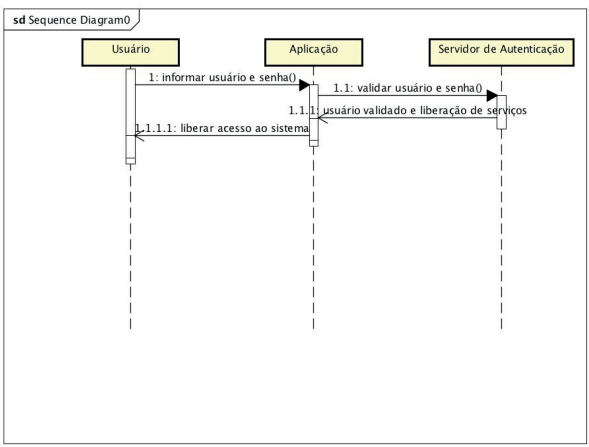
Descrição da situação-problema

Um analista deseja implementar um sistema de autenticação de acesso para seu sistema de aula a distância. Ele está apresentando algumas dificuldades de entendimento do problema e procurou sua ajuda para auxiliá-lo neste problema. De que maneira você poderia ajudá-lo na compreensão do problema?

Resolução da situação-problema

Para resolver o problema de entendimento do sistema de autenticação de internet, você deve elaborar, primeiramente, um diagrama de casos de uso e, na sequência, utilizar um diagrama de sequência para detalhamento do caso de uso. A Figura 2.20 ilustra a criação do diagrama de sequência.

Figura 2.20 | Diagrama de sequência



Fonte: captura de tela do software Astah Community, elaborada pelo autor.

Faça valer a pena

1. Uma mensagem é uma maneira de comunicação entre dois objetos e é implementada como uma operação. Na fase de análise, uma mensagem pode ser vista como solicitação de serviço ou informação. Em programação orientada a objetos, a mensagem é a invocação de uma operação (função).

A mensagem é representada como uma seta conectando dois objetos. Assinale a alternativa que apresenta corretamente os tipos de seta para representação de mensagens:

a) Apenas simples e síncrona.
b) Apenas síncrona e assíncrona.
c) Apenas simples, síncrona e assíncrona.
d) Apenas simples e síncrona com retorno imediato.
e) Apenas simples, síncrona, assíncrona e síncrona com retorno imediato.

2. O diagrama de sequência apresenta interação entre objetos com foco na ordem de tempo das mensagens. Na fase de análise, os diagramas de sequência podem ser usados para fazer a modelagem do fluxo de controle de um caso de uso.

O diagrama de sequência possui elemento(s). Assinale a alternativa que contém esse(s) elemento(s):

a) Apenas objetos.
b) Apenas mensagens.
c) Objetos e mensagens.
d) Objetos, mensagens e atores.
e) Objetos e atores.

3. O diagrama de sequência faz parte da modelagem dinâmica, buscando capturar o comportamento do sistema ao longo do tempo. O comportamento de sistemas orientados a objetos é observado pela comunicação entre objetos, ocorrendo a comunicação através da troca de mensagens.

Assinale a alternativa que apresenta o que ocorre quando uma mensagem síncrona é recebida por um objeto.

a) Quando uma mensagem é recebida, tem-se uma ativação, iniciando uma atividade no objeto receptor em que o chamador aguarda o fim da operação.
b) Quando uma mensagem é recebida, verifica-se se o objeto está ativo.
c) Quando uma mensagem é recebida, tem-se uma ativação, iniciando uma atividade no objeto de origem da mensagem.
d) Quando uma mensagem é recebida, inicia-se uma atividade nos objetos.
e) Quando uma mensagem é recebida, verifica-se se o objeto está ativo, iniciando uma atividade no objeto receptor.

Referências

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

DENNIS, A.; WIXOM, B. H.; ROTH, R. M. **Análise e Projeto de Sistemas**. 5. ed. Rio de Janeiro: LTC, 2014.

ESTUDO NA WEB. **Entenda o Diagrama de Sequência | #10**. [S.l.], 2 nov. 2017. Disponível em: <https://bit.ly/2mE4Sww>. Acesso em: 24 jul. 2018.

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

FRANCO, Matheus. **Diagrama de Sequência**. Disponível em: <https://www.youtube.com/watch?v=UKfdmCMVda4>. Acesso em: 15 mai. 2018

LARMAN, C. **Utilizando UML e Padrões**: uma introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

MARTINS. L. E. G. **Notas de Aulas da Disciplina de Engenharia de Software II**. Pós-Graduação UNIMEP, 2006.

PADUA FILHO, W. de P. **Engenharia de Software**: fundamentos, métodos e padrões. 3. ed. Rio de Janeiro: LTC, 2009.

PRESSMAN, R. S; MAXIM, Bruce R. **Engenharia de Software**: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

SCHACH, S. R. **Engenharia de Software**: os paradigmas clássico e orientado a objetos. 7. ed. Porto Alegre: AMGH, 2010.

Modelagem da análise para projeto – diagrama de classes

Convite ao estudo

Prezado aluno, você já refletiu sobre como a produção de sistemas, antes voltada apenas em ambiente desktop, evoluiu para outros ambientes, tais como a web (websites e e-commerce), dispositivos móveis (smartphones e tablets), relógios (*smartwatches*) e televisões (*smart TVs*)? Percebeu como a complexidade aumentou? Desenvolver sistemas e aplicativos que possam interconectar todos esses dispositivos, apesar de não ser uma tarefa fácil, é totalmente viável e desafiadora. Na construção de uma solução de software, após determinar os requisitos (funcionais e não funcionais) e realizar a modelagem das funcionalidades do sistema por meio do diagrama de casos de uso, o próximo passo consiste em modelar as classes do sistema e seus relacionamentos, assuntos a serem abordados nesta unidade. A partir desse conhecimento, você será capaz de aplicar a modelagem e mapeamento do diagrama de classe, utilizando, para isso, a UML (Linguagem de Modelagem Unificada) e ferramentas específicas.

A fim de ajudá-lo em seu aprendizado, apresentamos a seguinte situação: um depósito de beleza, que está expandindo suas atividades, comercializa toda a linha produtos de higiene pessoal, perfumaria, tinturas, maquiagens, dentre outros. A proprietária necessita de um sistema de controle de estoque para o depósito, porém é totalmente leiga em relação a sistemas. Para poder auxiliá-la, você foi contratado para realizar a análise e projeto do sistema e deverá refletir sobre as seguintes questões: como elaborar diagramas de classes?

Quais aspectos estruturais devem ser utilizados no diagrama de classe? Como realizar relacionamentos e associações entre classes? Como realizar o refinamento dos diagramas de classe? Como realizar o mapeamento das classes para o modelo relacional? Essas e outras questões serão trabalhadas nesta unidade. Para respondê-las, você aprenderá a definição de diagrama de classe, refinamento dos aspectos estruturais das classes, tais como atributos e operações, e passagem da modelagem da análise para o projeto. Bom trabalho!

Seção 3.1

Diagrama de classes: conceito e aplicações

Diálogo aberto

Você já observou que quando vamos construir uma casa começamos nosso vocabulário com os elementos de construção básicos, tais como, fundação, lajotas, paredes, piso, janelas, portas, vigas e telhado? Esses itens são estruturais (paredes têm altura, largura e espessura), mas, de alguma forma, comportamentais (fundação pode suportar diferentes cargas, as portas abrem e fecham). O processo de determinar a arquitetura da casa envolve a união desses itens de forma a satisfazer os requisitos funcionais e não funcionais. As plantas do projeto que você cria para visualizar sua casa e especificar os detalhes destinados aos construtores são apresentações gráficas desses itens e relacionamentos. A construção de um software tem muitas dessas características e, utilizando a UML, você pode elaborar diagramas de classes (de forma semelhante às plantas do projeto da casa) para visualizar os aspectos estáticos desses blocos de construção e seus relacionamentos e para especificar detalhes da construção.

Como vimos anteriormente, você foi contratado para modelar o sistema de um depósito de beleza. Esse sistema deverá fazer o controle de estoque de todas as mercadorias comercializadas. Como realizar o processo de modelagem da análise para projeto do sistema, com os seus aspectos estáticos, isto é, as classes do sistema? Quais as prováveis classes e relacionamentos para esse projeto? Como elaborar um diagrama de classes?

Nesta seção você aprenderá a realizar a passagem da modelagem de análise de requisitos para projeto, utilizando o diagrama de classe. Você vai aprender a definição de diagrama de classes e o refinamento dos aspectos estáticos e estruturais das classes. Ao final, deverá produzir um relatório com as respostas dos questionamentos expostos e apresentá-lo ao seu professor.

Essas são etapas fundamentais para o processo de análise e projeto de um sistema, sendo de extrema importância para seu aprendizado e para as próximas etapas a serem realizadas, então, bom trabalho!

Não pode faltar

De acordo com Fowler (2005), o diagrama de classes descreve as classes do sistema e os relacionamentos existentes entre elas. Os diagramas de classes apresentam os atributos (propriedades) e as funcionalidades (operações) de uma classe e restrições que se aplicam às classes.

Segundo Larman (2007), a UML tem o diagrama de classes para apresentar classes, interfaces e suas associações, sendo utilizado para a modelagem estática de objetos.

Diagrama de classes e propriedades básicas

De acordo com Booch, Rumbaugh e Jacobson (2005), os principais itens do diagrama de classes são:

- Classes.
- Relacionamentos.

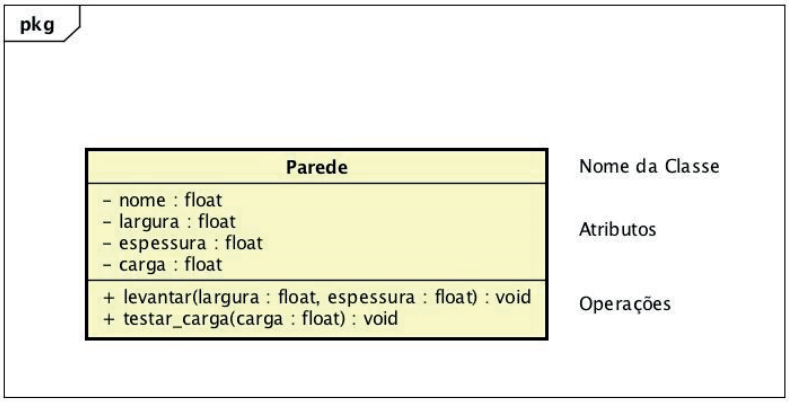
Classes

Segundo Booch, Rumbaugh e Jacobson (2005), classes são blocos de construção mais importantes em sistemas orientados a objetos. Uma classe apresenta um grupo de objetos com atributos e operações semelhantes. As classes são utilizadas para a obtenção do vocabulário do sistema em desenvolvimento, podendo incluir abstrações (partes do domínio do problema), assim como as classes que as implementam.

Considerando o exemplo da construção da casa, apresentado anteriormente, fundação, paredes, portas e janelas são projetadas como classes. Uma classe representa uma sociedade de objetos, não sendo um objeto individual, mas um conjunto inteiro deles. Assim, podemos pensar em "parede" como uma classe de objetos com determinadas propriedades comuns, como altura, largura, espessura, resistência e assim por diante.

A UML permite a representação gráfica de classes, conforme mostra a Figura 3.1.

Figura 3.1 | Representação visual de uma classe



Fonte: captura de tela do programa Astah Community, elaborada pelo autor.

Conforme a Figura 3.1, uma classe é representada através de um retângulo particionado em três partes, sendo o primeiro usado para nomear a classe "parede", o do meio para os atributos (características) "nome", "largura", "espessura" e "carga" e o último compartimento para as operações (funções) "levantar" e "testar_carga".

Pesquise mais

Para um maior conhecimento sobre diagramas de classe, introdução, conceitos de classes e tipos de relacionamentos, realize a leitura do material do Prof. Givanaldo Rocha de Souza, da IFRN, indicado a seguir.

SOUZA, Givanaldo Rocha de. **Diagrama de classe**. Disponível em: <https://goo.gl/StjBjp>. Acesso em: 1 ago. 2018.

Segundo Booch, Rumbaugh e Jacobson (2005), um atributo é uma característica de uma classe, logo, a classe pode ter vários atributos. Por exemplo, toda parede tem altura, largura e espessura.

Uma operação é uma funcionalidade que o objeto da classe pode executar e uma classe pode ter várias delas. Por exemplo, toda parede é levantada com uma largura e espessura e é feito o teste de carga.

Reflita

Comparando o uso do diagrama de casos de uso e o diagrama de classes, qual diagrama tem uma maior aplicação para a implementação de sistemas?

Relacionamentos

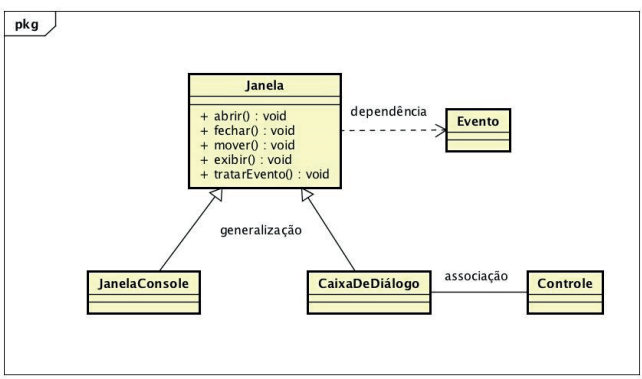
De acordo com Booch, Rumbaugh e Jacobson (2005), ao construir abstrações, você descobrirá que as classes não trabalham sozinhas. A grande maioria delas se relaciona com outras de várias formas, portanto, ao modelar de um sistema, além de verificar os itens que formam vocabulário dele, será necessário modelar como esses itens se relacionam.

Assimile

Dificilmente a classe é utilizada sozinha, pois cada uma funciona em colaboração com outras, para a realização de alguma semântica maior do que cada uma delas individualmente.

Na modelagem orientada a objetos temos três tipos de relacionamentos: dependências, que representam relacionamentos de dependência entre as classes; generalizações, que relacionam classes gerais e específicas; e associações, que representam relacionamentos entre objetos. A Figura 3.2 apresenta os três tipos de relacionamentos.

Figura 3.2 | Relacionamentos entre classes



Fonte: Booch, Rumbaugh e Jacobson (2005, p. 65).

A dependência é um tipo de relacionamento de utilização, o qual determinando que um item (por exemplo, uma classe "Janela"), da Figura 3.2, depende das informações e serviços de outro item (por exemplo, a classe "Evento"). A dependência é um relacionamento representado graficamente com linhas tracejadas se conectando ao item do qual o outro depende.

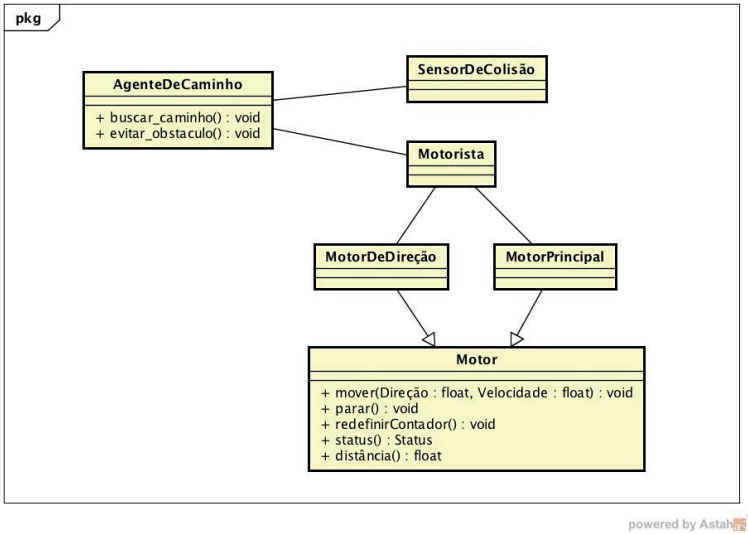
Uma generalização é um tipo de relacionamento que ocorre entre superclasses (itens gerais) e subclasses (itens mais específicos). A generalização representa que os objetos da classe-filha são utilizados em qualquer lugar no qual a classe-mãe ocorrer. A classe-filha herda os comportamentos da classe-mãe, principalmente seus atributos e operações. No exemplo da Figura 3.2, a classe geral é "Janela", que contém os métodos herdados por suas classes-filha "JanelaConsole" e "CaixaDeDiálogo", que são classes específicas.

Uma associação é um relacionamento que representa objetos de duas ou mais classes conectadas. De uma associação conectando duas classes, os objetos das classes podem ser acessados. No exemplo da Figura 3.2, a classe "CaixaDeDiálogo" está associada com a classe "Controle".

Exemplificando

A Figura 3.3 apresenta um conjunto de classes, baseado em Booch, Rumbaugh e Jacobson (2005), estabelecido a partir da implementação de um robô autônomo. A figura tem como foco as classes envolvidas no mecanismo para mover o robô por um determinado caminho. Existe uma classe abstrata, a "Motor", com duas classes-filha concretas, a "MotorDeDireção" e a "MotorPrincipal". Essas duas classes herdam as cinco operações da classe-mãe "Motor". A classe "AgenteDeCaminho" tem uma associação com "Motorista" e uma associação com "SensorDeColisão".

Figura 3.3 | Modelagem de colaboração entre classes



Fonte: Booch, Rumbaugh e Jacobson (2005, p. 112).

De acordo com Booch, Rumbaugh e Jacobson (2005), para fazer a modelagem de um esquema, é necessário:

- Identificar as classes existentes em seu modelo.
- Criar um diagrama de classes contendo essas classes.
- Ampliar os detalhes estruturais dessas classes, especificando os detalhes de seus atributos e focando as associações.

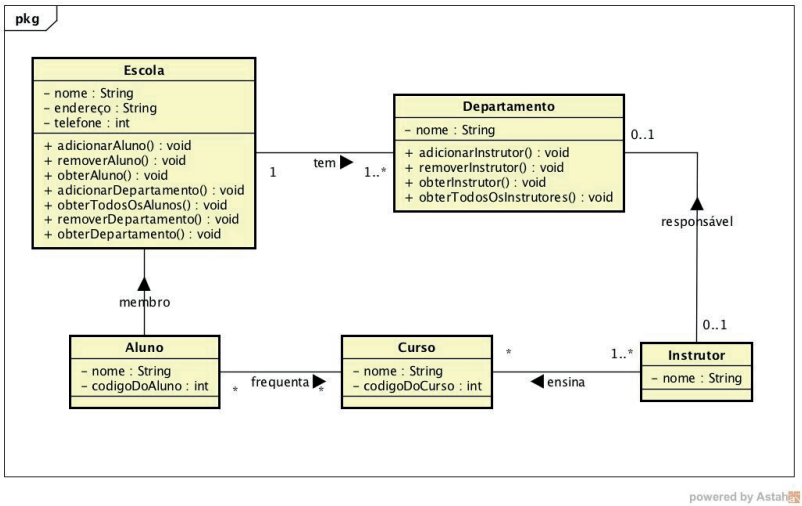
Ao criar um diagrama de classes, deve-se:

- Atribuir-lhe um nome que indique seu propósito.
- Evitar o cruzamento de linhas ao criar o diagrama.
- Organizar espacialmente seus elementos de maneira que os itens semanticamente relacionados fiquem fisicamente próximos.
- Procurar não exibir uma quantidade excessiva de tipos de relacionamentos.

A Figura 3.4 apresenta um conjunto de classes definidas a partir de um sistema de informações de uma escola. O diagrama

apresenta o relacionamento de associação entre as classes, destacando os atributos e operações.

Figura 3.4 | Modelagem de colaboração entre classes



Fonte: Booch, Rumbaugh e Jacobson (2005, p. 112).

A Figura 3.4 apresenta as classes e relacionamentos para um sistema acadêmico. No diagrama, a classe "Escola" se relacionando com a classe "Departamento", em que um objeto da classe "Escola" se relaciona com um ou muitos da classe "Departamento". Um objeto da classe "Departamento" se relaciona com apenas um objeto da classe "Escola". A classe "Departamento" possui zero ou um "Instrutor". O "Instrutor" é responsável por zero ou um objeto da classe "Departamento". Um objeto da classe "Instrutor" se relaciona com muitos objetos da classe "Curso". O "Curso" é ensinado por um ou muitos objetos da classe "Instrutor". O "Curso" é frequentado por muitos objetos da classe "Aluno" e o aluno frequenta vários "Cursos".

Até o momento, aprendemos sobre classes e suas propriedades básicas, tais como atributos, operações e relacionamentos de associação, generalização e dependência entre classes.

Sem medo de errar

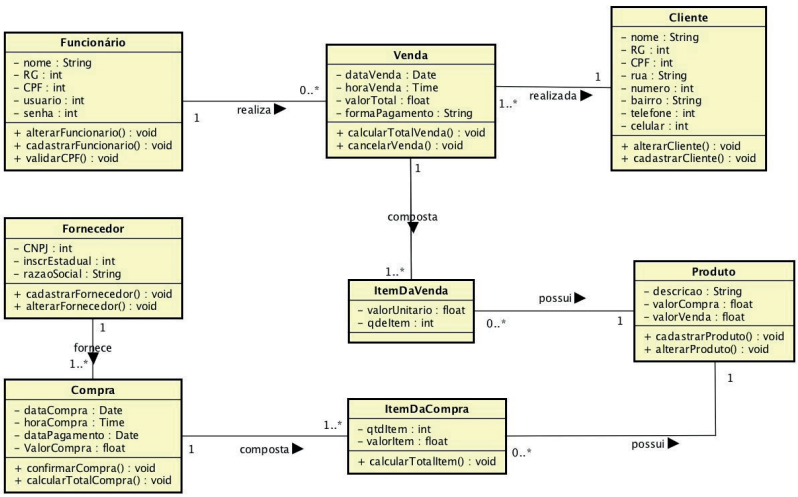
Você foi contratado pela proprietária do depósito de cosméticos para fazer a modelagem de um sistema para controle de estoque. Para

realizar a modelagem da análise para o projeto sob o aspecto estático do sistema, como você deverá identificar as classes? Quais serão os atributos e métodos? Como realizar o relacionamento dessas classes?

As classes desse sistema seriam: Cliente, Funcionário, Fornecedor, Produto, Venda, ItemDaVenda, Compra e ItemDaCompra.

O diagrama de classe para esse sistema ficaria conforme apresentado na Figura 3.5:

Figura 3.5 | Diagrama de classes para sistema de controle de estoque



Fonte: captura de tela do programa Astah Community elaborada pelo autor.

A Figura 3.5 apresenta as classes identificadas para o sistema de controle de estoque, que são: Funcionário, Fornecedor, Compra, Venda, ItemDaVenda, ItemDaCompra, Cliente e Produto. Cada uma delas apresenta os atributos e métodos expostos no diagrama. Um Funcionário realiza zero ou muitas Vendas para um Cliente. A Venda é composta de um ou muitos Itens da Venda e cada Item da Venda tem um Produto. O Fornecedor fornece uma ou muitas Compras, sendo cada uma composta de um ou muitos Itens da Compra e cada item inclui um Produto.

Assim, você realizou a modelagem do diagrama de classe para o sistema de controle de estoque do depósito de cosméticos, podendo aplicar esse tipo de modelagem para qualquer sistema de informação.

Avançando na prática

Construindo diagrama de classes

Descrição da situação-problema

Uma escola gostaria de criar um sistema para controle acadêmico e você foi contratado para realizar o projeto desse sistema utilizando o diagrama de classes. Após levantar os requisitos, você obteve aos seguintes dados:

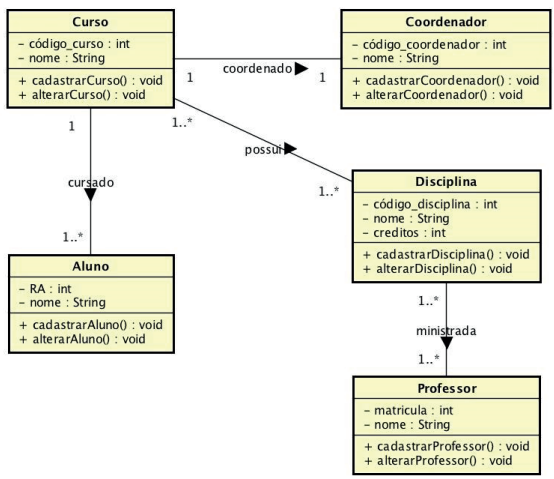
- Cada curso é coordenado por um coordenador.
- O aluno cursa apenas um curso.
- O curso é composto de disciplinas, que são ministradas por professores.

Considerando essas informações, faça a identificação de classes, atributos, métodos e seus relacionamentos e proponha um diagrama de classes.

Resolução da situação-problema

Para a resolução do problema do sistema para controle acadêmico, as classes identificadas são: Curso, Coordenador, Disciplina, Aluno e Professor. O diagrama de classe que apresenta a proposta das classes e relacionamentos é apresentado na Figura 3.6:

Figura 3.6 | Diagrama de classes para sistema de controle acadêmico



Fonte: captura de tela do programa Astah Community elaborada pelo autor.

Cada classe identificada tem sua lista de atributos e métodos, conforme apresentado no diagrama, e realiza seus relacionamentos. Um objeto da classe "Curso" se relaciona com um objeto da classe "Coordenador", um ou muitos objetos da classe "Disciplina" e um ou muitos objetos da classe "Aluno". A classe "Disciplina" se relaciona com um ou muitos objetos da classe "Professor".

Dessa forma, você elaborou o diagrama de classe para o sistema de controle acadêmico, identificando as classes, atributos, métodos e os relacionamentos entre as classes.

Faça valer a pena

1. Classes são blocos de construção mais importantes em sistemas que usam a orientação a objetos. Uma classe apresenta um grupo de objetos que possuem os mesmos atributos e operações. As classes podem incluir abstrações, que são partes do domínio do problema, assim como as classes que realizam a implementação.

A UML permite a representação de classes por meio de diagrama de classes. A classe é representada por um retângulo dividido em três compartimentos. Assinale a alternativa que apresenta corretamente os três compartimentos.
a) Nome da classe, atributos e operações.
b) Nome da classe, atributos e objetos.
c) Nome do objeto, atributos e operações.
d) Nome da classe, atributos e relacionamentos.
e) Nome do objeto, atributos e relacionamentos.

2. As classes colaboram entre si de várias formas, portanto, ao realizar a modelagem de um sistema, será necessário, além de identificar os itens que formam o vocabulário de sistema, modelar como esses itens se relacionam.

Na modelagem orientada a objetos, existem três tipos de relacionamentos. Assinale a alternativa que apresenta corretamente cada um deles.
a) Associação, generalização e dependência.
b) Associação, herança e dependência.
c) Associação, generalização e multiplicidade.
d) Associação, herança e multiplicidade.
e) Associação, agregação e multiplicidade.

3. A maioria das classes colabora com outras de várias formas, portanto, ao realizar a modelagem de um sistema, será necessário, além de identificar os itens que formam o vocabulário de sistema, modelar como esses itens se relacionam.

Um dos três tipos de relacionamentos é a generalização. Assinale a alternativa que apresenta corretamente a sua definição.
a) Uma generalização é um relacionamento entre superclasses (itens gerais) e subclasses (itens mais específicos).
b) Uma generalização é um relacionamento entre superclasses (itens específicos) e subclasses (itens mais genéricos).
c) Uma generalização é um relacionamento entre subclasses (itens gerais) e superclasses (tipos mais específicos).
d) Uma generalização é um relacionamento de dependência entre superclasses (itens gerais) e subclasses (tipos mais específicos).
e) Uma generalização é um relacionamento de agregação entre superclasses (itens gerais) e subclasses (tipos mais específicos).

Seção 3.2

Diagrama de classes avançado

Diálogo aberto

Prezado aluno, você já reparou que quando construímos uma casa nos preocupamos primeiramente com os materiais básicos, tais como cimento, areia, ferro, lajota, etc.? Quando avançamos na construção, percebemos que outros materiais mais detalhados devem ser adicionados ao projeto, tais como componentes elétricos e hidráulicos, os quais devem ser compatíveis com o que projetamos inicialmente. Já imaginou ter de passar um cano de água fria exatamente no local em que existe uma viga maciça de concreto? Ou ter de colocar uma caixa d'agua com tamanho superior ao reservado no telhado? Assim, realizar aprimoramentos em um projeto residencial e adicionar detalhes se faz necessário.

A construção de um software se assemelha muito à de uma casa, pois a princípio identificamos classes gerais do sistema e não nos preocupamos tanto com os detalhes dos atributos, métodos e relacionamentos. Dentro desse contexto, uma das formas de aprimorar as classes do sistema é realizar a modelagem refinada do diagrama de classes, que você aprenderá nesta seção.

Você foi contratado para auxiliar uma proprietária de um depósito de beleza a desenvolver um sistema de controle de estoque. Até o momento, você já identificou as classes do sistema e agora pode realizar a modelagem do diagrama de classes mais refinado, incluindo aspectos detalhados de atributos, operações e relacionamentos entre classes. Para realizar essa tarefa, buscaremos responder às seguintes questões: o que podemos fazer para refinar o diagrama de classes? Como modelar o relacionamento entre classes? Quais relacionamentos podem ser aplicados? Ao final da seção, você deverá criar um documento com o diagrama de classes refinado para o projeto do depósito de beleza.

A seguir, você aprenderá sobre o detalhamento de classes, tais como visibilidade e sintaxe formal para atributos e operações, bem como relacionamentos avançados de composição e agregação.

A modelagem do diagrama de classes refinado necessita de seu esforço, então bom trabalho!

Não pode faltar

Detalhamento de classes

Anteriormente, vimos que uma classe é composta de atributos e operações, os quais podem ser detalhados, por exemplo, pela visibilidade. Segundo Booch, Rumbaugh e Jacobson (2005), a visibilidade é um dos mais importantes detalhes que você pode especificar para atributos e operações. Ela especifica que objetos de classes podem visualizar e utilizar os atributos e operações. Esse conceito na programação orientada a objetos é chamado de encapsulamento, que é uma forma de "proteger" e "esconder" detalhes de uma classe. A UML determina, conforme os autores já citados, quatro níveis de visibilidade:

- Público: qualquer objeto de classe pode visualizar e utilizar os atributos ou operações com este tipo de visibilidade. É especificado pelo símbolo + antecedendo o nome do atributo ou operação.
- Protegido: qualquer descendente da classe (herança, visto anteriormente na generalização) é capaz de visualizar e utilizar os atributos ou operações com este tipo de visibilidade. É especificado pelo símbolo # antecedendo o nome do atributo ou operação.
- Privado: somente objetos da própria classe podem visualizar e utilizar os atributos ou operações com este tipo de visibilidade. É especificado pelo símbolo – antecedendo o nome do atributo ou operação.
- Pacote: somente objetos declarados no mesmo pacote (classes que estão dentro do mesmo diretório) podem usar os atributos ou operações com este tipo de visibilidade. É especificado pelo símbolo ~ antecedendo o nome do atributo ou operação.

Além da definição da visibilidade de atributos e operações, estes podem ser detalhados, conforme Booch, Rumbaugh e Jacobson (2005), com as seguintes sintaxes formais:

Atributos (sintaxe formal) – visibilidade:

$nome\_atributo : tipo = valor\_inicial \{domínio\}$:

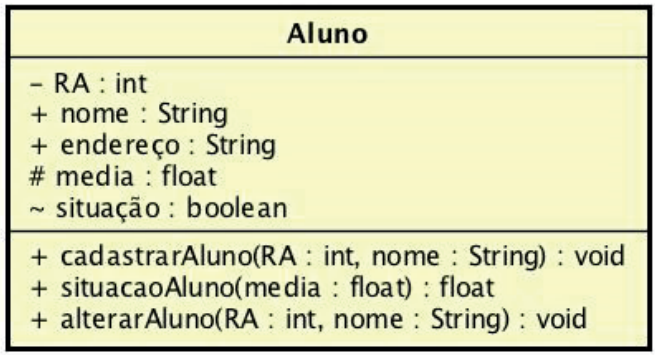
- A visibilidade é uma das vistas anteriormente, podendo ser de nível público, privado, protegido ou pacote.
- O $nome\_atributo$ é um identificador válido, tal como nome, endereço, etc.
- O $tipo$ é um tipo primitivo (inteiro, real, *boolean*, etc.), construído (*arrays* e registros) ou do tipo classe.
- $valor\_inicial$ corresponde a valores que podem ser usados para inicializar o atributo.
- $domínio$: são os valores válidos para o atributo.

Operações (sintaxe formal) – visibilidade $nome\_operação$ (lista de parâmetros): tipo de retorno {domínio}:

- A visibilidade é uma das vistas anteriormente, podendo ser de nível público, privado, protegido ou pacote.
- O $nome\_operação$ é um nome utilizado para identificar a operação, como cadastrar(), alterar(), etc.
- Lista de parâmetros são atributos (valores) que podem ser enviados para uma determinada operação, por exemplo, ao cadastrar um aluno, a função cadastrar recebe por parâmetro o RA e o nome do aluno.
- Tipo de retorno {domínio} é o tipo de retorno da operação. Uma função do tipo *void* não retorna parâmetros, mas no caso de retorno, deve ser especificado seu tipo, por exemplo a operação calculaMedia(float nota1, float nota2): *float*, é uma função que calcula a média do aluno, considerando a nota1 e a nota2 recebidas por parâmetro, e retorna um *float* (real), que é a média calculada.

A Figura 3.7 apresenta a construção de uma classe Aluno com a aplicação dos conceitos e visibilidade, e sintaxe formal para atributos e operações.

Figura 3.7 | Classe Aluno detalhada



Fonte: captura de tela do programa Astah Community, elaborada pelo autor.

A Figura 3.7 apresenta a classe Aluno com detalhe para os atributos e operações. O atributo "RA" tem visibilidade privada (símbolo −), podendo ser acessado apenas pelo objeto da própria classe. O tipo desse atributo é int (inteiros), podendo receber apenas valores inteiros (positivos, negativos e zero). Os atributos "nome" e "endereço" têm visibilidade pública (símbolo +), podendo ser acessados por qualquer objeto de qualquer classe. O tipo desses atributos é *String* (cadeia de caracteres). O atributo "media" tem visibilidade protegida (símbolo #), podendo ser acessado por classes que herdem a classe Aluno. O tipo desse atributo é *float* (números reais), recebendo valores reais (números fracionados e inteiros). O atributo "situação" tem visibilidade pacote (símbolo ~), podendo ser acessado apenas por classes que façam parte do mesmo pacote. O atributo é do tipo *boolean* (booleano, que aceita valores *true* ou *false*).

A classe Aluno apresenta as operações "cadastrarAluno()" que apresenta visibilidade pública, recebe dois parâmetros (RA e nome) e apresenta tipo de retorno void, isto é, não apresenta retorno. A operação "situacaoAluno()" apresenta visibilidade privada, pode ser acessada apenas por objetos da classe Aluno, recebe por parâmetro um parâmetro (media) e tem o tipo de retorno *float* (números reais), isto é, retorna um atributo do tipo *float*. A operação "alterarAluno()" apresenta visibilidade pública, recebe dois parâmetros (Ra e nome) e apresenta tipo de retorno *void*.

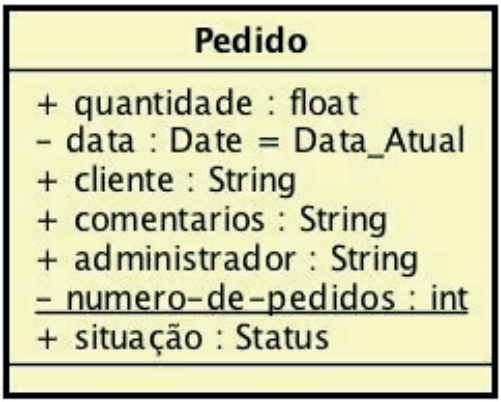
Assimile

Para um melhor entendimento de quando usar determinada visibilidade, lembre-se de que há vários papeis que os programadores podem assumir:

- Se a intenção é permitir que apenas você, quem escreve a classe, possa usá-la, então utilize *private*.
- Se a intenção é dar acesso a programadores que estenderão sua classe, então utilize *protected*.
- Se a intenção é compartilhar o que você está fazendo com outros programadores em um mesmo pacote, então utilize *package*.
- Se a intenção é que qualquer programador use a classe que você criou, então utilize *public*.

Além da sintaxe formal apresentada para os atributos e operações, outro importante detalhe, conforme Booch, Rumbaugh e Jacobson (2005), é poder especificar para os atributos o seu escopo. Por padrão, todo escopo é de instância, isto é, cada instância de objetos utiliza um valor único para atributos e operações. Visualmente não requer nenhuma notação adicional à que já vimos. A outra forma de especificar os atributos é por meio do escopo de estática, o qual representa que o atributo ou operação manterá apenas um valor para todas as instâncias dos objetos daquela classe. Também é conhecido por escopo de classe e sua notação é o sublinhado. Na Figura 3.8, é apresentada uma classe "Pedido", que contém vários atributos, tais como "quantidade", "data", "cliente", "comentários", "administrador", "numero-de-pedidos" e "situação". Perceba que o atributo "numero-de-pedidos" está sublinhado, representando que esse é um atributo estático, ou seja, seu valor é compartilhado por todos os objetos instanciados da classe "Pedido", tendo um valor único para todas as instâncias.

Figura 3.8 | Atributo com escopo de estática



Fonte: captura de tela do programa Astah Community, elaborada pelo autor.

Reflita

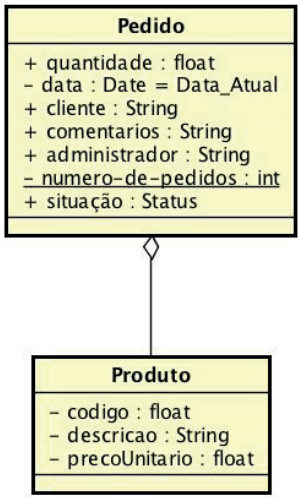
A propriedade de visibilidade da UML atende à semântica comum à maioria das linguagens de programação. Quais linguagens de programação suportam a propriedade de visibilidade?

Relacionamentos avançados

De acordo com Booch, Rumbaugh e Jacobson (2005), uma agregação é um tipo específico de associação que tem por objetivo diferenciar a parte do todo. Ela indica um relacionamento "parte-todo" entre os objetos das classes. Percebe-se que há agregação a partir de relacionamentos que incluem expressões como "consiste de", "contém" ou "é parte de".

Graficamente é representado por um pequeno losango (diamante) na extremidade da associação que tem a classe agregadora. A Figura 3.9 apresenta um exemplo de agregação, representando que um "Produto" está agregado a um "Pedido". Conseguimos notar que um produto "é parte de" um pedido, representando um relacionamento do tipo parte (Produto) e todo (Pedido).

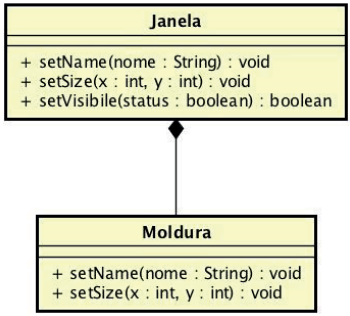
Figura 3.9 | Agregação entre classes



Fonte: captura de tela do programa Astah Community, elaborada pelo autor.

Uma agregação de composição ou simplesmente composição é uma variação da agregação. Isso significa que esse tipo de agregação indica que as partes somente existem se o todo existir, as partes "vivem" dentro do todo. Por exemplo, em um sistema de janelas, uma Moldura pertence exatamente a uma única Janela. Assim, ao criar uma Moldura em um sistema de janelas, você deve anexá-la a uma Janela que a conterá. Dessa maneira, quando você "destrói" a Janela e você acaba por destruir também a Moldura. A Figura 3.10 apresenta o exemplo de composição entre a classe Janela e Moldura.

Figura 3.10 | Composição entre classes



Fonte: captura de tela do programa Astah Community, elaborada pelo autor.

O Quadro 3.1 apresenta um detalhamento e utilização da simbologia de agregação e composição utilizado na UML.

Quadro 3.1 | Simbologia para agregação e composição

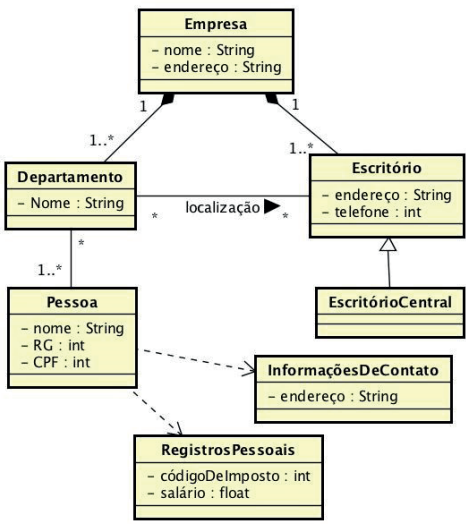
Símbolo	Utilização
	Símbolo utilizado para representar agregação entre classes. Identificamos agregação em relacionamentos do tipo "consiste de", "contém" ou "é parte de". O losango está na extremidade da associação que contém a classe agregadora.
	Símbolo utilizado para representar a composição entre classes. É uma variação da agregação e indica que as partes somente existem se o todo existir. O losango está extremidade da associação que contém a classe de composição.

Fonte: elaborado pelo autor.

Exemplificando

Considere o cenário de uma empresa que contém vários departamentos e escritórios. Os departamentos têm pessoas trabalhando nele e as pessoas possuem registradas informações de contato e registros pessoais. O escritório apresenta um escritório específico, um escritório central. O diagrama de classes que reflete o cenário apresentado é mostrado na Figura 3.11.

Figura 3.11 | Diagrama de classes e relacionamentos

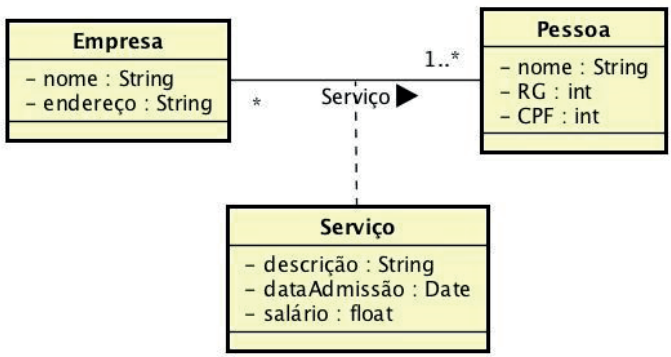


Fonte: Booch, Rumbaugh e Jacobson (2005, p. 108).

Em uma associação entre duas classes, a associação poderá ter propriedades.

Considere, por exemplo, um relacionamento entre Empresa e uma Pessoa: existe um Serviço que representa as propriedades desse relacionamento que se aplicam a exatamente um único par específico de Empresa e Pessoa. Poderíamos pensar em conectar Empresa a Serviço, e Pessoa a Serviço, porém, isso não ligaria uma determinada instância de Serviço ao par específico de Empresa e Pessoa. Na UML podemos modelar esse tipo de associação utilizando uma classe de associação, representada com um símbolo de classe anexado a uma associação por uma linha tracejada. A Figura 3.12 apresenta o exemplo de classe de associação Serviço, conectado às classes Empresa e Pessoa.

Figura 3.12 | Classes de associação



Fonte: adaptado de Booch, Rumbaugh e Jacobson (2005, p. 150).

Pesquise mais

Para uma melhor compreensão dos elementos avançados do diagrama de classes, realize a leitura do material a seguir:

BELL, D. **Fundamentos básicos de UML**: o diagrama de classes. IBM. 19 dez. 2016. Disponível em: <https://goo.gl/1HFaFs>. Acesso em: 18 ago. 2018.

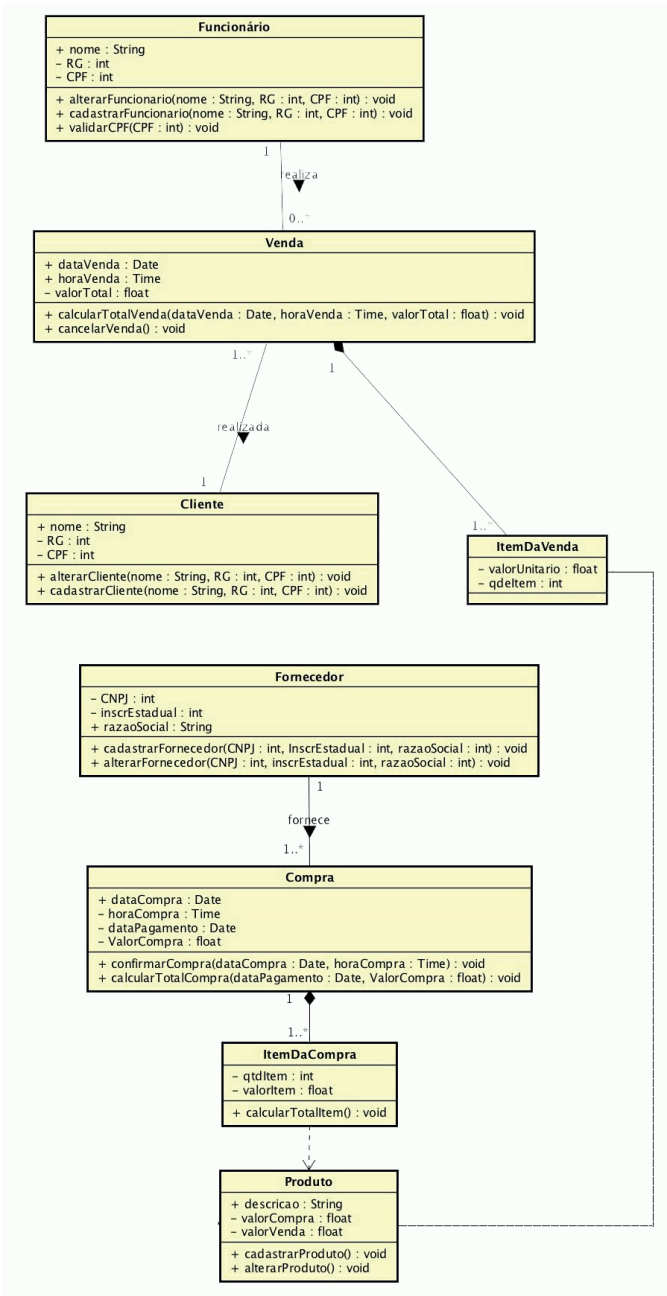
Nesta seção, você aprendeu sobre o detalhamento de classes, estudando sobre visibilidade e sintaxes formais para atributos e métodos. Você também estudou sobre os relacionamentos avançados de agregação e composição.

Sem medo de errar

Retomando a situação apresentada no início deste material, você foi contratado para auxiliar uma proprietária de um depósito de produtos de higiene pessoal para desenvolver um sistema de controle de estoque. Para tanto, você deveria responder às seguintes questões: como podemos realizar o refinamento do diagrama de classes e que relacionamentos podem ser aplicados entre as classes? Por fim, solicitou-se que você criasse um documento com o diagrama de classes refinado para o projeto do depósito de produtos de beleza.

Para poder refinar o diagrama de classes você deve primeiramente detalhar os seus atributos e operações. O conceito de visibilidade (pública, privada, protegida e pacote) deve ser aplicado aos atributos e operações, bem como deve-se usar a sintaxe formal destes. Os relacionamentos entre classes que podem ser aplicados são os de associação, generalização, dependência, agregação e composição, e cada um deles apresenta a forma que as classes se conectam. Uma sugestão para refinamento do diagrama de classes, com conceitos de visibilidade, sintaxe formal de atributos e operações e relacionamentos, é apresentado na Figura 3.13. Observe que foram detalhadas as classes com a visibilidade dos atributos e operações e também com a utilização dos relacionamentos de composição entre as classes ItemDaVenda e Venda (ItemDaVenda apresenta uma composição com Venda) e ItemDaCompra e Compra.

Figura 3.13 | Diagrama de classes refinado



Fonte: captura de tela do programa Astah Community, elaborada pelo autor.

Agora que você já tem uma visão do detalhamento das classes, complemente o diagrama de classes com mais detalhes, tais como parâmetros de operações, outros atributos, etc.

Avançando na prática

Detalhando o diagrama de classes

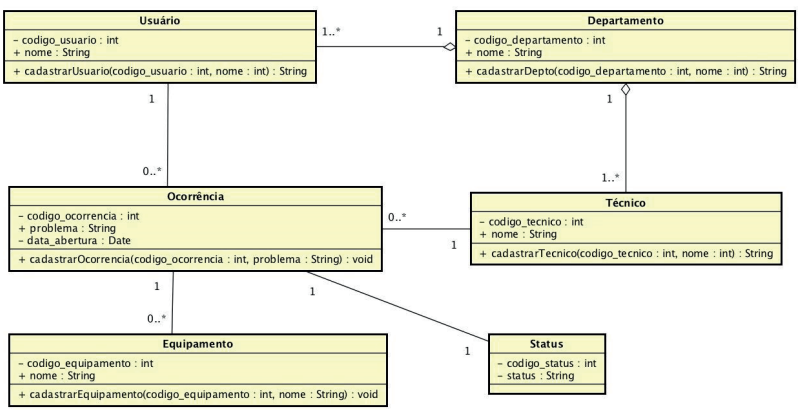
Descrição da situação-problema

O departamento de TI de uma fábrica gostaria de criar um sistema para registrar as solicitações de reparos de forma on-line. Você atua como analista de sistemas desse departamento e realizou os seguintes levantamentos de requisitos: o usuário de um determinado departamento, ao solicitar reparos, tem sua requisição direcionada a um determinado técnico. As ocorrências podem necessitar de equipamentos e cada ocorrência pode ser consultada por seu status. Com base nesse cenário, faça a proposta de um diagrama de classes detalhado.

Resolução da situação-problema

A proposta de diagrama de classes detalhado para o problema do sistema de registro de ocorrências on-line é apresentada na Figura 3.14.

Figura 3.14 | Diagrama de classes detalhado



Fonte: captura de tela do programa Astah Community, elaborada pelo autor.

Conforme apresentado, as classes identificadas são: Usuário, Departamento, Ocorrência, Técnico, Equipamento e Status. Cada classe contém atributos e operações com suas respectivas visibilidades e sintaxes formais. Os relacionamentos entre as classes são apresentados, havendo agregação entre as classes Usuário e Departamento, e Técnico e Departamento.

Faça valer a pena

1. Uma classe é composta de atributos e operações, os quais podem ser detalhados, por exemplo, pela visibilidade. A visibilidade é um dos mais importantes detalhes que você pode especificar para atributos e operações.

Assinale a alternativa correta, a qual apresenta os tipos de visibilidade que podem ser aplicados a atributos e operações.
a) Apenas público e privado.
b) Apenas público e protegido.
c) Apenas público e pacote.
d) Apenas público, privado e protegido.
e) Público, privado, protegido e pacote.

2. Além da sintaxe formal apresentada para os atributos e operações, outro importante detalhe é poder especificar para os atributos o seu escopo. O escopo determinará de que forma os atributos e operações são utilizados pelas instâncias de classes.

Assinale a alternativa que apresenta corretamente o(s) tipo(os) de escopo que pode(m) ser aplicado(s) a atributos e métodos.
a) Apenas escopo de instância.
b) Apenas escopo de estática.
c) Escopo de instância e escopo de estática.
d) Escopo de instância e escopo público.
e) Escopo de instância e escopo privado.

3. Uma agregação é um tipo específico de associação que tem por objetivo diferenciar a parte do todo. Ela indica um relacionamento "parte-todo" entre os objetos das classes. Percebe-se que há agregação a partir de relacionamentos que incluem expressões como "consiste de", "contém" ou "é parte de".

Assinale a alternativa que apresenta corretamente a variação da agregação a qual indica que as partes somente existem se o todo existir.

a) Associação.
b) Generalização.
c) Dependência.
d) Agregação.
e) Composição.

Seção 3.3

Mapeamento de classes

Diálogo aberto

Prezado aluno, você já reparou que quando construímos uma casa, precisamos comprar materiais básicos e de acabamento para a nossa obra, tais como, cimento, areia, ferro, lajota, madeira, pisos, revestimentos, etc. Quando vamos comprar esses materiais, todos os detalhes dos produtos estão armazenados em computador, tais como: quantidades, valores unitários, peso, dimensões, etc. Imagina se não houvesse como armazenar todos esses produtos e todo o estoque precisasse ser controlado manualmente? Dessa forma, os sistemas fazem uso de banco de dados, que armazenam todos os dados que o sistema deve manipular. Dentro desse contexto, uma das formas de realizar a modelagem de banco de dados é realizando o mapeamento de classes para o modelo relacional, utilizando as regras do mapeamento de classes e produzindo tabelas no formato de esquema.

Lembre-se de que você foi contratado para auxiliar uma proprietária de um depósito de produtos beleza a desenvolver um sistema de controle de estoque. Agora que já temos o diagrama de classes detalhado, podemos fazer o mapeamento de classes para o modelo relacional. Como podemos realizar essa tarefa? Quais tabelas serão produzidas a partir do diagrama de classes detalhado? Por fim, você deverá criar um documento com o mapeamento do diagrama de classes detalhado para o modelo relacional, para o projeto desse depósito.

Nesta seção você aprenderá sobre conceitos e aplicações de mapeamento de classes para o modelo relacional, regras do mapeamento das classes e mapeamento do diagrama de classes para tabelas no formato de esquema.

O processo de mapeamento de classes para o modelo relacional necessita de seu esforço, então, bom trabalho!

Não pode faltar

Mapeamento de classes para o modelo relacional

Segundo Booch, Rumbaugh e Jacobson (2005), em muitos sistemas que serão modelados, haverá dados que deverão ser armazenados em bancos de dados para serem recuperados posteriormente. Frequentemente, serão utilizados bancos de dados relacionais para armazenamento de dados. A UML é adequada para a modelagem de esquemas lógicos de bancos de dados, além dos próprios bancos de dados físicos.

Um banco de dados é uma coleção de dados relacionados, organizada de forma a possibilitar a manipulação de dados. Bancos de dados fazem parte do nosso cotidiano. Considere, por exemplo, uma matrícula em uma universidade. Nessa operação, são armazenados seus dados pessoais, tais como nome, endereço, RG, CPF, telefone, e-mail, bem como dados relacionados ao seu curso, como disciplinas, professores, notas e frequência.

Assim como a UML faz uso de diagramas para modelagem de diferentes aspectos do sistema, como o diagrama de classes, representando classes e relacionamento, para o projeto de banco de dados utilizamos um modelo de dados, que é um conjunto de conceitos para descrever a estrutura de um banco de dados, as operações para manipular essas estruturas e certas restrições as quais o banco de dados deve seguir. Um exemplo muito utilizado de modelo de dados é o modelo relacional.

Reflita

Dentre os sistemas gerenciadores de bancos de dados (SGBDs) mais utilizados comercialmente, estão os SGBDs relacionais. Além deles, quais outros tipos de SGBDs são utilizados?

O modelo relacional foi proposto em 1970 por EF Codd (IBM) e é utilizado em vários produtos comerciais, tais como DB2, Oracle, SQL Server, MySQL e PostgreSQL. Está relacionado com vários padrões do SQL (Structured Query Language), que é a linguagem de pesquisa declarativa padrão para banco de dados relacional.

O paradigma orientado a objetos é o mais difundido para o processo de desenvolvimento de software, por outro lado,

no mercado, a persistência é dominada por bancos de dados relacionais. Assim, o mapeamento de classes para o modelo relacional é uma necessidade cada vez mais importante no processo de desenvolvimento.

A abordagem relacional se baseia no princípio de que as informações em uma base de dados são consideradas relações matemáticas e estão representadas de maneira uniforme com o uso de tabelas bidimensionais.

Os SGBDs relacionais apresentam como vantagens o fato de serem extremamente confiáveis e eficientes, e o modelo de dados relacional foi criado para possibilitar a representação de uma grande variedade de problemas usando um conjunto de conceitos simples. Utiliza-se o SQL para fazer busca e recuperação dos dados de forma eficiente. Como desvantagem, podemos considerar a dificuldade em abstrair um problema do mundo real e traduzi-lo para um modelo de tabelas. Outro grande desafio ao utilizar o modelo relacional é que são necessários dois projetos, sendo um para o sistema e outro para o banco de dados. Um projeto é feito a partir do outro, por meio de regras que permitem realizar o mapeamento.

Assimile

Os bancos de dados surgiram da necessidade de armazenamento das informações de forma persistente e padronizada. Eles proporcionam uma maior facilidade de manutenção, segurança e integridade dos dados. A criação de um banco de dados ocorre a partir do levantamento e análises de requisitos para a implementação de uma aplicação; também deve-se realizar a modelagem conceitual dos dados, que corresponde à abstração das informações relevantes para que o sistema seja construído.

Regras para mapeamento

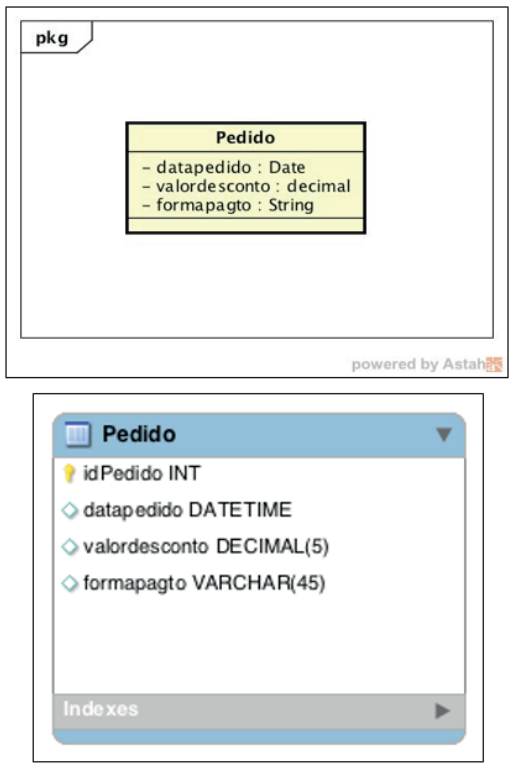
As classes são unidades que encapsulam atributos e operações. Os bancos de dados representam atributos, mas não representam operações. Podemos fazer uma analogia entre classes e tabelas e entre atributos e colunas.

Para realizar o mapeamento de classes para o modelo relacional, algumas regras devem ser seguidas:

- **Tabelas:** todas as tabelas devem ter uma chave primária, que é um campo da tabela o qual identifica unicamente um registro, não podendo ter seu valor repetido por nenhum outro campo. Uma chave primária para um aluno, por exemplo, seria o seu registro acadêmico (RA), que tem um valor numérico único para cada aluno. Assim, para o mapeamento de uma classe para o modelo relacional, deve ser criado um atributo chave.

Na Figura 3.15 é apresentado o mapeamento da classe Pedido para a tabela Pedido. A classe Pedido não apresenta um atributo chave, então, ao ser mapeada para a tabela Pedido, é criado um campo chave "idPedido". Os demais atributos da classe Pedido foram mapeados para campos da tabela, com seus respectivos tipos "datapedido", "valor desconto" e "formapagto".

Figura 3.15 | Mapeamento classe Pedido para tabela Pedido



Fonte: capturas de tela dos programas Astah Community e MySQL Workbench, elaborada pelo autor.

Pesquise mais

Para um melhor entendimento do mapeamento de um modelo de classes para um banco de dados relacional, faça a leitura do artigo da Revista SQL Magazine, *Mapeando um modelo de classes para um banco de dados relacional*, indicado a seguir:

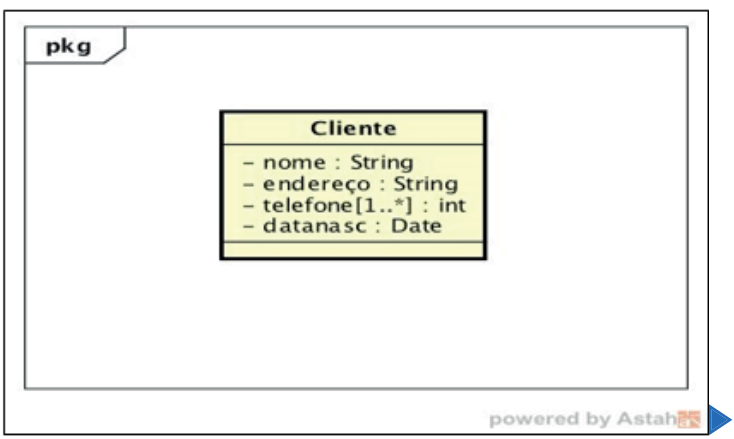
ZIMBRÃO, Geraldo. Mapeando um modelo de classes para um banco de dados relacional. **SQL Magazine**, v. 5, 2008. Disponível em: <https://goo.gl/PbohcY>. Acesso em: 18 ago. 2018.

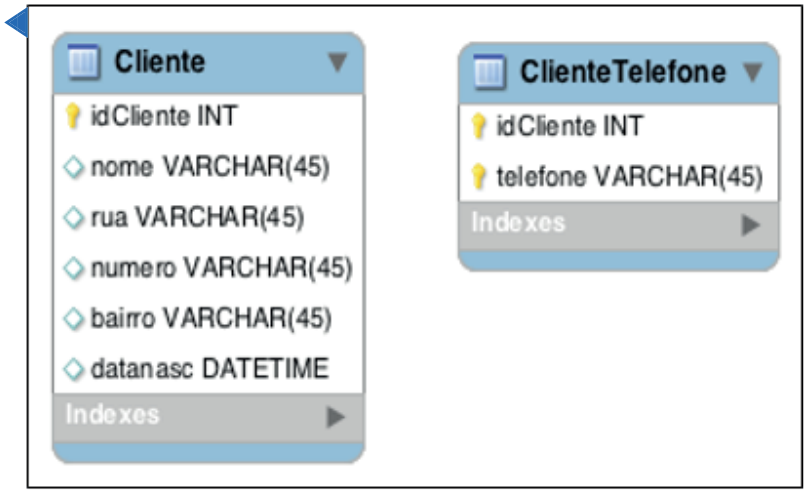
- Atributos: para o mapeamento de atributos, deve-se considerar que há três tipos:

- Os atributos simples são mapeados como coluna das tabelas.
- Os atributos compostos, como o endereço, que é composto de rua, número e bairro, são mapeados com as partes que o compõem.
- Os atributos multivalorados, como o telefone (pode haver vários números de telefone) são mapeados para uma nova tabela, contendo a chave primária da tabela de origem e pelo atributo multivalorados, ambos como chave.

A Figura 3.16 apresenta o mapeamento dos vários tipos de atributos.

Figura 3.16 | Mapeamento de atributos de classes para tabelas



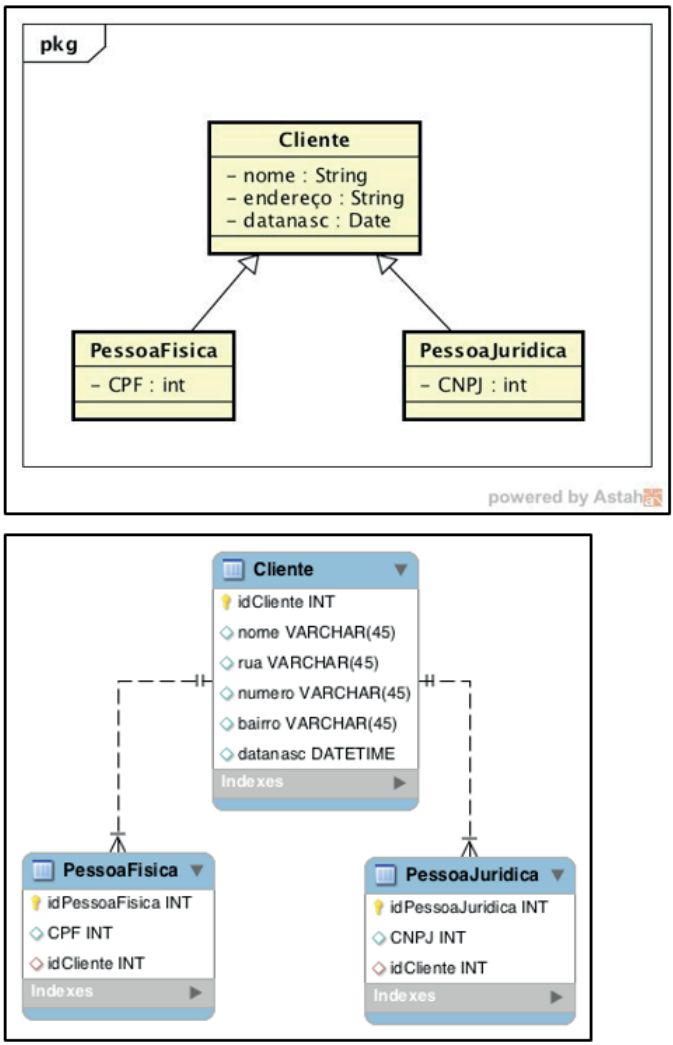


Fonte: capturas de tela dos programas Astah Community e MySQL Workbench, elaborada pelo autor.

Conforme apresenta a Figura 3.16, a classe "Cliente" é mapeada para a tabela cliente, contendo um campo chave "idCliente" e o mapeamento dos atributos simples e compostos (endereço, que é composto de rua, número e bairro) "nome", "rua", "numero", "bairro" e "datanasc". O atributo multivalorado "telefone" é mapeado para uma nova tabela "ClienteTelefone", contendo os atributos chave "idCliente" e "telefone".

- **Herança (generalização):** mapeamos a herança entre classes criando uma tabela para cada classe. Os atributos da tabela "pai" são os atributos gerais e os das tabelas "filhas" são os atributos específicos da classe mais a coluna de chave estrangeira que referencia a chave primária da tabela pai. A Figura 3.17 ilustra a representação de generalização do mapeamento do diagrama de classes para o diagrama entidade relacionamento. Cada classe foi mapeada para uma tabela, contendo os campos (atributos das classes). As tabelas "PessoaFisica" e "PessoaJurica" recebem como chave estrangeira (uma chave primária em outra tabela) "idCliente", que é a chave primária da tabela pai "Cliente".

Figura 3.17 | Mapeamento de herança de classes para tabelas

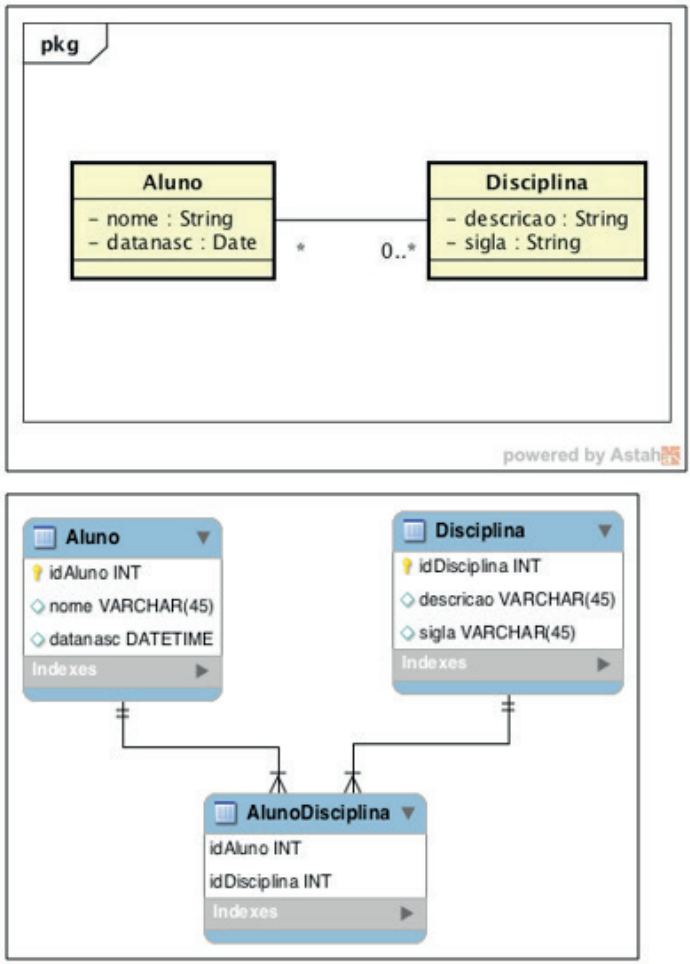


Fonte: capturas de tela dos programas Astah Community e MySQL Workbench, elaborada pelo autor.

- **Associações de muitos para muitos:** em relacionamentos de classes do tipo muitos para muitos, devemos criar uma tabela associativa que conterá a chave primária composta pelas chaves estrangeiras das tabelas associadas. A Figura 3.18 apresenta um relacionamento de muitos para muitos, em que um Aluno pode cursar zero ou muitas Disciplinas e uma Disciplina é cursada por muitos Alunos. Dessa forma,

o mapeamento, além das tabelas "Aluno" e "Disciplina", criou a tabela "AlunoDisciplina", que contém como chave estrangeira as chaves primárias "idAluno" e "idDisciplina".

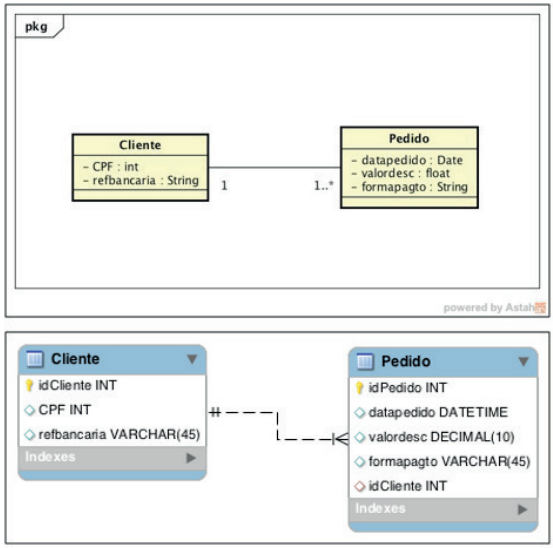
Figura 3.18 | Mapeamento de relacionamento muitos para muitos para tabelas



Fonte: capturas de tela dos programas Astah Community e MySQL Workbench, elaborada pelo autor.

- Associações de um para muitos: nesse tipo de associação, a chave primária da tabela do lado um é propagada como chave estrangeira para a tabela do lado muitos. Na Figura 3.19, a chave primária de Cliente "idCliente" é mapeada como chave estrangeira para a tabela Pedido.

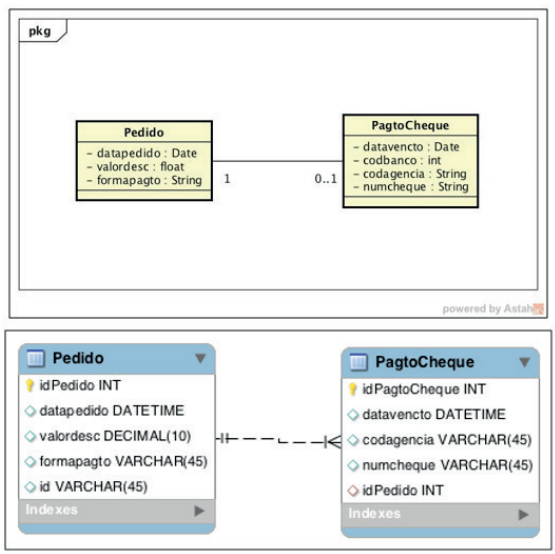
Figura 3.19 | Mapeamento de relacionamento um para muitos para tabelas



Fonte: capturas de tela dos programas Astah Community e MySQL Workbench, elaborada pelo autor.

- **Associações de um para um:** deve-se gerar as duas tabelas e uma delas receberá a chave estrangeira.

Figura 3.20 | Mapeamento de relacionamento um para um para tabelas

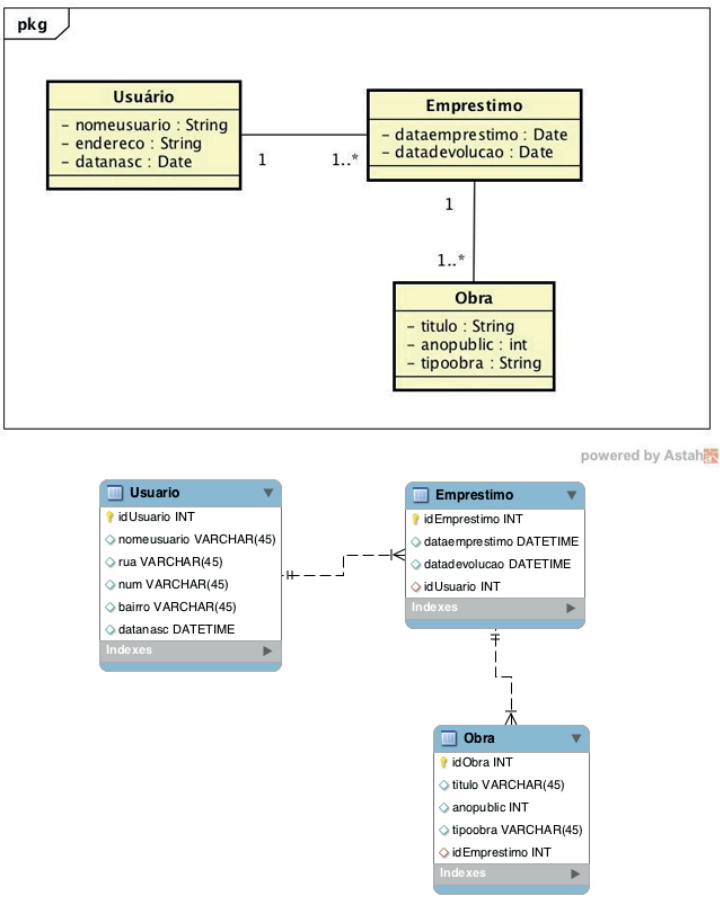


Fonte: capturas de tela dos programas Astah Community e MySQL Workbench, elaborada pelo autor.

Exemplificando

Considere o diagrama de classes que apresenta as classes e relacionamentos para uma biblioteca. O usuário realiza o empréstimo de uma obra e, em seguida, é apresentado o mapeamento desse diagrama de classes para o modelo relacional.

Figura 3.21 | Exemplo de mapeamento de classes para modelo relacional



Fonte: capturas de tela dos programas Astah Community e MySQL Workbench, elaborada pelo autor.

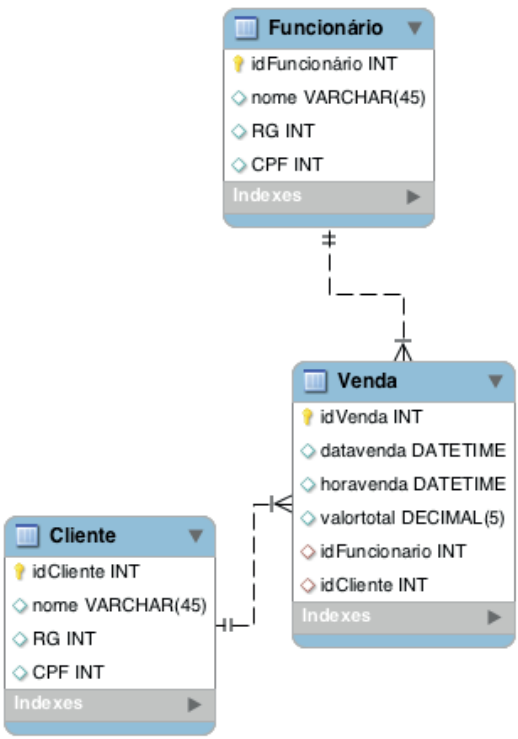
Como resultado do mapeamento, a tabela Emprestimo recebe a chave estrangeira "idUsuario" da tabela Usuario (relacionamento de um para muitos); e a tabela Obra recebe a chave estrangeira "idEmprestimo" da tabela Emprestimo (relacionamento de um para muitos).

Assim, aprendemos o processo de mapeamento do diagrama de classes para o modelo relacional, obtendo as tabelas e seus relacionamentos.

Sem medo de errar

Você foi contratado para auxiliar uma proprietária de um depósito de produtos de beleza a desenvolver um sistema de controle de estoque. Como podemos realizar o mapeamento das classes para o modelo relacional? Quais tabelas serão produzidas a partir do diagrama de classes detalhado? Podemos realizar o mapeamento do diagrama de classes para o modelo relacional seguindo as regras de mapeamento. Uma possível solução para o mapeamento das tabelas a partir do diagrama de classes detalhado é apresentada a seguir:

Figura 3.22 | Mapeamento das tabelas a partir do diagrama de classes detalhado



Fonte: captura de tela do programa MySQL Workbench, elaborada pelo autor.

Foram realizados os mapeamentos das tabelas Cliente, Venda e Funcionario. A tabela Venda recebe a chave estrangeira "idFuncionario" da tabela Funcionario (relacionamento de um para muitos) e a chave estrangeira "idCliente" da tabela Cliente (relacionamento de um para muitos).

Agora que você já visualizou um exemplo de modelagem, complete o diagrama entidade relacionamento com as demais tabelas que estão faltando, tais como Funcionário, Produto, Venda, etc.

Avançando na prática

Modelagem do diagrama de classes para o modelo relacional

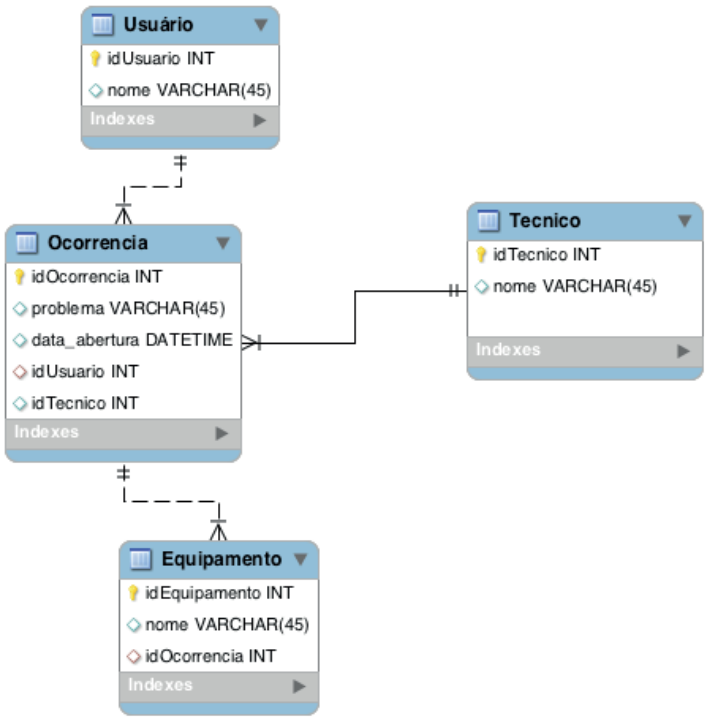
Descrição da situação-problema

Um projetista realizou a proposta de um diagrama de classes detalhado para criar um sistema a fim de registrar as solicitações de reparos de forma on-line. O usuário de um determinado departamento, ao solicitar reparos, tem sua requisição direcionada a um determinado técnico. As ocorrências podem necessitar de equipamentos e cada ocorrência pode ser consultada por seu status. Após a modelagem do sistema, você foi contratado para realizar a modelagem do diagrama de classes para o modelo relacional. Com base nesse diagrama, faça a proposta do diagrama entidade relacionamento.

Resolução da situação-problema

A proposta de modelagem do diagrama de classes para o modelo relacional é apresentada na figura a seguir:

Figura 3.23 | Modelagem do diagrama de classes para o modelo relacional



Fonte: captura de tela do programa MySQL Workbench, elaborada pelo autor.

Agora que você já visualizou um exemplo de modelagem, complete o diagrama entidade relacionamento com as demais tabelas que estão faltando, tais como, Departamento e Status.

Faça valer a pena

1. Assim como a UML faz uso de diagramas para modelagem de diferentes aspectos do sistema, por exemplo, o diagrama de classes que representa as classes e os relacionamentos, para o projeto de banco de dados, utilizamos um modelo de dados, que é um conjunto de conceitos para descrever a estrutura de um banco de dados.

Assinale a alternativa que apresenta, corretamente, o modelo de dados utilizado para o projeto de banco de dados.

a) Modelo relacional.
b) Modelo de entidades.
c) Modelo de dados.
d) Modelo relacionamento.
e) Modelo de tabelas.

2. Em muitos sistemas que serão modelados, haverá dados que deverão ser armazenados em bancos de dados para serem recuperados posteriormente. Frequentemente, serão utilizados bancos de dados relacionais para esse armazenamento.

Assinale a alternativa que apresenta, corretamente, os tipos de atributos mapeados.
a) Apenas atributos simples.
b) Apenas atributos simples e compostos.
c) Apenas atributos simples, compostos e multivalorados.
d) Apenas atributos simples e multivalorados.
e) Apenas atributos compostos e multivalorados.

3. O mapeamento de classes para o modelo relacional é uma necessidade cada vez mais importante no processo de desenvolvimento. A abordagem relacional se baseia no princípio de que as informações em uma base de dados são consideradas relações matemáticas e estão representadas de maneira uniforme com o uso de tabelas bidimensionais.

Assinale a alternativa que apresenta, corretamente, como é realizado o mapeamento de um para muitos.
a) A chave primária da tabela do lado um é propagada como chave estrangeira para a tabela do lado muitos.
b) A chave primária da tabela do lado muitos é propagada como chave estrangeira para a tabela do lado um.
c) Deve-se criar uma tabela associativa que contenha a chave primária composta pelas chaves estrangeiras das tabelas associadas.
d) A chave primária da tabela de qualquer um dos lados é propagada como chave estrangeira.
e) Não há propagação de chaves estrangeiras.

Referências

ANDRE OLIMPIO. **UML Diagrama de classe usando Astah**. Disponível em: <https://www.youtube.com/watch?v=d5tzVWvfpzI>. Acesso em: 1 ago. 2018.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: guia do usuário. Rio de Janeiro: Elsevier, 2005.

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

LARMAN, Craig. **Utilizando UML e padrões**: uma introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

OLIVEIRA, Lucas. **Entenda o diagrama de classes**. Disponível em: <https://www.youtube.com/watch?v=Tj3fNdXL-NE>. Acesso em: 1 ago. 2018.

Arquitetura de sistema de software

Convite ao estudo

Prezado aluno, já parou para analisar como o padrão de desenvolvimento de sistemas mudou nos últimos anos? Antigamente, o processo de desenvolvimento não fazia uso de diferentes tipos de arquiteturas de desenvolvimento, não utilizava diferentes elementos arquiteturais e padrões para orientar suas composições e restrições. Assim, desenvolver sistemas adequando o projeto a uma das várias arquiteturas disponíveis na literatura, apesar de não ser uma tarefa fácil, é totalmente viável e desafiador. Na construção de uma solução de software, após determinar os requisitos (funcionais e não funcionais) e realizar a modelagem das funções do sistema, aplicando o diagrama de casos de uso e de classe, o próximo passo é determinar a arquitetura de sistema de software, conhecendo os diferentes estilos e projetos de arquitetura, por meio de frameworks. Esse conhecimento vai lhe permitir conhecer e compreender arquiteturas relacionadas a projetos de sistemas, através das diferentes categorias e estilos arquiteturais.

Dois jovens empreendedores estão prestes a inaugurar um Pub em sua cidade. Nele, comercializarão comidas e bebidas ao estilo inglês. Já está tudo pronto com a instalação física do Pub, porém um sistema de controle também se faz totalmente necessário. Os jovens fazem questão de que o sistema tenha uma arquitetura sólida e confiável. Como você definiria a arquitetura do sistema? Quais estilos de arquitetura podem ser utilizados em um projeto? Quais frameworks e padrões de projeto podem ser aplicados? Como definir o design da arquitetura do software, considerando os elementos de

software e hardware? Você irá trabalhar esta e outras questões nesta unidade de ensino.

Para cumprir seu trabalho você aprenderá, nesta unidade, sobre o projeto da arquitetura, estilos de arquitetura e design da arquitetura de sistemas. Bom trabalho!

Seção 4.1

Arquitetura do projeto de sistemas

Diálogo aberto

Ao construirmos uma casa, primeiramente decidimos que tipo de construção iremos realizar: se uma edícula, uma casa térrea ou um sobrado, por exemplo. Para cada tipo de construção, determina-se o tipo de fundação (arquitetura) que será aplicada ao projeto. Para uma edícula ou casa térrea, pode-se optar por uma fundação utilizando baldrame com blocos de fundação (sapatas), já para um sobrado, utilizam-se estacas e brocas. O que vai determinar o tipo de fundação são as cargas (peso da casa) e resistências (tipo de solo). A construção de um software tem muitas dessas características, pois a arquitetura depende do tipo de software a ser construído, que é determinado pela elicitação e pelo projeto dos requisitos.

Você foi contratado para definir a arquitetura de um sistema para um Pub. Como você poderia definir a arquitetura do sistema por meio de elementos e padrões que direcionem seu desenvolvimento? Dentre os estilos de arquitetura, tais como, baseado em camadas (*Layered-based architecture*), modelo de visão e controle (MVC - *Model-View-Controller*), arquitetura orientada a serviço (SOA – *Service Oriented Architecture*) e desenvolvimento baseado em componentes (CBD – *Component-based Development*), quais podem ser aplicados ao seu projeto?

Para que essas perguntas sejam respondidas e possam auxiliar na definição da arquitetura do sistema para o Pub, você aprenderá, nesta seção, sobre o projeto da arquitetura, estilo de arquitetura em camadas (MVC), estilo de arquitetura orientada a serviços (SOA) e o desenvolvimento baseado em componentes (CBD). Ao final, produza um relatório com as respostas dos questionamentos e apresente aos proprietários do Pub.

Essas são etapas fundamentais para o processo de análise e projeto de um sistema, sendo de extrema importância para seu aprendizado e para as próximas etapas a serem realizadas. Bom trabalho!

Não pode faltar

De acordo com Pfleeger (2002), a arquitetura do software está relacionada à descrição de elementos arquiteturais sob os quais os sistemas serão construídos, às relações entre os elementos, aos padrões que orientem suas composições e às restrições sobre os padrões. A arquitetura do software permite definir quais são os elementos arquiteturais em termos computacionais e como eles se relacionam.

Segundo Pfleeger (2002), os elementos arquiteturais são: bancos de dados, servidores e clientes, e como eles se relacionam por meio de chamadas de funções, acesso a variáveis, banco de dados, etc. Na etapa de projeto do sistema, nos preocupamos com a arquitetura da aplicação e tecnologia. Segundo o modelo *Rational Unified Process* (RUP) a arquitetura do software objetiva:

- Definir a estrutura e composição do sistema;
- Definir a colaboração entre os elementos;
- Representar elementos estruturais em subsistemas.

A arquitetura é influenciada por fatores de implementação (PFLEEGER, 2002):

- Arquitetura de computador;
- Sistema operacional;
- Sistema Gerenciador de Banco de Dados (SGBD);
- Protocolos de rede;
- Linguagem de programação;
- Ambiente de interface gráfica;
- Necessidades de performance, portabilidade, etc.

O responsável pela arquitetura de software deve possuir alguns conhecimentos (PFLEEGER, 2002):

- Conhecimento dos requisitos das aplicações;

- Conhecimento das tecnologias para construção da arquitetura;
- Conhecimento dos processos de desenvolvimento para a aplicação a ser desenvolvida.

Segundo Pfleeger (2002), os estilos de arquitetura criam um conjunto de regras de projeto que determinam tipos de componentes, conectores e restrições para a sua composição, que podem ser usados para compor um conjunto de sistemas e subsistemas. Exemplos:

- Estilo arquitetural em camada;
- Estilo arquitetural orientado a serviços;
- Estilo arquitetural baseado em componentes.

Estilo Arquitetural em Camada

De acordo com Pfleeger (2002), os componentes são alocados em camadas, e cada um se comunica com os das camadas vizinhas. Por exemplo, um sistema de gerência de versões possui as seguintes camadas:

- Camada de sistema de gerenciamento de configuração;
- Camada de sistema de gerenciamento de objetos;
- Camada de sistema de banco de dados;
- Camada de sistema operacional.

O estilo arquitetural em camada apresenta as seguintes vantagens e desvantagens:

Vantagens

- Facilidade de entendimento compreensão;
- Facilidade de reparo;
- Desenvolvimento independente;

- Fácil reutilização.

Desvantagens

- Funcionalidade duplicada;
- Dificuldade em estruturar sistemas em camadas.

Model-View-Controller (MVC)

Segundo Pfleeger (2002), o MVC, criado na década de 70, é um padrão de arquitetura para aplicação, que busca separar o projeto em três camadas: modelo, visão e controlador. A separação permite maior manutenção do código e reutilização em outros projetos.

MVC – *Model-View-Controller* são sistemas interativos:

- *Model*: define o significado da aplicação e como ela se comporta;
- *View*: apresentação da parte visual da aplicação;
- *Controller*: realiza a gerência das interações do usuário como modelo e visão da aplicação.

O uso do MVC traz como benefício o isolamento de regras de negócio da lógica e da interface com o usuário. Assim, podemos ter muitas interfaces com o usuário sem a necessidade de modificar as regras de negócios, aumentando as possibilidades de reuso das classes.

As interfaces e regras de negócio se comunicam por um controlador, e é ele quem separa as camadas. Quando um evento de clique em botão, por exemplo, for executado na interface gráfica, a interface vai ser comunicar com o controlador, que irá se comunicar com as regras de negócios.

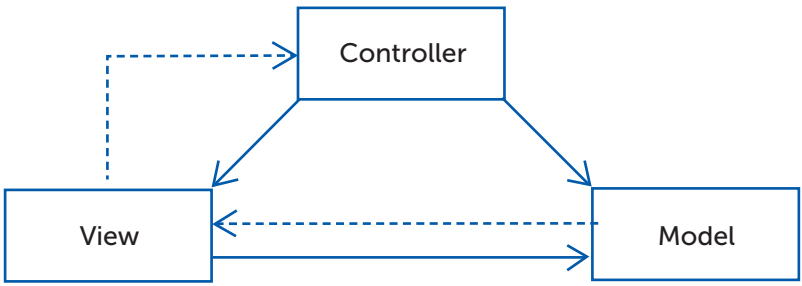
Exemplificando

Pense em uma aplicação financeira que realiza diversos tipos de cálculos, entre eles, o cálculo de juros. O *View* é o responsável pela interface gráfica, permitindo que o usuário insira os valores para os cálculos e escolha o cálculo que será realizado. Para o sistema saber

que um cálculo deve ser realizado, pode ser usado um botão que, quando clicado, irá gerar um evento.

O evento é como uma requisição a um mediador que realiza o preparo das informações para o cálculo, chamado de *Controller*. O controlador sabe quem deve realizar o cálculo. A operação matemática será executada pelo *Model*, que vai realizar a operação matemática e retornar o valo calculado para Controller, que repassará para a interface gráfica *View* para que seja exibido para o usuário.

Figura 4.1 | MVC e suas interações



Fonte: <https://goo.gl/JqZ1Vi>.

A Figura 4.1 apresenta os objetos do padrão MVC. O objeto *Controller* interpreta as entradas de mouse ou teclado e direciona as ações de usuário enviadas para *Model* ou para a janela de visualização (*View*). O objeto *Model* faz o gerenciamento de vários elementos de dados, sabendo o que o aplicativo quer realizar, sendo ele quem modela o problema que se quer resolver. O objeto *View* é o responsável por exibir ao usuário as informações por meio de gráficos e textos.

Service Oriented Architecture (SOA)

De acordo com Pfleeger (2002), a arquitetura orientada a serviços é um modelo para planejamento de estratégia alinhado com as necessidades de negócios da organização. Serviços são módulos de negócios ou funções que apresentam interfaces que são chamadas por mensagens. Todas as funções que serão implementadas devem

estar disponíveis no formato de serviços. Os serviços se conectam por um barramento de serviços (*Enterprise Service Bus* – ESB). O ESB fornece as interfaces ou contratos, estando disponíveis por *web services*. Um dos elementos mais importantes de SOA é o ESB (Barramento de Serviços Corporativos), que oferece uma camada de abstração que permite integrar aplicativos.

Segundo Pfleeger (2002), a SOA é uma arquitetura que integra a tecnologia com componentes de serviços e apresenta facilidade para alterações, redução de custo e reutilização de serviços. O foco da SOA é construir e disponibilizar serviços de negócio, evitando replicação de dados. A SOA tem por objetivo a organização e utilização de recursos distribuídos em diferentes domínios, permitindo que as aplicações compartilhem dados, mesmo que executadas em sistemas operacionais e linguagens diferentes.

Considere, por exemplo, uma empresa que possui diversos departamentos usando sistemas e arquiteturas diferentes, processando dados que necessitam ser acessíveis aos clientes. A metodologia SOA visa padronizar a forma dos sistemas se comunicarem, considerando as diferentes plataformas e linguagens, e aprensenta algumas vantagens e desvantagens:

Vantagens

- Reutilização: o serviço pode ser reutilizado por outras aplicações;
- Produtividade: com do reuso, há um ganho no tempo de desenvolvimento;
- Alinhamento com negócio: a área de negócio visualiza os processos alinhados com a tecnologia;
- Padronização: baseia-se no uso de padrões.

Desvantagens

- Complexidade: necessidade de gerenciamento de grande quantidade de serviços;
- Performance: depende do servidor em que o serviço está publicado, bem como da rede;

- Disponibilidade: queda na rede ou servidor deixa os serviços indisponíveis;
- Segurança: como os serviços estão em rede, qualquer aplicativo pode usar esse serviço e os dados podem ser interceptados.

Assimile

Usar tecnologias e aplicações heterogêneas em corporações é uma realidade. A SOA permite a reutilização de aplicações e opera entre aplicações e tecnologias heterogêneas, mas esse conceito se depara com algumas barreiras técnicas. Alguns problemas comuns:

- Desconhecimento da complexidade da SOA;
- Falta de profissionais capacitados para utilizar a SOA;
- Mudanças na organização.

Component-based Development (CBD)

De acordo com Pfleeger (2002), o desenvolvimento baseado em componentes (CBD) tem por objetivo o desenvolvimento de software utilizando partes já existentes. A construção de novas soluções pela combinação dos componentes, faz a qualidade aumentar e permite maior suporte ao rápido desenvolvimento. O CBD possibilita uma melhor manutenção dos sistemas de software, permitindo que os sistemas recebam atualizações por meio da integração de novos componentes ou da atualização de componentes existentes.

Pfleeger (2002) afirma que um componente é uma parte reutilizável do sistema. Componentes de softwares reutilizáveis executam funções específicas, com documentação e condição de reuso definida. O desenvolvimento baseado em componentes melhora a produtividade e qualidade do software desenvolvido, pois usa partes de código ou serviços existentes, facilitando a atualização do software, substituindo os componentes existentes por atualizados.

O repositório de componentes é uma biblioteca de componentes de software reusáveis que devem atender às necessidades dos

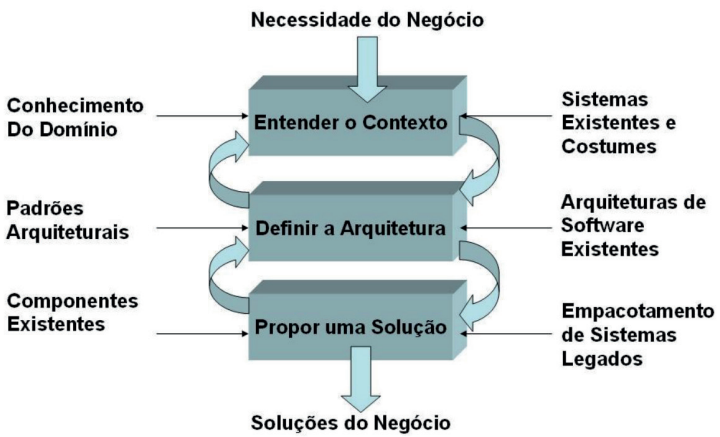
desenvolvedores de aplicações. Para que seja possível a reutilização de um componente, o mesmo deve ser identificável. Alguns benefícios relacionados à reutilização:

- Reduzir o custo e tempo de desenvolvimento;
- Componentes são desenvolvidos paralelamente e possuem serviços independentes e bem definidos;
- Aumento da qualidade com a utilização de componentes testados;
- Facilidade de manutenção e atualização.

Reflita

Dentre os padrões de arquitetura estudados, qual modelo se adequa melhor ao padrão de desenvolvimento orientado a objetos, e de que forma esse modelo se destaca dos demais?

Figura 4.2 | Elementos de uma abordagem CBD



Fonte: Brown (2000 apud BLOIS, 2006, p. 15).

A Figura 4.2 apresenta um conjunto de elementos e alternativas para desenvolvimento baseado em componentes. Uma vez definido o contexto, define-se a arquitetura e, ao se propor uma solução, verificam-se os componentes existentes para a solução do negócio.

Pesquise mais

Para um maior conhecimento de *Model-View-Controller* (MVC), Software *Oriented Architecture* (SOA) e *Component-basead Development* (CBD), faça a leitura do material a seguir:

BACELO, A. **Arquitetura de Software:** Conceitos e Tendências. Jornada Acadêmica – FACIN. 25 a 27 ago. 2010. Disponível em: <https://www.inf.pucrs.br/jornada.facin/jafacin_2010/palestras/ArquiteturaDeSoftware.pdf>. Acesso em: 18 jul. 2018.

Dessa forma, você estudou sobre a arquitetura de software, conhecendo os estilos arquiteturais em camada (MVC - *Model-View-Controller*), orientado a serviços (SOA - *Service-Oriented Architecture*) e baseado em componentes (CBD - *Component-basead Development*), avaliando suas aplicações, vantagens e desvantagens.

Sem medo de errar

Você foi contratado para definir a arquitetura de um sistema para um Pub. Como você realizaria a definição da arquitetura do sistema por meio de elementos e padrões que direcionem seu desenvolvimento? Quais estilos de arquitetura podem ser aplicados ao seu projeto?

Para responder a essas perguntas, você deve conhecer as principais arquiteturas de sistemas de software, tais como camadas *Model-View-Controller* (MVC), Service *Oriented Architecture* (SOA) e *Component-based Development* (CBD). Após conhecer os padrões, você deve avaliar qual padrão se aplicaria ao sistema do Pub. Por exemplo, você poderia escolher o desenvolvimento baseado em componentes (CBD), que utiliza componentes prontos e apresenta redução de custo e tempo de desenvolvimento, com componentes sendo desenvolvidos paralelamente, e que possui serviços independentes, bem definidos e com aumento da qualidade, com a utilização de componentes testados. Crie uma apresentação contendo os seguintes conteúdos:

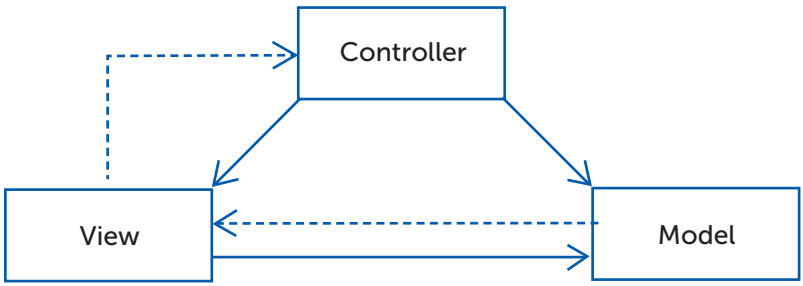
1. Arquitetura do sistema;
2. Estilos de arquitetura;

3. Estilo de arquitetura aplicada ao projeto do Pub.

Seguem algumas dicas para elaboração do relatório:

Retomando a Figura 4.1, que caracteriza o padrão MVC e suas interações, podemos notar que o objeto *Controller* interpreta as entradas de mouse ou teclado e direciona as ações de usuário enviadas para o *Model* ou para janela de visualização (*View*).

Figura 4.1 | MVC e suas interações



Fonte: <https://goo.gl/JqZ1Vi>.

Tratando-se do estilo de arquitetura, a SOA tem por objetivo organizar e utilizar recursos distribuídos em diferentes domínios, permitindo que as aplicações compartilhem dados, mesmo que executadas em sistemas operacionais e linguagens diferentes.

Leve em consideração que na abordagem CBD é necessário definir o contexto para depois definir a arquitetura e, ao propor uma solução, verificam-se os componentes existentes para a solução do negócio

Avançando na prática

Arquitetura de Sistema para Gerenciamento de Notas e Frequências

Descrição da situação-problema

Um professor gostaria de criar um sistema que determinasse automaticamente as médias e frequências dos alunos. Utilizando a arquitetura MVC, você foi contratado para projetar a arquitetura do

sistema para o professor. De que forma podemos utilizar a arquitetura MVC para elaborar a arquitetura do sistema de gestão automática de médias e frequências? Como você realizaria essa tarefa?

Resolução da situação-problema

Uma possível solução é iniciar o processo pela *View*, criando a interface gráfica que permita ao professor inserir as notas e frequências dos alunos e, ao final, com um clique em um botão, calcular notas e frequência para gerar o evento. O evento é como um pedido a um intermediador que prepara as informações para o cálculo, chamado de *Controller*. O controlador sabe quem executará o cálculo. O cálculo das médias e frequências é executado pelo *Model*. O *Model* vai realizar o cálculo da média e frequência dos alunos e retornar os valores calculados para o *Controller*, que os repassará para a interface gráfica *View*, que, por sua vez, irá exibir para o professor a situação do aluno.

Faça valer a pena

1. A arquitetura de software está relacionada à descrição de elementos arquiteturais sob os quais os sistemas serão construídos, às relações entre os elementos, aos padrões que orientem suas composições e às restrições sobre os padrões. A arquitetura do software permite definir quais são os elementos arquiteturais em termos computacionais e como eles se relacionam.

Assinale a alternativa que indica corretamente os estilos de arquitetura de sistemas:

a) Apenas arquitetura em Camadas.
b) Apenas arquitetura orientada a Serviços.
c) Apenas arquitetura baseada em Componentes.
d) Apenas arquitetura em Camadas e orientada a Serviços.
e) Arquitetura em Camadas, orientada a Serviços e baseada em Componentes.

2. O uso do MVC traz como benefício o isolamento de regras de negócio da lógica e da interface com o usuário. Assim, pode-se haver várias interfaces com o usuário sem a necessidade de modificar as regras de negócios, aumentando as chances de reuso das classes.

Assinale a alternativa que apresenta corretamente os componentes do MVC:
a) *Model, View, Controller.*
b) *Model, View, Composition.*
c) *Maps, View, Controller.*
d) *Maps, View, Composition.*
e) *Model, Verify, Controller.*

3. Arquitetura orientada a serviços é um modelo de planejamento de estratégia alinhado com as necessidades de negócios de uma organização. Serviços são módulos de negócios ou funções que apresentam interfaces que são chamadas por mensagens.

Os serviços são conectados por meio de um barramento de serviços. Assinale o nome desse barramento de serviço.

a) *Enterprise Serial Bus.*
b) *Enterprise Service Bus.*
c) *Enterprise Task Bus.*
d) *International Service Bus.*
e) *International Serial Bus.*

Seção 4.2

Arquitetura de frameworks e padrões de projeto

Diálogo aberto

Prezado aluno, ao começarmos a fase de acabamento de uma casa ou apartamento, podemos optar por vários tipos de padrões. Por exemplo, podemos realizar diversos tipos de acabamento, utilizando pisos e revestimentos cerâmicos para área externa (quintal e garagem), interna úmida (banheiros, cozinhas e lavanderias) e interna seca (salas e quartos), com tipo de acabamento polido, esmaltado, esmaltado acetinado ou rústico, com espaçamento entre peças *bold* (arredondado) ou retificado, e com diversos tipos de tamanhos e estampas. De modo similar, quando vamos construir um software, podemos optar por vários tipos de padrões para a implementação (programação) orientada a objeto de um projeto.

Você foi contratado para definir a arquitetura de um sistema para um Pub. Uma vez definidos os estilos de arquitetura que podem ser utilizados em nosso projeto, é chegada a hora de conhecer os principais padrões de projeto. Quais padrões de projeto podem ser aplicados ao sistema para o Pub? Nesta seção, você aprenderá sobre as característica e aplicações do projeto da arquitetura, bem como os principais padrões de projeto, tais como *Composite*, *Observer*, *Strategy*, *Factory Method*, *Mediator* e *Façade*.

Essas são etapas fundamentais para o processo de análise e projeto de um sistema, sendo de extrema importância para seu aprendizado e para as próximas etapas a serem realizadas. Bom trabalho!

Não pode faltar

Projeto da Arquitetura

Segundo Gamma et al. (2007), os bons projetistas não resolvem cada novo problema a partir do zero, em vez disso, reutilizam soluções que já funcionavam no passado. Com isso, temos padrões de classes que são usadas frequentemente em muitos sistemas orientados a objetos.

Os padrões tornam os projetos orientados a objetos mais flexíveis e reutilizáveis, ajudando os projetistas a reutilizar projetos e arquiteturas, baseando novos projetos na experiência anterior, ajudando a escolher alternativas de projeto que possibilitam a um sistema ser reutilizável e melhorar a documentação e manutenção de sistemas, ao fornecer uma especificação explícita de interações entre classes e objetos.

Um padrão descreve um problema e sua solução de tal forma que você possa usar a solução muitas vezes. Em geral, um padrão possui quatro elementos essenciais, de acordo com Gamma et al. (2007):

- Nome do padrão: referência que usamos para descrever um padrão de projeto, suas soluções e consequências. Atribuir um nome a um padrão aumenta o vocabulário do projeto e permite projetar em um nível mais alto de abstração. O nome torna mais fácil pensar sobre projetos e comunicá-los;
- Problema: descreve como aplicar o padrão, explicando o problema e seu contexto. Ele pode descrever problemas de projeto específicos, tais como representar algoritmos como objetos ou descrever estruturas de classes ou objetos;
- Solução: descreve os componentes do padrão de projeto, seus relacionamentos, responsabilidades e colaborações. O padrão fornece uma descrição abstrata de um projeto e de como elementos gerais (classes e objetos) o resolvem;
- Consequências: são os resultados e análises das vantagens e desvantagens da aplicação do padrão.

O padrão de projeto faz as identificações das classes e instâncias participantes, de seus papéis, colaborações e distribuição de responsabilidades. Cada padrão de projeto é voltado a um problema ou tópico particular do projeto orientado a objetos, descrevendo em qual situação pode ser aplicado, bem como os custos e benefícios de sua utilização.

Pesquise mais

Para um maior conhecimento sobre padrões de projeto, realize a leitura do artigo a seguir, focando nos vários padrões de criação, estruturais e comportamentais.

LEITE, A. F. Conheça os Padrões de Projeto. **DevMedia.** 2005. Disponível em: <https://www.devmedia.com.br/conheca-os-padroes-de-projeto/957>. Acesso em: 27 ago. 2018.

Dentre os principais padrões de projeto, destacam-se, segundo Gamma et al. (2007):

- *Composite*: compõem objetos que representam hierarquias do tipo parte-todo, utilizando a estrutura de árvore de dados;
- *Façade:* fornece uma interface unificada para um conjunto de interfaces em um subsistema. O *Façade* define uma interface de nível mais alto, que torna o subsistema mais fácil de usar;
- *Factory Method:* define uma interface para criar um objeto, mas permite às subclasses decidirem qual classe a ser instanciada;
- *Mediator:* define um objeto que encapsula a forma como um conjunto de objetos interage;
- *Observer:* define uma dependência um-para-muitos entre objetos, de modo que, quando um objeto mudar de estado, seus dependentes serão automaticamente atualizados;
- *Strategy:* define uma família de algoritmos, encapsulando cada um deles e os torna intercambiáveis. O *Strategy* permite que o algoritmo varie independentemente dos clientes que o utilizam.

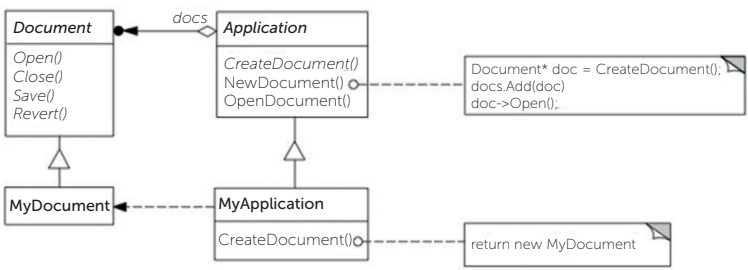
De acordo com Gamma et al. (2007), cada padrão possui uma finalidade, podendo ser de criação, estrutural ou comportamental:

- Padrões de criação realizam o processo de criação de objetos. *Factory Method* é um exemplo de padrão de criação;
- Padrões estruturais realizam a composição de classes ou objetos. *Composite* e *Façade* são exemplos de padrões estruturais;
- Padrões comportamentais caracterizam como classes e objetos interagem entre si e distribuem responsabilidades. *Mediator*, *Observer* e *Strategy* são exemplos de padrões comportamentais.

Factory Method

De acordo com Gamma et al. (2007), define uma interface para criar um objeto, permitindo que suas subclasses escolham que classe instanciar. O *Factory Method* permite aditar a instanciação para subclasses. Considere um *framework* para aplicações que apresenta múltiplos documentos para o usuário. Duas abstrações-chave nesse framework são as classes *Application* e *Document*. As duas classes são abstratas, e os clientes devem criar as subclasses para realizar suas implementações. Por exemplo, para criar uma aplicação de desenho, definimos as classes *DrawingApplication* e *DrawingDocument*. A classe *Application* faz a administração de *Documents*. Como a subclasse *Document* a ser instanciada é própria da aplicação específica, a classe *Application* sabe quando um documento deve ser criado, mas não que tipo de *Document* criar. Um dilema surge: o *framework* deve instanciar classes, mas ele somente tem conhecimento de classes abstratas, as quais não pode instanciar. O padrão *Factory Method* oferece uma solução, pois encapsula o conhecimento sobre a subclasse de *Document* que deve ser criada e movendo esse conhecimento para fora do *framework*. A Figura 4.3 ilustra o padrão *Factory Method*.

Figura 4.3 | Padrão *Factory Method*



Fonte: Gamma et al. (2007, p.113).

A Figura 4.3 ilustra a subclasse de *Application* (*MyApplication*), que redefine a operação abstrata *CreateDocument* que retorna a subclasse apropriada de *Document*. Uma vez instanciada a subclasse de *Application*, pode-se então instanciar *Documents*. Chamamos *CreateDocument* de um ***factory method***, por que ele é responsável pela construção de um objeto. Utiliza-se o padrão *Factory Method*, segundo Gamma et al. (2007), quando:

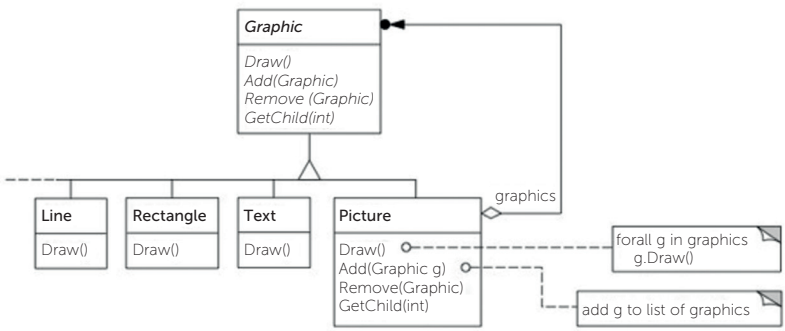
- Uma classe não consegue antecipar a classe de objetos a ser criado;
- Uma classe quer permitir às suas subclasses especificarem os objetos que criam;
- Classes responsabilizam subclasses auxiliares e você quer localizar o conhecimento de qual subclasse auxiliar foi delegada.

Composite

De acordo com Gamma et al. (2007), *Composite* é uma forma de representar hierarquias do tipo parte-todo, utilizando a estrutura de dados árvore, permitindo aos clientes tratarem de maneira uniforme objetos individuais e composições de objetos.

Aplicações gráficas permitem aos usuários construir diagramas complexos a partir de componentes simples, em que o usuário pode agrupar componentes para formarem componentes maiores. Uma implementação poderia definir classes primitivas gráficas, tais como, Texto, Linhas, Retângulos, além de outras classes que funcionam como containers para essas primitivas. O padrão *Composite* descreve como usar a composição recursiva. A Figura 4.4 ilustra o padrão *Composite*.

Figura 4.4 | Padrão *Composite*



Fonte: Gamma et al. (2007, p.160).

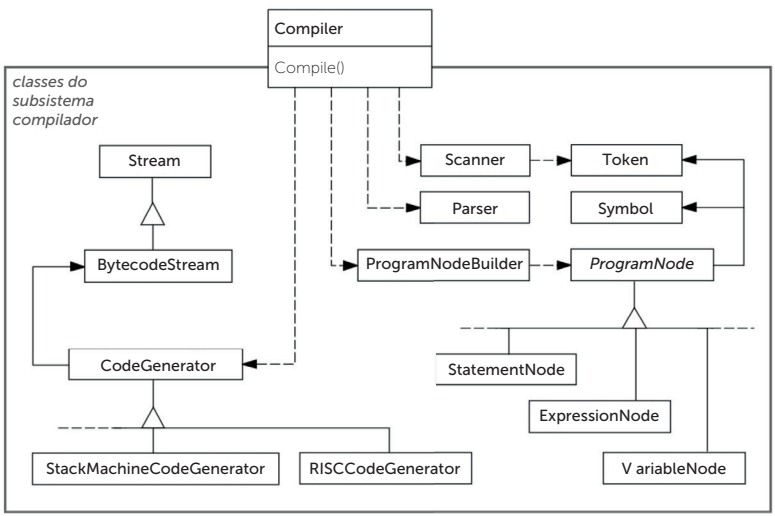
Conforme apresentado na Figura 4.4, o padrão *Composite* é uma classe abstrata, chamada *Graphic*. A *Graphic* declara

operações tais como *Draw*, que são específicas de objetos gráficos. As subclasses *Line*, *Rectangle* e *Text* definem objetos gráficos primitivos, implementando *Draw* para desenhar linhas, retângulos e textos. A classe *Picture* define um agregado de objetos *Graphic*, implementando *Draw* em seus filhos. Usamos o padrão *Composite,* segundo Gamma et al. (2007), para representar hierarquias partes-todo de objetos;

Façade

Segundo Gamma et al. (2007), apresenta uma interface unificada para um conjunto de interfaces em um subsistema. *Façade* define uma interface de nível mais alto, que torna o subsistema mais fácil de ser usado. Estruturar um sistema utilizando subsistemas reduzir sua complexidade. A Figura 4.5 ilustra o padrão *Façade*.

Figura 4.5 | Padrão *Façadea*



Fonte: Gamma et al. (2007, p.180).

A Figura 4.5 apresenta um ambiente de programação que fornece acesso às aplicações para o seu subsistema compilador. Esse subsistema contém classes, tais como, *Scanner*, *Parser*, *ProgramNodeBuilder* e *CodeGenerator*, que implementam o compilador. Para fornecer uma interface de nível mais alto, que possa isolar os clientes dessas classes, o

sistema faz uso de uma classe *Compiler*, que funciona como um *Façade* (fachada), oferecendo aos clientes uma interface única e simples para o subsistema compilador.

Usamos o padrão *Façade,* segundo Gamma et al. (2007), quando:

- Há a necessidade de oferecer uma interface simples para um subsistema complexo;
- Queremos estruturar seus subsistemas em camadas.

Mediator

Segundo Gamma et al. (2007), define um objeto que encapsula a forma de interação de um conjunto de objetos. O *Mediator* promove o acoplamento fraco, evitando que os objetos se refiram uns aos outros explicitamente.

O projeto orientado a objetos incentiva a distribuição de comportamento entre vários objetos. Essa distribuição resulta em uma estrutura de objetos com muitas conexões entre eles e, na pior das situações, cada objeto conhece todos os outros objetos.

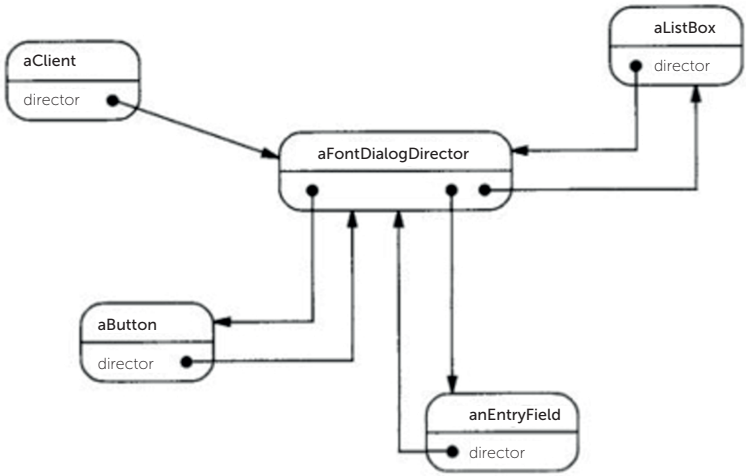
Considere como exemplo a implementação de caixas de diálogo em uma interface gráfica de usuário. Uma caixa de diálogo usa uma janela para apresentar uma coleção de *widgets*, tais como botões, menus e campos de entrada, conforme a Figura 4.6 a seguir:

Figura 4.6 | Interface gráfica de usuário

Fonte: Gamma et al. (2007, p.258).

A Figura 4.7 apresenta a *FontDialogDirector* que pode ser um mediador entre os *widgets* numa caixa de diálogo. Um objeto *FontDialogDirector* conhece os *widgets* de um diálogo e coordena sua interação, funcionando como um centro de comunicações para os *widgets*:

Figura 4.7 | Padrão *Mediator*



Fonte: Gamma et al. (2007, p.259).

Usamos o padrão *Mediator* quando:

- A reutilização de um objeto é difícil porque ele referencia e se comunica com muitos outros objetos;
- Um comportamento que est á distribuído entre várias classes deveria ser customizável ou adaptável, sem excessiva especialização em subclasses.

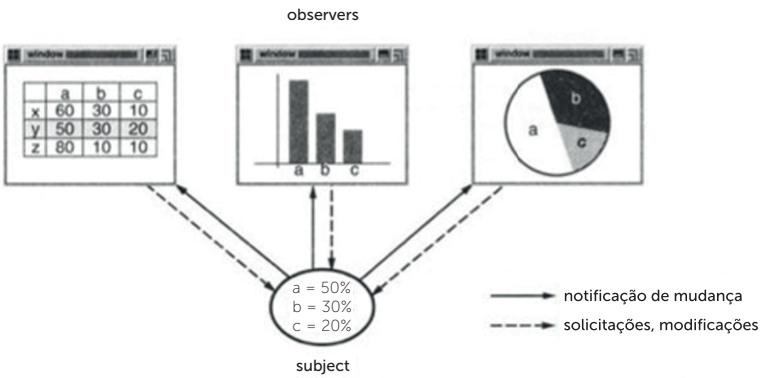
Observer

De acordo com Gamma et al. (2007), permite definir dependência entre objetos, do tipo um-para-muitos, de forma que, quando um objeto muda de estado, todos os seus dependentes são atualizados automaticamente. Um problema enfrentado do particionamento de um sistema em uma coleção

de classes é a necessidade de manter a consistência entre objetos relacionados. Por exemplo, muitos *toolkits* para construção de interfaces gráficas de usuário separam os aspectos de apresentação da interface do usuário dos dados da aplicação subjacente, assim as classes que definem dados da aplicação e da apresentação podem ser reutilizadas independentemente.

O padrão *Observer* descreve como estabelecer o relacionamento. Os objetos-chave nesse padrão são *subject* (assunto) e *observer* (observador). Um *subject* pode ter um número qualquer de observadores dependentes. Todos os observadores são notificados quando o *subject* sofre uma mudança de estado. A Figura 4.8 ilustra esse conceito.

Figura 4.8 | Padrão *Observer*



Fonte: Gamma et al. (2007, p.275).

Usamos o padrão *Observer,* de acordo com Gamma et al. (2007), em qualquer uma das seguintes situações:

- A abstração tem dois aspectos e um depende do outro. Realizar o encapsulamento desses aspectos em objetos separados, permite-se variá-los e reutilizá-los independentemente;
- Quando um objeto deveria ser capaz de notificar outros objetos.

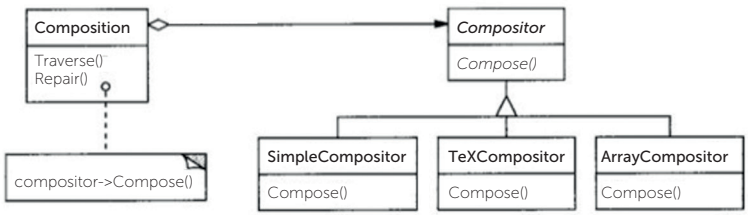
Reflita

Agora que você já conheceu a maioria dos padrões de criação comportamentais e estruturais, qual tipo de padrão você acha mais adequado para o desenvolvimento de sistemas?

Strategy

Segundo Gamma et al. (2007), define uma família de algoritmos que encapsula cada uma delas e as torna intercambiáveis. Existem muitos algoritmos, por exemplo, para quebrar um *stream* de texto em linhas. Podemos definir classes que encapsulam diferentes algoritmos de quebra de linhas ao invés de um único algoritmo. Um algoritmo encapsulado dessa maneira é chamado *Strategy* (estratégia). A Figura 4.9 ilustra o conceito do padrão *Strategy*.

Figura 4.9 | Padrão *Strategy*



Fonte: Gamma et al. (2007, p.293).

Suponha, na Figura 4.9, que a classe *Composition* seja responsável pela manutenção e atualização das quebras de linhas de texto exibidas em um visualizador de texto. As estratégias de quebra de linhas não são implementadas pela classe *Composition*. Em vez disso, são implementadas separadamente por subclasses da classe abstrata *Compositor*. Subclasses de *Compositor*, segundo Gamma et al. (2007), implementam diferentes estratégias:

- *SimpleCompositor*: determina a quebra de linhas uma por vez;
- *TeXCompositor*: realiza quebra de linhas por parágrafo;
- *ArrayCompositor*: implementa uma estratégia que seleciona quebras, de maneira que cada linha tenha um número fixo de itens.

Usamos o padrão *Strategy*, de acordo com Gamma et al. (2007), quando:

- Necessitamos de variantes de um algoritmo;

- Uma classe define muitos comportamentos, e estes aparecem em suas operações como múltiplos comandos condicionais da linguagem.

Exemplificando

Para a escolha de um padrão de projeto a ser utilizado:

- Considere como os padrões de projeto solucionam os problemas;
- Verifique qual o objetivo do padrão, o que faz o padrão de projeto, seus princípios e qual problema ele soluciona;
- Estude como os padrões se relacionam;
- Estude as semelhanças existentes entre os padrões.

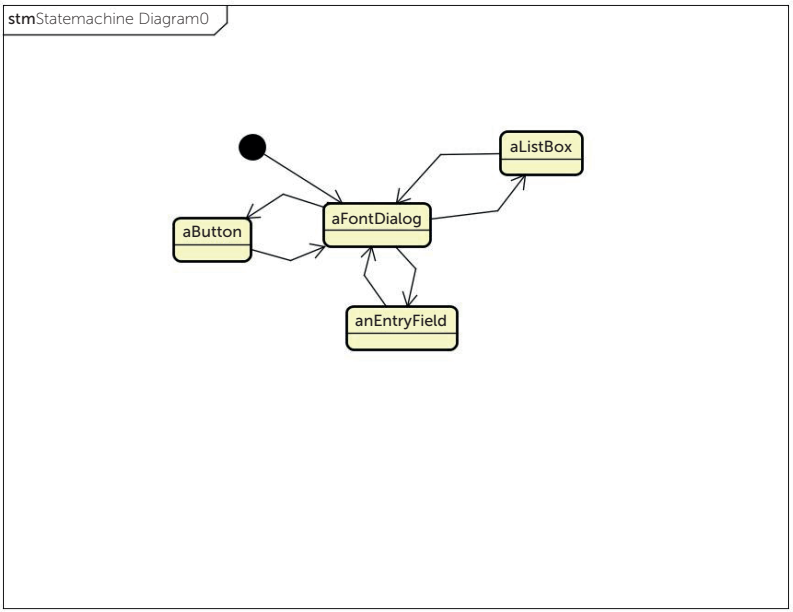
Assim, você aprendeu sobre as característica e aplicações do projeto da arquitetura, bem como os principais padrões de projeto, tais como *Composite*, *Observer*, *Strategy*, *Factory Method*, *Mediator* e *Façade*.

Sem medo de errar

Você foi contratado para definir a arquitetura de um sistema para um Pub. Quais são os principais padrões de projeto que podem ser utilizados em seu projeto? Para responder a essas perguntas, você deve realizar um levantamento dos principais padrões de projeto, tais como *Composite*, *Observer*, *Strategy*, *Factory Method*, *Mediator* e *Façade*, e avaliar a que tipo de projeto eles se destinam. Para o sistema do Pub, podemos utilizar o padrão *Mediator*, que é o responsável pelo controle e coordenação das interações de um grupo de objetos.

O mediador funciona como um intermediário, evitando que os objetos do grupo referenciem uns aos outros explicitamente. Os objetos conhecem somente o *Mediator*, reduzindo assim o número de interconexões. Considere, por exemplo, uma interface em que o usuário pode escolher seu prato, bebida e selecionar um botão para enviar o pedido. O padrão *Mediator* pode ser aplicado conforme a imagem abaixo:

Figura 4.10 | Padrão *Mediator*



Fonte: elaborada pelo autor.

Avançando na prática

Determinando a escolha de um padrão de projeto

Descrição da situação-problema

Uma empresa de desenvolvimento de software está interessada em aplicar padrões de projeto com o objetivo de melhorar a qualidade de desenvolvimento. O gerente de projetos gostaria que você apresentasse, em uma reunião de equipe, os principais tipos de padrão de projeto e seus objetivos. Você foi designado para elaborar esse estudo e apresentação à equipe durante a reunião.

Resolução da situação-problema

Para a resolução desse problema, crie uma apresentação com os principais tipos de padrões estudados e suas aplicações.

Considere os padrões: *Composite*, *Observer*, *Strategy*, *Factory Method*, *Mediator* e *Façade*. Considere as seguintes aplicações de cada padrão:

- Utilize o padrão *Factory Method* quando:
- Uma classe não consegue antecipar a classe de objetos a ser criado;
- Uma classe quer permitir às suas subclasses especificar os objetos que criam;
- Classes responsabilizam subclasses auxiliares, e você quer localizar o conhecimento de qual subclasse auxiliar foi delegada.

Use o padrão *Composite* quando:

- Precisar representar hierarquias partes-todo de objetos;

Use o padrão *Façade* quando:

- Necessitar oferecer interface simples para um subsistema complexo;
- Quiser estruturar seus subsistemas em camadas.

Use o padrão *Mediator* quando:

- A reutilização de um objeto for difícil porque ele referência e se comunica com muitos outros objetos;
- Um comportamento que está distribuído entre várias classes deveria ser customizável ou adaptável, sem excessiva especialização em subclasses.

Use o padrão *Observer* em qualquer uma das seguintes situações:

- A abstração tiver dois aspectos em que um depende do outro. Realizar o encapsulamento desses aspectos em objetos separados, permite-se variá-los e reutilizá-los independentemente;
- Quando um objeto deveria ser capaz de notificar outros objetos.

Faça valer a pena

1. Os padrões ajudam projetistas a reutilizarem projetos e arquiteturas bem-sucedidos ao basearem novos projetos na experiência anterior. Um padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca o fazer da mesma maneira.

Assinale a alternativa que apresenta de forma correta os quatro elementos essenciais de um padrão.
a) Nome do padrão, problema, solução e consequências.
b) Nome do padrão, problema, solução e resultados.
c) Nome do padrão, problema, consequências e resultados.
d) Nome do padrão, problema, solução e validação.
e) Nome do padrão, problema, solução e verificação.

2. Um padrão de projeto nomeia, abstrai e identifica aspectos chave de uma estrutura de projeto comum, para torná-los úteis para a criação de um projeto orientado a objetos reutilizável. O padrão de projeto identifica as classes e instâncias participantes, seus papéis, colaborações e distribuição de responsabilidades.

Assinale a alternativa que apresenta corretamente as finalidades que o padrão pode ter:
a) Criação, desenvolvimento ou comportamental.
b) Criação, estrutural ou comportamental.
c) Desenvolvimento, estrutural ou comportamental.
d) Desenvolvimento, estrutural ou criação.
e) Criação, estrutural ou padronização.

3. Cada padrão possui uma finalidade, que pode ser de criação, estrutural ou comportamental. Cada padrão de projeto foca em um problema ou tópico particular do projeto orientado a objetos, descrevendo em qual situação pode ser aplicado, bem como os custos e benefícios de utilização.

Assinale a alternativa que apresenta corretamente os padrões comportamentais:
a) *Composite e Observer.*
b) *Factory e Façade.*
c) *Mediator, Observer e Strategy.*
d) *Strategy, Factory e Mediator.*
e) *Mediator e Façade.*

Seção 4.3

Design de arquitetura

Diálogo aberto

Prezado aluno, você já percebeu que a maioria dos sistemas de informações é transmitida por meio de dois ou mais computadores? Um sistema baseado na Web, por exemplo, roda no navegador em seu computador pessoal, mas interagirá com o servidor Web pela Internet. Um sistema que opere inteiramente dentro da rede de uma empresa pode estar interagindo com outro sistema Web em outro lugar na rede da empresa. Uma etapa importante da fase de design é a criação do design da arquitetura, o planejamento de como os componentes do sistema de informação serão distribuídos entre os vários computadores e quais hardwares e software de sistema operacional e softwares de aplicativos serão usados para cada computador.

Você foi contratado para definir a arquitetura de um sistema para um Pub. Uma vez definidos os estilos de arquitetura que podem ser utilizados em nosso projeto e os principais padrões de projetos, está na hora de finalizar o projeto da arquitetura do nosso sistema, conhecendo e aplicando o design da arquitetura do software.

Nesta seção você aprenderá sobre design da arquitetura, elementos de um design de arquitetura, requisitos operacionais, requisitos de desempenho, requisitos de segurança e requisitos culturais e políticos. Ao final, produza um relatório com as respostas dos questionamentos e apresente ao contratante. Essas são etapas fundamentais para o processo de análise de projeto de um sistema, sendo de extrema importância para seu aprendizado e para as próximas etapas a serem realizadas, então bom trabalho.

Não pode faltar

Design de Arquitetura

Segundo Dennis, Wixom e Roth (2014), um dos componentes importantes da fase de projeto (design) é o design de arquitetura,

que descreve hardware, software e ambiente de rede do sistema. Ele se fundamenta principalmente nos requisitos não funcionais, como requisitos operacionais, de desempenho, de segurança, culturais e políticos.

O objetivo do projeto de arquitetura é determinar como os componentes de software do sistema de informação serão atribuídos aos dispositivos de hardware do sistema.

Os componentes arquitetônicos mais importantes de qualquer sistema são o software e o hardware, que precisam ser identificados e, em seguida, alocados aos vários componentes de hardware nos quais o sistema será executado. Todos os sistemas de software podem ser divididos em quatro funções básicas:

Armazenagem de dados – A maioria dos sistemas de informações requer que os dados sejam armazenados e recuperados, seja em um pequeno arquivo ou em banco de dados que armazena os registros de recursos humanos de uma organização. Essas são as entidades de dados documentadas nos Diagramas Entidades Relacionamentos (DERs).

Lógica de acesso aos dados – É o processamento exigido para acessar os dados, significando, frequentemente, consultas a banco de dados em SQL (*Structured Query Language*).

Lógica de aplicação – A lógica documentada nos casos de uso e requisitos funcionais.

Lógica de apresentação – É a forma de exibição das informações para o usuário e a aceitação dos comandos do usuário (a interface com o usuário).

Essas quatro funções são os blocos básicos de construção de qualquer sistema de informação. De acordo com Dennis, Wixom e Roth (2004), os três principais componentes de hardware de um sistema são os computadores clientes, os servidores e a rede que os conecta.

Os **computadores** são os dispositivos de entrada/saída empregados pelos usuários, sendo, em geral, computadores de mesa (desktop) ou portáteis (laptops), mas podem ser telefones

celulares ou terminais com finalidades específicas. Os **servidores** são computadores multiusuários, usados para armazenar softwares e dados, que podem ser acessados por meio de permissão. A **rede** que conecta os computadores pode apresentar diferentes velocidades, desde uma conexão lenta, como a discada, por telefone, passando por redes *frame relay* permanente (*always-on*) de velocidade média, até conexões rápidas de banda larga, como circuitos de modem por cabo ou DSL de alta velocidade.

Assimile

Para se determinar corretamente as quatro funções básicas de um software, tais como processo de armazenagem de dados, lógica de acesso aos dados, lógica de aplicação e lógica de apresentação, é necessário primeiramente obter os requisitos funcionais, que são as funcionalidades que o sistema deve executar, e os requisitos não funcionais, como os operacionais, de desempenho, de segurança, culturais e políticos.

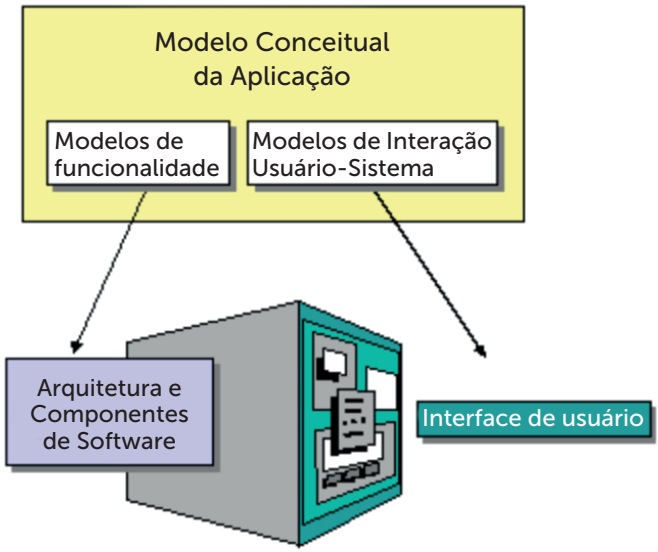
Criando um Design de Arquitetura

De acordo com Dennis, Wixom e Roth (2014), o design de arquitetura especifica a arquitetura como um todo e a disposição do software e do hardware que serão usados. A criação de um design de arquitetura inicia com os requisitos não funcionais. A primeira etapa é refiná-los em requisitos mais detalhados, que serão empregados para ajudar na seleção da arquitetura a ser usada (baseada em servidor, baseada em cliente ou cliente-servidor) e de quais componentes de software serão colocados em cada máquina.

Segundo Leite (2000), o design compreende atividades de concepção, especificação e prototipação de um artefato de software. O software como um produto precisa ser concebido, especificado e prototipado. Design pode ser traduzido tanto como projeto como por desenho do software. A função de designer de software compreende a parte externa, composta pelo modelo conceitual da aplicação e pela interface de usuário, e a parte interna, composta pela arquitetura de componentes de softwares e pelos algoritmos e estruturas de dados que implementem esses componentes. A Figura 4.11 ilustra o modelo conceitual da aplicação, em que são definidas

as funcionalidades do sistema e os modelos de interação usuário-sistema. Por meio das funcionalidades, define-se a arquitetura e os componentes de software, e por meio do modelo de interação usuário-sistema, define-se a Interface de usuário.

Figura 4.11 | Modelo conceitual da aplicação



Fonte: <https://goo.gl/5AqfuH>.

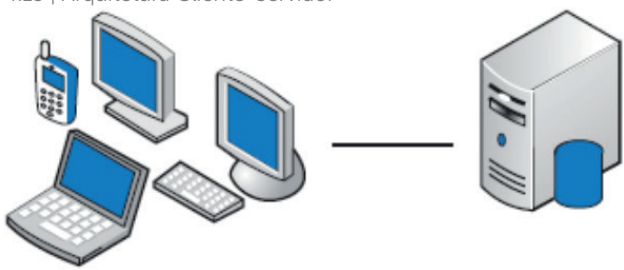
Segundo Dennis, Wixom e Roth (2014), a arquitetura baseada em servidor é aquela que possui um computador servidor (normalmente um computador *mainframe* central) executando as quatro funções da aplicação. Os clientes (normalmente terminais) permitem que os usuários enviem e recebam mensagens do computador servidor, apenas capturando o que é digitado, enviando ao servidor para processamento e aceitando instruções do servidor sobre o deve exibir. A Figura 4.12 ilustra a arquitetura baseada em servidor.

Figura 4.12 | Arquitetura baseada em servidor

Fonte: Dennis, Wixom e Roth (2014, [s.p.]).

De acordo com Dennis, Wixom e Roth (2014), a arquitetura cliente-servidor é a que a maioria das empresas utiliza hoje em dia, sendo aquela que equilibra o processamento entre os dispositivos clientes e um ou mais dispositivos servidores. Nessas arquiteturas, o cliente é responsável pela lógica de apresentação e o servidor é responsável pela lógica de acesso aos dados e armazenagem de dados. A lógica da aplicação pode residir no cliente, no servidor ou ser dividida em ambos. A Figura 4.13 ilustra a arquitetura cliente-servidor.

Figura 4.13 | Arquitetura Cliente-Servidor



Fonte: Dennis, Wixom e Roth (2014, [s.p.]).

De acordo com Dennis, Wixom e Roth (2014), as arquiteturas cliente-servidor apresentam vantagens, como a capacidade de serem redimensionáveis (ou escaláveis), podendo aumentar ou diminuir as capacidades de armazenamento e processamento dos servidores, e o suporte a diversos tipos de clientes e servidores, sendo possível conectar computadores que usem sistemas operacionais diferentes para não ficar restrito a um revendedor.

Segundo Dennis, Wixom e Roth (2014), na arquitetura baseada em cliente, os clientes são microcomputadores em uma rede local e o computador servidor é um servidor na mesma rede. O software de aplicação nos computadores clientes é responsável pela lógica de apresentação, lógica de aplicação e lógica de acesso aos dados, e o servidor apenas fornece o armazenamento dos dados. A Figura 4.14 ilustra a arquitetura baseada em cliente.

Figura 4.14 | Arquitetura baseada em cliente

Computador cliente
Responsável pela lógica da apresentação, lógica da aplicação e lógica de acesso aos dados

Computador servidor
Responsável pelo armazenamento dos dados

Fonte: Dennis, Wixom e Roth (2014, [s.p.]).

A arquitetura baseada em cliente funciona muito bem em situações com número reduzido de usuários ou exigências limitadas de acesso aos dados. De acordo com Dennis, Wixom e Roth (2014), o problema na arquitetura baseada em cliente é que todos os dados no servidor devem passar para o cliente a fim de serem processados.

Dennis, Wixom e Roth (2014) afirmam que há quatro tipos principais de requisitos não funcionais que podem ser importantes na elaboração do design de arquitetura: requisitos operacionais, requisitos de desempenho, requisitos de segurança e requisitos culturais e políticos.

Os requisitos operacionais especificam o ambiente operacional em que o sistema deve ser executado e como ele pode mudar ao longo do tempo. Normalmente se refere aos sistemas operacionais, ao software dos sistemas e aos sistemas de informações com os quais o sistema deve interagir. As quatro principais áreas do requisito operacional são (DENNIS; WIXOM; ROTH, 2014):

- **Requisitos do Ambiente Técnico**: especificam os tipos de hardware e software com os quais o sistema funcionará. Esses requisitos normalmente estão relacionados ao software do sistema operacional (por exemplo, Windows, Linux, Mac OS), ao software do sistema de banco de dados (por exemplo, MySQL, Oracle) e a qualquer outro software do sistema (por exemplo, Internet Explorer);

- **Requisitos de Integração de Sistemas**: exigem que o sistema opere com outros sistemas de informações, dentro ou fora da empresa, especificando as interfaces pelas quais os dados serão trocados com os outros sistemas. Exemplo: o sistema deve ser capaz de importar e exportar planilhas Excel;

- **Requisitos de Portabilidade**: definem como os ambientes operacionais técnicos podem evoluir ao longo do tempo e como o sistema deve responder (pode ser que o sistema seja executado atualmente no Windows 7, mas no futuro seja executado no Windows 10);

- **Requisitos de Manutenção**: especificam modificações dos requisitos empresariais que podem ser previstas. Nem todas as mudanças são previsíveis, mas algumas são. Por exemplo: suponha que uma pequena empresa tenha somente uma instalação fabril, mas preveja a construção da segunda nos próximos cinco anos. O sistema deve ser capaz de oferecer suporte a mais uma instalação fabril depois de um aviso prévio de seis meses.

De acordo com Dennis, Wixom e Roth (2014), os requisitos de desempenho referem-se às questões de desempenho, como tempo de resposta, capacidade e confiabilidade. O analista define os requisitos de desempenho para o sistema e a realização de teste desses requisitos é fundamental (DENNIS; WIXOM; ROTH, 2014):

- **Requisitos de Velocidade**: dizem respeito à velocidade em que o sistema deve ser executado. O tempo de resposta do sistema é o tempo que o sistema leva para responder a uma solicitação do usuário. A maioria dos usuários compreende que certas partes de um sistema responderão rápido, enquanto outras serão mais lentas. As ações executadas localmente no computador do usuário devem ser quase imediatas (por exemplo, digitação, arrastar e soltar) e outras, que exigem comunicação por meio de uma rede, podem ter tempos de resposta maiores (por exemplo, solicitação Web). Exemplo: o

tempo de resposta deve ser 4 segundos ou menos para qualquer transação por meio da rede;

- **Requisitos de Capacidade**: tentam prever para quantos usuários o sistema terá de dar suporte, tanto no total como simultaneamente. Os requisitos de capacidade são importantes no entendimento do tamanho dos bancos de dados, da capacidade de processamento necessário, etc. Exemplo: haverá o máximo de 2 mil usuários simultâneos nos instantes de utilização máxima;

- **Requisitos de Disponibilidade e Confiabilidade**: determinam até que ponto os usuários podem presumir que o sistema estará disponível para uso. Embora alguns sistemas destinem-se à utilização apenas durante a semana de trabalho de 40 horas, outros sistemas são destinados à utilização por pessoas ao redor do mundo. Exemplo: o sistema apresentará o desempenho com 99% do tempo em atividade e a manutenção programada não deve ultrapassar um período de 6 horas por mês.

Pesquise mais

Para maior conhecimento sobre design de arquitetura de software, faça a leitura do artigo a seguir:

SILVA FILHO, A. M. Arquitetura de Software: desenvolvimento orientado para arquitetura. **DevMedia**. 2008. Disponível em: <https://www.devmedia.com.br/arquitetura-de-software-desenvolvimento-orientado-para-arquitetura/8033>. Acesso em: 28 ago. 2018.

- **Requisitos de Segurança:** protegem o sistema de informações contra interrupções e perda de dados, causadas por ação intencional (por exemplo, um hacker ou ataque terrorista) ou acontecimento aleatório (por exemplo, falha de disco, tempestades). A segurança é essencialmente responsabilidade do grupo de operações, ou seja, o pessoal responsável por instalar e operar controles de segurança, como firewalls, sistemas de detecção de intrusos e operações de backups de rotina e recuperação.

Segundo Dennis, Wixom e Roth (2014), a segurança é um problema que cresce constantemente no mundo moldado pela internet. A maior ameaça tem vindo do interior das próprias empresas. As ameaças externas se tornaram um problema mais significativo no início dos anos 2000. Em geral, a segurança interna dos sistemas se concentra em especificar quem tem acesso aos dados e a que dados alguém pode ter acesso, identificar a necessidade de criptografia e autenticação e assegurar que as aplicações evitem a propagação de vírus:

- **Valor do sistema**: os ativos computacionais mais importantes em qualquer organização não são os equipamentos de informática, mas os dados da organização. Exemplo: suponha que alguém destrua todos os registros dos alunos de sua universidade, de modo que ninguém saiba nada sobre as disciplinas cursadas por alguém ou suas respectivas notas;

- **Requisitos de Controle de Acesso**: alguns dados armazenados no sistema precisam ser mantidos confidenciais: alguns dados precisam de controles especiais sobre quem está autorizado a modificá-los ou excluí-los. Registros de pessoal, por exemplo, deveriam estar disponíveis para leitura somente para o departamento pessoal e o supervisor dos empregados. Os requisitos de controle de acesso determinam quem pode acessar os dados, quais dados alguém pode acessar e que tipo de acesso é permitido;

- **Requisitos de Criptografia e Autenticação**: uma das melhores maneiras de evitar acesso não autorizado a dados é a criptografia, que é um meio de ocultar informações pelo uso de algoritmos matemáticos. A criptografia pode ser usada para proteger dados armazenados em bancos de dados ou dados que estão trafegando em uma rede, de um banco de dados para um computador. Há dois tipos de criptografia: simétrica, em que a chave usada para criptografar uma mensagem é a mesma usada para descriptografar, e assimétrica, que usa chave para criptografar dados diferentes da chave para descriptografar.

Exemplificando

Considere a exemplificação dos requisitos de seguranças propostos por Dennis, Wixom e Roth (2014).

Quadro 4.1 | Requisitos de Segurança de Sistemas

Tipo de Requisito	Definição	Exemplos
Estimativas do Valor do Sistema	Valor estimado do negócio do sistema e de seus dados.	Estima-se que uma perda completa de todos os dados do sistema custe US$ 20 milhões.
Requisitos de Controle de Acesso	Limitações sobre quem pode acessar e quais dados podem ser acessados	Apenas os gerentes de departamentos poderão alterar os itens do estoque dentro de seu próprio departamento.
Requisitos de Criptografia e Autenticação	Define que dados serão criptografados onde e se a autenticação será necessária para o acesso do usuário	Será exigida autenticação para os usuários que entrarem no sistema fora de seu escritório.
Requisitos de Controle de Vírus	Controla a propagação de vírus	Todos arquivos atualizados serão examinados quanto à presença de vírus antes de serem salvos no sistema.

Fonte: Dennis, Wixom e Roth (2014, [s.p.]).

Segundo Dennis, Wixom e Roth (2014), os **requisitos culturais e políticos** são específicos de cada país onde o sistema será usado. No ambiente empresarial globalizado, as empresas estão expandindo seus sistemas para atingir usuários em todo o mundo. Os requisitos culturais e políticos são:

- **Requisitos multilíngues**: as aplicações globais apresentam requisitos multilíngues, o que significa que devem oferecer suporte aos usuários em diferentes idiomas ou permitir o uso de caracteres diferentes do inglês. Exemplo: o sistema será executado em inglês, francês e espanhol;

- **Requisitos de Personalização**: a equipe de projeto precisa dedicar atenção aos requisitos de personalização, determinando quanto da aplicação será gerenciado por um grupo central e quanto será gerenciado localmente. Por exemplo: algumas empresas permitem que as subsidiárias em alguns países personalizem a aplicação, omitindo ou adicionando certos recursos;

- **Normas implícitas**: muitos países possuem normas implícitas que não são compartilhadas internacionalmente. Deve-se mencionar explicitamente suposições que diferem de um país para outro. Exemplo: todos os campos de datas trarão a informação explícita de que estão no formato mês-dia-ano;

- **Requisitos legais**: os requisitos legais são impostos pelas leis e regulamentos governamentais. Exemplo: as informações pessoais sobre os clientes não podem ser transferidas dos países da União Europeia para os Estados Unidos.

Reflita

De acordo com Dennis, Wixom e Roth (2014), a Lithonia Lighting, localizada nos arredores de Atlanta, é o maior fabricante mundial de acessórios de iluminação, com mais de US$1 bilhão em vendas anuais. Uma tarde, o transformador de energia que alimentava a matriz da empresa explodiu e deixou todo o complexo comercial, incluindo a central de dados, sem energia elétrica. O transformador foi substituído rapidamente e a energia restabelecida, porém, a paralisação por 3 horas do sistema de vendas custou US$1 milhão em perda de vendas potenciais. O que você recomendaria para evitar perdas semelhantes no futuro?

Nesta seção, você conheceu sobre o design da arquitetura, que se baseia nos requisitos operacionais, de desempenho, de segurança, culturais e políticos.

Sem medo de errar

Você foi contratado para definir a arquitetura de um sistema para um Pub. Que tipo de arquitetura pode ser utilizado neste projeto?

Quais requisitos operacionais, de desempenho, de segurança, culturais e políticos podem ser aplicados ao sistema do Pub?

Para esse sistema, a arquitetura cliente-servidor é uma boa opção, com o cliente como o responsável pela lógica de apresentação e o servidor como o responsável pela lógica de acesso aos dados e armazenagem de dados. A principal vantagem é usar vários clientes no sistema do Pub, podendo utilizar sistemas clientes em tablets.

Para os requisitos operacionais, podemos propor um sistema com desenvolvimento em linguagem Java, suportando vários sistemas operacionais (Windows, Linux e Mac OS) e banco de dados MySQL. Com isso, atendemos aos requisitos de portabilidade e manutenção, por meio da atualização do Java e do MySQL.

Com relação aos requisitos de desempenho, a comunicação será realizada por meio de uma rede, devendo ter tempo de resposta de aproximadamente 4 segundos para as transações em rede. A capacidade deve permitir o acesso de aproximadamente 50 usuários simultâneos e o sistema deve apresentar desempenho em 99% do tempo.

Sobre a segurança, o sistema deve contar com sistema de controle de acesso, permitindo que apenas a gerência tenha acesso aos produtos vendidos. Para que se evite acesso não autorizado aos dados, como vendas diárias e folha de pagamento, o sistema deve realizar a criptografia para proteger os dados armazenados em banco de dados.

Avançando na prática

Criando Design de Arquitetura

Descrição da situação-problema

Uma empresa multinacional gostaria de criar um sistema Web para oferta de cursos online para seus funcionários. Por meio desse sistema, o funcionário acessaria uma base de treinamentos e escolheria quais cursos gostaria de realizar. Ao término de determinado curso, o sistema emitiria um certificado digital, que ficaria visível na área do funcionário.

Você é analista de sistemas dessa organização e, como já determinou os requisitos funcionais desse sistema, determine agora os requisitos operacionais, de desempenho e de segurança para o sistema Web para oferta de cursos online.

Resolução da situação-problema

1. Requisitos Operacionais

Ambiente Técnico

1.1 O sistema funcionará no ambiente Web.

1.2 Os clientes precisão apenas do RA em seus computadores.

Integração de Sistemas

1.3 O sistema de Internet obterá e gravará informações no banco de dados principal de clientes.

Portabilidade

1.4 O sistema precisará permanecer atualizado com os padrões Web em constante evolução.

Manutenção

1.5 Não estão previstos requisitos de manutenção.

2. Requisitos de Desempenho

Velocidade

2.1 O tempo de resposta deve ser menor que 7 segundos.

Capacidade

2.2 Haverá o máximo de 100 usuários simultâneos em horários de pico.

Disponibilidade e Confiabilidade

2.3 O sistema terá desempenho de 99% do tempo em atividade.

3. Requisitos de Segurança

Controle de Acesso

3.1 Os clientes podem ter acesso a suas contas com nome de usuário e senha.

Criptografia

3.2 As informações de pagamento dos clientes devem ser transmitidas no modo de segurança.

Faça valer a pena

1. Os componentes arquitetônicos mais importantes de qualquer sistema são o software e o hardware, que precisam ser identificados e, em seguida, alocados aos vários componentes de hardware nos quais o sistema será executado.

Assinale a alternativa que apresenta corretamente as quatro funções básicas em que os sistemas de software podem ser divididos:
a) Armazenagem de dados, lógica de acesso aos dados, lógica de aplicação e lógica de apresentação.
b) Armazenagem de dados, lógica de acesso aos dados, lógica de aplicação e lógica de programação.
c) Armazenagem de dados, lógica de acesso aos dados, lógica de aplicação e lógica de segurança.
d) Armazenagem de dados, lógica de acesso aos dados, lógica de aplicação e lógica de normas.
e) Armazenagem de dados, lógica de acesso aos dados, lógica de aplicação e lógica de exibição.

2. O design de arquitetura especifica a arquitetura como um todo, bem como a disposição do software e do hardware que vão ser usados. A criação de um design de arquitetura se inicia com os requisitos não funcionais. A primeira etapa é refinar os requisitos não funcionais em requisitos mais detalhados, que serão empregados para ajudar na seleção da arquitetura a ser usada.

Assinale a alternativa que apresenta corretamente os tipos de arquitetura de software:
a) Baseada em serviços, baseada em cliente ou cliente-servidor.
b) Baseada em servidor, baseada em cliente ou cliente-servidor.

c) Baseada em servidor, baseada em cliente ou servidor-servidor.
d) Baseada em servidor, baseada em cliente ou cliente-cliente.
e) Baseada em servidor, baseada em cliente-cliente ou cliente-servidor.

3. O design de arquitetura especifica a arquitetura como um todo, bem como a disposição do software e do hardware que vão ser usados. A criação de um design de arquitetura se inicia com os requisitos não funcionais. A primeira etapa é refinar os requisitos não funcionais em requisitos mais detalhados, que serão empregados para ajudar na seleção da arquitetura a ser usada.

Assinale a alternativa que apresenta corretamente os quatro tipos principais de requisitos não funcionais que podem ser importantes na elaboração do design de arquitetura:
a) Requisitos operacionais, requisitos de desempenho, requisitos de segurança e requisitos culturais e políticos.
b) Requisitos operacionais, requisitos de desempenho, requisitos de segurança e requisitos culturais e sociais.
c) Requisitos operacionais, requisitos de desempenho, requisitos de segurança e requisitos de hardware.
d) Requisitos operacionais, requisitos de desempenho, requisitos de segurança e requisitos de software.
e) Requisitos operacionais, requisitos de desempenho, requisitos de segurança e requisitos hardware e software.

Referências

BLOIS, A. P. T. B. **Uma Abordagem de Projeto Arquitetural Baseado em Componentes no Contexto de Engenharia de Domínio.** 2006. 205 p. Tese (Doutorado em Engenharia de Sistemas e Computação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2006.

DENNIS, A.; WIXOM, B. H.; ROTH, R. M. **Análise e Projeto de Sistemas.** 5. ed. Rio de Janeiro: LTC, 2014.

GAMMA, E. et al. **Padrões de Projeto:** soluções reutilizáveis de software orientado a objetos. 1. ed. Porto Alegre: Bookman, 2007.

LEITE, J. C. **Design Conceitual de Software.** Notas de aula de Engenharia de Software da UFRN. Disponível em: <https://www.dimap.ufrn.br/~jair/ES/c5.html>. Acesso em: 16 out. 2018.

PFLEEGER, S. L. **Engenharia de Software**: Teoria e Prática. 2. ed. Rio de Janeiro: Prentice Hall, 2002;

PRESSMAN, R. S; MAXIM, B. R. **Engenharia de Software:** uma abordagem profissional. 8 ed. Porto Alegre: AMGH, 2016;

SCHACH, S. R. **Engenharia de Software:** os paradigmas clássico e orientado a objetos. 7. ed. Porto Alegre: AMGH, 2010.

UNIVESP. **Aula 13 –** Projeto da arquitetura de software (parte 1). 22 set. 2017. Disponível em: <https://www.youtube.com/watch?v=2R4l995J2Jw>. Acesso em: 25 jun. 2018;

UNIVESP. **Aula 14 –** Projeto da arquitetura de software (parte 2). 22 set. 2017. Disponível em: <https://www.youtube.com/watch?v=PwRvSNgbibU>. Acesso em: 25 jun. 2018.

ISBN 978-85-522-1168-6