



MODELAGEM DE SOFTWARE

Professora Me. Vanessa Ravazzi Perseguine

UNICESUMAR

Av. Guedner, 1610 - Jardim Aclimação
Cep 87050-900 - MARINGÁ - PARANÁ
unicesumar.edu.br
44 3027.6360

UNICESUMAR EDUCAÇÃO A DISTÂNCIA

NEAD - Núcleo de Educação a Distância
Bloco 4 - MARINGÁ - PARANÁ
unicesumar.edu.br
0800 600 6360

as imagens utilizadas neste
livro foram obtidas a partir
do site SHUTTERSTOCK.COM

FICHA CATALOGRÁFICA

C397 **CENTRO UNIVERSITÁRIO DE MARINGÁ.** Núcleo de Educação a Distância; **PERSEGUINE**, Vanessa Ravazzi.

Modelagem de Software. Vanessa Ravazzi Perseguine.

Maringá-Pr.: UniCesumar, 2016. Reimpresso em 2019.

172 p.

"Graduação - EaD".

1. Modelagem. 2. Software. EaD. I. Título.

ISBN 978-85-459-0205-8

CDD - 22 ed. 005
CIP - NBR 12899 - AACR/2

Ficha catalográfica elaborada pelo bibliotecário
João Vivaldo de Souza - CRB-8 - 6828

Impresso por:

Reitor

Wilson de Matos Silva

Vice-Reitor

Wilson de Matos Silva Filho

Pró-Reitor de Administração

Wilson de Matos Silva Filho

Pró-Reitor Executivo de EAD

William Victor Kendrick de Matos Silva

Pró-Reitor de Ensino de EAD

Janes Fidélis Tomelin

Presidente da Mantenedora

Cláudio Ferdinandi

NEAD - Núcleo de Educação a Distância**Diretoria Executiva**

Chrystiano Mincoff

James Prestes

Tiago Stachon

Diretoria de Design Educacional

Débora Leite

Diretoria de Graduação e Pós-graduação

Kátia Coelho

Diretoria de Permanência

Leonardo Spaine

Head de Produção de Conteúdos

Celso Luiz Braga de Souza Filho

Gerência de Produção de Conteúdo

Diogo Ribeiro Garcia

Gerência de Projetos Especiais

Daniel Fuverki Hey

Supervisão do Núcleo de Produção de Materiais

Nádila Toledo

Supervisão Operacional de Ensino

Luiz Arthur Sanglard

Coordenador de Conteúdo

Fabiana de Lima

Designer Educacional

Paulo Victor Souza e Silva

Projeto Gráfico

Jaime de Marchi Junior

José Jhonnny Coelho

Arte Capa

Arthur Cantareli Silva

Ilustração Capa

Bruno Pardinho

Editoração

Thomas Hudson Costa

Qualidade Textual

Keren Pardini

Ilustração

André Luís Onishi

Bruno Pardinho



Professor
Wilson de Matos Silva
Reitor

Em um mundo global e dinâmico, nós trabalhamos com princípios éticos e profissionalismo, não só para oferecer uma educação de qualidade, mas, acima de tudo, para gerar uma conversão integral das pessoas ao conhecimento. Baseamo-nos em 4 pilares: intelectual, profissional, emocional e espiritual.

Iniciamos a Unicesumar em 1990, com dois cursos de graduação e 180 alunos. Hoje, temos mais de 100 mil estudantes espalhados em todo o Brasil: nos quatro campi presenciais (Maringá, Curitiba, Ponta Grossa e Londrina) e em mais de 300 polos EAD no país, com dezenas de cursos de graduação e pós-graduação. Produzimos e revisamos 500 livros e distribuímos mais de 500 mil exemplares por ano. Somos reconhecidos pelo MEC como uma instituição de excelência, com IGC 4 em 7 anos consecutivos. Estamos entre os 10 maiores grupos educacionais do Brasil.

A rapidez do mundo moderno exige dos educadores soluções inteligentes para as necessidades de todos. Para continuar relevante, a instituição de educação precisa ter pelo menos três virtudes: inovação, coragem e compromisso com a qualidade. Por isso, desenvolvemos, para os cursos de Engenharia, metodologias ativas, as quais visam reunir o melhor do ensino presencial e a distância.

Tudo isso para honrarmos a nossa missão que é promover a educação de qualidade nas diferentes áreas do conhecimento, formando profissionais cidadãos que contribuam para o desenvolvimento de uma sociedade justa e solidária.

Vamos juntos!



Professor

Janes Fidélis Tomelin

Pró-Reitor de
Ensino de EAD

Professora

Kátia Solange Coelho

Diretoria de Graduação
e Pós-graduação

Seja bem-vindo(a), caro(a) acadêmico(a)! Você está iniciando um processo de transformação, pois quando investimos em nossa formação, seja ela pessoal ou profissional, nos transformamos e, consequentemente, transformamos também a sociedade na qual estamos inseridos. De que forma o fazemos? Criando oportunidades e/ou estabelecendo mudanças capazes de alcançar um nível de desenvolvimento compatível com os desafios que surgem no mundo contemporâneo.

O Centro Universitário Cesumar mediante o Núcleo de Educação a Distância, o(a) acompanhará durante todo este processo, pois conforme Freire (1996): "Os homens se educam juntos, na transformação do mundo".

Os materiais produzidos oferecem linguagem dialógica e encontram-se integrados à proposta pedagógica, contribuindo no processo educacional, complementando sua formação profissional, desenvolvendo competências e habilidades, e aplicando conceitos teóricos em situação de realidade, de maneira a inseri-lo no mercado de trabalho. Ou seja, estes materiais têm como principal objetivo "provocar uma aproximação entre você e o conteúdo", desta forma possibilita o desenvolvimento da autonomia em busca dos conhecimentos necessários para a sua formação pessoal e profissional.

Portanto, nossa distância nesse processo de crescimento e construção do conhecimento deve ser apenas geográfica. Utilize os diversos recursos pedagógicos que o Centro Universitário Cesumar lhe possibilita. Ou seja, acesse regularmente o Studeo, que é o seu Ambiente Virtual de Aprendizagem, interaja nos fóruns e enquetes, assista às aulas ao vivo e participe das discussões. Além disso, lembre-se que existe uma equipe de professores e tutores que se encontra disponível para sanar suas dúvidas e auxiliá-lo(a) em seu processo de aprendizagem, possibilitando-lhe trilhar com tranquilidade e segurança sua trajetória acadêmica.

Professora Me. Vanessa Ravazzi Perseguine

Mestre em Ciências da Computação. Especialista em Gestão e Coordenação Escolar. MBA em Gestão de Projetos. Graduada em Ciências da Computação. Experiência profissional na Gestão de TI, coordenação de cursos acadêmicos e gestão de projetos. Experiência acadêmica na área de Ciências da Computação e Administração, atuando nas áreas: Engenharia de Software e Gerência de Projetos, Sistemas de Informação.

MODELAGEM DE SOFTWARE

SEJA BEM-VINDO(A)!

Olá! Bem-vindo(a) à disciplina Modelagem de Software. Neste curso, iremos abordar os conceitos e principais técnicas de modelagem de software.

Iniciaremos nossos estudos, na unidade I, com o objetivo de compreender a importância da modelagem ao perceber que, ao desenvolver um projeto, oferecemos condições de personalização total de acordo com todas as exigências do cliente, quanto às características especiais do software, e possibilitamos uma visualização completa da arquitetura antes dela ser construída, assim, todos temos a certeza de que será desenvolvido exatamente o que é pretendido. Para se projetar, torna-se essencial a modelagem de processos. Os modelos oferecem uma visão simplificada da realidade complexa do software, permitindo uma melhor compreensão dessa realidade.

Modelar é um meio utilizado para analisar e projetar sistemas de software. O modelo pode representar o software como um todo e partes dele. Essas partes representam as várias visões que podem ser abstraídas do sistema. Os detalhes diferem de acordo com a perspectiva do profissional que cria o modelo e sua necessidade.

Na unidade II, estudaremos os vários tipos de modelos possíveis de serem criados de acordo com cada perspectiva.

Estudaremos que é necessário utilizar uma linguagem para a notação dos modelos. Com esse objetivo, utilizaremos a UML® (Unified Modeling Language), padrão de aceitação internacional de linguagem para modelagem utilizada no desenvolvimento de software. A utilização de modelos UML® tende à padronização e estruturação, o que facilita a comunicação entre equipe de desenvolvimento, entre os membros e também com o cliente.

Na unidade III, abordaremos a UML®, suas especificações e diagramas.

Na unidade IV, teremos o contato com algumas ferramentas CASE de modelagem de software disponíveis no mercado, e então, na unidade V, aplicaremos todos os conceitos apresentados em um estudo de caso simples.

Vamos começar nossos estudos?

SUMÁRIO

UNIDADE I

OBJETIVO DA MODELAGEM DE SOFTWARE

-
- 15 Introdução
 - 16 Conceitos e Objetivos da Modelagem de Software
 - 21 Modelagem de Software e o Processo de Desenvolvimento
 - 24 Importância da Modelagem de Software
 - 26 Modelos e o Que Eles Modelam
 - 27 Considerações Finais

UNIDADE II

TIPOS DE MODELOS

-
- 33 Introdução
 - 34 Modelos de Contexto
 - 41 Modelos de Interação
 - 49 Modelos Estruturais
 - 56 Modelos Comportamentais



SUMÁRIO

UNIDADE III

UML®

69 Introdução

70 Pilares da Orientação a Objetos

82 Linguagem de Modelagem Unificada - UML®

94 Diagramas UML®

112 Considerações Finais

UNIDADE IV

FERRAMENTAS DE MODELAGEM

121 Introdução

122 Ferramentas Case

123 Modelagem de Software Utilizando Ferramentas Case

124 Ferramentas de Modelagem de Software

134 Comparativo

136 Considerações Finais



SUMÁRIO

 UNIDADE V

MODELAGEM ORIENTADA A OBJETO COM UML® E ASTAH COMMUNITY

143 Introdução

144 Projeto de Desenvolvimento de Software

160 Considerações Finais

165 CONCLUSÃO

167 REFERÊNCIAS

172 GABARITO



OBJETIVO DA MODELAGEM DE SOFTWARE

Objetivos de Aprendizagem

- Conceituar modelagem de software.
- Identificar a importância da modelagem de software..

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Conceitos e objetivos da modelagem de software
- Modelagem de software e o processo de desenvolvimento
- Importância da modelagem de software
- Modelos e o que eles modelam

INTRODUÇÃO

Olá, caro(a) aluno(a)! Começamos nossos estudos apresentando a contextualização da modelagem e estabelecendo sua importância dentro do processo de desenvolvimento de software.

O sucesso de um projeto de software pode ser determinado por aspectos técnicos e pelo cumprimento das especificações dadas pelo planejamento, entretanto a satisfação geral das partes interessadas é o medidor principal, uma vez que, mesmo cumprindo prazos e custos, se não for entregue um software que resolva os problemas dos interessados, esse projeto pode ser considerado um fracasso. A modelagem de software entra como um processo importante que oferece um canal de comunicação, por meio dos diagramas, promovendo a interação entre os engenheiros de software e o cliente, garantindo a correta interpretação entre o que se pede e o que será desenvolvido.

A modelagem pode ser desenvolvida a qualquer momento dentro do processo de desenvolvimento de software, ela auxilia na interpretação e no levantamento de requisitos e pode atuar como documentação final para a manutenção e rastreabilidade dos requisitos após a implantação.

Estudaremos, nesta unidade, que a modelagem de software tem como objetivo principal dividir o software em partes, utilizando os requisitos como se fossem peças de um quebra-cabeça, para obter visões específicas e direcionadas para diferentes contextos de análise.

Vamos lá!?

CONCEITOS E OBJETIVOS DA MODELAGEM DE SOFTWARE

Ao pesquisar sobre modelagem de software é comum encontrar como definição algo do tipo: um modelo é uma simplificação da realidade. Eu não concordo por um único motivo, um modelo de software busca retratar uma ação ou procedimento que foi descrito e definido por um requisito, geralmente aquele de difícil interpretação, com o objetivo de visualizar todas as ações decorrentes dele. Um modelo não simplifica a realidade, ele a traduz, e realidade, nesse contexto, pode ser encarada como “problema” a ser resolvido, ou melhor, como a função a ser desempenhada; e a representa de uma forma mais lúdica, abstraindo alguns detalhes e focando outros, dependendo da perspectiva de visão e entendimento desejada.

Nunes e O’Neill (2000) apoiam minha teoria quando dizem que a modelagem se traduz como a representação dos aspectos estruturais e das possíveis funções de um programa computacional, por meio de símbolos padronizados com significados específicos, ou seja, a modelagem de software tem como objetivo principal dividir o sistema em partes menores, usando os requisitos como se fossem peças de um quebra-cabeça, para obter uma “imagem” da estrutura e comportamento daquele pedaço específico.

Para promover uma base para a conceituação e entendermos melhor o objetivo da modelagem de software, recorro ao método de associar o significado da palavra a suas tarefas. Entendendo o significado da palavra, fica mais fácil associá-la a sua função no contexto em que é aplicada. O dicionário Michaelis online apresenta a definição a seguir para o verbo modelar:

modelar¹mo.de.lar¹**(modelo+ar²) vtd 1** Fazer o modelo ou o molde de: **Modelar uma estátua.****Modelou a imagem em cera. vtd 2 Pint** Imitar com muita exatidão o relevo ou os contornos de. **vtd 3** Amoldar: **Modelar o bronze. vtd 4** Ajustar-se a, cobrir ou envolver, unir-se bem a, deixando ver a forma do conteúdo: **Uma blusa modelava-lhe o corpo. vtd e vpr 5** Tomar como modelo: **Modelai os vossos atos pelos ensinos do Divino Mestre. Modelava-se pelo exemplo dos grandes homens. vtd 6** Delinear, regular, traçar intelectualmente: **Modelou poemas segundo padrões clássicos.****modelar²**mo.de.lar²**adj (modelo+ar³)** Que serve de modelo; exemplar.

Fonte: Modelar – Michaelis (online)

O Dicionário Online de Português apresenta a seguinte definição:

v.t. Fazer o modelo ou o molde de uma peça.

Denunciar ou insinuar as formas.

Fig. Formar de acordo com um modelo.

Fig. Formar, criar, produzir, dar forma e contornos a.

Fonte: Modelar – Dicionário Online de Português (online).

As definições formais para o verbo modelar nos ensartam ao modelo. Modelar significa fazer um modelo, formar algo. Modelar software é o processo de criação de modelos que representam um software existente ou um que será desenvolvido.



SAIBA MAIS

O não sucesso de projetos de software tem relação com aspectos únicos e específicos de cada projeto, mas todos os projetos de sucesso são semelhantes em vários aspectos. Existem vários elementos que contribuem para um projeto bem-sucedido; um desses componentes é a utilização de modelagem. A escolha dos modelos a serem criados tem profunda influência sobre a maneira como um determinado problema é atacado e como uma solução é definida.

Aprofunde seu conhecimento com a leitura do estudo “Um guia para a criação de modelos de software no Praxys Synergia”, disponível em: <<https://www.dcc.ufmg.br/pos/cursos/defesas/871M.PDF>>. Acesso em: 29 ago. 2015.

Fonte: Rodrigues (2007).

Os objetivos gerais de um modelo de software são relacionados por Pressman (2010, p. 145):

1. Descrever o que o cliente exige;
2. Estabelecer a base para a criação de um projeto de software;
3. Definir um conjunto de requisitos que possam ser validados quando o software for construído.

Sommerville (2011, p. 83), por sua vez, classifica os modelos em função de sua utilização:

1. Como forma de facilitar a discussão sobre um sistema existente ou proposto;
2. Como forma de documentar um sistema existente;
3. Como uma descrição detalhada de um sistema, que pode ser usada para gerar uma implementação do sistema.

Podemos observar que ambos os autores, de sua maneira, apresentam classificações semelhantes para os objetivos da modelagem de software. Quando Pressman sugere que o objetivo do modelo é descrever o que o cliente deseja, coincide com a classificação de Sommerville, que diz que o modelo é uma forma de facilitar a discussão. Ambas garantem que o objetivo é promover a interação entre os engenheiros de software e o cliente, garantindo a correta interpretação entre o que se pede e o que será desenvolvido. O objetivo de Pressman, estabelecer a base para a criação de um projeto de software, é semelhante ao de Sommerville, que afirma que o modelo pode ser usado para gerar uma implementação do sistema. Significa que a partir do modelo é possível ser gerado o código-fonte do sistema, quando ele for desenvolvido com os critérios e detalhes necessários para isso.

Outra palavra bastante associada à modelagem de software e importante de ser conceituada é o termo **abstração**. Abstrair na informática corresponde ao isolamento, à desconsideração de determinados aspectos ou características, com o objetivo de simplificar sua avaliação ou sua interpretação.

Os modelos de software podem ser desenvolvidos em vários níveis de abstração, por exemplo, os engenheiros podem desenvolver um modelo específico para o ponto de vista dos usuários, abstraindo informações técnicas e focando somente cenários de leitura rápida e fácil. A modelagem pode ser baseada em classes, definindo objetos, atributos e relacionamentos, já operando em um nível mais técnico, ou ainda, pode-se desenhar um modelo que representa o fluxo de dados, indicando como os dados são transformados durante a execução das funções do sistema.

A abstração é o aspecto mais importante de um modelo, afirma Sommerville (2011, p. 83). Um modelo deixa de fora alguns detalhes do sistema, portanto ele é uma abstração do sistema. Um modelo seleciona as características mais relevantes e a representa em uma linguagem geralmente gráfica que possibilita a visão do software de diferentes perspectivas, por exemplo:

1. Perspectiva externa: modelagem do ambiente do sistema.
2. Perspectiva de interação: modelagem das interações entre o sistema e o ambiente, ou entre os seus componentes.
3. Perspectiva estrutural: modelagem considerando a estrutura de dados processados pelo sistema.

4. Perspectiva comportamental: modelagem que considera o comportamento dinâmico do sistema, sua reação aos eventos.

Espíndola (online) apresenta esse conceito de uma forma bastante simples em seu artigo “A Importância da Modelagem de Objetos no Desenvolvimento de Sistemas” para o site Linha de Código. Ele diz que o software pode ser analisado sob diferentes aspectos, de acordo com o interesse e necessidade de quem realiza a análise. Por exemplo, o profissional responsável pela criação e manutenção do banco de dados direciona seus modelos para os relacionamentos entre as entidades e tabelas, dando atenção especial aos processos de armazenamento e aos eventos que os iniciam. Já o foco será para os algoritmos, e o fluxo de dados de um evento, para o outro, se a modelagem for designada para as análises de um profissional responsável pela estrutura interna do software, que tem preocupações em manter um código simples e eficiente. Já se o sistema for orientado a objetos, uma modelagem necessária seria em relação à interação entre as classes e o funcionamento delas em conjunto.

Na unidade II, estudaremos os principais tipos de modelos nos aprofundando um pouco mais nesse assunto.



REFLITA

A palavra escrita é um veículo magnífico de comunicação, mas não é necessariamente o melhor modo de representar os requisitos de software para computador.

Fonte: Pressman (2010, p.144).

MODELAGEM DE SOFTWARE E O PROCESSO DE DESENVOLVIMENTO

O modelo de processo de desenvolvimento de software tradicional recomenda que a modelagem deve ser feita a partir da primeira versão aprovada do documento de requisitos. Na academia, dividimos as etapas desse processo em disciplinas distintas e as ensinamos em sequência para facilitar a didática; já, na prática, a modelagem de software pode ocorrer a qualquer momento, por exemplo, durante o levantamento de requisitos, o engenheiro, que não domina a regra de negócio, precisa de mecanismos e ferramentas que o auxiliem na interpretação de uma determinada rotina, junto com o especialista, ou o cliente, ele representa o fluxo de informações que demandam daquele requisito por meio de desenhos, isto é, um modelo.

Outra situação, quando apresentada, por exemplo, é uma solicitação de mudança ou quando é solicitada uma inclusão de rotina em softwares já em operação; para analisar o impacto dessas alterações por todo o sistema e para encontrar a melhor proposta de código e banco de dados, o analista de sistemas irá observar todas as tabelas e suas ligações, uma forma prática de ser fazer isso é por meio de representações gráficas, que são os modelos. O ponto aonde quero chegar é que, mesmo não utilizando uma linguagem de modelagem formal para desenvolver um software, sempre é feito algum tipo de modelo, entretanto esses modelos informais nem sempre apresentam uma linguagem comprehensível por leitores que não participam do processo.

Sommerville (2011, p. 82) diz que os modelos são usados durante o processo de engenharia de requisitos auxiliando no levantamento, são usados também durante a fase de projeto, auxiliando na interpretação dos requisitos, e também como documentação, registrando a estrutura e a operação do software - a ser desenvolvido ou de um já existente. Pressman (2010, p. 145) acrescenta que, a partir dos requisitos, o modelador (engenheiro ou analista) pode construir modelos que descrevam cenários de usuários, atividades funcionais, classes de problemas e seus relacionamentos, comportamento do sistema e das classes, e fluxo dos dados, à medida que são transformados. Nesse sentido, o modelo de software fica entre uma descrição em nível de sistema e o projeto de software.

A Figura 1 apresenta esse relacionamento.

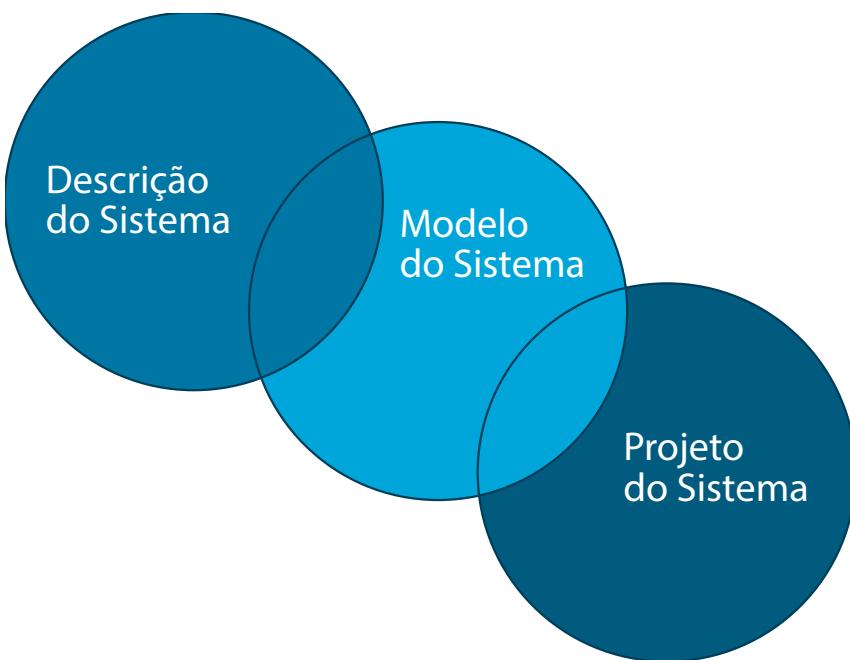


Figura 1: Modelo de Software como uma ponte entre a descrição e o projeto do sistema
Fonte: adaptado de Pressman (2010, p. 146).

Existem no mercado muitas propostas, com vários formatos, roteiros e artefatos, para o processo de desenvolvimento de softwares; cada profissional ou cada empresa vai decidir pelo processo que melhor se adéque a sua realidade e necessidade. Por exemplo, Engholm (2010, p. 66) sugere um roteiro para o desenvolvimento de softwares orientados a objeto, contendo os seguintes processos: elicitação de requisitos, análise de requisitos, arquitetura, design, implementação e testes, propõe que o desenvolvimento de software é a transformação do modelo mental dos envolvidos no projeto (*stakeholders*) para o código e, utilizando a linguagem padronizada UML® (Estudaremos a UML® na unidade V), usa modelos em todas as fases do seu processo.

Morais (2011) observa que os modelos são bem aceitos dentro das metodologias ágeis de desenvolvimento de software. Ele diz que a modelagem ágil procura focar o desenvolvimento de artefatos apenas quando agregarem valor

real ao projeto, diferente da tradicional, que os utiliza. Ela não é prescritiva, fornece propostas para ganhar efetividade. A modelagem ágil procura o equilíbrio entre a criação dos artefatos e o custo de manutenção desses artefatos e, conforme Moraes (2011) afirma, toma por base dois princípios: (1) O objetivo primário de um projeto de software é o próprio software, e não um grande conjunto de documentos sobre ele; (2) um artefato é criado primordialmente para permitir a comunicação e a troca de informações entre a equipe e permitir a discussão e refinamento do modelo do sistema.

Deboni (2003, p. 26) considera a modelagem necessária em todas as etapas do processo de desenvolvimento e nos diz que o processo de desenvolvimento de software é realizado com base na evolução de uma visão que os engenheiros e desenvolvedores constroem em conjunto com o cliente sobre o problema a ser resolvido. Essa visão é traduzida pelos modelos e deve evoluir juntamente com a evolução do processo de desenvolvimento. Complementa que uma única visão não é suficiente para traduzir todas as necessidades e a complexidade do problema, para isso, é necessário um conjunto coerente de modelos em escala diferentes e criadas de diferentes pontos de vista. Ele propõe três tipos de modelo, a fim de garantir a introdução da complexidade nos modelos aos poucos, na medida em que a análise evolui:

1. **Modelo de contexto:** inicia com a definição do problema, e é construído em alto nível de abstração, onde a maioria dos problemas não é considerada. Utilizado para definir as fronteiras do sistema e contextualizar o ambiente.
2. **Modelo conceitual:** esse modelo reúne todos os conceitos dos problemas existentes, construído em um nível menor de abstração que o de contexto. O objetivo nesse ponto da análise é a construção de um modelo simplificado de classes que contenha os principais componentes e suas relações. Esse modelo contém a proposta de solução do problema, mas abstrai os detalhes de implementação.
3. **Modelo detalhado:** esse modelo representa todos os detalhes de uma versão projetada do software, e pode possuir uma equivalência ao código. Desenvolvido em sistemas automatizados de manutenção de modelagem, pode sofrer alterações quando o código for alterado.



IMPORTÂNCIA DA MODELAGEM DE SOFTWARE

A modelagem de software comumente representa o software por meio de algum tipo de notação gráfica. A notação mais comumente utilizada como base para a modelagem é a UML®, que é apresentada por Cardoso (2003, p. 3) como um conjunto de diagramas com suas finalidades, porém, sem nenhuma ligação ou sequência definida pela linguagem, o que não orienta o processo de desenvolvimento, isto é, a UML® oferece suporte gráfico para o entendimento do sistema no ponto em que ele é mais necessário.

Para promover o entendimento sobre a importância dos modelos no desenvolvimento de software, recorro à teoria da imagem para estabelecer um referencial e reforçar meu ponto de vista. Os primeiros vestígios de comunicação humana que se tem conhecimento foram feitas por meio de imagens, as pinturas rupestres (antigas representações pictóricas, datadas do período Paleolítico (100.000-10.000 a.C.).



SAIBA MAIS

A modelagem, do ponto de vista ágil, é um método eficiente que tem como objetivo tornar mais produtivos os esforços da tarefa de modelar, tão comum nos projetos de software. Os valores, princípios e práticas da Modelagem Ágil podem auxiliar as equipes na definição de componentes técnicos de alto e baixo nível que farão parte do desenvolvimento de software. Artefatos sofisticados elaborados por ferramentas de alto custo nem sempre são os melhores para ajudar no desenvolvimento do software. Uma boa prática é modelar o software em grupo e com a participação dos usuários, utilizando rascunhos.

Aprofunde seu conhecimento lendo este artigo na íntegra, disponível em: <<http://www.devmedia.com.br/modelagem-em-uma-visao-agil-engenharia-de-software-32/19006>>. Acesso em: 20 out. 2015.

Fonte: Moraes (2011, online).

Silva (2008) apresenta, em seu estudo sobre a importância da imagem no ensino-aprendizagem, que quando se compara as definições de imagem ao longo

da história é possível perceber que a imagem é conceituada como sendo algo capaz de representar visualmente outra coisa, que pode estar ausente. Seguindo essa linha de raciocínio e considerando a complexidade de se definirem rotinas, algoritmos, cardinalidade, interações por meio da linguagem escrita, a utilização de gráficos e diagramas para a compreensão das funções de um software assume grande importância, como facilitadores na transmissão de informação.

Outro fator que pode ser usado para mensurar a importância dos modelos no processo de desenvolvimento de software é que, mesmo para aqueles leitores não especialistas, a mensagem vinculada na forma de imagem possibilita um entendimento mais fácil, mais direto. Um projeto de software não é composto apenas por elementos tecnológicos, um importante e decisivo elemento que compõe o projeto de software é o social. Os elementos tecnológicos são responsáveis pela construção do software enquanto os elementos sociais, pelo relacionamento entre os desenvolvedores e os técnicos, e devemos sempre considerar que o software é desenvolvido por pessoas para ser utilizado por outras pessoas (DEBONI, 2003, p. 19). A comunicação facilitada pelos modelos pode garantir o sucesso do projeto de software.

Aproveitando ainda o trabalho de Silva (2008), constatamos que a leitura visual sempre foi importante para o homem, desde os primórdios da humanidade, uma vez que as imagens significam mais do que representações de objetos, elas despertam considerações que excedem a percepção visual. Isso quer dizer que o processo de assimilação e retenção da informação transmitida por uma imagem ocorre de forma emocional e subliminar e, por isso, é bem mais fácil de interpretar que o de uma redação ou palavra.

Projetar software é construir um modelo. Um modelo que representa de uma forma mais simples o que se pretende construir. A engenharia de software procura trazer para a ciência da computação o que já é tão utilizado nas outras disciplinas de engenharia, como a civil, onde a figura clássica de um engenheiro civil é uma pessoa envolvida por plantas e diagramas enquanto comanda uma construção (DEBONI, 2003, p. 18).

Concluo essa discussão com as palavras de Confúcio: “uma imagem vale mais que mil palavras”¹, sendo assim, a modelagem de software é a representação gráfica, é a imagem do que será codificado.

¹ Disponível em: <<http://pensador.uol.com.br/frase/NTcxMjMz/>>. Acesso em: 03 set. 2015.



MODELOS E O QUE ELES MODELAM

É óbvio o que eu vou afirmar, mas um modelo é sempre um modelo de alguma coisa. A “coisa” que está sendo modelada, genericamente falando, pode ser considerada um elemento dentro de algum domínio qualquer. O modelo fará declarações sobre esse elemento abstraindo detalhes a partir de certo ponto de vista e para um determinado fim. Não se preocupe, estudaremos todos esses conceitos nas próximas unidades.

Para um sistema existente, o modelo pode representar uma análise das propriedades e do comportamento daquele sistema. Para um sistema em planejamento, o modelo pode representar uma especificação de como o sistema será construído e como deverá se comportar.

Um método de modelagem deve considerar três categorias nas suas propostas de visões (OMG, 2015, p. 12):

- Objetos: suporte para a descrição de um conjunto de objetos que possuem identidade e relacionamentos com outros objetos.
- Eventos: um evento descreve um conjunto de possíveis ocorrências.
- Comportamentos: um comportamento descreve um conjunto de possíveis execuções.

Para modelar, precisamos adotar uma notação padrão: uma ferramenta e uma linguagem que forneça os recursos necessários para a representação de toda a complexidade envolvida no desenvolvimento de um software.

CONSIDERAÇÕES FINAIS

Nesta unidade, estudamos os principais conceitos referentes à atividade de modelagem de software. Foi possível observar que a modelagem deve estar relacionada com o processo de engenharia de requisitos, garantindo uma ponte entre as etapas de definição do sistema e projeto, que representam as atividades mais importantes do processo de desenvolvimento de software.

Sabemos que a perfeita interpretação dos requisitos é essencial para o sucesso do desenvolvimento do software. Os modelos se mostram, nesse contexto, ótimos aliados para promoção da discussão entre a equipe técnica e usuários, pois promovem uma especificação clara e precisa. Dentro do processo de desenvolvimento de software o modelo completa o processo de desenvolvimento garantindo como resultado uma metodologia de desenvolvimento. Outro fator que pode ser usado para mensurar a importância dos modelos no processo de desenvolvimento de software é que, mesmo para aqueles leitores não especialistas, a mensagem vinculada na forma de imagem possibilita um entendimento mais fácil, mais direto. Sistemas de informação estão em constante mudança. Essas mudanças ocorrem por vários fatores, como, por exemplo: os clientes solicitam modificações constantes ou por exigência do mercado que está em mudança constante. Assim, um sistema precisa de documentação detalhada, atualizada e que seja fácil de ser mantida. A modelagem se apresenta também como uma forma bastante eficiente de documentação.

Vimos também que a modelagem de software comumente representa o software por meio de algum tipo de notação gráfica. A notação mais comumente utilizada como base para a modelagem é a UML® e será estudada com mais critério na unidade V. Finalmente estudamos que os modelos de software podem ser desenvolvidos em vários níveis de abstração, que possibilitam a visão do software de diferentes perspectivas, por exemplo: perspectiva externa, perspectiva de interação, perspectiva estrutural e perspectiva comportamental.

ATIVIDADES



1. Defina, com suas palavras, o termo Modelagem de Software.
2. Quais são os objetivos gerais de um modelo de software?
3. O que significa abstração no contexto da modelagem de software?



POR QUE OS PROJETOS DE TI FRACASSAM?

O sucesso de um projeto de TI pode ser determinado pela satisfação geral das partes interessadas (NELSON, 2005). A satisfação das partes interessadas pode ser mensurada a partir da combinação de 6 critérios: critérios de processo – tempo (cumprimento de cronograma), custo (execução dentro do orçamento) e produto (atendimento aos requisitos com qualidade aceitável) – e critérios de resultado – uso (produto efetivamente usado), aprendizado (extração de lições aprendidas relevantes para a organização) e valor (melhoria efetiva do negócio) (NELSON, 2005). É responsabilidade do gerente de projetos estabelecer junto às partes interessadas as métricas de avaliação dos critérios de sucesso. No entanto, a partir de entrevista realizada a um número significativo de grupos de partes interessadas de diferentes projetos, conclui-se que os critérios mais relevantes para o sucesso são produto, uso e valor (NELSON, 2005).

Por outro lado, o fracasso de um projeto de TI pode ser definido como o abandono total do projeto antes ou logo depois de o produto ter sido entregue (CHARETTE, 2005).

Diversos são os fatores que levam projetos de TI ao fracasso. Tais fatores têm impacto direto em um ou mais critérios de satisfação das partes interessadas. Planos de projeto medíocres, que não incluem um planejamento de riscos adequado, *business cases* que não evidenciam a conexão entre as necessidades do projeto e do negócio, baixo suporte e envolvimento da alta gestão e usuários, imaturidade das tecnologias adotadas e indefinição quanto ao responsável pelo negócio estão entre os fatores críticos que levam projetos de TI ao fracasso (WHITTAKER, 1999; YARDLEY, 2002).

Fonte: Atalla (online).

MATERIAL COMPLEMENTAR



LIVRO

Engenharia de Software: Uma Abordagem Profissional

Roger S. Pressman

Editora: Bookman

Sinopse: Versão concebida tanto para alunos de graduação quanto para profissionais da área. Oferece uma abordagem contemporânea sobre produção do software, gestão da qualidade, gerenciamento de projetos, com didática eficiente e exercícios de grande aplicação prática.



TIPOS DE MODELOS

Objetivos de Aprendizagem

- Compreender por que diferentes tipos de modelos são necessários.
- Apresentar as perspectivas fundamentais de modelagem de software

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Modelos de contexto
- Modelos de interação
- Modelos estruturais
- Modelos comportamentais

INTRODUÇÃO

Olá! Vamos para mais uma unidade de estudos? Na unidade I, estudamos sobre o auxílio que os modelos oferecem na interpretação e na visualização de um software em projeto, um pedaço dele ou ainda um existente, oferecendo uma ilustração de diferentes perspectivas e níveis de abstração. Quando estamos iniciando na área de engenharia de software, às vezes é complicado entender o que significa “diferentes perspectivas” ou “níveis de abstração”, para facilitar, vamos considerar que estamos na fase de levantamento de requisitos, da engenharia de requisitos, e vamos seguir a classificação feita por Sommerville (2011) para a modelagem de sistemas.

Ele organiza a atividade de modelagem de dados partindo do nível mais alto de abstração, que é o contexto até o nível mais interno, que ele chama de comportamental. Nesta unidade, estudaremos sobre os tipos de modelagem possíveis de desenvolvimento, considerando as perspectivas, ou visões, relacionadas por Sommerville (2011, p. 83): perspectiva externa, perspectiva de interação, perspectiva estrutural, perspectiva comportamental.

Os modelos de contexto representam a perspectiva externa do software, isto é, o seu relacionamento com os elementos exteriores a ele que irão impactar e sofrer impactos pelo seu funcionamento, ou seja, define os limites externos do sistema.

Os modelos de interação são voltados para a perspectiva de interação; oferecem uma visão de como é feito o diálogo entre o sistema e o usuário. Serão apresentados o diagrama de casos de uso e o de sequência da UML®.

Os modelos estruturais representantes da visão estrutural do desenvolvimento de software registram a estrutura interna do software, seus componentes, elementos de banco de dados e os relacionamentos entre eles. Será apresentado o diagrama de classes da UML®. Por último, estudaremos os modelos comportamentais representantes dos procedimentos dinâmicos do software.

MODELOS DE CONTEXTO

Qualquer discussão sobre contexto deve começar focando o termo contexto. Contexto dentro da informática pode assumir diferentes posturas, dependendo do ambiente em que é empregado. Por exemplo, em sistemas computacionais, contexto é um instrumento de apoio à comunicação entre os sistemas e seus usuários, onde, a partir da compreensão do contexto, o sistema pode mudar a sequência de ação, o tipo de informação oferecida ao usuário. As áreas da Computação Ubíqua e Inteligência Artificial foram as pioneiras nos estudos e na utilização do conceito de contexto (VIEIRA; TEDESCO; SALGADO, online). Ainda falando de sistemas adaptativos, os modelos de contexto representam características atuais do usuário e do ambiente (MACHADO; OLIVEIRA, 2013). Para o nosso estudo, contexto também representa o ambiente, mas como ferramenta limitadora.

Os modelos de contexto representam as perspectivas externas de onde será modelado o ambiente do sistema, os seus limites e suas relações técnicas e também as relações não técnicas. Por exemplo, um limite do sistema pode ser estabelecido de forma que o processo da análise ocorra em um site, ou pode ser definido de forma que um gerente, particularmente difícil, não precise ser consultado (SOMMERVILLE, 2011, p. 84).

Nem sempre as barreiras entre um software e seu ambiente são claras, por isso é muito importante que a definição do contexto do sistema seja feita juntamente com os *stakeholders* do cliente. Nesse sentido, os modelos de contexto podem ser desenvolvidos juntamente com as duas primeiras tarefas da engenharia de requisitos, Levantamento e Negociação - estudados na disciplina de Engenharia de Requisitos. Outras atividades da engenharia de requisitos podem ser usadas para identificar interessados, definir escopo do problema, especificar os objetivos operacionais e descrever os objetos que serão manipulados pelo sistema (PRESSMAN, 2010, p. 153).

REFLITA



“Tornou-se chocantemente óbvio que a nossa tecnologia excede a nossa humanidade.”

Fonte: Albert Einstein.

O modelo de contexto, então, pode ser utilizado como um instrumento da análise funcional que irá identificar os seguintes subsídios:

1. Os elementos externos que irão interagir com o sistema.
2. O fluxo de informação existente entre o sistema e o ambiente externo.
3. Os limites do sistema.
4. Os eventos do ambiente externo que impactam (provocam alterações) o sistema provocando respostas.

Um modelo de contexto pode ser montado utilizando dois componentes, um diagrama de contexto e uma lista de eventos. O diagrama de contexto é uma representação gráfica do sistema com seu ambiente, e a lista de eventos relaciona os episódios do contexto externo que o sistema deve obrigatoriamente considerar.

A Figura 2 representa um diagrama de contexto genérico.



Figura 2: Diagrama de Contexto Genérico

Fonte: a autora.

A Tabela 1, Lista de Eventos Genérica, sugere como uma lista de eventos pode ser construída.

| ESTÍMULO | RESPOSTA | EVENTO |
|----------|----------|--------|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

Tabela 1: Lista de Eventos Genérica

Fonte: a autora.

Ações para a construção dos artefatos:

1. Identificar os eventos externos.
2. Identificar os fluxos.
3. Produzir a tabela de eventos.
4. Analisar se todos os eventos foram considerados.

Para facilitar a compreensão, vamos analisar alguns exemplos práticos, primeiro o desenvolvido por Sommerville (2011, p. 84), que modela o contexto do projeto de um Sistema de Gerenciamento da Saúde Mental de Pacientes (MHC-PMS).

O sistema destina-se a gerenciar informações sobre os pacientes que procuram clínicas de saúde mental e os tratamentos prescritos. Ao desenvolver a especificação para esse sistema, é preciso decidir se ele deve se concentrar exclusivamente em coletar informações sobre as consultas (usando outros sistemas para coletar informações pessoais sobre os pacientes) ou se deve também coletar informações pessoais dos pacientes.

A Figura 3 mostra o contexto do sistema de forma simples que mostra como o MHC-PMS irá se relacionar com os demais sistemas (ambiente externo ao MHC-PMS).

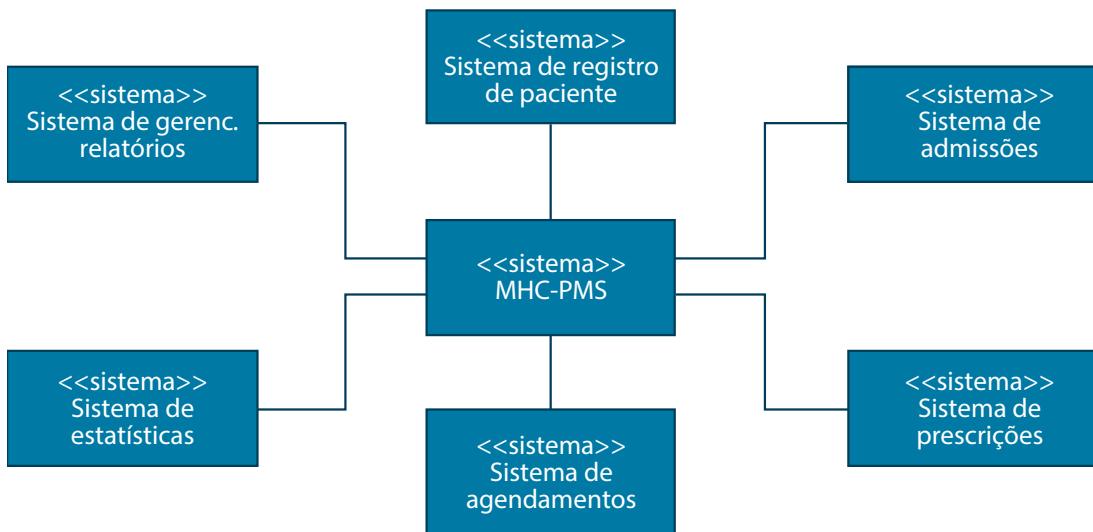


Figura 3: Contexto de um sistema MHC-PMS

Fonte: adaptada de Sommerville (2011, p. 84).



Os modelos de desenvolvimento de software orientado ao planejamento, ágil e livre, têm características em comum e diferenças significativas, mas nenhum deles é efetivo para todos os projetos. Apesar da maioria das propostas existentes para a conciliação de processos de desenvolvimento de software envolver uma combinação rígida das práticas dos diferentes modelos, esse trabalho de pesquisa utiliza a gestão de contexto para obter o equilíbrio entre a colaboração e a disciplina dos processos de desenvolvimento. Para esse balanceamento, é necessário entender e caracterizar o contexto que envolve as pessoas e o ambiente em que o desenvolvimento de software ocorre.

Saiba mais sobre a influência do contexto acessando a pesquisa no endereço disponível em: <http://reuse.cos.ufrj.br/files/publicacoes/relatorioTecnico/RT_VanessaAndrea_ModelagemContexto.pdf>. Acesso em: 20 out. 2015.

Fonte: Nunes, Magdaleno e Werner (2010, online).

Observe na Figura 3 que o MHC-PMS se conecta com os outros sistemas: sistema de agendamento, sistema de registro de pacientes, sistema de admissões, sistema para gerenciamento de relatórios, sistema de estatística e sistema de prescrições. Esses sistemas definem os limites do MHC-PMS e interagem com ele, inserindo e recolhendo informações, portanto o MHC-PMS sofre e promove influência desses outros sistemas. Nesse contexto, os sistemas são o ambiente externo do MHC-PMS, pois todos os fluxos de dados destinados a ele virão dos sistemas. Nas próximas unidades, esse processo será mais detalhado.

Agora vamos utilizar, como exemplo, a especificação de um software comercial. Considere um sistema de controle de inadimplência simplificado, que deverá executar as rotinas de: registrar pagamentos de clientes; emitir comprovantes de pagamento para os clientes que efetuarem o pagamento; emitir extrato para os clientes quando solicitado; emitir relatório de inadimplentes quando solicitado. A Figura 4 representa o modelo do contexto do software de controle de inadimplência.

Observe o seguinte:

- O cliente é fornecedor e receptor de informações do sistema.
- O sistema gera informações para algum responsável pelo relatório de inadimplência.
- O sistema recebe informações do cliente quando efetivado um pagamento.
- O sistema fornece recibo e extrato para o cliente

O Cliente e o Responsável pelo relatório de inadimplência são entidades externas, fontes e receptores das informações geradas pelo e para o sistema. O pagamento do Cliente é um fluxo de dados que o sistema interpreta – podemos considerar como um fluxo de entrada do sistema. O Recibo, o Extrato e o Relatório de Inadimplência são fluxos de dados que o sistema emite – fluxo de saída do sistema.



Figura 4: Diagrama de Contexto do Sistema de Controle de Inadimplência

Fonte: a autora.

Com base na Figura 4, vamos analisar o Diagrama de Contexto em relação ao que foi registrado como ambiente externo. Consideramos como ambiente externo o Cliente e o Responsável pela cobrança, pois não fazem parte do sistema, mas interagem com ele, isto é, geram impacto e provocam respostas do sistema.

Nesse exemplo, é possível incrementar no sistema rotinas que executem a cobrança, incorporando essa responsabilidade no escopo do sistema. Se assim for decidido, a entidade responsável pela cobrança deixaria de ser ambiente externo e não comporia o diagrama de contexto. Essas são as decisões que devem ser tomadas no momento da concepção do sistema e durante a engenharia de requisitos, onde o modelo auxilia na compreensão dos elementos do sistema e facilita a comunicação entre o engenheiro e o cliente.

Para a elaboração do Diagrama de contexto, considere sempre que os fluxos são dados em movimento e podem representar dados de entrada (fluxos de entrada), mas também podem representar dados de saída (fluxos de saída). As entidades externas são as geradoras desses fluxos, elas afetam ou são afetadas pelo sistema, isto é, elas podem fornecer ou receber dados do sistema.

Vamos agora construir uma Lista de Eventos. A Lista de Eventos é composta por três elementos: os Eventos, os Estímulos e as Respostas. Os eventos são ações que estimulam uma reação do sistema. Eles podem ser externos e temporais. Os externos, como a própria denominação insinua, ocorrem fora do sistema, por exemplo, quando o cliente paga a conta. Você deve estar se perguntando, mas o registro do pagamento da conta não é feito no sistema, isso não é interno? Sim, mas o processo vem de um elemento externo; o cliente precisa efetivamente

efetuar o pagamento, que então gera a reação do sistema, que é processar essa ação. As características de um evento externo são: ocorre fora do sistema, provoca uma reação no sistema, gera uma resposta do sistema. Os eventos temporais não estão atrelados a ações externas ao sistema, eles são eventos programados e relacionados ao tempo, por exemplo, um sistema de controle de tráfego que precisa ter acesso a um satélite de hora em hora para acessar fotos e atualizar suas informações.

Os estímulos são os fluxos de dados encarregados de comunicar ao sistema que um evento externo ocorreu. Esse fluxo carrega todos os dados da ação executada pelo evento externo e provoca a reação de resposta no sistema. Um estímulo está sempre ligado a um evento externo. Um evento temporal nunca está ligado a um estímulo.

As respostas são os fluxos de dados encarregados de carregar as informações de saída do sistema, o resultado do processamento. São geradas respostas tanto na ocorrência de estímulos externos quanto na dos temporais.

Ao compor a Lista de Eventos, nomeie os eventos utilizando uma frase simples que indique a ação que será executada fora do sistema, por exemplo, Cliente realiza pagamento. Para relacionar os estímulos, considere a ação executada em si. Para compor as respostas na lista, verifique qual foi a resposta do sistema para aquele estímulo. A Tabela 2 ilustra as ideias apresentadas.

| ESTÍMULO | RESPOSTA | EVENTO |
|----------------------------------------------|-------------|----------------------------|
| Cliente executa pagamento | Pagamento | Recibo de pagamento |
| Cliente emite extrato | Solicitação | Extrato de pagamento |
| Responsável emite relatório de inadimplentes | Solicitação | Relatório de inadimplentes |

Tabela 2: Lista de Eventos do Sistema de Controle de Inadimplência
Fonte: a autora.

Claro que essa é uma versão muito simplificada de um sistema de cobrança, muitos outros eventos externos podem e devem acontecer. Esse foi um exemplo para auxiliar a compreensão do conteúdo.

A lista de eventos pode ser substituída ou aprimorada para uma especificação de requisitos e pode evoluir conforme evoluí a interpretação das funções e limites do sistema.

Podemos considerar os modelos de contexto como resultado de um primeiro contato entre o cliente e o engenheiro de software e a equipe de desenvolvimento; depois de refinados e levantados todos os eventos, seus estímulos e suas respostas, é possível ter uma boa noção do que o cliente deseja do sistema e, então, é possível definir o documento preliminar de requisitos apoiado no Diagrama de Contexto e seguir para a próxima etapa, que é definir como será a interação entre o sistema e os usuários.



REFLITA

"Acho que vírus de computador deve contar como vida. Creio que dizem algo sobre a natureza humana que a única forma de vida que criamos até agora é puramente destrutiva. Nós criamos vida à nossa própria imagem."

Fonte: Stephen Hawking.

MODELOS DE INTERAÇÃO

As interações são os aspectos dinâmicos do sistema, isto é, são os fluxos de dados – também denominados na literatura por mensagens – trocados entre os elementos que compõem o sistema com o objetivo de realizar alguma ação. No exemplo do sistema de controle de inadimplência que usamos no tópico anterior, o fluxo de dados entre o cliente e o sistema (efetua pagamento) é uma interação entre o usuário e o sistema; no exemplo do MHC-PMS, as interações representadas lá ocorrem entre sistemas. Vamos entender melhor isso?

Sommerville (2011, p. 86) diz que todos os sistemas envolvem algum tipo de interação, que podem ser do usuário, do tipo entrada e saída (fluxos de entradas e fluxos de saída), entre o software que está sendo desenvolvido e outros softwares, ou ainda interações entre os próprios componentes do software.



SAIBA MAIS

A interação entre máquinas e humanos faz parte da história da humanidade. Essa necessidade impulsiona o desenvolvimento de interfaces sempre mais elaboradas e sempre mais simples de utilização.

Interface, até pouco tempo, era referência de softwares que tinham como objetivo a função de interpretar comandos humanos e reproduzi-los digitalmente e vice-versa. Atualmente, essa abordagem agrega aspectos como processamento perceptual, motor, viso-motor e cognitivo do usuário. A IHC é uma subárea da Engenharia de Software (ES) que propõe modelos de processos, métodos, técnicas e ferramentas para o desenvolvimento de sistemas interativos.

Leia mais sobre o artigo Interação Humano-Computador no link disponível em: <<http://www.devmedia.com.br/artigo-engenharia-de-software-16-interacao-humano-computador/14192#ixzz3mBrKY3H0>>. Acesso em: 20 out. 2015.

Fonte: Moreira (online).

Os diagramas de interação representam a ação interna do software para que o usuário alcance a resposta esperada. A modelagem de um sistema geralmente demanda vários diagramas de interação, o conjunto de todos esses diagramas, segundo Bezerra (2014), constitui o seu modelo de interações.

Os principais objetivos de um diagrama de interação são: (1) levantar informações adicionais para complementar os modelos estruturais e comportamentais; (2) prover aos programadores uma visão detalhada dos objetos e das mensagens envolvidas na realização de uma determinada função. Nesse sentido, a autoridade máxima da interação é a **mensagem**, uma vez que as funcionalidades somente serão realizadas após a interação entre os elementos envolvidos pela troca de mensagens.

Os Casos de Uso e outros diagramas, como os diagramas de sequência e o diagrama de comunicação, são utilizados para modelar as interações de um sistema em projeto. Os diagramas de caso de uso e o diagrama de sequência representam as interações em níveis diferentes de detalhamento e, usados juntos, se complementam mutuamente. O diagrama de comunicação também representa as interações, mas é uma alternativa ao diagrama de sequência, tanto que algumas ferramentas Case de Modelagem de Dados (estudaremos as ferramentas de modelagem na unidade IV) geram um diagrama de comunicação a partir de um diagrama de sequência.

O Guia para Modelagem de Interações Metodologia CELEPAR (2009) reforça essa informação explicando que a modelagem de cada interação pode ser feita de duas formas: dando-se ênfase à ordem temporal (diagrama de sequência) ou dando-se ênfase à sequência das mensagens, conforme elas ocorrem no contexto da estrutura de composição dos objetos (diagrama de comunicação). Portanto, como ambos, o diagrama de sequência e o diagrama de comunicação, são derivados das mesmas informações, são semanticamente equivalentes e podem ser convertidos um no outro sem prejuízo de informações.

É importante ressaltar que eles oferecem abstrações distintas do cenário que estão representando, isto é, são pontos de vista diferentes e a escolha entre um e outro vai depender do que se pretende visualizar do sistema em um determinado momento do projeto ou da análise. Não trataremos, em nossos estudos, de exemplos utilizando o diagrama de comunicação, na unidade III, quando tratarmos da UML, ele será apresentado e conceituado.

CASO DE USO

O caso de uso é uma técnica de descoberta de requisitos, como já citado, criada por Jacobson em 1987; em sua forma mais simples, identifica os atores envolvidos em uma interação e nomeia essa interação. O conjunto de casos de uso é registrado em um Diagrama de Casos de Uso que representa todas as possíveis interações que serão consideradas pelo documento de requisitos do sistema. Esse diagrama tem como objetivo descrever um modelo funcional de alto nível do software em análise.

Segundo Deboni (2003, p. 80), os casos de uso são utilizados em todas as fases do desenvolvimento de um sistema: na fase inicial, no levantamento dos requisitos, durante a fase de *design*, quando são usados para auxiliar na criação de novas visões, além da funcionalidade do sistema; durante a codificação, quando os casos podem e devem ser explorados para promover a validação de cada nova funcionalidade implementada, verificando se está de acordo com o especificado pelo usuário.

Portanto, um caso de uso descreve um cenário de uso específico, utilizando uma linguagem promovida direto do ponto de vista de um ator definido. Mas como traduzir um requisito em casos de uso? Para iniciar o desenvolvimento de um conjunto de casos de uso, que mostrem valor como ferramenta de modelagem, as funções necessárias para o sistema devem estar listadas (PRESSMAN, 2010, p. 153). Vamos retomar o exemplo da modelagem do sistema MHC-PMS proposto por Sommerville (2011, p. 75). A descrição a seguir é representada pelo Caso de Uso Agendar Consulta representado pela Figura 5.

Agendar a consulta permite que dois ou mais médicos de consultórios diferentes possam ler o mesmo registro ao mesmo tempo. Um médico deve escolher, em um menu de lista de médicos on-line, as pessoas envolvidas. O prontuário do paciente é então exibido em suas telas, mas apenas o primeiro médico pode editar o registro. Além disso, uma janela de mensagem de texto é criada para ajudar a coordenar as ações. Supõe-se que uma conferência telefônica para comunicação por voz será estabelecida separadamente.

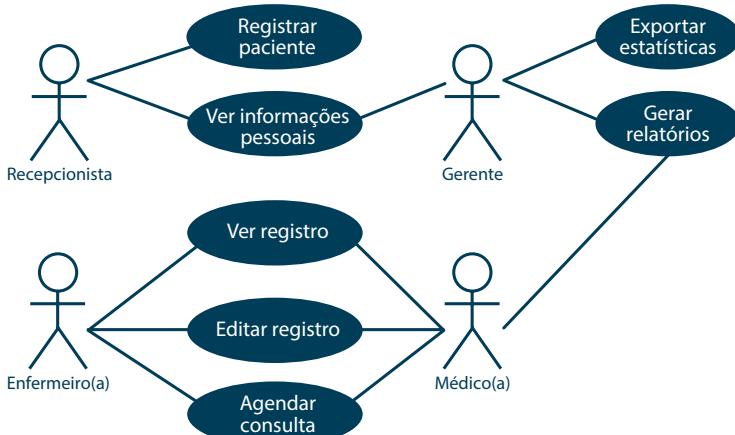


Figura 5: Casos de Uso Agendar Consulta.

Fonte: Sommerville (2011, p. 75).

Observe que temos quatro atores e sete casos de uso nessa visão e conseguimos, de maneira fácil, interpretar as mensagens que estão efetuando com o sistema. A Figura 6, Caso de Uso Transferência de dados, representa o exemplo de um caso específico de troca de mensagem, considerando especificação feita para o exemplo apresentado pelo contexto demonstrado pela Figura 3 – Diagrama de Contexto MHC-PMS.

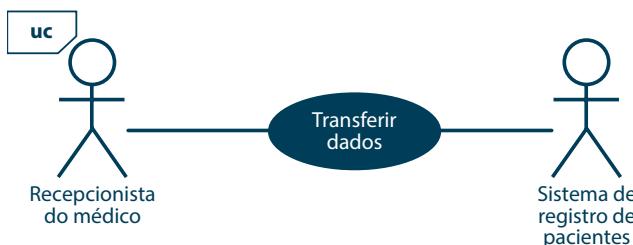


Figura 6: Use Case Transferência de Dados

Fonte: Sommerville (2011, p. 86).

Nesse caso de uso, um dos atores é outro sistema. Os diagramas de caso de uso representam a interação por meio de linhas sem setas, pois na UML as setas representam a direção do fluxo de dados (direção da troca de mensagens). Um diagrama de caso de uso abstrai esse detalhe.

Ficou claro que um caso de uso garante uma visão bastante simplificada da interação, portanto é necessária alguma complementação, conforme a necessidade de detalhes. Essa complementação pode ser uma simples descrição textual, mediante uma tabela, ou outro diagrama, como o de sequência. A decisão deve ser adequada à realidade e necessidade do projeto.

Para encerrar nossa discussão sobre o caso de uso, a Figura 7, Caso de Uso Efetua Pagamento, mostra o caso de uso para a modelagem do sistema de controle de inadimplência, mostrado pelo diagrama de contexto representado pela Figura 4.

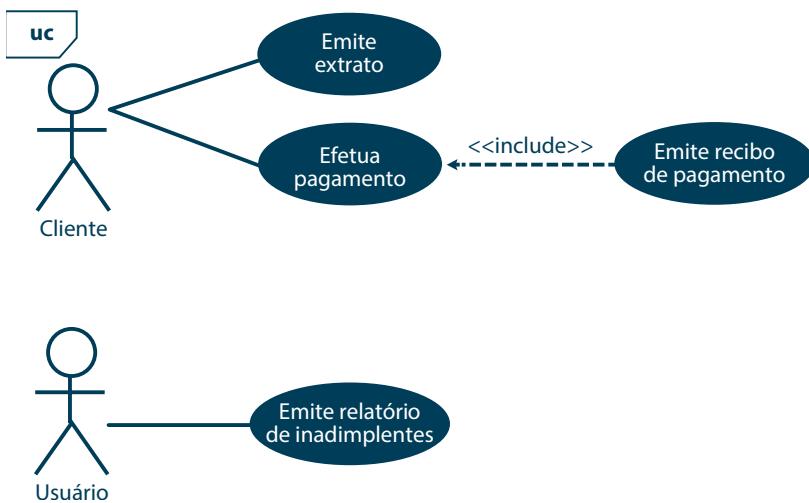


Figura 7: Caso de Uso Efetuar Pagamento

Fonte: Sommerville (2011, p. 86).

Observe que esse caso de uso apresenta uma notação diferente, o <<include>>. O include representa um caso de uso que será executado sempre que um outro for, isso quer dizer que o caso de uso Emite recibo de pagamento é essencial para o comportamento do caso Efetua pagamento. A UML permite dizer que Emite recibo de pagamento_é_parte_de_Efetua_pagamento. Observe também que lá no contexto a emissão de relatório de inadimplentes estava relacionada a um “responsável pela cobrança”, aqui foi substituído por um usuário, porque estamos considerando que essa pessoa responsável pela cobrança possa ser um ou vários usuários do sistema que forem designados para isso, então generalizamos, da mesma maneira que generalizamos cliente.

DIAGRAMA DE SEQUÊNCIA

Os diagramas de sequência na UML® são utilizados para representar as interações – troca de mensagens – entre os atores (atores que representam os elementos externos, ou os que representam outros sistemas) e os objetos, e também entre os próprios objetos. Quando falo objetos, estou me referindo aos elementos que compõem o sistema internamente.

Um diagrama de sequência é montado considerando duas dimensões: uma horizontal, que representa os objetos, e outra vertical, que representa o tempo. Como o nome indica, descreve a sequência e as interações que representam o comportamento dos objetos do sistema, que se relacionam pela troca de mensagens, sequencializando-as no tempo. Cada diagrama representa uma visão (um cenário) das mensagens organizadas por objetivos, ou seja, por funcionalidade do sistema.

Os objetos e atores considerados são listados na parte superior do diagrama, uma linha tracejada na vertical a partir deles representa o tempo. O fluxo de dados – a troca de mensagens – é representado por uma seta, que indica o sentido. Sommerville (2011, p. 87) complementa a descrição acrescentando que o retângulo na linha tracejada indica a linha da vida do objeto. A leitura deve ser feita de cima para baixo. As anotações sobre as setas indicam as chamadas para os objetos, seus parâmetros e os valores de retorno. A Figura 8, Diagrama de Sequência Transferir Dados, é um exemplo de Diagrama de Sequência que representa a troca de mensagens que ocorre para o Caso de Uso Transferir Dados, mostrado na Figura 6.

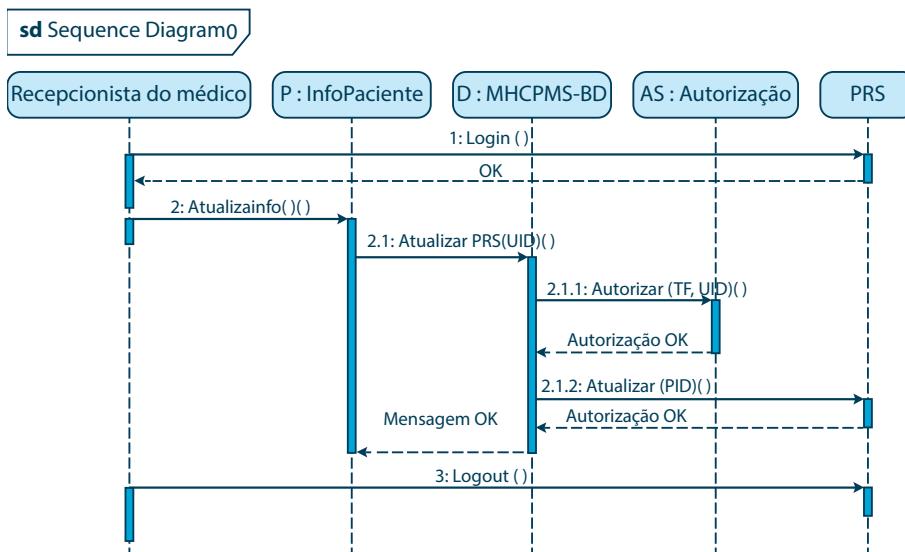


Figura 8: Diagrama de Sequência Transferir Dados
Fonte: adaptada de Sommerville (2011, p. 89).

A leitura desse diagrama pode ser feita da seguinte forma:

1. O Ator Recepção do Médico inicia uma sessão fazendo o login no sistema PRS.
2. A permissão desse Ator é verificada.
3. As informações pessoais do paciente são transferidas para o sistema banco de dados do sistema PRS.
4. Após a conclusão da transferência, uma mensagem de status é enviada pelo PRS para o Ator e a sessão de login é encerrada.

A Figura 9, Diagrama de Sequência Efetuar Pagamento, representa o diagrama de sequência para o Caso de Uso Emitir Pagamento, representado pela Figura 7. Observe que esse diagrama de sequência representa duas possibilidades de interação com o sistema pelo cliente, ele pode efetuar o pagamento, como também pode solicitar a emissão do extrato do cliente. Para atender à solicitação de emissão de extrato, perceba que os objetos trocam mensagens entre si internamente, para então enviar uma resposta ao cliente.

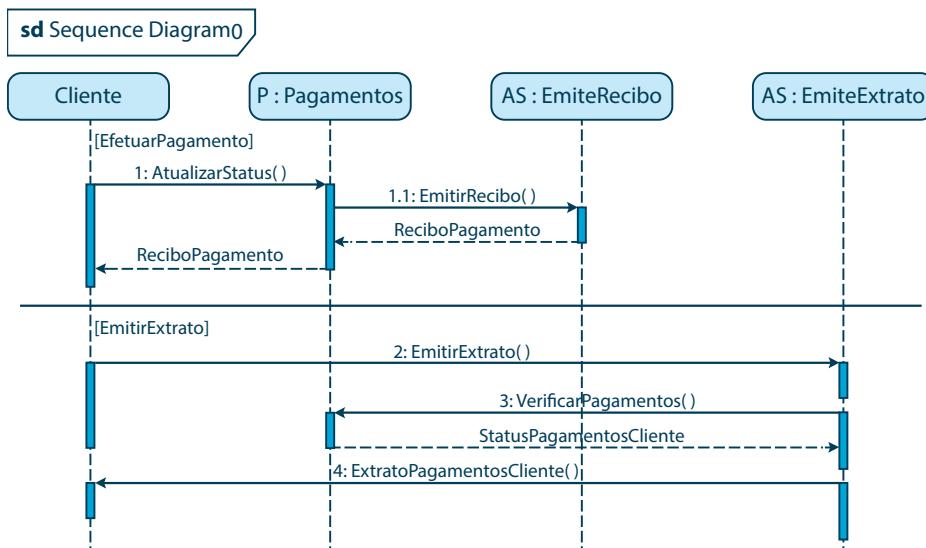


Figura 9: Diagrama de Sequência Efetuar Pagamento
Fonte: a autora.

A leitura desse diagrama pode ser feita da seguinte forma:

Na interação Efetuar pagamento:

1. O Ator Cliente efetua o pagamento e provoca uma alteração de status, antes devedor, para pago.
2. A alteração do status do cliente dispara uma solicitação para a emissão do recibo de pagamento.
3. O recibo de pagamento do cliente é emitido.

Na interação Emite extrato:

1. O Ator Cliente solicita a emissão de extrato.
2. O objeto EmiteExtrato busca as informações de pagamento do cliente e recebe a resposta do objeto Pagamentos.
3. O Extrato é emitido para o cliente.

O nível de detalhamento utilizado no diagrama de sequência depende de como ele será utilizado, se ele será utilizado como base de codificação ou como documentação; todas as interações e todos os parâmetros e mensagens devem ser considerados. Se for utilizado para apoio na engenharia de requisitos, o nível de abstração pode ser mais alto.

MODELOS ESTRUTURAIS

Passando para a perspectiva estrutural, os modelos estruturais representam a organização, a disposição e ordem dos elementos essenciais que compõem o sistema. Os modelos estruturais podem ser estáticos, que mostram a estrutura do projeto do sistema, ou dinâmicos, que mostram a organização do sistema quando está em execução (Sommerville 2011 pag. 89). Nós estudaremos nesta disciplina a modelagem da estrutura estática dos objetos em um software.



SAIBA MAIS

A engenharia dirigida a modelos (MDE, do inglês model-based-engineering) é uma abordagem do desenvolvimento de software segundo a qual os modelos, em vez de programas, são as saídas principais do processo de desenvolvimento. Os programas executados em um hardware/software são, então, gerados automaticamente a partir dos modelos. Os defensores da MDE argumentam que esta aumenta o nível de abstração na engenharia de software, e, dessa forma, os engenheiros não precisam mais se preocupar com detalhes da linguagem de programação ou com as especificidades das plataformas de execução. A engenharia dirigida a modelos tem suas raízes na arquitetura dirigida a modelos (MDA, do inglês model-driven-architecture), proposta pelo Object Management Group (OMG), em 2001, como novo paradigma de desenvolvimento de software. A engenharia e a arquitetura dirigidas a modelos são, frequentemente, vistas como a mesma coisa. No entanto, penso que a MDE tem um alcance maior que a MDA.

Saiba mais sobre a engenharia dirigida a modelos lendo os capítulos 6, 18 e 19 de Engenharia de Software, de Ian Sommerville.

Fonte: Sommerville (2011, p. 96).

DIAGRAMA DE CLASSES

A partir deste ponto, nossos objetos passam a se chamar “classes”. Em uma definição bastante generalizada, as classes representam um conjunto de objetos com as mesmas características, são matrizes de objetos, identificam grupos de elementos do sistema que compartilham as mesmas propriedades. O diagrama de classes é a representação fundamental da modelagem orientada a objeto e evolui de uma visão conceitual para uma visão detalhada durante a evolução do projeto (DEBONI, 2003).

No primeiro estágio, chamado de conceitual, o foco está em identificar os objetos e classificá-los em classes específicas, considerando suas características semelhantes. Simula o domínio em estudo, em uma perspectiva destinada ao cliente. Nesse estágio de modelagem, o Diagrama de Classes é chamado de Conceitual.

A Figura 10 traz o Diagrama de Classes Conceitual do MHC-PMS apresentado por Sommerville (2011). Observe que a figura representa as classes e as associações entre elas são representadas por uma linha. A associação apresenta o vínculo que incide, geralmente, em duas classes (binária), mas pode acontecer de uma classe se associar com ela mesma (unária) ou de se associar com mais de uma classe (n-ária).

O diagrama especifica também a multiplicidade que determina qual das classes envolvidas em uma associação fornece informações para as outras. Estudaremos mais sobre a multiplicidade na unidade III, quando analisaremos mais detalhadamente os diagramas da UML®. Outro exemplo do Diagrama de Classes no nível conceitual é representado pela Figura 11, Diagrama de Classes Conceitual Caixa Eletrônico.

A evolução do nível conceitual do diagrama de classes, chamado de especificação, tem foco nas principais interfaces da arquitetura e nos principais métodos, mas ainda não tem preocupação em como eles serão implementados, em uma perspectiva destinada àqueles envolvidos que não precisam dos detalhes da codificação, como o gerente de projeto ou o patrocinador. Para exemplificar a evolução, a Figura 12, Diagrama de Classes Especificação Caixa Eletrônico, mostra como fica o diagrama acrescido das principais interfaces, identificados por <<interfaces>>, e os principais métodos, relacionados na última partição da classe.

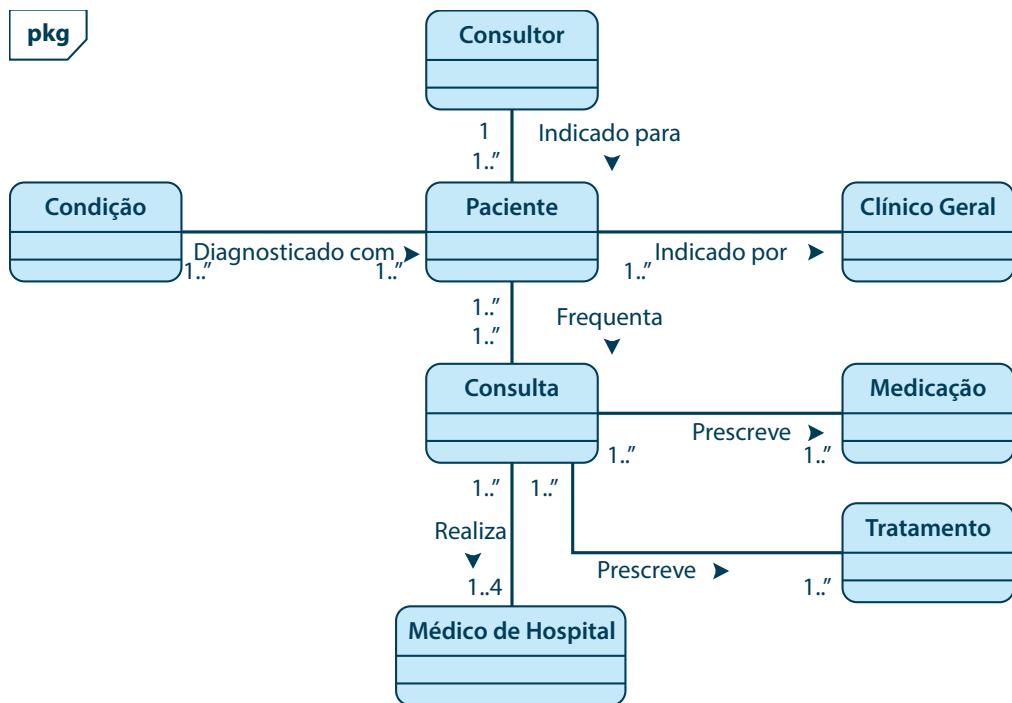


Figura 10: Diagrama de Classes Conceitual do MHC-PMS

Fonte: adaptada de Sommerville (2011, p. 91).

O último nível de detalhamento do diagrama de classes considera os requisitos em nível de implementação, como naveabilidade, tipo dos atributos, métodos etc. É uma perspectiva designada à equipe de programação. A Figura 13, Diagrama de Classes Implementação Caixa Eletrônico, mostra todos os detalhes necessários para a equipe de programação ser capaz de gerar o código do sistema a partir dele.

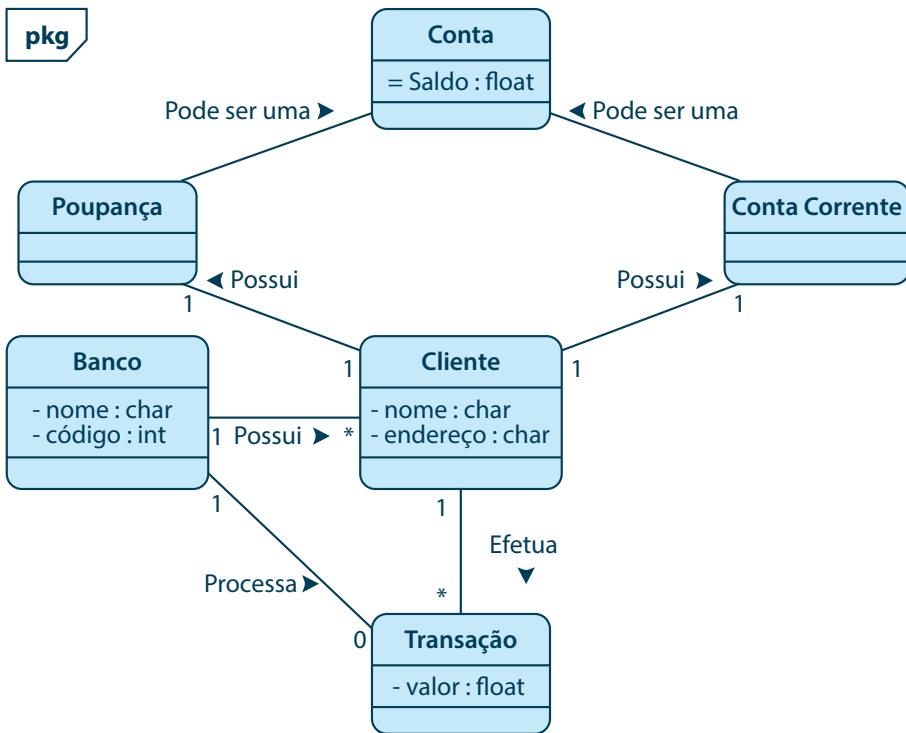


Figura 11: Diagrama de Classes Conceitual Caixa Eletrônico

Fonte: adaptada de UFCG (online).

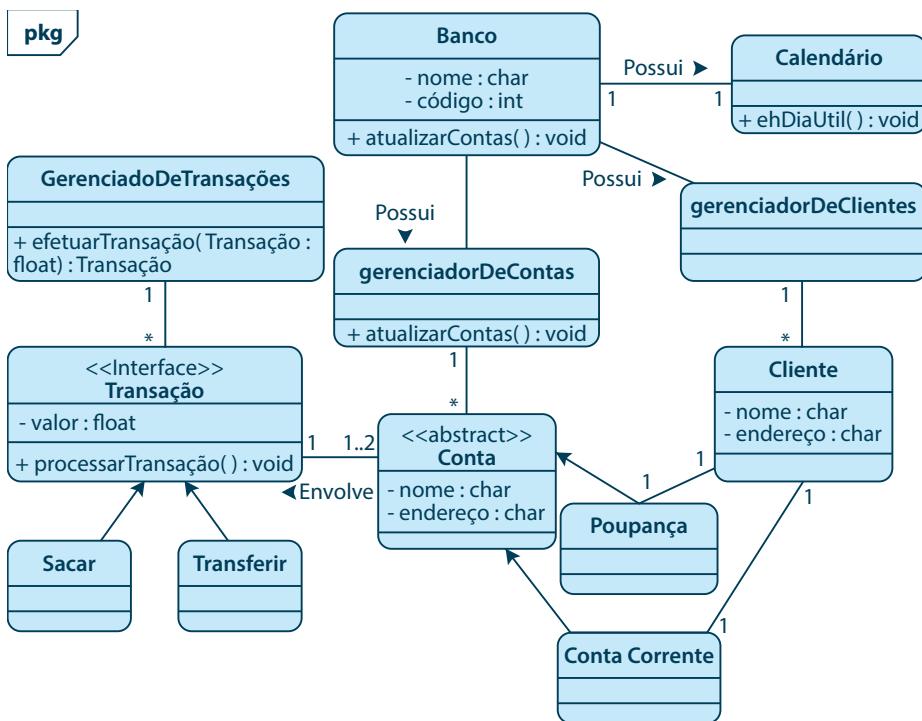


Figura 12: Diagrama de Classes Especificação Caixa Eletrônico
Fonte: adaptada de UFCG (online).

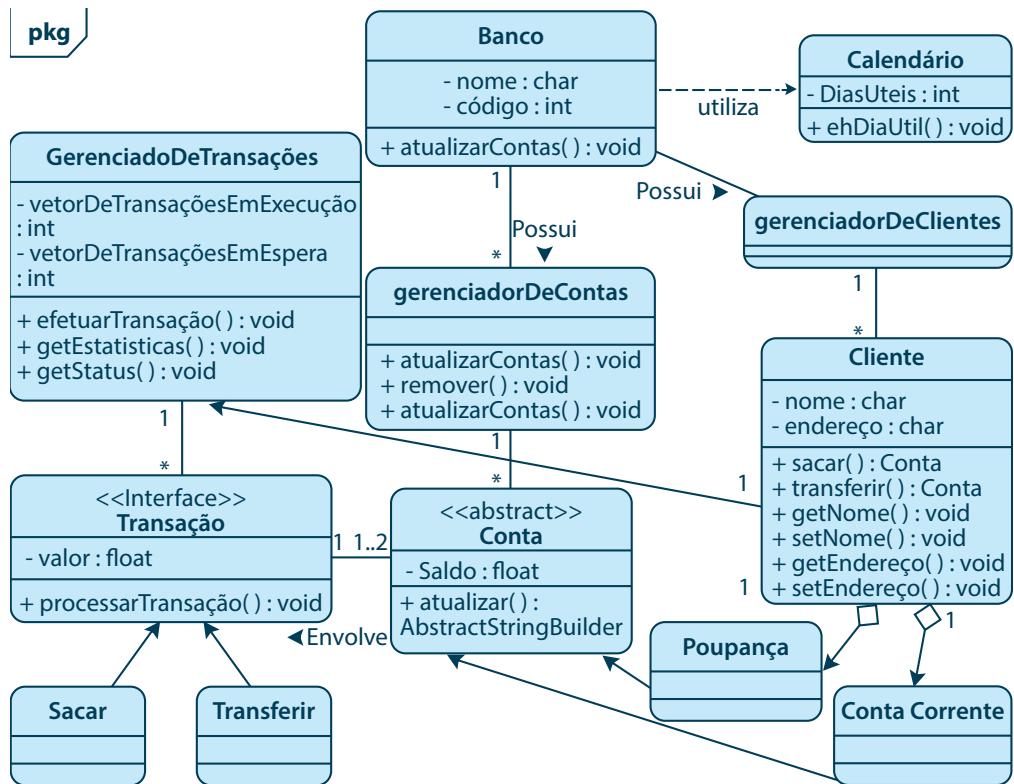


Figura 13: Diagrama de Classes Implementação Caixa Eletrônico

Fonte: adaptada de UFCG (online).

Reforço novamente que os Diagramas apresentados possuem um nível de complexidade baixo, desconsiderando várias funções possíveis para os sistemas sugeridos, tendo um objetivo didático e não comercial. Para o desenvolvimento de sistemas comerciais, todas as funcionalidades devem ser consideradas.

MODELOS COMPORTAMENTAIS

Os modelos estruturais discutidos no tópico anterior representam elementos estáticos, chegou a hora de progredir para o comportamento dinâmico do sistema. Os modelos comportamentais representam o comportamento dinâmico do sistema, o que acontece ou o que deve acontecer quando o sistema responde a um estímulo. Representa a perspectiva – visão – do comportamento dos dados mediante a ação das funções. A modelagem comportamental irá retratar os estados e os eventos que transformam esses estados.

A maioria dos sistemas computacionais é orientada a dados, isto é, são controlados pela entrada de dados com carga relativamente baixa de processamento de eventos. Sommerville (2011, p. 93) nos orienta a considerar dois tipos de estímulos:

- a. Dados: aqueles que chegam e precisam ser processados (transformados).
- b. Eventos: aqueles que geram o processamento do sistema, mas nem sempre estão acompanhados de dados.

A orientação por dados é fácil, estudamos isso desde o início da nossa vida acadêmica, uma função do sistema é ativada quando recebe um dado para ser processado, que então é transformado em novos dados de saída, isto é, são sistemas direcionados por um fluxo de controle padronizado. Agora, e o processamento de eventos? Ao estudarmos os Modelos de Contexto, vimos que eventos estavam relacionados com a interação entre o ambiente externo, ou de outro sistema, e o sistema. Essa definição vale aqui também, mas um evento, em sistemas orientados a eventos, não precisa ser necessariamente de usuário ou de outro sistema, pode ser de hardware, de *socket*, de *timers*, ou ainda de qualquer outro objeto da programação. A programação orientada a eventos é um dos paradigmas de programação, isto é, é uma maneira básica de se programar, na qual a execução do programa é direcionada por eventos, normalmente sensível a sensores. É bastante aplicado no desenvolvimento de software de interface com o usuário. Para exemplificar uma aplicação, apresento a seguir um trecho do artigo de Nagao (2009, online):

Esta forma de se programar é, na minha humilde opinião, a base de todos os sistemas de UI (User Interface) sofisticados. Por exemplo, aquela barrinha do MAC OSX que todos adoram e suas similares: [...] ela sim-

plesmente possui um sensor que avisa “Olha, programa, detectei que o mouse entrou na posição (x,y) da barra”, ao receber esta mensagem o programa faz o zoom e mostra o label dos ícones na posição. Este parece um exemplo bobo, mas acho que ilustra bem a diferença entre a programação em lote e a programação orientada a eventos.

Nesse sentido, a modelagem deve respeitar o comportamento interno do programa, no caso, se orientado a dados ou se orientado a eventos.

MODELAGEM ORIENTADA A DADOS

Para falar de modelagem orientada a objetos, temos que dar um pulinho no passado. A modelagem de dados foi uma das propostas pioneiras como solução para a crise de software que aconteceu em meados da década de setenta. A análise estruturada de sistemas passa a ser, então, um conjunto de técnicas e de ferramentas, tendo como objetivo apoiar a análise e a definição dos sistemas computacionais. Sua base conceitual é a construção de um modelo do sistema utilizando técnicas gráficas: Diagrama de Fluxo de Dados (DFD) e o Dicionário de Dados (DD). A UML® não oferece suporte ao DFD, porque sua proposta é modelar o processamento de dados, e, portanto, foca somente as funções do sistema e não reconhece os objetos do sistema.

Para contornar essa situação, uma vez que os sistemas orientados a dados são a maioria no mundo dos negócios, a UML® desenvolveu o Diagrama de Atividades – muito parecido com o DFD – em que é possível modelar as etapas do processamento (atividades) e os dados (objetos) navegando entre as etapas. A Figura 14, Diagrama de Atividade Caixa Eletrônico – Cartão de Crédito, mostra o fluxo de atividades em um único processo, ele registra como uma atividade depende de outra.

Para associar um objeto ao modelo, o diagrama contém espaços chamados *swimlanes*. Dentro de cada *swimlane* estão registradas as atividades relativas ao objeto daquela região. No Exemplo, as *swimlanes* são o cliente, o caixa eletrônico e o banco. Outro detalhe do diagrama é que as atividades são conectadas por setas (transições ou *threads*), essas setas representam as dependências entre elas. O exemplo representa, de forma simplificada, a função de retirar dinheiro de um caixa eletrônico (utilizando cartão de crédito).

act Activity Diagramo

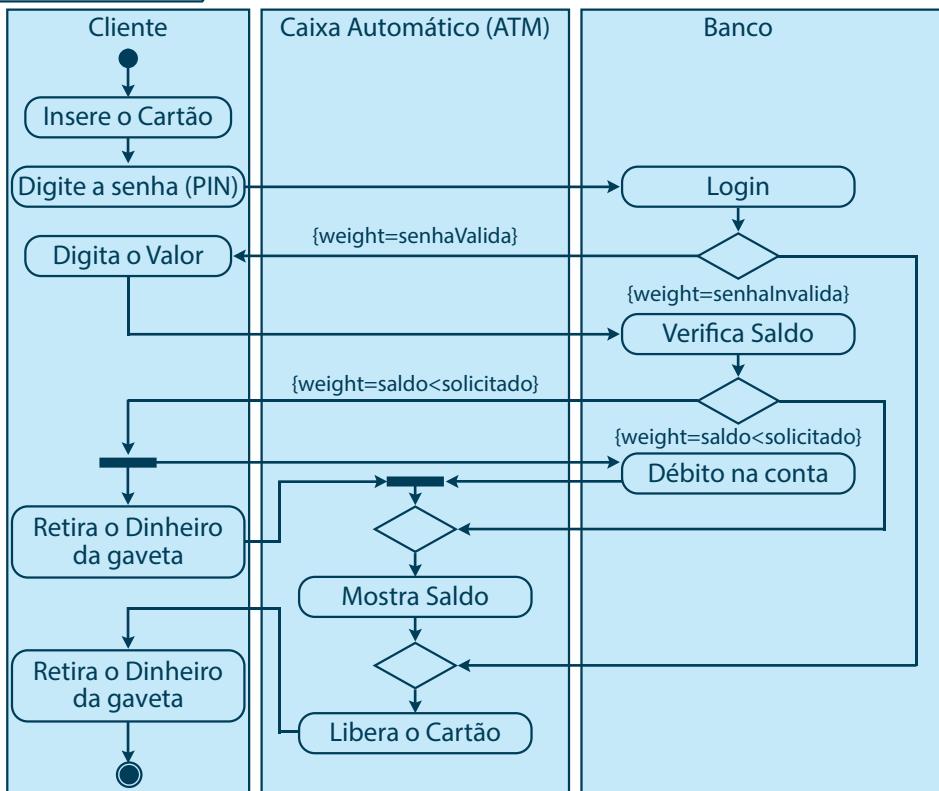


Figura 14: Diagrama de Atividade Caixa Eletrônico – Cartão de Crédito.

Fonte: adaptada de UFCG – Diagrama de Atividades (online).

O diagrama de sequência da UML® também pode ser utilizado para modelar o fluxo de dados de um sistema.

MODELAGEM ORIENTADA A EVENTOS

Se um sistema orientado a dados reage a dados, então um sistema orientado a eventos irá reagir a eventos. Como vimos, eventos podem partir de usuários, de hardware, de sensores, de *timers* etc. O que difere aqui, Sommerville (2011, p. 94) diz, é que a modelagem dirigida a eventos supõe que um software tem um número finito de estados e que os eventos causam uma transição entre esses estados.

O sistema que controla o *airbag* dos nossos carros pode traduzir essa afirmação de uma forma muito simples – e dramática – quando o estado do *airbag* é desarmado (estado 1) e quando o estímulo (batida) é recebido pelo sensor, ele passa para o estado armado. Podemos considerar também a situação de uma válvula de pressão, que está desligada, mas no momento que receber o estímulo, que pode vir do operador ou do próprio software, passa para ligada. São estados finitos.

A UML® usa o Diagrama de Estados para a modelagem de sistemas orientados a eventos (PRESSMAN, 2010, p. 177). Para criar um modelo de comportamento orientado a eventos, o engenheiro precisa executar os seguintes passos:

Avaliar todos os casos de uso para entender plenamente a sequência de interação dentro do sistema.

1. Identificar os eventos que dirigem a sequência de interação e entender como esses eventos se relacionam a classes específicas.
2. Criar uma sequência para cada caso de uso.
3. Construir um diagrama de estado para o sistema.
4. Revisar o modelo comportamental para verificar a precisão e a consistência.

A Figura 15, Diagrama de Estado da Conta Bancária, mostra de maneira simplificada os estados que a conta de um usuário bancário pode assumir, complementando os exemplos de modelagem para caixa eletrônico, apresentados até agora.

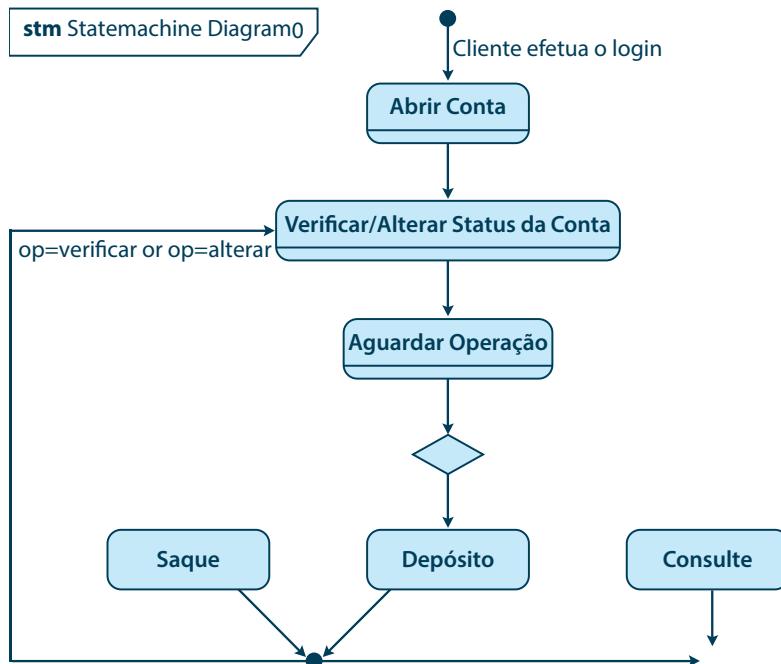


Figura 15: Diagrama de Estado da Conta Bancária
Fonte: adaptada de UFMA – Estudo de caso (online).

Para complementar e empregar outra abordagem de eventos, vamos analisar o exemplo oferecido por Sommerville. Ele mostra o diagrama de estados de um software simplificado para o controle de um forno micro-ondas representado na Figura 16, Diagrama de Estados Forno Micro-ondas.

Forno de micro-ondas reais são muito mais complexos, [...] esse micro-ondas simples tem um interruptor para selecionar a potência total ou meia, um teclado numérico para inserir o tempo de cozimento, um botão iniciar/cancelar e um display alfanumérico. Vamos assumir que a sequência de ações no uso do micro-ondas seja: (1) Selecionar o nível de potência (ou meia potência ou potência total). (2) Introduzir o tempo de cozimento usando um teclado numérico. (3) Pressionar ‘iniciar’, e os alimentos são cozidos no tempo selecionado. [...] o forno não opera com a porta aberta, e uma campainha é acionada no final do cozimento (SOMMERVILLE, 2011, p. 96).

Observe que o sistema parte do estado de espera e responde ao botão de potência total ou meia potência. Após a seleção, é possível mudar de ideia e pressionar outro botão. O tempo é definido, se a porta estiver fechada, o botão iniciar é habilitado. Ao ser pressionado o botão iniciar, a operação começa e o cozimento ocorrerá durante o tempo definido, finalizando assim esse ciclo, e o sistema volta para o estado de espera (SOMMERVILLE, 2011, p. 95).

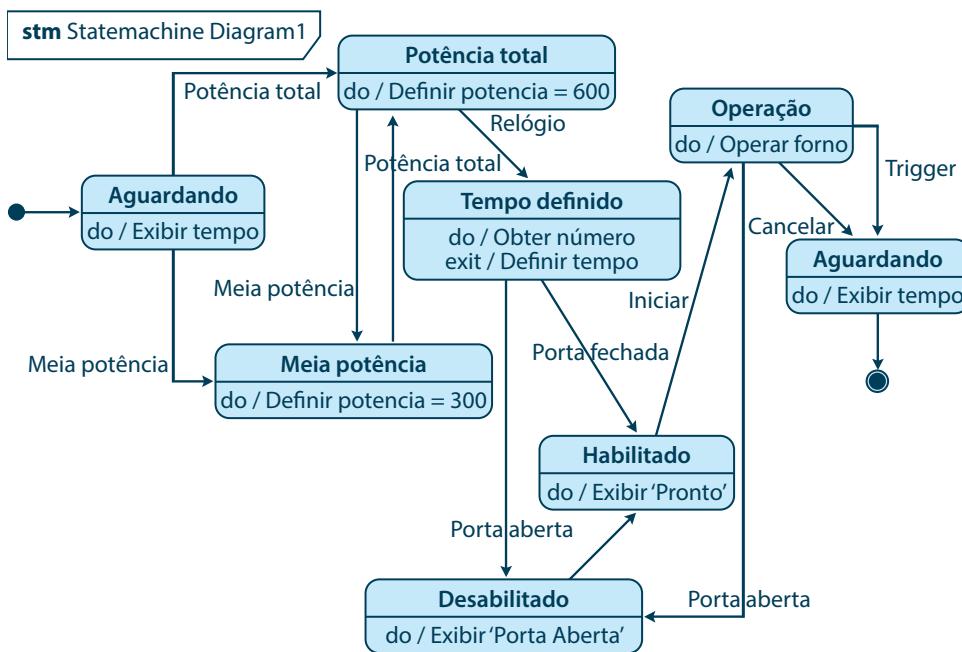


Figura 16: Diagrama de Estados Forno Micro-ondas
Fonte: adaptada de Sommerville (2011, p. 96).

CONSIDERAÇÕES FINAIS

Nesta unidade, analisamos os principais tipos de modelos utilizados nos projetos de desenvolvimento de software e percebemos que diferentes tipos de modelos são necessários, porque existem várias formas de se observar o problema em desenvolvimento. O conjunto de visões tem o objetivo de propiciar formas diferentes de se observar a mesma coisa, sempre considerando a ótica do interessado ou da necessidade naquele momento.

Os modelos de contexto, por exemplo, fornecem visões que interessam aos *stakeholders* e aos engenheiros analistas responsáveis por definir ou identificar as funções essenciais do software, sem se preocupar com as especificidades da estrutura ou implementação. Uma visão mais lógica, do mesmo pedaço do software, seria mais interessante para os programadores, assim podem definir as melhores estruturas para atender às necessidades de forma eficiente, e assim acontece para todas as etapas envolvidas no projeto de desenvolvimento de software.

Estudamos, para isso, as perspectivas fundamentais de modelagem de softwares: perspectiva externa, que mostra como um sistema que está sendo modelado está posicionado, referente a outros sistemas ou processos, auxiliando na definição dos seus limites operacionais; perspectiva de interação, na qual vimos que casos de uso e diagramas de sequência ilustram por simbologias lógicas a interação entre o sistema e os atores externos, no caso de uso, ou entre os objetos do sistema, no diagrama de sequências, também que os modelos estruturais representam a organização e arquitetura do software, e que os diagramas de classe são utilizados para representar a estrutura estática das classes e seus relacionamentos; por fim, a perspectiva comportamental, que traz que os diagramas de estado são utilizados para modelar o comportamento de um software em resposta a eventos promovidos por elementos internos ou externos.

ATIVIDADES



1. Qual é a importância da participação do cliente no processo de desenvolvimento de software?
2. Exemplifique dois softwares que podem ser modelados por meio de várias visões. Faça um breve descritivo dessas visões.
3. Os diagramas de caso de uso da UML® são importantes para a modelagem do contexto de um sistema. Qual seria outra utilidade dos diagramas de caso de uso?
 - a) Testes de sistemas executáveis.
 - b) Definição dos limites do sistema.
 - c) Gerenciamento de versões.
 - d) Modelagem da visão estática da implantação de um sistema.
4. No desenvolvimento de softwares, diversas atividades estão envolvidas, uma delas pode ser definida como: criação de modelos que permitam ao engenheiro, programador e cliente entenderem mais claramente os requisitos do software e o projeto que vai atender a esses requisitos. Essa atividade é conhecida como:
 - a) Modelagem
 - b) Construção
 - c) Comunicação
 - d) Planejamento
 - e) Análise
5. Em relação à modelagem de software, assinale a opção correta.
 - a) Um modelo é uma abstração elaborada para entender um problema antes de implementar uma solução. As abstrações são subconjuntos da realidade, selecionados para determinada finalidade.
 - b) Modelos de contexto são usados para mostrar como os dados ocorrem por uma sequência de etapas de processamento.
 - c) A capacidade de reproduzir fielmente a complexidade do problema é uma das principais motivações para a realização da modelagem.
 - d) Uma forma comum de modelagem de programas procedurais é por meio de fluxogramas de objeto.



METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE: UMA ANÁLISE NO DESENVOLVIMENTO DE SISTEMAS NA WEB

O desenvolvimento de software objetiva a criação de sistemas de software que correspondam às necessidades de clientes e usuários (VASCONCELOS, 2006). Dessa forma, se torna fundamental que se realize uma correta especificação dos requisitos do software para se obter o sucesso do processo. Assim, é cada vez mais utilizado dentro das organizações o analista de requisitos, desempenhando um papel de crucial importância. Com isso, surgem as novas Metodologias de Desenvolvimento de Software (MDS), que dividem o processo de desenvolvimento de software, a fim de organizá-lo e facilitar seu entendimento. Assim, segundo Souza Neto (2004) e Soares (2004), divide-se em duas áreas de atuação:

Desenvolvimento “tradicional”, o qual se fundamenta na análise e no projeto, que conserva tudo em documentação, no entanto, não é vantajoso para mudanças.

Desenvolvimento ágil, baseado em código, inteiramente adaptável a mudanças nos requisitos, mas deficiente na esfera contratual e de documentação.

Maia (2007) define a engenharia de software como um processo para a produção organizada que utiliza uma coleção de técnicas predefinidas e convenções de notação. Para Sommerville (2008), a engenharia de software é uma área da engenharia que se ocupa de todos os aspectos produtivos do software, desde os estágios iniciais de

especificação e entendimento do sistema até sua manutenção depois que ele entrou em operação.

Podemos verificar que a engenharia de software se preocupa com a implantação de métodos, ferramentas e técnicas no processo produtivo de sistemas, buscando a eficácia e eficiência dos recursos, melhorando a qualidade do produto final, bem como reduzindo o prazo e custo de produção.

Essa pesquisa demonstra que o processo utilizado não é responsável direto pelo sucesso do projeto, e sim sua adequação ao ambiente onde foi implantado, atendendo à demanda do cliente da melhor maneira possível, seja com a entrega de um produto apenas ao final do cronograma ou com entregas sucessivas ao longo do projeto. Verifica-se que os clientes não têm interesse no processo de produção da aplicação, descartando ou minimizando a importância dos artefatos e valorizando apenas o software. Esse motivo dá-se principalmente pelo fato de que a documentação e o processo de desenvolvimento não agregam valor ao negócio, e sim a aplicação que objetiva resolver seus problemas negociais.

Essa visão tem causado problemas em relação à aderência do produto às especificações e necessidades do cliente, bem como custo com retrabalho, atrasos e insucessos no projeto. Como forma de ampliar os conhecimentos acerca do assunto, sug-



re-se que se realize um levantamento dos processos utilizados para o desenvolvimento de software nas áreas pública e privada, relacionando com o tamanho dos projetos e sua taxa de sucesso.

Essa pesquisa pode levantar um quadro atual do desenvolvimento no mercado brasileiro e sugerir novas soluções para atender às novas necessidades.

Fonte: Oliveira e Seabra (2015, online).

MATERIAL COMPLEMENTAR



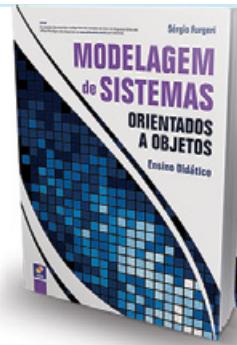
LIVRO

Modelagem de Sistemas: Orientados a Objetos - Ensino Didático

Sergio Furgeri

Editora: Érica

Sinopse: A modelagem de sistemas tem se tornado uma forte aliada na comunicação entre membros de um time de software. Nos últimos anos, os processos de desenvolvimento, a UML e as ferramentas CASE têm ajudado a equipe a ser mais eficiente, participativa e flexível, reduzindo o tempo de produção e, consequentemente, o custo do projeto como um todo. O livro apresenta uma proposta de ensino diferente, com a qual o leitor aprende os elementos mais importantes do processo de modelagem, começando com a captura de requisitos e seu impacto no código de programação, fornecendo exemplos na linguagem Java. Em todo o livro se estabelecem relações entre o mundo conceitual e o prático, abordando os principais diagramas da UML e Java, uma maneira de mapear o mundo gráfico com a programação Java. Analogias também são bastante usadas para facilitar o entendimento. Contempla estudos de caso e exemplos que envolvem os diagramas da UML, o diagrama de entidades e relacionamentos para modelagem do banco de dados, ambos usando o Visual Paradigm for UML 10.0, uma das principais ferramentas CASE do mercado com suporte à modelagem de sistemas. Além disso, dois apêndices demonstram a utilização do Visual Paradigm for UML 10.0 e a IDE NetBeans versão 4 para criação de alguns exemplos em Java presentes no livro. Destina-se aos profissionais da área de informática e estudantes de ensino técnico, tecnológico e universitário. Independente do nível escolar, o aprendizado da UML e da modelagem de sistemas é fundamental. O conteúdo do livro é útil também às empresas que selecionam candidatos para a área e concursos públicos. O estudo da modelagem e da orientação a objetos é aprimorado por diversos exemplos e exercícios, inclusive resolvidos. As respostas dos exercícios, o código-fonte dos exemplos em Java e os diagramas feitos com o Visual Paradigm estão disponíveis em: <www.editoraerica.com.br> para download.





Objetivos de Aprendizagem

- Revisar principais conceitos de orientação a objeto.
- Compreender os objetivos dos principais diagramas da linguagem de modelagem UML®.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Pilares da orientação a objeto
- Linguagem de modelagem unificada UML®
- Diagramas UML®

INTRODUÇÃO

Muito bem! Já entendemos que os modelos são importantes, estudamos também que diferentes aspectos do software são representados por diferentes modelos, a fim de atingir diferentes objetivos. O próximo passo agora é compreendermos a linguagem utilizada nos modelos.

A UML® começou a ser desenhada quando Jim Rumbaugh e Grady Booch resolveram combinar dois métodos populares de modelagem orientada a objeto: Booch e OMT (*Object Modeling Language*). Ivar Jacobson, o criador do método Objectory, uniu-se aos dois para a concepção da primeira versão da linguagem UML (*Unified Modeling Language*). A UML® foi adotada em 1997 pela OMG (*Object Management Group*), que a mantém até hoje. A UML® acrônimo de *Unified Modeling Language*, é uma linguagem para modelagem de dados orientado a objetos, que tem por principal objetivo apoiar e incentivar as boas práticas para os projetos de desenvolvimento de software.

Estudaremos nesta unidade os diagramas propostos pela UML® e suas estruturas, com o objetivo de identificar sua aplicabilidade nas diferentes etapas do processo de desenvolvimento de software. Por ser orientada a objetos, passaremos brevemente por uma revisão dos principais conceitos relacionados a esse paradigma.

Perceberemos, por fim, que, apesar de recomendar diagramas específicos para cada estágio do processo de desenvolvimento de software, organizando-os em duas grandes categorias – Diagramas Estruturais e Diagramas Comportamentais – a especificação UML® não restringe a mistura de diferentes tipos de diagramas, por exemplo, combinar elementos estruturais e comportamentais para mostrar uma máquina de estado aninhado em um caso de uso. Portanto, não existem fronteiras entre a utilização de um ou outro modelo pelo seu objetivo principal possibilitando a utilização de acordo com o necessário para o projeto e as necessidades em questão.

Venha comigo nesta aventura, vamos conhecer os principais diagramas UML®!

PILARES DA ORIENTAÇÃO A OBJETOS

Para entender os conceitos da linguagem UML®, é necessário revisar alguns da orientação a objetos. A linguagem UML é baseada nos princípios da orientação a objetos e trata da representação gráfica parcial de um sistema na sua fase de projeto, implementação ou de sistemas existentes.

Antes de iniciarmos nossa discussão sobre a orientação a objetos e a UML®, convido-o(a) a relembrar dois conceitos importantes necessários para “costurar” a modelagem enquanto análise e a programação. O primeiro deles é o termo paradigma. Provavelmente, você já se deparou com a expressão paradigma de software ou paradigma de linguagem de programação ou ainda paradigmas de programação. Você sabe exatamente o que isso significa? Vamos lá!

O termo paradigma tem origem no grego “*paradeigma*”, que significa modelo ou padrão. Vamos observar o que o dicionário nos apresenta no quadro a seguir:

paradigma

pa.ra.dig.ma

sm (gr *parádeigma*) 1 Modelo, padrão, protótipo. 2 **Ling** Conjunto de unidades suscetíveis de aparecerem num mesmo contexto, sendo, portanto, comutáveis e mutuamente exclusivas. No paradigma, as unidades têm, pelo menos, um traço em comum (a forma, o valor ou ambos) que as relaciona, formando conjuntos abertos ou fechados, segundo a natureza das unidades. No primeiro caso temos os paradigmas lexicais e, no segundo, gramaticais.

Exemplo de paradigma lexical:

A **bela casa/alta/grande/verde**. Exemplo de paradigma gramatical: **and-a/and-as/and-a/and-amos**.

Fonte: Paradigma (online).

A linguagem de programação é uma forma padronizada de repassar comandos para um computador. É um conjunto de regras de sintaxes e de semântica que possibilita a interação entre o homem e a máquina. Cada linguagem de programação é diferente da outra, contendo suas próprias “palavras” de código. Sobrepondo as definições, chegamos à conclusão de que um paradigma de linguagem de programação é um modelo, um conjunto de unidades capaz de estar em um mesmo contexto.

Um paradigma é o que determina o ponto de vista da realidade e como se atua sobre ela, os quais são classificados quanto ao seu conceito de base, podendo ser: Imperativo, funcional, lógico, orientado a objetos e estruturado. Cada qual determina uma forma particular de abordar os problemas e de formular respectivas soluções. Além disso, uma linguagem de programação pode combinar dois ou mais paradigmas para potencializar as análises e soluções. Deste modo, cabe ao programador escolher o paradigma mais adequado para analisar e resolver cada problema (JUNGTHON; GOULART, s/d, p. 1).

Um desses paradigmas é a orientação a objetos. O paradigma de programação orientada a objetos fundamenta-se na utilização de objetos, que colaboram entre si, para a construção do software. A colaboração entre os objetos é feita por meio da troca de mensagens. O paradigma orientado a objeto possibilitou a criação de várias soluções, entre outras, Engholm (2010) cita: programação OO, modelagem OO, banco de dados OO, interfaces OO, metodologias de desenvolvimento de software OO. Por exemplo, esse padrão de programação é seguido – incorporado – por linguagens de programação como *Smalltalk* (a pioneira), *Python*, *Ruby*, *C++*, *Object Pascal*, *Java*, *C#*, *Oberon*, *Ada*, *Eiffel*, .NET e *Simula*.

A orientação a objetos está sustentada nos seguintes pilares: abstração, encapsulamento, herança e polimorfismo. Conceitos que estudaremos a seguir.

SAIBA MAIS



Sobre os paradigmas de desenvolvimento no paradigma estruturado, temos procedimentos (ou funções) que são aplicados globalmente em nossa aplicação. No caso da orientação a objetos, temos métodos que são aplicados aos dados de cada objeto. Essencialmente, os procedimentos e métodos são iguais, sendo diferenciados apenas pelo seu escopo.

A programação estruturada, quando bem feita, possui um desempenho superior ao que vemos na programação orientada a objetos. Isso ocorre pelo fato de ser um paradigma sequencial, em que cada linha de código é executada após a outra, sem muitos desvios, como vemos na POO. Entretanto, a programação orientada a objetos traz outros pontos que acabam sendo mais interessantes no contexto de aplicações modernas. Essa difusão se dá muito pela questão da reutilização de código e pela capacidade de representação do sistema muito mais perto do que veríamos no mundo real. Entenda quais são as vantagens e desvantagens de cada um dos paradigmas de programação. Leia mais no link disponível em: <<http://www.devmedia.com.br/programacao-orientada-a-objetos-versus-programacao-estruturada/32813>>. Acesso em: 21 out. 2015.

ABSTRAÇÃO

Novamente o termo abstração é apresentado. Enquanto para a modelagem abstração significa níveis de detalhamento para cada visão específica, para a orientação a objetos, a abstração representa uma entidade do mundo real. Ela representa as características essenciais de um objeto que o diferenciam de outros (ENGHOLM, 2010). Mas ambas as definições remetem o conceito de abstração ao isolamento de propriedades que se deseja representar fora do seu ambiente complexo. Em orientação a objetos, isolamos um objeto do ambiente real e representamos somente as características interessantes para o problema em questão. A abstração está fortemente relacionada com outro conceito da Orientação a Objeto conhecido como Acoplamento. Quanto mais abstrações utilizadas estratégicamente, mais baixo é o nível de Acoplamento¹.

A abstração de um objeto dentro da orientação a objetos deve possuir três características: Identidade, Propriedades e Métodos.

¹ Acoplamento em orientação a objeto significa o grau de dependência entre dois objetos. Por exemplo, Uma classe com alto nível de acoplamento é uma classe dependente de várias outras para ser executada.

- Identidade - a identidade de uma abstração é o seu nome, sua identificação dentro do código e do sistema como um todo. Aqui cabe passear pelo conceito da padronização. Um código bem escrito é fácil de entender, portanto, fácil de manter. Existem regras, padrões e convenções disponíveis para algumas linguagens, por exemplo, a linguagem de programação Java, por convenção, diz que toda classe deve começar com uma letra maiúscula e, de preferência, não pode conter letras não ASCII. A linguagem de programação Delphi preconiza que a classe deve ser iniciada pela letra T maiúscula acompanhada pelo nome seguindo o padrão infix caps. Por exemplo, Pessoa. Essa pode ser a identidade de um objeto que irá representar pessoas do mundo real.
- Propriedade - as propriedades são as características do objeto em abstração (também podem ser encontradas na literatura, definidas como atributos). Tendo em mente que um objeto representa algo do mundo real e, se no mundo real esse objeto possui características que o definem, quando o abstraímos, trazemos para a orientação a objeto as características que nos interessam. Por exemplo, o objeto Pessoa, que definimos acima. Para o programa em desenvolvimento, foi analisada a necessidade de se registrar as seguintes características/propriedades: nome, RG, CPF, data de nascimento, endereço e telefone.
- Métodos - os métodos representam o que esse objeto pode fazer, ou seja, as ações que ele deverá executar, seu comportamento. Por exemplo, quais são as ações possíveis de se realizar para o objeto Pessoa? Inserir(), Alterar(), Excluir(), Listar().

A modelagem para a abstração do objeto Pessoa é representada pela Figura 18 – Classe Pessoa.

É importante considerar nesse momento os termos Objeto e Classe. Ficou fácil perceber que o Objeto representa uma abstração do mundo real envolvido no problema a ser resolvido pelo software. Um objeto é essencialmente uma coleção de atributos e métodos válidos para manipular outros objetos (ENGHOLM, 2010, p. 119). Uma Classe representa um conjunto de objetos que possuem as mesmas propriedades. Considere o objeto Pessoa que foi utilizado como exemplo para a abstração. Esse objeto pode ser dividido (especializado) em Pessoas diferentes, como Clientes, Fornecedores, Alunos, Professores, Médico, Paciente, Enfermeiro etc., mas suas propriedades e métodos permanecem iguais. Portanto,

objetos são organizados em classes, e objetos da mesma classe possuem as mesmas propriedades e os mesmos métodos. Aproveitando a Figura 17, para ilustrar, observe que a Classe é Pessoa, e os objetos poderiam ser: Maria -> Atributos: 21-01-1998, Rua das Flores, Maria dos Anjos, 5.749.846-89, 000.145.568-89, 44 3222 2015. Métodos: Alterar(), Excluir(), Inserir(), Listar(). João 10-10-1960, Avenida Roxa, Pedro do Céu, 1.111.222-11, 000.256.145-10, 44 3221 2112. Métodos Alterar(), Excluir(), Inserir(), Listar(), pois, apesar de possuírem valores diferentes, compartilham os mesmos atributos e métodos.

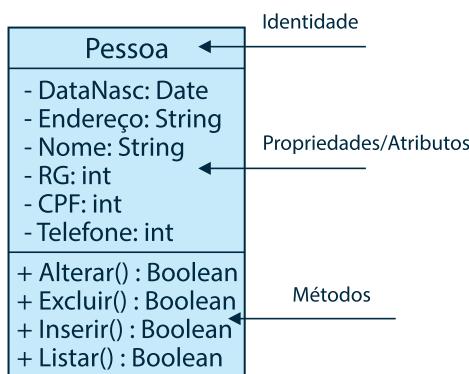


Figura 17: Classe Pessoa

Fonte: a autora.

ENCAPSULAMENTO

“As pessoas que usam os objetos não precisam se preocupar em saber como eles são constituídos internamente acelerando o tempo de desenvolvimento” (CORREIA; TAFNER, 2006, p. 13). Esse é o grande propósito do encapsulamento. Na orientação a objetos, os objetos possuem tanto dados quanto métodos. Como vimos na definição de métodos, eles são pedaços de códigos, são funções que dizem o que aquele objeto pode fazer. O encapsulamento oculta essa codificação e o trata como um “componente”, que permite que você use suas propriedades e métodos, mostrando como as partes do objeto se relacionam com o exterior, e você não tem que se preocupar com dados e códigos que implementam o comportamento dos objetos da classe. Muitos autores utilizam o sinônimo “caixa preta”

para o encapsulamento, onde se vê o exterior, mas não precisa se preocupar com o que acontece lá dentro.

O encapsulamento, na maioria das linguagens de programação orientada a objetos, é praticado utilizando métodos especiais conhecidos por *getters* e *setters*, onde o método *setter* é responsável por alterar ou inserir valores nos dados do objeto, e o método *getter* é utilizado para recuperar valores encapsulados no objeto e pode ser implementado em três níveis de acesso: Privado, Público e Protegido.

- Público (public): os atributos e métodos da classe são visíveis dentro da classe e para classes externas.
- Privado (private): os atributos e métodos da classe não são visíveis a nenhuma classe externa. São somente visíveis dentro da classe.
- Protegido (protect): os atributos e métodos da classe não são visíveis a nenhuma classe externa. São visíveis dentro da classe e para as classes herdeiras.

Observe na Figura 18, Exemplo de Encapsulamento, a modelagem de um objeto em que todos os atributos são Privados e os métodos são públicos. Essa configuração impede, por exemplo, uma quantidade de meses para a aplicação financeira diferente da estabelecida pela regra de negócio, codificada (ELGHOLM, 2010, p. 125).

| AplicaçãoFinanceira |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> - idAplicaçãoFinanceira : int - descrição : String - taxasDeJuros : double - quantidadeDeMeses : int - valorAplicação : double </pre> |
| <pre> + setIdOperacaoFinanceira() : void + getIdOperacaoFinanceira() : void + setDescrição(descrição : String) : String + getDescrição() : String + setTaxaDeJuros(taxaDeJuros : double) : Double + getTaxaDeJuros() : double + setQuantidadeDeMeses(quantidadeDeMeses : int) : int + getQuantidadeDeMeses() : int + setValorAplicacao() : double + getValorFuturoAplicaco() : double </pre> |

Figura 18: Exemplo de Encapsulamento

Fonte: adaptado de Elgholm (2010, p. 125).



REFLITA

“Trocava toda minha tecnologia por uma tarde com Sócrates.”

Fonte: Steve Jobs.

HERANÇA

O terceiro pilar da orientação a objetos é a possibilidade de compartilhamento de atributos, métodos entre as classes que compõem o sistema. Esse mecanismo, denominado Herança, permite criar novas classes a partir de classes já existentes (ELGHOLM, 2010, p. 128).

Na modelagem, essa característica é representada como **especialização** e funciona da seguinte maneira: considere a modelagem de um sistema financeiro, que deverá administrar tanto funcionários quanto os clientes de um banco. Durante o processo de análise, o engenheiro de software identifica que Clientes e Funcionários possuem várias características similares e também que várias ações deverão ser aplicadas a ambos os objetos, sobrando somente características exclusivas dos objetos que os diferenciam. A Tabela 3, Propriedades e Métodos dos Objetos Clientes e Funcionários, destaca essas características em comum.

PROPRIEDADES DOS OBJETOS

| Funcionário | Cliente |
|-----------------------|------------------------------|
| <i>Nome</i> | <i>Nome</i> |
| <i>Endereço</i> | <i>Endereço</i> |
| <i>dataNascimento</i> | <i>dataNascimento</i> |
| <i>CPF</i> | <i>CPF</i> |
| <i>RG</i> | <i>RG</i> |
| <i>telefone</i> | <i>telefone</i> |
| <i>cargo</i> | <i>dataCadastroCliente</i> |
| <i>salário</i> | <i>contaCorrente</i> |
| <i>dataAdmissao</i> | <i>aplicacoesFinanceiras</i> |
| | <i>emprestimosBancarios</i> |

| MÉTODOS DOS OBJETOS | |
|----------------------------------|---------------------------------------------|
| Funcionário | Cliente |
| <code>setNome()</code> | <code>setNome()</code> |
| <code>getNome()</code> | <code>getNome()</code> |
| <code>setEndereco()</code> | <code>setEndereco()</code> |
| <code>getEndereço()</code> | <code>getEndereço()</code> |
| <code>setDataNascimento()</code> | <code>setDataNascimento()</code> |
| <code>getDataNascimento()</code> | <code>getDataNascimento()</code> |
| <code>setCPF()</code> | <code>setCPF()</code> |
| <code>getCPF()</code> | <code>getCPF()</code> |
| <code>setRG()</code> | <code>setRG()</code> |
| <code>getRG()</code> | <code>getRG()</code> |
| <code>setTelefone()</code> | <code>setTelefone()</code> |
| <code>getTelefone()</code> | <code>getTelefone()</code> |
| <code>setCargo()</code> | <code>insereEmprestimo()</code> |
| <code>getCargo()</code> | <code>insereContaCorrente()</code> |
| <code>setSalario()</code> | <code>insereAplicacaoFinanceira()</code> |
| <code>getSalario()</code> | <code>retornaEmprestimosBancarios()</code> |
| <code>setDataAdmissao()</code> | <code>retornaContasCorrentes()</code> |
| <code>getDataAdmissao()</code> | <code>getContaCorrente()</code> |
| | <code>getEmprestimosBancarios()</code> |
| | <code>retornaAplicacoesFinanceiras()</code> |
| | <code>getAplicacoesFinanceiras()</code> |
| | <code>setDataCadastroCliente()</code> |
| | <code>getDataCadastroCliente()</code> |

Tabela 3: Propriedades e Métodos dos Objetos Clientes e Funcionários

Fonte: a autora.

Para aperfeiçoar o código e não ser necessário repetir os atributos e os métodos comuns para as duas classes, entra o mecanismo da Herança. Cria-se uma entidade em que se incluirá o que é comum aos dois objetos e, a partir dela, derivar especializações.

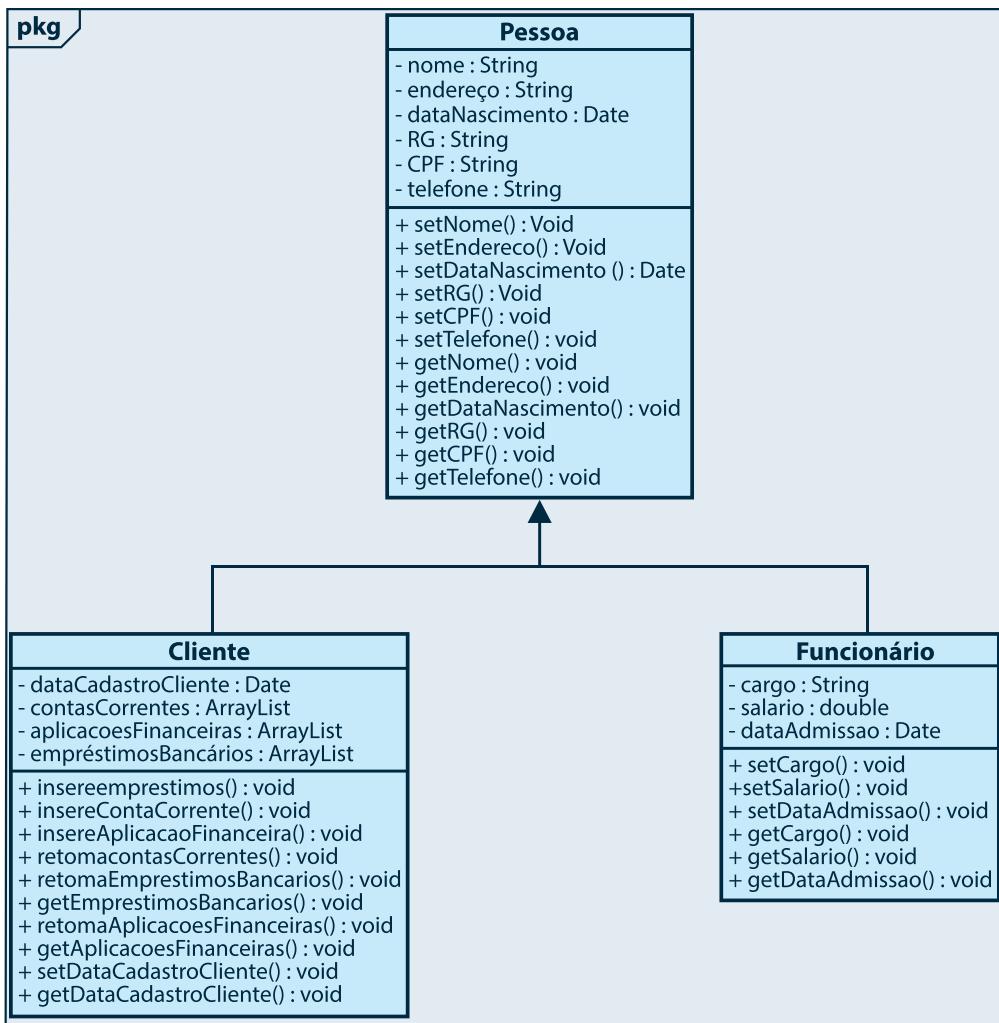


Figura 19: Especialização da Classe Pessoa

Fonte: a autora.

Na engenharia de software, a classe de onde as outras são especializadas é chamada **superclasse**, e as classes especialistas são chamadas de **subclasses**. Ao construir uma classe a partir de outra, a nova classe pode herdar os estados e os

comportamentos da superclasse, e pode usá-las além das próprias (ENGHOLM, 2010, p. 128). Na Figura 19, a classe Pessoa é a superclasse (generalização) e as classes Funcionário e Cliente são subclasses (especializações de Pessoas).



REFLITA

"Meus filhos terão computadores, sim, mas antes terão livros. Sem livros, sem leitura, os nossos filhos serão incapazes de escrever - inclusive a sua própria história."

Fonte: Bill Gates.

POLIMORFISMO

Para explicar o conceito de polimorfismo, apelo novamente ao meu sistema de vincular o significado da palavra à sua função no contexto da engenharia de software. Polimorfismo é a qualidade ou o estado de ser capaz de assumir diferentes formas. Na orientação a objeto, ele está extremamente vinculado ao mecanismo de herança, pois ele possibilita que métodos herdados de uma superclasse sejam reescritos, isto é, uma mesma mensagem pode ser interpretada de maneiras diferentes. Em outras palavras, o polimorfismo consiste na alteração do funcionamento interno de um método herdado de um objeto pai (GASPAROTTO, 2014).

Para exemplificar, vamos utilizar o problema proposto pelo Professor José Carlos Macoratti, no seu artigo Programação Orientada a Objetos em 10 lições práticas – Parte 07 (2014, online).

Problema: Você deseja controlar sua conta bancária pessoal registrando os saques, depósitos e controlando o saldo da conta.

Análise: Classe Conta responsável por definir os comportamentos e atributos de qualquer Conta. Propriedades: tipoConta. Métodos básicos: Sacar(); Depositar(). Superclasse: Conta; Subclasses: ContaPoupança; ContaInvestimento. Como mostra a Figura 20, Classes para Polimorfismo.

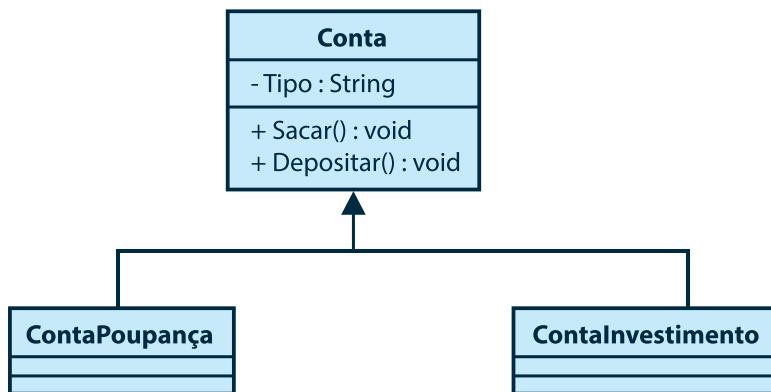


Figura 20: Classes para Polimorfismo
Fonte: adaptada de Macoratti (2014).

Até aqui, sem novidades, modelamos uma herança. Não temos como modelar o polimorfismo, por isso vou apresentar o código do Professor Macoratti na linguagem VB .NET. O polimorfismo será promovido usando a herança e reescrevendo os métodos. Observe os trechos de código nas Figuras 21, Código para Polimorfismo em VB.NET, e Figura 22, Código para Polimorfismo em VB.NET 2.

```

1  Public Class ContaPoupança
2      Inherits Conta
3      Public Sub New(tipoConta As String)
4          MyBase.New(tipoConta)
5      End Sub
6      Public Overrides Sub Sacar()
7          Console.WriteLine("Sacando da conta de poupança")
8      End Sub
9
10     Public Overrides Sub Depositar()
11         Console.WriteLine("Depositando na conta de poupança")
12     End Sub
13 End Class
  
```

Figura 21: Código para Polimorfismo em VB.NET.
Fonte: adaptada de Macoretti (2014).

```

1 Module Module1
2 Sub Main()
3     Dim conta As Conta() = New Conta(1) {}
4     conta(0) = New ContaPoupanca("Poupança do Macoratti")
5     conta(1) = New ContaInvestimento("Conta de Investimento do Macoratti")
6     MovimentarConta(conta(0))
7     MovimentarConta(conta(1))
8     Console.ReadKey()
9 End Sub
10 Public Sub MovimentarConta(_conta As Conta)
11     Console.WriteLine(_conta.Tipo)
12     _conta.Sacar()
13     _conta.Depositar()
14 End Sub
15 End Module

```

file:///C:/vbn/OOP_Polimorfismo1/OOP_Polimorfismo1...
 Poupança do Macoratti
 Sacando da conta de poupança.
 Depositando na conta de poupança.
 Conta de Investimento do Macoratti
 Sacando da conta de investimento.
 Depositando na conta de investimento.

Figura 22: Código para Polimorfismo em VB.NET 2

Fonte: Macoratti (2014).

O Professor Macorati (2014, online) explica,

no método MovimentarConta estamos usando o método Sacar() e Depositar() para cada tipo de conta que foi instanciada e cada objeto sabe realizar o comportamento em resposta à mesma chamada do método. Aqui estamos usando o conceito de polimorfismo, pois o método Sacar() e o método Depositar() são executados e a decisão de qual comportamento será usado ocorre em tempo de execução. Para usar o polimorfismo os objetos precisam executar as mesmas ações (métodos) mesmo que possuam comportamentos diferentes.

Concluindo essa rápida revisão sobre o paradigma da programação orientada a objetos, generalizando, temos a orientação a objetos como sendo um conjunto de encapsulamento com abstração e polimorfismo (ENGHOLM, 2010).

Muitas linguagens de programação modernas suportam o conceito de abstração de dados, porém, o uso de abstração juntamente com polimorfismo e herança, como suportado em orientação a objetos, é um mecanismo muito mais poderoso.



SAIBA MAIS

Coesão e Acoplamento em Sistemas Orientados a Objetos

Muitos desenvolvedores de software percorreram caminhos que passaram pelo desenvolvimento procedural antes de chegarem à orientação a objetos.

Diferentemente do paradigma orientado a objetos, no qual a subdivisão de sistemas é baseada no mapeamento de objetos do domínio do problema para o domínio da solução, o desenvolvimento procedural não possui uma semântica forte que oriente a subdivisão de sistemas. Nesse contexto, muitos conceitos e métricas foram definidos para avaliar e auxiliar a subdivisão de sistemas. Dois desses conceitos são especialmente importantes por terem grande influência na qualidade dos sistemas desenvolvidos: coesão e acoplamento.

Leia mais em: Artigo Java Magazine 77 - Coesão e Acoplamento em Sistemas Orientados a Objetos, disponível em:<<http://www.devmedia.com.br/artigo-javamagazine-77-coesao-e-acoplamento-em-sistemas-orientados-a-objetos/16167#ixzz3lqLLQdz9>>. Acesso em: 21 out. 2015.

Fonte: Luque (online).

LINGUAGEM DE MODELAGEM UNIFICADA - UML®

Antes de iniciarmos nossa discussão sobre a UML®, gostaria de deixar claro que não é objetivo desse conteúdo a descrição completa de cada conceito. Pretendo abordá-lo como complemento para nosso estudo sobre a modelagem de software.

É comum você encontrar na literatura a UML® definida como um método, ela até pode ser encarada como parte dos procedimentos e técnicas necessários para o desenvolvimento de um software, mas, por concepção, a UML® é uma linguagem.

Ela disponibiliza um meio sistemático, um conjunto de símbolos com o intuito de comunicar ideias. O objetivo da UML® é fornecer aos profissionais de software ferramentas para análise, projeto e implementação de sistemas. As versões iniciais da UML® surgiram de três principais métodos orientados a objeto

(Booch, OMT e OOSE) e incorporaram vários artefatos baseados nas melhores práticas de modelagem de *designer*, programação orientada a objetos e modelagem de arquiteturas.

A UML® nos possibilita cinco visões. Observe a Figura 23, Visões UML®.

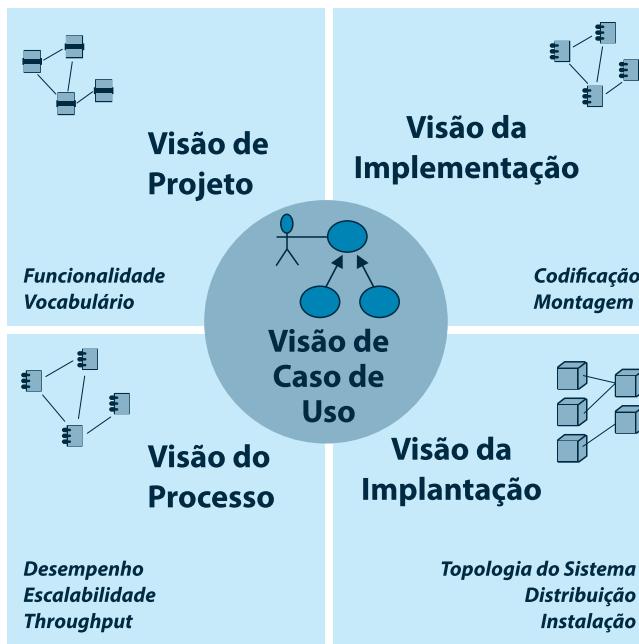


Figura 23: Visões UML®

Fonte: UML (online).

A visão do caso de uso envolve os casos de uso que descrevem o comportamento do sistema pelo ponto de vista de seus usuários. Não representa a organização interna do software, mas determina, de forma preliminar, sua arquitetura. Os aspectos estáticos dessa visão na UML® são representados pelos diagramas de caso de uso; os aspectos dinâmicos são representados pelos diagramas de interação, diagrama de gráfico de estados e diagrama de atividades.

A visão do projeto representa as classes e suas colaborações. Nessa visão, as funcionalidades que o sistema irá desempenhar são modeladas. A UML® utiliza, para captar os aspectos estáticos dessa visão, os diagramas de classes e de objetos. E, para os aspectos dinâmicos, os diagramas de interações, os diagramas de estados e os diagramas de atividades.

A visão do processo envolve os elementos relacionados ao desempenho do sistema. Mecanismos de concorrência, processos etc. são representados na UML® por meio dos mesmos diagramas utilizados para a visão do projeto, mas com o foco nas classes ativas e na representação de seus processos e concorrência.

A visão da implementação modela os componentes e arquivos que, reunidos, produzem o sistema executável. A visão da implantação representa como os componentes e arquivos que compõem o sistema serão organizados e distribuídos para a sua instalação. Os diagramas UML® que modelam os aspectos estáticos dessa visão são os diagramas de implantação, e os aspectos dinâmicos podem ser representados nos diagramas de interações, diagramas de atividades e gráficos de estados.

Essas visões são representadas por diagramas, que estão organizados em duas grandes categorias: diagramas que modelam a estrutura do sistema (estáticos) e diagramas que modelam o comportamento do sistema (dinâmicos).

- Diagramas estáticos: diagrama de classes, diagrama de objetos, diagrama de componentes e diagrama de distribuição.
- Diagramas dinâmicos: diagrama de casos de uso, diagramas de interação – sequência e colaboração – diagrama de atividades e diagrama de estados.

Cada visão é descrita em um número de diagramas que contém informação enfatizando um aspecto particular do sistema. Analisando-se o sistema por meio de visões diferentes é possível se concentrar em um aspecto de cada vez.

ESPECIFICAÇÃO UML®

A especificação da UML® é composta por duas partes:

- Semântica - especifica a sintaxe abstrata dos conceitos de modelagem estática e dinâmica de objetos.
- Notação - especifica a notação gráfica para a representação visual da semântica, composta pelos itens, relações e diagramas.

Os itens podem ser estruturais, representando a parte mais estática do modelo: classes, interface, colaborações, caso de uso, classes ativas, componentes, nós. Podem ser comportamentais, que representam um comportamento no tempo e no espaço: interação, máquina de estado. De agrupamento: pacote, que representam um grupo de outros itens, que podem ser estruturais ou comportamentais; por último, itens de notação são comentários utilizados para descrever, esclarecer sobre quaisquer elementos do modelo.

São quatro os tipos de relacionamentos contidos na UML®: Dependência, Associação, Generalização e Realização.

Os Diagramas, como estudamos acima, estão organizados em duas grandes categorias: diagramas que modelam a estrutura do sistema (estáticos) e diagramas que modelam o comportamento do sistema (dinâmicos).

ITENS ESTRUTURAIS

Classe

Estudamos no tópico Pilares da Orientação a Objetos que uma Classe pode ser vista como um *template* que representa um conjunto de objetos que compartilham os mesmos atributos, métodos e relacionamentos (ENGHOLM, 2010, p. 121). A representação de uma classe é apresentada pela Figura 17 – Classe Pessoa.

Na UML®, a representação de um atributo tem que conter no mínimo uma identificação e pode ser complementada por outras propriedades, como, por exemplo, o seu tipo ou um valor inicial.

A notação para a visibilidade dos elementos que compõem as classes (atributos, e métodos) na linguagem UML® é:

- + para atributos públicos
- - para atributos privados
- # para atributos protegidos

Interface

Em modelagem UML®, interfaces são as estruturas que definem o conjunto de operações que outras estruturas do modelo, como classes ou componentes, devem implementar. Cada interface especifica um conjunto de operações bem definido que possuem visibilidade pública. É possível especificar os seguintes tipos de interfaces (IBM Knowledge Center):

- Interfaces fornecidas: representa o que a classe faz.
- Interfaces requeridas: como uma classe faz as tarefas.

Na prática, lemos a interface da seguinte maneira: “quem desejar ser autenticável precisa saber autenticar dado um inteiro e retornando um booleano”. Ela é um contrato no qual quem assina se responsabiliza por implementar esses métodos (INTERFACES, online). A Figura 24, Interface UML®, representa o símbolo para a modelagem da Interface pela UML®.



Figura 24: Interface UML

Fonte: a autora.

Colaborações

As colaborações definem as interações. Representam os elementos que funcionam em conjunto a fim de garantir um comportamento cooperativo maior que a soma de todos os elementos, isto é, a implementação do padrão que forma o sistema.

Caso de uso

Um caso de uso representa um conjunto de atividades que irão produzir um resultado para o ator relacionado. Eles modelam as funções que o sistema deve realizar a partir das interações entre os usuários e o sistema. O caso de uso se preocupa somente em dizer “o que” o sistema deve fazer e não “como” ele deve fazer. A Figura 25, Caso de uso UML®, representa o símbolo para a modelagem do caso de uso pela UML®.



Figura 25: Caso de uso UML®

Fonte: a autora.

Classes ativas

Uma classe ativa se difere da Classe, pois seus objetos representam elementos que são concorrentes com outros elementos. A Figura 26, Classe Ativa UML®, representa o símbolo para a modelagem da classe ativa pela UML®.

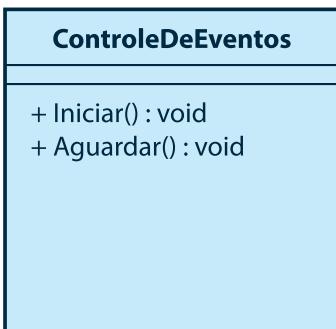


Figura 26: Classe Ativa UML®

Fonte: a autora.

Componentes

Um componente é uma parte física de um sistema, sua função é garantir a realização de um conjunto de interfaces; representa um empacotamento físico de elementos relacionados logicamente (normalmente classes). Por exemplo, executáveis, bibliotecas, tabelas, banco de dados etc. A Figura 27, Componente UML®, representa o símbolo para a modelagem do componente pela UML®.

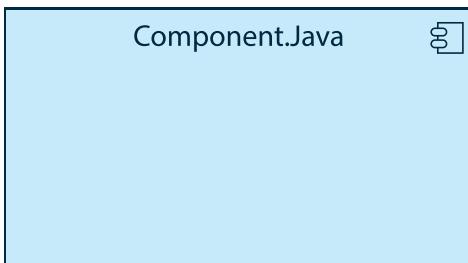


Figura 27: Componente UML®

Fonte: a autora.

Nó

Um nó representa um elemento computacional físico, com alguma capacidade de processamento e memória. Por exemplo, um computador, uma rede etc. A Figura 28, Nó UML®, representa o símbolo para a modelagem do nó pela UML®.



Figura 28: Nó UML®

Fonte: a autora.

ITENS COMPORTAMENTAIS

Interação

Interação representa as mensagens trocadas entre os objetos. É representada por uma seta.

Máquina de estado

Representa a sequência de estados que os objetos assumem no decorrer do tempo em resposta aos eventos aos quais são submetidos. A Figura 29, Máquina de

Estado UML®, representa o símbolo para a modelagem da máquina de estado pela UML®.

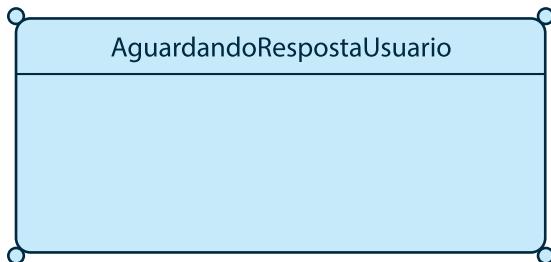


Figura 29: Máquina de Estado UML®

Fonte: a autora.

ITENS DE AGRUPAMENTOS

Chamados Pacotes, os itens de agrupamento têm o propósito de organizar os elementos em grupos. Um Pacote é puramente conceitual, não representa um comportamento de execução. A Figura 30, Pacote UML®, representa o símbolo para a modelagem do pacote pela UML®.

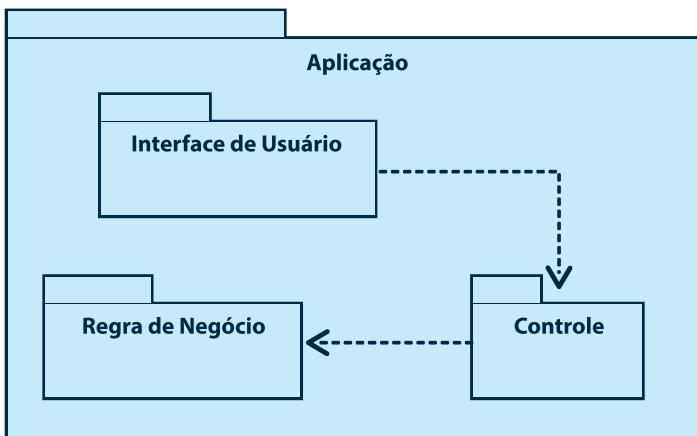


Figura 30: Pacote UML®

Fonte: adaptada de UML (online).

ITENS ANOTACIONAIS

São os símbolos utilizados para fazer anotações, descrever, esclarecer sobre qualquer elemento do modelo. A UML® os define como Nota. A Figura 31, Nota UML®, representa o símbolo para a modelagem da nota pela UML®.

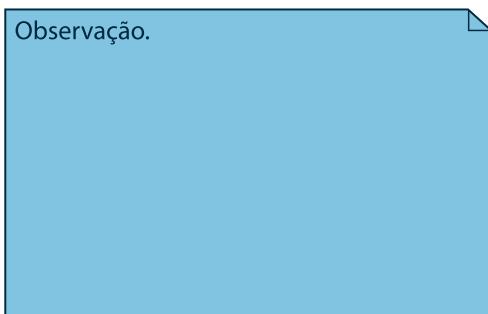


Figura 31: Nota UML®

Fonte: a autora.

RELACIONAMENTOS

Os relacionamentos conectam os objetos entre si, criando relações lógicas entre eles.

Dependência

Na UML®, um relacionamento de dependência é um relacionamento no qual um elemento usa ou depende de outro elemento. Indica que qualquer alteração no objeto, que chamaremos aqui de “pai”, causa uma alteração no objeto dependente. Um relacionamento de dependência também pode ser utilizado para representar precedência, em que um elemento deve preceder outro (IBM Knowledge Center).

Geralmente, os relacionamentos de dependência não são identificados. A Figura 32, Dependência UML®, ilustra uma dependência, representada por uma linha tracejada com uma seta aberta que aponta do objeto “pai” para o objeto dependente. Resumindo de uma forma bastante objetiva, relacionamento de

dependência ocorre quando a existência de uma classe depende da existência de outra. Por exemplo, em um sistema comercial que controla Pedidos de Clientes, só existirá um Item de Pedido se existir um Pedido.



Figura 32: Dependência UML®

Fonte: a autora.

Associação

Uma associação é um relacionamento entre dois objetos como classes ou casos de uso que tem por objetivo representar os motivos e as regras que conduzem os objetos ao relacionamento. As associações registram as propriedades dos objetos. Por exemplo, você pode utilizar um recurso de navegabilidade de uma associação para mostrar como um objeto de uma classe obtém acesso a um objeto de outra classe ou, em uma associação reflexiva, para um objeto da mesma classe. Cada extremidade de um relacionamento de associação possui propriedades que especificam sua função, multiplicidade, visibilidade, navegabilidade e restrições (IBM Knowledge Center).

Quanto à classificação, uma associação pode ser:

- **Normal/Simples:** representa somente a colaboração entre os objetos de elementos diferentes; a navegabilidade pode ser unidirecional ou bidirecional e é representada por um traço simples.
- **Qualificada:** um qualificador de associação é um atributo do elemento-alvo, que identifica uma instância entre as demais.
- **Recursiva:** acontece quando um objeto do elemento se conecta a outro contínuo nele próprio.
- **Agregação:** indica que um elemento é parte ou está contido em outro elemento, mas são independentes entre si (parte existe sem todo).
- **Composição:** onde um elemento está contido em outro, e a execução de um depende do outro (todo controla a execução da parte).

Quanto à multiplicidade, nos extremos do traço que representa a associação, é definido o número de instâncias permitidas para aqueles objetos. Uma multiplicidade do final da associação pode ter um dos seguintes valores: (1) um, (0) zero, (*) muitos.

O nome de uma associação deve descrever, de forma nítida, a natureza do relacionamento e deve ser um verbo ou frase simples. Uma associação é representada como uma linha sólida entre dois elementos. A boa prática recomenda sempre incluir o nome da associação ou um papel que a identifique; a definição do seu papel é muito útil na geração do código quando utilizadas ferramentas CASE. A Figura 33, Associação UML®, ilustra uma associação simples entre classes. A Figura 34 ilustra os relacionamentos do tipo agregação e composição.

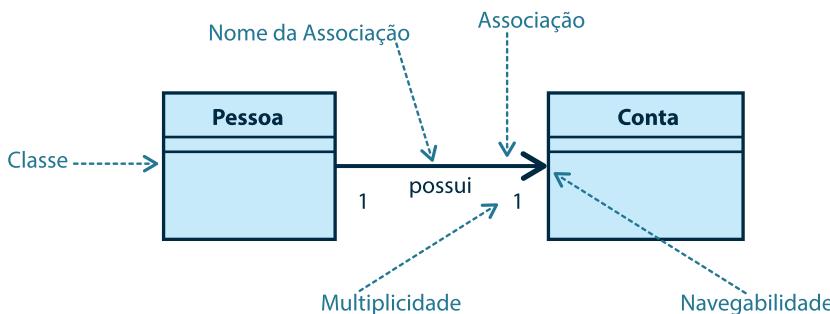


Figura 33: Associação UML®

Fonte: Barcelar (online).

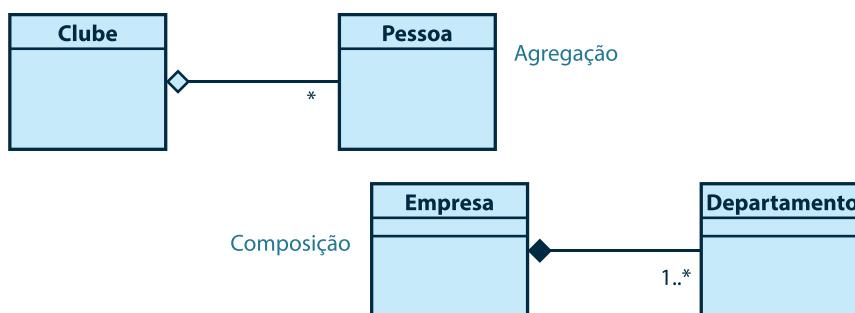


Figura 34: Agregação e Composição

Fonte: Barcelar (online).

Generalização

O relacionamento Generalização está fortemente relacionado com a característica Herança da orientação a objetos. Esse relacionamento representa a especialização de um objeto a partir de um outro com atributos mais gerais. Por exemplo, a especialização do objeto Pessoa para o objeto Funcionário, como vimos no exemplo representado pela Figura 19, Especialização da Classe Pessoa, quando estudamos orientação a objeto no tópico anterior.

Realização

Na modelagem UML®, um relacionamento de realização é aquele entre dois elementos, no qual um deles realiza o comportamento que o outro elemento especificou. É possível utilizar os relacionamentos de realização nos diagramas de classe e diagramas de componentes. Esse relacionamento é representado por uma seta tracejada com a cabeça de seta fechada e vazia que aponta do realizador para o especificador. A Figura 35, Realização UML®, ilustra uma realização.



Figura 35: Realização UML

Fonte: a autora.

DIAGRAMAS UML®

DIAGRAMAS ESTRUTURAIS

Diagramas de estrutura representam a estrutura estática do sistema e de partes do sistema em diferentes níveis de abstração e de implementação e também mostram como essas partes estão relacionadas umas com as outras. Os elementos em um diagrama de estrutura representam os conceitos significativos de um sistema e podem incluir abstrato, mundo real ou conceitos de implementação. Diagramas de estrutura utilizam conceitos de tempo, não mostram os detalhes de comportamento dinâmico.

Diagrama de Classe

O objetivo do diagrama de classe é representar a estrutura de um sistema em fase de projeto, de um subsistema ou ainda de um componente, como, por exemplo, as classes e interfaces relacionadas; ele inclui na denotação as características, limitações e relacionamentos (associações, generalizações, dependências etc.). Os Elementos que compõem o diagrama de classe são os seguintes: classe, relacionamentos e interfaces.

A estrutura de uma classe divide-se em três compartimentos, no primeiro fica registrado o nome da classe, no segundo, as propriedades (atributos), no terceiro, as operações (métodos). A UML® permite representar as classes com mais ou menos detalhes. Para representar, de forma plena, as características dos objetos, as classes são dotadas de várias simbologias. Observe a Figura 36, ela identifica todos os componentes discricionais da classe.

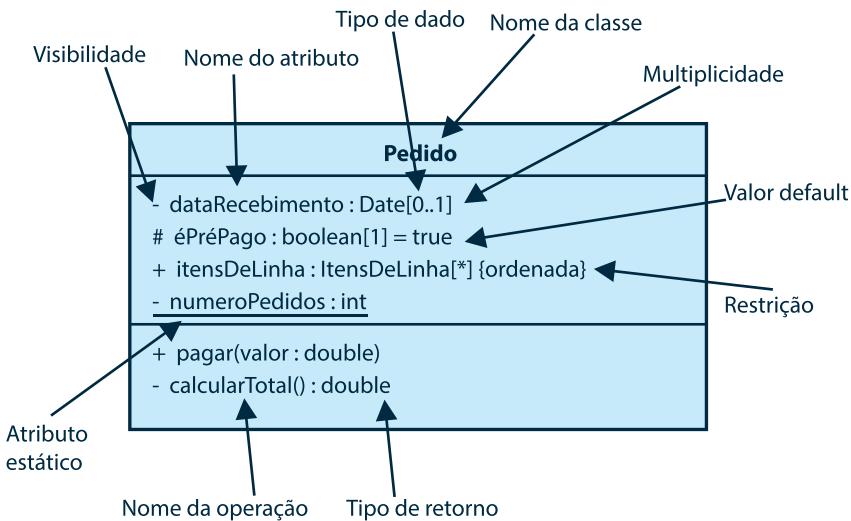


Figura 36: Estrutura da classe

Fonte: Barcelar (online).

Os relacionamentos ligam as classes entre elas, criando relações lógicas, como vimos, são classificados como: Associação (simples, agregação e composição), Generalização, Dependência e Realização.

A Figura 37 ilustra um diagrama de classes e seus componentes.

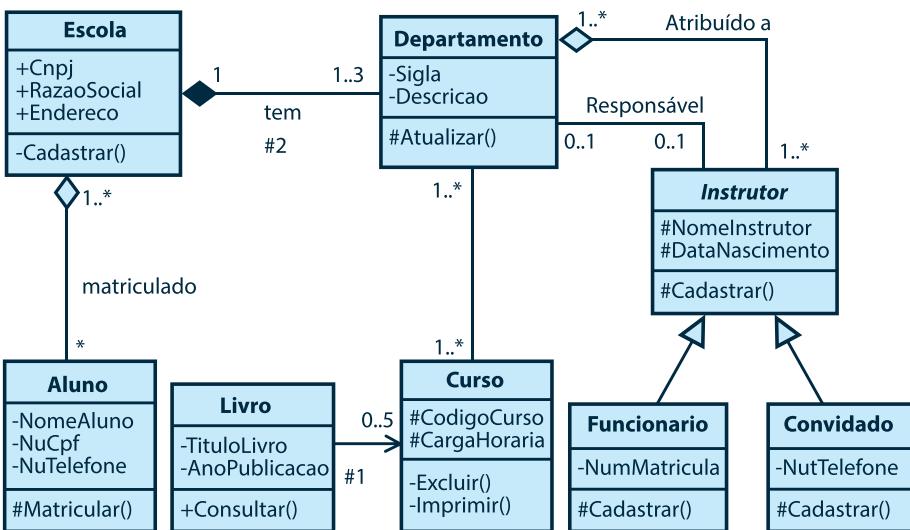


Figura 37: Diagrama de Classes

Fonte: Barcelar (online).

Diagrama de objetos

O diagrama de objetos é uma variação do diagrama de classes, ele representa uma instância do objeto, isto é, representa o sistema em um determinado momento de sua execução. Sua definição oficial está presente somente na UML® versão 1.4.2, as versões mais novas não contemplam essa definição.

um gráfico de instâncias, incluindo objetos e valores de dados, diagrama de objeto estático é uma instância de um diagrama de classes, com o objetivo de detalhar um estado do sistema em um ponto no tempo².

A mesma notação do diagrama de classes é utilizada no diagrama de objetos com duas exceções: a primeira, em que os objetos são escritos com seus nomes sublinhados, e na segunda, todas as instâncias em um relacionamento são mostradas.

Ambos os diagramas de classes e de objetos são úteis para exemplificar objetos e comportamentos complexos de uma classe diagramas complexos de classes auxiliando na sua compreensão. A Figura 38 ilustra um diagrama de objetos.

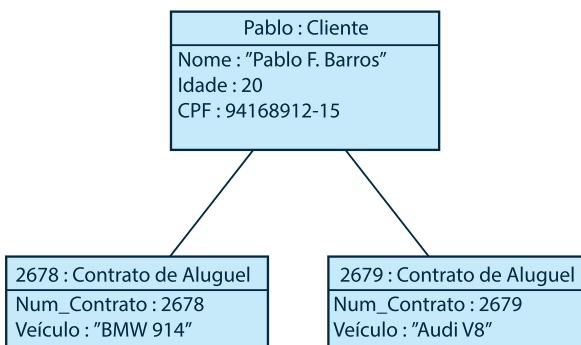


Figura 38: Diagrama de Objetos

Fonte: Linguagem d Modelagem Unificada (online).

Diagrama de pacotes

Um pacote é um conjunto de elementos agrupados. Esses elementos podem ser classes, diagramas, ou até mesmo outros pacotes. Ele é muito utilizado para ilustrar a arquitetura de um sistema, representando os pedaços do sistema repartidos em agrupamentos lógicos e suas dependências (ENGHOLM, 2010, p. 225).

² Disponível em: <<http://www.omg.org/spec/UML/2.5/Beta1/PDF/>>. Arquivo em PDF para download.

Os elementos que compõem esse diagrama são pacotes, dependências, pacotes de elementos, elementos importados, pacotes importados, pacotes mesclados. As Figuras 39, Diagrama de pacotes, e 40, Diagrama de pacotes (estrutura), ilustram formas diferentes de modelagem.

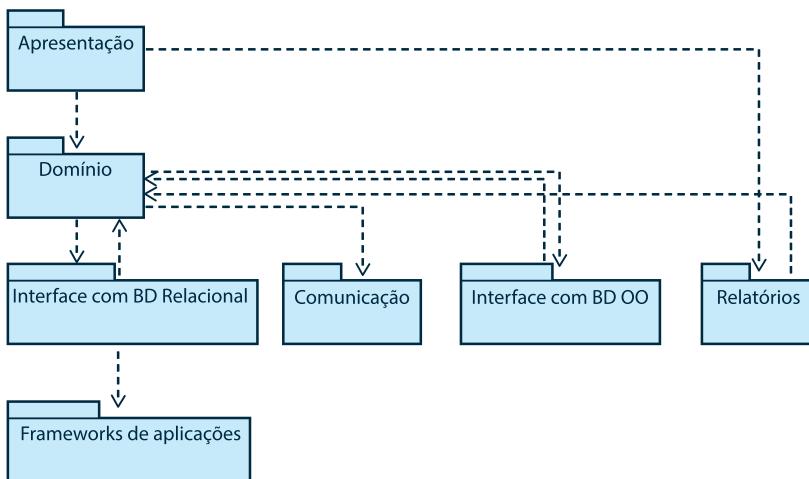


Figura 39: Diagrama de Pacotes

Fonte: adaptada de Diagrama de Pacotes (online).

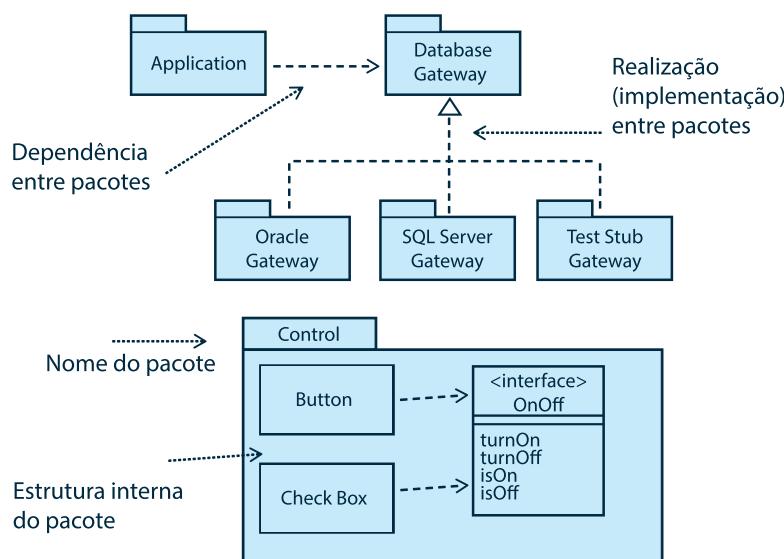


Figura 40: Diagrama de pacotes (estrutura)

Fonte: Barcelar (online).

Diagrama de componentes

Componentes de software podem ser definidos como pedaços de código que contêm um conjunto de interfaces comuns, por exemplo, os executáveis, as bibliotecas, as tabelas do banco de dados, *JavaBeans* etc. O diagrama de componentes permite decompor o sistema em subsistemas que detalham o funcionamento interno.

Os componentes representam a implementação na arquitetura física (linguagem de programação) dos conceitos e das funcionalidades definidas na arquitetura lógica (classes, objetos, relacionamentos). O Diagrama de componentes auxilia no processo de engenharia reversa pela organização do sistema e seus relacionamentos (ENGHOLM, 2010, p. 226). Os elementos que compõem um diagrama de componentes são: componentes, dependências, interface, interface provida, interface requerida, classes, portas, conectores, artefatos, componentes de realização.

Esse tipo de diagrama é muito utilizado para o Desenvolvimento Baseado em Componentes (CBD) e para descrever sistemas com *Service Oriented Architecture* (SOA). A Figura 41 exemplifica um diagrama de componentes.

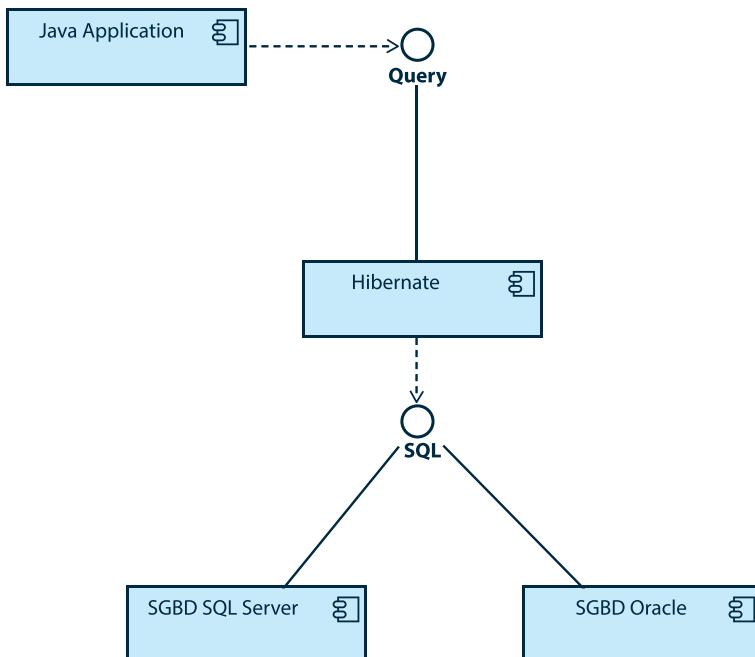


Figura 41: Diagrama de Componentes
Fonte: adaptada de Barcelar (online).

Diagrama de implantação

Mostra a arquitetura do sistema como implantação (distribuição) de artefatos de software para destinos de implementação. Representa a distribuição dos pacotes de sistema em execução nos equipamentos (hardwares). Pode ser utilizado, por exemplo, para mostrar diferenças em implementações de ambientes de desenvolvimento, teste ou de produção com os nomes de compilação específica ou servidores ou dispositivos de implantação (UML-diagrams).

Os elementos que compõem esses diagramas são:

- Nós, representando dispositivos ou ambientes de execução.
- Artefatos, representando código fonte, código binário, executáveis etc.

A Figura 42 ilustra a modelagem de um diagrama de implantação.

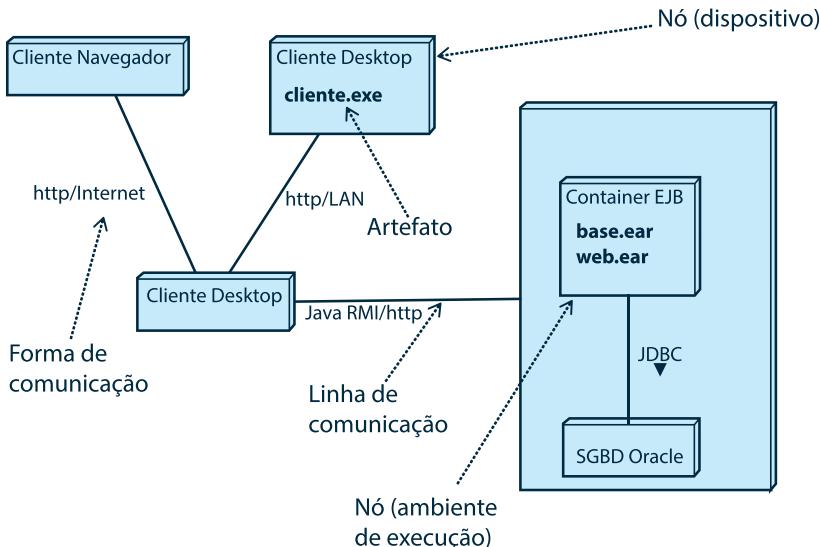


Figura 42: Diagrama de Implantação
Fonte: Barcelar (online).

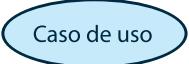
Diagramas comportamentais

Diagramas de comportamento mostram o comportamento dinâmico dos objetos em um sistema, comportamentos que podem ser descritos como uma série de mudanças no sistema ao longo do tempo (UML-diagrams).

Diagrama Caso de Uso

Os diagramas de casos de uso descrevem as funções principais de um sistema e identificam as interações entre o sistema e seu ambiente externo, representado por Atores. Esses Atores podem ser pessoas, organizações, máquinas ou outros sistemas externos (IBM Knowledge Center).

O diagrama é composto por Atores, Casos de uso e relacionamentos:

-  Representação do Ator. Um ator é um usuário do sistema, que pode ser pessoas, organizações, máquinas ou outros sistemas.
-  Representação do caso de uso. Um caso de uso define uma função do sistema.

Os relacionamentos no diagrama de caso de uso podem acontecer entre os atores, entre os casos de uso e entre um ator e um caso de uso. Pode ser do tipo associação, generalização, realização e entre casos de uso, do tipo *Include* e *Extend*.

- <<include>> representa que um caso de uso é essencial para o comportamento de outro.
- <<extend>> representa que um caso de uso pode ser acrescentado para descrever o comportamento de outro, mas não é essencial para isso.

A Figura 43, Diagrama de caso de uso Realizar Pedido Online, ilustra todos os elementos que o compõem.

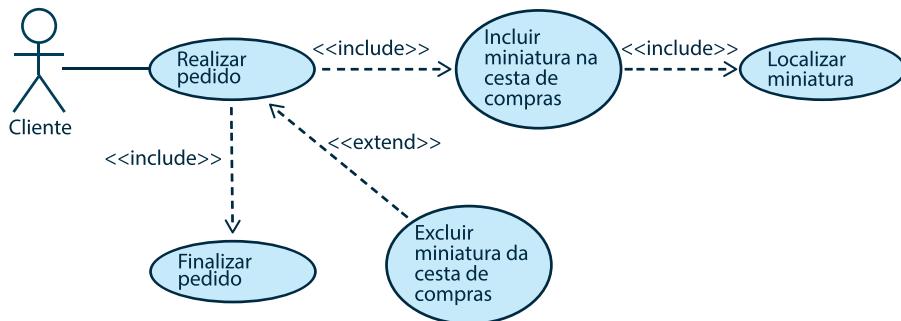


Figura 43: Diagrama de caso de uso Realizar Pedido Online

Fonte: adaptada de Engholm (2010, p. 217).

Um retângulo pode ser utilizado para delimitar os limites do sistema, isto é, até onde as operações são internas e a partir de onde existe interação de atores externos ao sistema. Observe a Figura 44, Diagrama de caso de uso Clínica médica, que representa os limites do sistema.

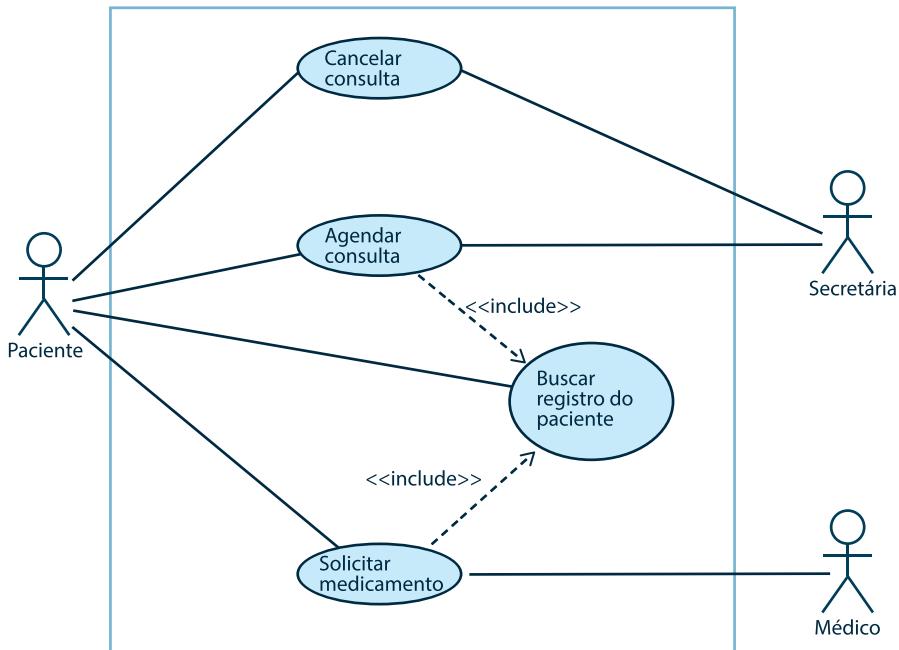


Figura 44: Diagrama de caso de uso Clínica médica
Fonte: adaptado de UFCG – Casos de Uso (online).

Diagrama de atividade

Um diagrama de atividades modela as atividades dos métodos, algoritmos ou, ainda, processos completos. Até a versão 2.0 da UML® esse diagrama era tratado como um caso especial do diagrama de estado, por isso grande parte dos elementos que compõem um diagrama de atividades foi herdada dos diagramas de estados.

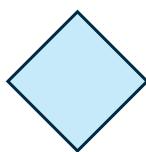
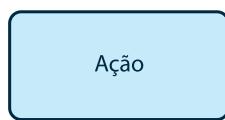
Um diagrama de atividade tem uma grande similaridade com os fluxogramas, ele é essencialmente um gráfico de fluxos. Enquanto os diagramas de sequência focam o fluxo de um objeto para outro, os diagramas de atividades enfatizam o fluxo de uma atividade para outra.

Faço aqui uma pequena pausa na discussão sobre o diagrama para lembrar que, didaticamente, a organização de apresentação dos diagramas tem que seguir uma sequência linear, mas, na prática, os diagramas podem ser utilizados concomitantemente, um em complemento do outro e em várias fases do projeto, por exemplo, os diagramas de atividades são úteis nas seguintes fases:

- Antes de iniciar um projeto, para modelar os fluxos de trabalho mais importantes.
- Durante a fase de requisitos, para ilustrar o fluxo de eventos descritos nos casos de uso.
- Durante as fases de análise e projeto, para auxiliar a definir o comportamento das operações.

Uma atividade na UML® é um elemento que descreve o nível mais alto do comportamento. O diagrama ilustra diversos nós de atividade e linhas de atividade que representam a sequência de tarefas em um fluxo de trabalho, resultando em um comportamento (IBM Knowledge Center).

Os elementos que compõem o diagrama são relacionados a seguir:



- Estado que representa um passo, uma atividade dentro de um fluxo de controle; não pode ser decomposto; não possui ações internas; não pode ser interrompido.
- Representa um ponto de decisão; realização de um teste para seguir um fluxo em detrimento de outro. Conforme a notação sugere, textos de identificação das condições devem ser envolvidos por colchetes.
- *Forks e Joins*. Representa um comportamento condicional. A diferença aqui, para o diagrama de atividades, é que uma bifurcação ou Fork representa uma atividade subdividida em outras; Join significa que as atividades sendo executadas em paralelo chegaram a um ponto de junção para então gerar uma nova atividade.

A construção do diagrama de atividades oferece elementos que promovem a representação da complexidade do software, como, por exemplo, as Raias (*Swimlanes*); a marcação da região de atividade interrompível, isto é, que existe a possibilidade de ser cancelada, interrompida; Zona de expansão; subatividades etc.

A Figura 45, Diagrama de Atividades Pedido, representa um diagrama de atividades com raias, representando os setores físicos de uma empresa que estão envolvidos no processo. A Figura 46, Diagrama Atividades Utilizando *Forks* e *Joins*, ilustra a utilização dos elementos representativos de comportamento condicional. No exemplo da Figura 46, é possível observar que, após a atividade de Receber Pedido, um *Fork* “reparte” as atividades, dando origem a duas outras (Preencher Pedido e Enviar Fatura), e um *Join* ilustra a condição de que somente com a finalização das atividades (Enviar Fatura e Receber Pagamento) é que se executará a próxima (Realizar Entrega).

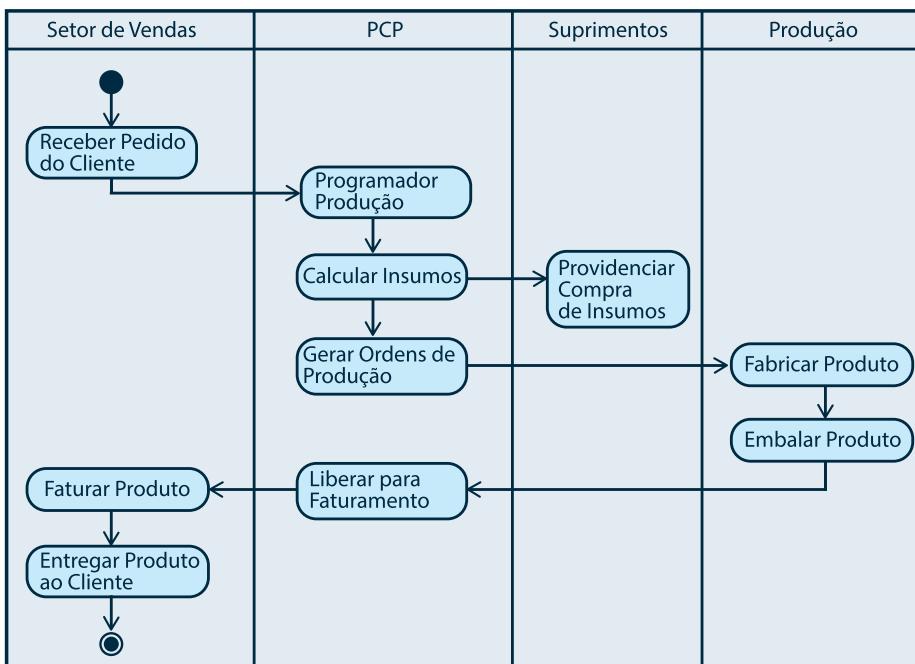


Figura 45: Diagrama de Atividades Pedido

Fonte: Artigo... (online).

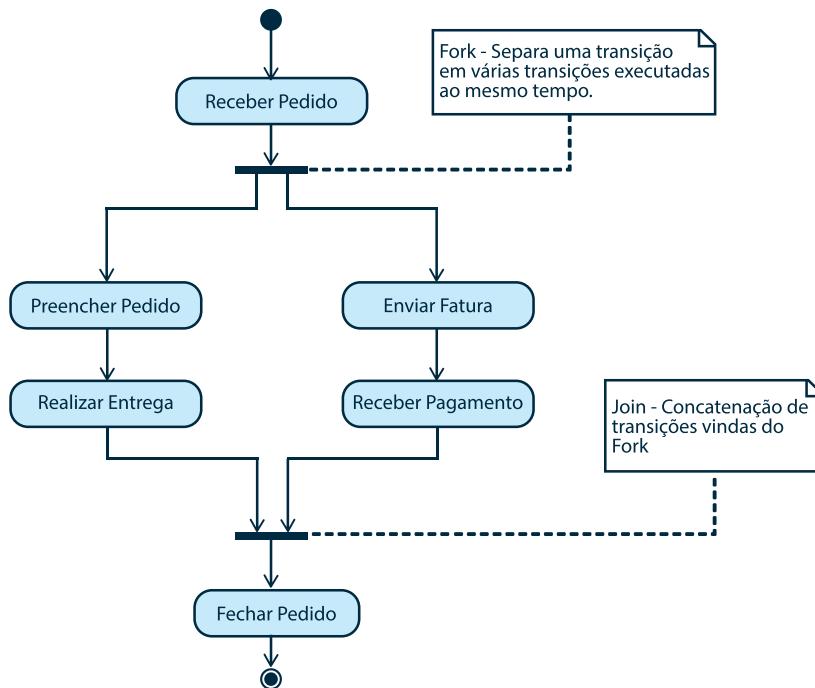


Figura 46: Diagrama Atividades Utilizando Forks e Joins
Fonte: Artigo... (online).

Reprodução proibida. Art. 184 do Código Penal e Lei 9.510 de 19 de fevereiro de 1998.

A Figura 47 ilustra a estrutura do Diagrama de Atividade.

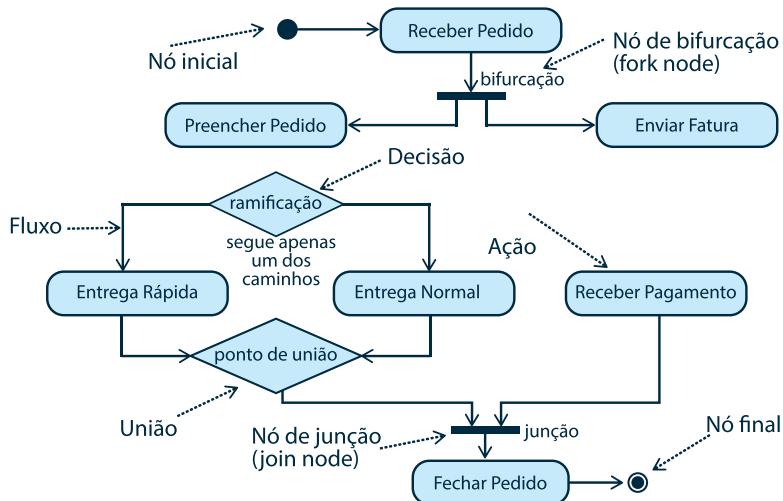


Figura 47: Diagrama de Atividade (estrutura)
Fonte: Barcelar (online).

Diagrama de estados

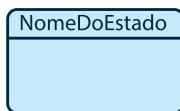
É uma representação gráfica da sequência de estados de um objeto e dos eventos que causam a transição de um estado para outro e também das ações resultantes da alteração de um estado (IBM Knowledge Center).

É usado para modelar o comportamento discreto por transições de estados finitos. Além de expressar o comportamento de uma parte do sistema, máquinas de estado também podem ser utilizadas para expressar o protocolo de utilização de parte de um sistema. Esses dois tipos de máquinas de estado são referidos como máquinas de estados comportamentais e máquinas de estado de protocolo (UML-diagrams).

Não é necessário representar o estado de todos os objetos, esse diagrama é recomendado para aqueles que possuem um comportamento mais complexo.

É muito aplicado na modelagem para o desenvolvimento de sistemas de tempo real ou dirigidos por eventos porque mostram o comportamento dinâmico dos objetos.

Os elementos que compõem um diagrama de estados estão relacionados a seguir:



- Um retângulo com as bordas arredondadas representa o estado do objeto.



- Representa um tipo especial de estado, o Estado inicial do objeto. É especial por que nenhum evento será capaz de devolvê-lo a este estado depois de iniciado o processo.



- Representa outro tipo especial de estado, o Estado final, mantendo o contexto; é um estado especial por que nenhum evento será capaz de devolvê-lo a ele depois de encerrado o processo.



- Transições, representadas por setas simples abertas.



- Forks e Joins, apresentados por um traço largo, representam um comportamento condicional. Fork (bifurcação), ou divisão dos estados, e Join (agrupamento) representam a junção de dois estados para proporcionar a transição para um novo.



SAIBA MAIS

O desenvolvimento de sistemas automatizados de informações, que apoiam as atividades de projeto e manufatura de produtos, deve seguir um modelo como referência para permitir uma melhor compatibilidade e portabilidade de tais sistemas, principalmente quando inseridos em um ambiente integrado de engenharia concorrente. Este artigo demonstra como a Linguagem de Modelagem Unificada (UML) pode ser aplicada em conjunto com o Modelo de Referência para Processamento Distribuído Aberto (ISO/RM-ODP), para o apoio ao desenvolvimento de sistemas de informações orientados a objetos. Enquanto o RM-ODP oferece um padrão para representação de diferentes pontos de vistas de tais sistemas, a UML é utilizada como notação para representação de cada uma dessas vistas. Um processo baseado em Use Cases é empregado para apoiar a evolução da representação das informações dentro desse modelo de referência. O ambiente de projeto de moldes de injeção é utilizado como exemplo para ilustração dos diagramas da UML.

Fonte: Costa (2001, online).

O estado em sua notação pode conter três repartições, no primeiro, como apresentado acima, mostra o nome do estado, os dois outros são opcionais e representam na sequência a variável do estado relacionando seus atributos (de acordo com o listado na classe) e a atividade (lista de eventos e ações). Um objeto passará por vários estados, por exemplo: momento em que foi criado, momento em que foi iniciado, momento em que executou uma solicitação, momento do seu encerramento etc.

Um estado tem várias propriedades: Nome, um rótulo para sua identificação e distinção de um estado para outro; Ações de entrada e saída, ações executadas ao entrar ou ao sair do estado; Transições internas, operações realizadas internamente e que não causam mudança de estado; Subestados e Eventos adiados, uma lista de eventos não manipulados, estado atual do objeto, que aguardam para serem manipulados pelo objeto em outro estado. Uma transição é um relacionamento entre dois estados que indica que após a execução das ações o objeto passará para um novo estado. O acionamento da transição acontece na mudança de estado, que delimita para o objeto o seu estado de origem e seu estado de

destino. Uma transição tem várias propriedades, não é nosso objetivo entrar nesse nível de detalhes. Um evento representa um valor, mensagem ou notificação que habilita a transição de estado. Por exemplo, uma interrupção do sistema operacional, uma função feita por outro objeto. Ele pode ser expresso por argumentos (valores recebidos junto com o evento), condição (expressão lógica, uma transição só ocorre se o evento acontecer e se a condição associada for verdadeira) e ação (cálculo, atribuição, envio de mensagem, etc.).

A Figura 48, Diagrama de Estado Registrar Pedido, ilustra um diagrama de estado.

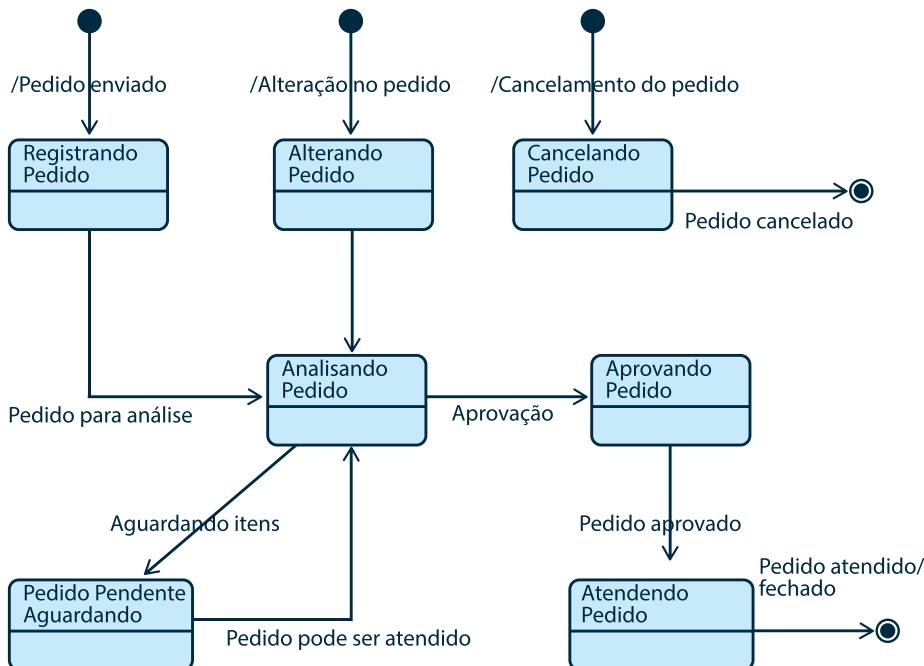


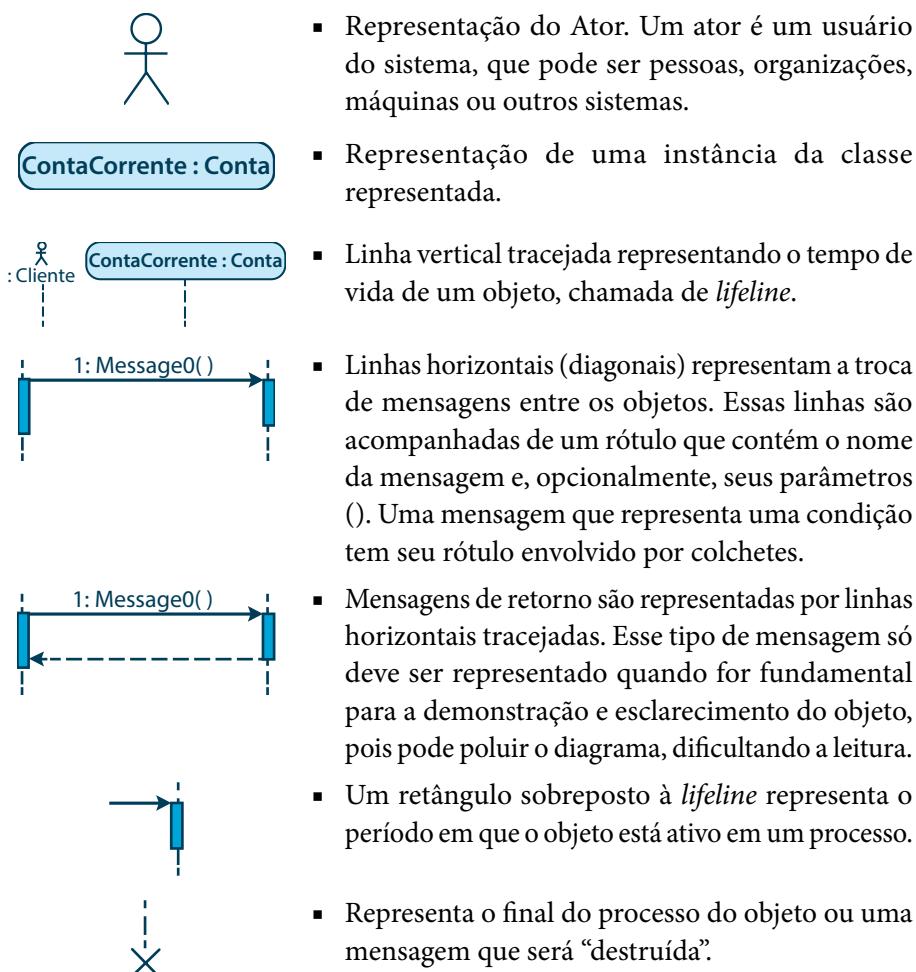
Figura 48: Diagrama de Estado Registrar Pedido
Fonte: adaptada de Diagrama de Estado (online).

Diagramas de Interação

Os diagramas de interação representam a colaboração dos objetos para um determinado comportamento do sistema, eles capturam o comportamento de um objeto dentro de um único caso.

DIAGRAMA DE SEQUÊNCIA

Em modelos UML, uma interação é um comportamento que representa a comunicação entre um ou mais elementos do diagrama (IBM Knowledge Center). Ilustram a sequência de acontecimentos dos eventos de um processo. Utilizado para visualizar e entender: quais condições devem ser satisfeitas, quais métodos devem ser disparados, qual a ordem em que os métodos são disparados. Um diagrama de sequência é composto pelos seguintes elementos:



As mensagens podem ser: síncronas (*wait*), assíncronas (*no wait*), fluxo de controle e de retorno. As trocas podem acontecer de Ator para Ator, de Ator para Objeto, de Objeto para Objeto, de Objeto para Ator.

Na unidade II, quando estudamos sobre os tipos de modelagem, conversamos sobre o diagrama de sequência, a Figura 8 - Diagrama de Sequência Transferir Dados é um exemplo de Diagrama de Sequência. A Figura 49, Diagrama de Sequência Conta Bancária, ilustra um diagrama de sequência.

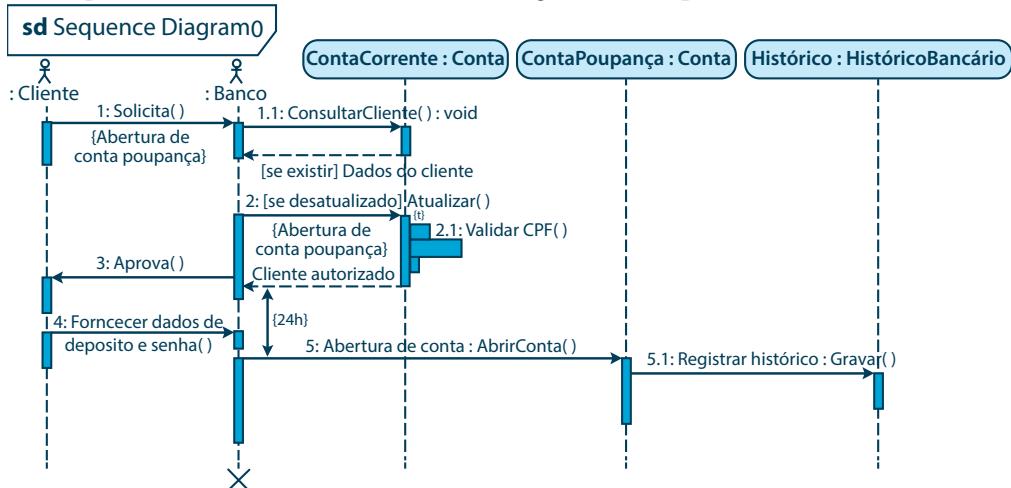


Figura 49: Diagrama de Sequência Conta Bancária

Fonte: a autora.

DIAGRAMA DE COMUNICAÇÃO

Esse diagrama substituiu o diagrama de colaboração das versões anteriores da UML®. Um diagrama de comunicação mostra as interações entre os objetos relacionados com linhas de vida e mensagens transmitidas entre linhas de vida.

O diagrama de comunicação é um diagrama de interação fornecendo uma visualização alternativa das mesmas informações oferecidas pelo diagrama de sequência, sendo que, no de sequência, o foco é ordenar as mensagens pelo tempo, aqui, no de comunicação, o foco é a estrutura de transmissão das mensagens entre os objetos. Esses diagramas ilustram o fluxo de mensagens entre objetos e os relacionamentos implícitos entre eles (IBM Knowledge Center).

Os diagramas de comunicação identificam os seguintes aspectos de uma interação:

- Objetos participantes
- Interfaces exigidas pelos objetos participantes
- Alterações estruturais requeridas por uma interação
- Dados transmitidos entre os objetos em uma interação



REFLITA

“Estou fazendo um sistema operacional gratuito (apenas um hobby, não será grande e profissional como GNU) para 386/486 AT.”

Fonte: Linus Torvalds.

O diagrama de comunicação assemelha-se muito com o diagrama de fluxo de dados da análise estruturada, embora o DFD e informações relacionadas não sejam parte formal da UML, eles podem ser usados para complementar os diagramas UML e fornecer visão adicional dos requisitos e fluxo do sistema. O DFD tem uma visão entrada-processo-saída de um sistema, ou seja, objetos de dados entram no software, são transformados por elementos de processamento, e os objetos de dados resultantes saem do software (PRESSMANN, 2010, p. 159).

Os elementos que compõem a notação do diagrama de comunicação são os seguintes:

- **:FormularioConta** Representa o objeto (classe ou método envolvido).
- Uma linha sólida, representando um relacionamento entre os objetos e uma troca de mensagem.
- Uma seta indica a direção da mensagem.
- Números cardinais, representando a sequência das mensagens.

A Figura 50, Diagrama de Comunicação Conta Bancária, ilustra um diagrama de comunicação.

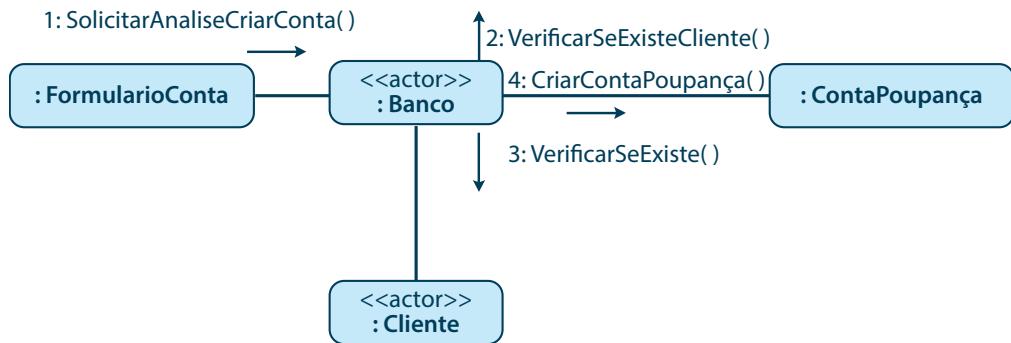


Figura 50: Diagrama de Comunicação Conta Bancária

Fonte: a autora.

Concluindo esta unidade, vimos que o conjunto de diagramas propostos pela UML® para representar a ideia do software, prover sua modelagem, são nove: diagrama de caso de uso, de classes, de objeto, de estado, de sequência, de colaboração, de atividade, de componente e de execução. Sabemos que todos os sistemas possuem uma estrutura estática e um comportamento dinâmico, então a organização lógica para a utilização dos diagramas é:

- Modelagem Estática: Diagrama de Classes; Diagrama de Objetos.
- Modelagem Dinâmica: Diagramas de Estado; Diagrama de Sequência; Diagrama de Comunicação; Diagrama de Atividade.
- Modelagem Funcional: Diagrama de caso de uso; Diagramas de Componente; Diagrama de Execução.

CONSIDERAÇÕES FINAIS

Estudamos nesta unidade os fundamentos da orientação a objetos e a linguagem de modelagem UML®. Observamos que, em um cenário orientado a objetos, a abstração é um dos aspectos mais importantes, pois permite estudar uma entidade complexa abstraindo dela somente os detalhes que interessam em um determinado momento, e, quando usada de forma estratégica, garante um código mais inteligente, mais coeso e com baixo nível de acoplamento, características que possibilitam, entre outras coisas, uma manutenção facilitada e o reuso de pacotes. Na orientação a objetos, os objetos se relacionam e fazem referência a outros objetos o tempo todo, princípio conhecido como dependência.

Na sequência dos estudos, fomos apresentados à UML®, uma linguagem visual de modelagem de dados orientada a objetos, ela é independente de linguagem de programação e também é independente de processo de desenvolvimento. Ficou claro em nossos estudos que a UML® não é uma linguagem de programação.

Um processo de desenvolvimento de software que assume a UML® como linguagem de modelagem deve promover a criação de vários documentos, tanto gráficos quanto textuais. A especificação UML® define dois principais tipos de diagramas: diagramas de estrutura e diagramas de comportamento.

Diagramas de estrutura mostram a estrutura estática do sistema e suas partes em diferentes níveis de abstração e de implementação. Registram como esses elementos estão relacionados uns aos outros e como se afetam no contexto em que estão inseridos. Os elementos em um diagrama de estrutura representam os conceitos significativos de um sistema e podem incluir abstrações, mundo real e conceitos de implementação.

Diagramas de comportamento mostram o comportamento dinâmico dos objetos em um sistema, que podem ser descritos como uma série de mudanças no sistema ao longo do tempo.

ATIVIDADES



1. Dê um exemplo da aplicação dos conceitos de generalização e especialização.
2. Qual é o pilar da orientação a objetos, automaticamente da modelagem orientada a objeto, que se preocupa em decompor um sistema complexo em suas partes fundamentais a fim de descrevê-las em uma linguagem simples e precisa?
 - a) Modularidade
 - b) Encapsulamento
 - c) Abstração
 - d) Herança
3. Dentro do paradigma orientação a objetos existe um mecanismo utilizado para impedir o acesso direto ao estado de um objeto. Assinale a alternativa que apresenta o nome desse mecanismo.
 - a) Mensagem
 - b) Herança
 - c) Polimorfismo
 - d) Encapsulamento
4. Observe a figura abaixo e identifique a alternativa correta a respeito do diagrama de classes.
 - a) Representa um diagrama de colaboração e representa a interação entre as classes Cliente, Funcionário e Pessoa.
 - b) A classe Cliente herda os atributos nome e CPF da classe Pessoa.
 - c) A classe Pessoa herda os atributos cargo, salário e dataAdmissao da classe Funcionário.
 - d) A classe Cliente herda os atributos +getNome() e +SetNome() de Pessoa.

ATIVIDADES



5. Assinale “F” para falso ou “V” para verdadeiro e marque a alternativa corrente:

- () A UML pode ser aplicada somente para modelar projetos relacionados ao desenvolvimento de software.
- () A UML é uma linguagem para especificação, documentação e visualização de softwares orientados a objetos.
- () Ao se modelar um sistema utilizando a UML, segundo normas do *Object Management Group* – OMG – seu mantenedor, deve-se obrigatoriamente utilizar no mínimo quatro de seus diagramas.
- () A UML é um método, isto é, ela diz o que fazer e como modelar um software.

A sequência está correta em:

- a) F, V, F, F.
- b) V, V, F, F.
- c) F, F, V, F.
- d) F, F, V, V.

6. São diagramas UML, EXCETO:

- a) Diagrama de Estado.
- b) Diagrama de Classe.
- c) Diagrama de Conformidade.
- d) Diagrama de Colaboração.

7. Analise a definição a seguir:

“... tem uma grande similaridade com os fluxogramas, ele é essencialmente um gráfico de fluxos. Enquanto os diagramas de sequência focam o fluxo de um objeto para outro, estes enfatizam o fluxo de uma atividade para outra”.

O diagrama descrito é o diagrama de:

- a) Sequência.
- b) Atividades.
- c) Casos de uso.
- d) Comunicação.



PROJETO DE SOFTWARE USANDO A UML

O desenvolvimento orientado a objetos começou em 1967 com a linguagem Simula-67 e desde então surgiram linguagens orientadas a objetos como Smalltalk e C++ entre outras. Nos anos 80, começaram a surgir metodologias de desenvolvimento orientadas a objetos para tirar vantagens desse paradigma. Entre 1989 e 1994 surgiram quase 50 métodos de desenvolvimento orientados a objetos, fenômeno que foi chamado: a guerra dos métodos.

Entre os mais importantes, estavam: o método de G. Booch, a *Object Modeling Technique* (OMT) de J. Rumbaugh, o método de Shlaer e Mellor e o método *Objectory* de I. Jacobson. I. Jacobson introduziu a modelagem de casos de uso em 1987 e criou o primeiro processo de desenvolvimento de software que utiliza casos de uso, chamado *Objectory*.

Cada método possuía uma notação própria, o que gerou uma infinidade de tipos de diagramas e notações. Isso causava problemas de comunicação, treinamento de pessoal e portabilidade. Um esforço de unificação começou em 1994 quando J. Rumbaugh e, logo após, I. Jacobson juntaram-se à G. Booch, na empresa *Rational Software Corporation*. O primeiro grande resultado desse esforço foi a criação da *Unified Modeling Language* (UML), apresentada, na sua versão 1.0, em 1997.

Outro grande resultado desse esforço de unificação de metodologias foi a criação, pela Rational, de um processo de desenvolvimento que usa a UML em seus modelos, chamado Processo Unificado. O Processo Unificado evoluiu do processo *Rational Objectory*, sendo inicialmente chamado de *Rational Unified Process* (RUP). O Processo Unificado, apesar de não ser um padrão, é amplamente adotado, sendo considerado como um modelo de processo de desenvolvimento de software orientado a objetos.

O ciclo de vida de um sistema consiste de quatro fases, divididas em iterações:

- Concepção (define o escopo do projeto)
- Elaboração (define os requisitos e a arquitetura)
- Construção (desenvolve o sistema)
- Transição (implanta o sistema)

Cada fase é dividida em iterações e cada iteração:

- é planejada;
- realiza uma sequência de atividades (de elicitação de requisitos, análise e projeto, implementação etc.) distintas;
- geralmente resulta em uma versão executável do sistema;
- é avaliada segundo critérios de sucesso previamente definidos.





O Processo Unificado é guiado por casos de uso.

- Os casos de uso não servem apenas para definir os requisitos do sistema.
- Várias atividades do Processo Unificado são guiadas pelos casos de uso:
 - planejamento das iterações;
 - criação e validação do modelo de projeto;
 - planejamento da integração do sistema;
 - definição dos casos de teste.

O Processo Unificado é baseado na arquitetura do sistema.

- Arquitetura é a visão geral do sistema em termos dos seus subsistemas e como estes se relacionam.
- A arquitetura é prototipada e definida logo nas primeiras iterações.
- O desenvolvimento consiste em complementar a arquitetura.
- A arquitetura serve para definir a organização da equipe de desenvolvimento e identificar oportunidades de reuso.

Fluxos de atividades

- atividades;
- passos;
- entradas e saídas;
- guias (de ferramentas ou não), *templates*;
- responsáveis (papel e perfil, não pessoa);
- artefatos.

Fonte: Pimentel (2015, online).

MATERIAL COMPLEMENTAR



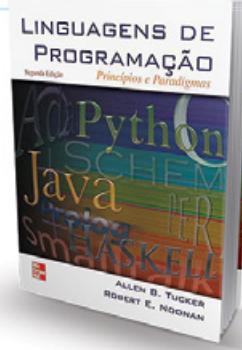
LIVRO

Linguagens de Programação: Princípios e Paradigmas

Allen Tucker e Robert Noonan

Editora: AMGH

Sinopse: Este texto enfatiza um tratamento prático e expandido das questões essenciais do projeto de linguagem de programação, oferecendo a professores e alunos uma mistura de experiências fundamentadas em explicações e implementações. Atualizado, cada capítulo inicia com a apresentação dos principais fundamentos, paradigmas e tópicos das linguagens, provendo tanto uma abordagem ampla quanto profunda dos princípios de projeto de linguagens, permitindo flexibilização na escolha de quais tópicos enfatizar. Inclui amplo tratamento dos quatro maiores paradigmas da programação – programação imperativa, orientada a objetos, funcional e lógica – incorporando algumas das linguagens mais atuais como Perl e Python. Tópicos especiais incluem manipulação de eventos, concorrência e ajuste.



FERRAMENTAS DE MODELAGEM

Objetivos de Aprendizagem

- Apresentar ferramentas CASE para modelagem de software.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Ferramentas CASE
- Modelagem de software utilizando ferramentas CASE
- Ferramentas de modelagem de software
- Comparativo

INTRODUÇÃO

Olá! Bem-vindo(a) a mais esta unidade. No decorrer das unidades, estudamos os conceitos e princípios envolvidos no processo de modelagem, agora, chegou a hora de aplicar todo esse aprendizado. Nesta unidade, discutiremos como uma ferramenta de modelagem de dados torna-se imprescindível para o processo de desenvolvimento de software que utiliza a UML® para a modelagem das suas fases.

Entenderemos como as ferramentas CASE (*Computer-Aided Software Engineering*) podem facilitar o processo de desenvolvimento de software tendo como objetivo principal garantir um nível de abstração alto para o engenheiro de software, eliminando a preocupação com detalhes intrínsecos do ambiente de desenvolvimento. Outro benefício das ferramentas CASE é a possibilidade de documentar o software em modelagem. A documentação e a fácil disponibilização das informações referentes ao projeto para os diferentes profissionais envolvidos garantem melhor comunicação entre eles próprios e com os *stakeholders*.

Analisaremos algumas ferramentas CASE para a modelagem de dados que suportam a linguagem UML®, promovendo a oportunidade de apresentar-lhe as principais ferramentas utilizadas no mercado atual.

O *IBM® Rational® Software Architect* é uma ferramenta avançada e abrangente de projeto, modelagem e desenvolvimento de aplicativos para a entrega de software. *Enterprise Architect*, da *Sparx Systems*, fornece modelagem de ciclo de vida completo para sistemas de negócios e engenharia de software; a *ArgoUML* é uma ferramenta de modelagem UML *Open Source* que oferece suporte a todos os diagramas UML. O *MagicDraw* está listado no site da OMG, como uma ferramenta para modelagem de processos de negócios, de software e de sistemas. Por último, a *Astah Community* (sucessor do *Jude Community*), ferramenta amplamente utilizada em sua versão gratuita. Vamos lá!?

FERRAMENTAS CASE

As ferramentas de modelagem são utilizadas para nortear e disciplinar o processo de desenvolvimento de software na fase do projeto. Como estudado nas unidades anteriores, a UML® alia vários diagramas para auxiliar no projeto e análise do sistema, entretanto, para ter de fato alguma vantagem na utilização da linguagem predita pela UML®, é necessário o uso das chamadas ferramentas CASE (*Computer-Aided Software Engineering*).

Quando uma ferramenta CASE é utilizada em sua plenitude, quero dizer, quando o engenheiro de software realmente incorpora no seu modelo todos os recursos da notação que a ferramenta disponibiliza, e não somente a utiliza, para desenhar uma imagem momentânea, os modelos podem ser feitos como uma representação gráfica para apresentação das funcionalidades para o cliente, até convertidos em tabelas para os bancos de dados e, dali, para o código-fonte. O que acontece, e muito, é que as ferramentas CASE não são integralmente exploradas do ponto de vista funcional, talvez pela complexidade dos conceitos envolvidos ou talvez pela usabilidade das mesmas.

Existem no mercado muitas ferramentas para modelagem de software baseadas na UML®. O site *Objects by Design*¹ apresenta uma relação com informações sobre seus fabricantes, versões, datas e plataformas, apesar de ser datada com última atualização em 2005, a lista apresenta os principais softwares.

CASE acrônimo de *Computer-Aided Software Engineering* ou Engenharia de Software Auxiliada por Computador classifica o conjunto de ferramentas que auxiliam as atividades de engenharia de software por meio da automação por computadores, auxiliando os profissionais de desenvolvimento de software durante uma ou mais fases no processo de desenvolvimento de software.

O termo CASE não se refere diretamente a um tipo de ferramenta, mas à implementação de um ambiente integrado que permita abordar desde a fase de concepção até a implementação do software.

¹ Disponível em: <http://www.objectsbydesign.com/tools/umltools_byProduct.html>. Acesso em: 22 out. 2015.

MODELAGEM DE SOFTWARE UTILIZANDO FERRAMENTAS CASE

É notório, até para quem não é da área, o aumento da complexidade dos sistemas de informação à medida que os recursos de hardware e de software se desenvolvem e se especializam. Quanto mais simples a interface e a interação com o usuário, mais complexos são os seus algoritmos e sua lógica. A construção desses sistemas envolve o esforço e a dedicação de diversos profissionais, o que é chamado de equipe multidisciplinar, assim cada um cuida daquilo em que é especialista. Controlar esse grupo de pessoas, produzindo concomitantemente em um mesmo projeto, não é uma tarefa fácil, bem por isso a indústria vem em um forte esforço implantando o planejamento, os projetos, padrões etc.

Uma das maneiras de enfrentar essas dificuldades é a adoção de uma metodologia de trabalho, um conjunto organizado de métodos e ferramentas com o objetivo de disciplinar o processo de desenvolvimento de software. Nesse contexto, se inserem as ferramentas CASE, que oferecem uma linguagem única de comunicação entre todos os membros da equipe envolvidos no processo de desenvolvimento, qualquer que seja a sua fase (levantamento de requisitos, modelagem, implementação etc.). Ferramentas CASE permitem a criação e manutenção de diversas camadas de modelagem, facilitando a integração entre os diversos profissionais envolvidos no desenvolvimento de software: analistas de sistemas, administradores de dados, DBAs, desenvolvedores de aplicações e gerentes (VALLE, 2007).

Uma das principais fases de um processo de desenvolvimento de software é a Engenharia de Requisitos, que tem como foco definir a funcionalidade do sistema esperada pelo usuário, para, a partir daí, identificar e mapear o problema, propor alternativas de soluções e analisar quais as melhores alternativas, considerando todo o conjunto de elementos que comporão o sistema final.

Como vimos na unidade I, a linguagem visual é a forma mais eficaz de comunicação; as ferramentas CASE promovem essa comunicação por meio do apoio para a modelagem das situações sob alguma linguagem visual, UML®, por exemplo. Existem vários benefícios da utilização de uma ferramenta CASE, como o aumento

na qualidade do produto final, o aumento da produtividade, a redução das atividades de programação, a agilidade no retrabalho, a diminuição da manutenção e do esforço para isso, a redução de custos e o mais óbvio deles é a facilidade para a produção dos artefatos a partir dos esforços de modelagem (VALLE, 2007).

FERRAMENTAS DE MODELAGEM DE SOFTWARE

Existem diversas ferramentas, desde soluções livres até ferramentas pagas. Dentre as ferramentas pagas que se destacam, é possível listar o *IBM® Rational® Architecture*, da antiga *Rational*, incorporado pela IBM, e o *Enterprise Architect*, da Sparx Systems, cujas licenças possuem altos valores, mas oferecem inúmeros recursos e garantia de manutenção, suporte e evolução. Dentre as ferramentas livres, destacam-se o ArgoUML e o Astah (antigo Jude), com download gratuito e acessível a qualquer indivíduo que queira fazer uso das ferramentas. Em comum, elas possuem as facilidades de ligação entre os diagramas e a geração de código básico baseado nos diagramas construídos (FERRAMENTAS..., 2013).

IBM® Rational® Software Architect

IBM® Rational® Software Architect é uma ferramenta avançada de modelagem e desenvolvimento, oferece suporte para todas as etapas do projeto de desenvolvimento de software, o que eles chamam de entrega *end-to-end*. A versão mais recente suporta tecnologias emergentes BPMN2, SOA e Java™ Enterprise Edition 5, e oferece integração com as soluções de gerenciamento de ciclo de vida de aplicativos da IBM. O *IBM® Rational® Software Architect* possui assistentes (*wizards*), *templates* e outros recursos para auxiliar na plena utilização da ferramenta.

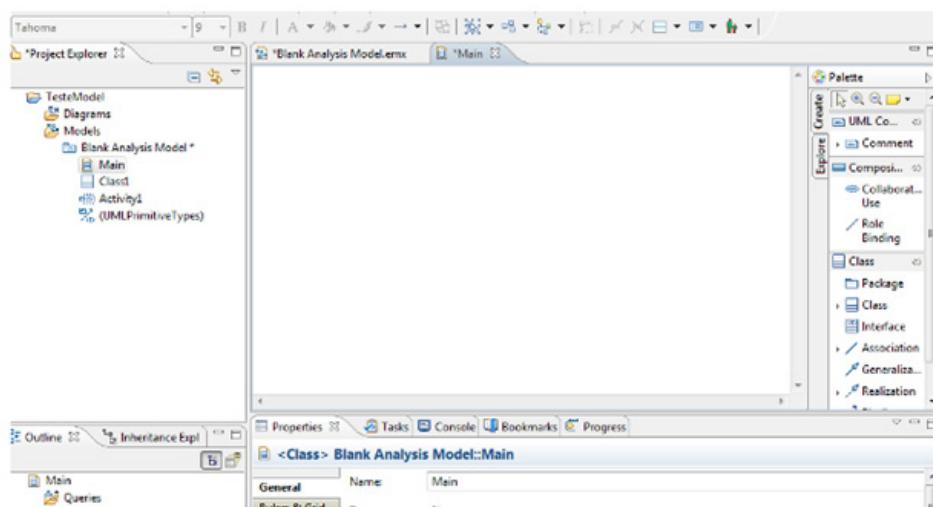
É um software proprietário, oferece versões *Trial* e uma versão *Academic Initiative*. A versão *Trial* oferece todas as funcionalidades da ferramenta durante um período de trinta dias. A versão *academic* garante licença definitiva dos produtos IBM Rational gratuitamente, desde que comprovado que sua utilização é para fins educacionais e de pesquisa.

É uma poderosa ferramenta baseada no projeto de *framework open-source*

do *Eclipse* e também dispõe dos recursos básicos herdados do *Eclipse*, tais como: criação de projetos web comuns, EJB's, JSE básico etc. (SOUZA, 2012). A interface do RSA é muito parecida com a do *Eclipse*, observe a Figura 51, Ambiente de trabalho do RSA. Quem já está familiarizado com o *Eclipse*, não encontrará dificuldade para operar a ferramenta.

Figura 51: Ambiente de trabalho do RSA

Fonte: a autora.



Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Recursos oferecidos pelo *IBM Rational Software Architect* (RSA):

Modelagem UML:

- Diagramas de Casos de Uso.
- Diagramas de Atividades.
- Diagramas de Classes.
- Diagramas de Interação.
- Diagramas de Estrutura Composta.
- Diagramas de Componentes.
- Diagramas de Implementação.
- Diagramas de Máquina de Estados.

Recursos adicionais de modelagem:

- Rastreabilidade e Análise Estática.
- Modelagem de Topologias de Implementação.
- Modelagem em Equipe.
- Simulação do Comportamento do Modelo.
- Esboço de design ágil.

Transformações:

- UML para SQL logical data model (IBM InfoSphere Data Architect).
- UML para XSD; XSD to UML.
- UML para Java, Java to UML;
- UML para JPA, JPA to UML;
- UML para C#, C# to UML;
- UML para VB.NET, VB.NET to UML;
- UML para CORBA.

Editores gráficos para:

- Java
- C#
- VB.NET
- Análise arquitetural, revisão e métricas
- Revisor C# e Métricas
- ALM através da integração com as ferramentas de ALM da IBM.

Ferramenta altamente recomendada para profissionais e projetos em JAVA, principalmente se a IDE utilizada for o *Eclipse*.

SAIBA MAIS



O IBM® Rational® Software Development Platform é um ambiente de desenvolvimento comum que contém o ambiente de trabalho de desenvolvimento e outros componentes de software que compartilham vários produtos. A plataforma de desenvolvimento inclui as seguintes ferramentas: Rational Application Developer; Rational Functional Tester; Rational Performance Tester; Rational Software Architect; Rational Software Modeler; Rational Systems Developer; Rational Tester for SOA Quality. Também disponível, mas que não faz parte da plataforma, está o Rational Manual Tester.

Fonte: IBM® Rational® Software Architect.

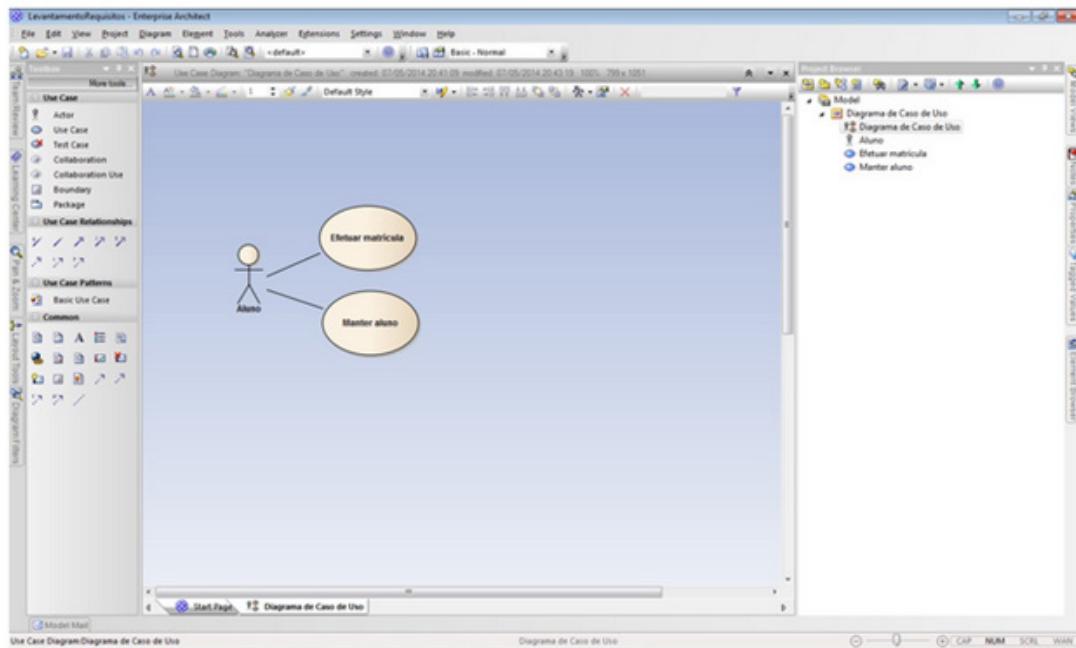
ENTERPRISE ARCHITECTURE

O *Enterprise Architect* é uma ferramenta CASE que gerencia equipes e é voltado para o pleno desenvolvimento do ciclo de vida do produto. Oferece alta performance visual e ferramentas para a modelagem de negócios, engenharia de sistemas, arquitetura corporativa, gerenciamento de requisitos, projeto de software, geração de código e testes.

O *Enterprise Architect* pode ser adquirido por número de licenças e utilizando um servidor disponibilizando-as em uma rede de computadores interna, a exemplo de softwares corporativos. Oferece uma versão *Trial* para a utilização por trinta dias.

Os modelos do EA podem ser mantidos em banco de dados. A ferramenta provê o recurso SVN (*Subversion*) suporte para versões do mesmo documento e controle dessas versões a partir de um repositório comum, em que são executados *merges* do conteúdo e registrados os *logs*. O *Enterprise Architect* oferece o recurso de comparação de modelos: entre modelos e banco de dados físico; entre dois bancos de dados físicos.

O *Enterprise Architect* gera a documentação em dois formatos: em HTML, que pode ser hospedada na intranet, ou em formato PDF. Garante integração (nativa) com Visual Studio.NET e Eclipse.



A Figura 52 ilustra a tela de trabalho do EA.



Modelagem Estratégica permite a uma organização planejar o futuro e tomar decisões de acordo com sua missão e valores. O EA pode modelar todas as fases do planejamento e desenvolvimento de processo, das “nuvens para o mundo real”. Esse Guia demonstra como criar modelos estratégicos, incluindo uma Mente Mapping Diagrama, Carta de organização, fluxograma, Cadeia de Valor, Balanced scorecard e Mapas Estratégicos. Utilizando o estudo de uma linha estratégica para um restaurante, ilustra como Enterprise Architect pode ligar vários modelos estratégicos, proporcionando rastreabilidade, permitindo que as partes interessadas possam localizar um requisito de forma direta para uma discussão ou definição de nova estratégia corporativa.

Fonte: Sparx Systems (2010, online).

Recursos oferecidos pelo *Enterprise Architect* (EA):

Modelagem UML:

- Diagramas de Casos de Uso.
- Diagramas de Atividades.
- Diagramas de Classes.
- Diagramas de Interação.
- Diagramas de Estrutura Composta.
- Diagramas de Componentes.
- Diagramas de Implementação.
- Diagramas de Máquina de Estados.

Padrões suportados:

- SysML
- BPMN
- Archimate
- ArcGIS
- TOGAF
- Zachman
- UPDM
- UML 2.5 plus

Supporte a projetos:

- Gerenciamento de requisitos.
- Gerenciamento de projetos.
- Manutenção.
- Gerenciamento de Testes.
- Documentação (Html, PDF).

- Engenharia de código.
- Simulações.
- *Debugging*.
- *Scripting*.
- *Profiling*.

Transformações:

- *C#*
- *DDL*
- *EJB*
- *Java*
- *JUnit*
- *NUnit*
- *WSDL*
- *XSD*

O EA é uma ferramenta completa e, para sua utilização plena, é necessário um treinamento intenso para quem vai operá-lo, tanto para a manipulação da ferramenta quanto nos conceitos UML®, sua licença possui um valor comercial relativamente alto.

ARGOUML

A ferramenta ArgoUML é livre, de código aberto. Escrito em JAVA, dentre as ferramentas de código aberto, é a mais complexa.

Uma vantagem do ArgoUML é a classe *Model*, responsável pelo provimento de independência entre as versões da UML e os demais subsistemas. A *Model* encapsula as informações sobre qual implementação da UML está sendo usada, assim as implementações da UML são vistas como *strategies*, o que garante uma evolução fácil da ferramenta para versões mais novas da UML.

A interface com o usuário é bastante intuitiva e agradável, oferece uma excelente documentação, fornecendo exemplos de extensão, explicações sobre os principais módulos e sobre sua arquitetura, e mantém comentários de código de ótima qualidade.

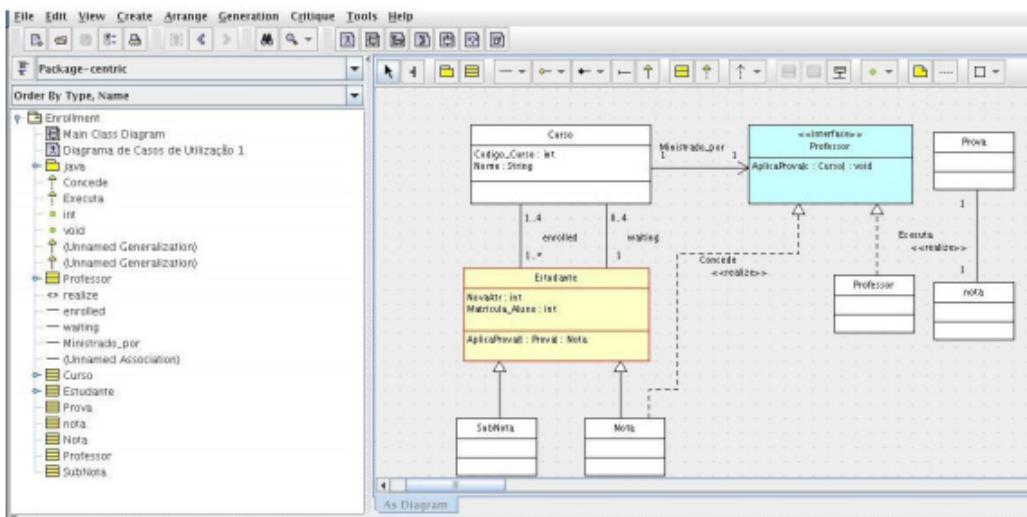


Figura 53: Ambiente de trabalho do ArgoUML
Fonte: a autora.

O ARgoUml tem suporte para XMI e garante a exportação dos gráficos nos formatos SVG ou imagens vetoriais, como GIF, PNG, PS, EPS e PGML. Pode ser executado em qualquer versão do Java 1.2. (Usa a licença BSD). O kit de ferramentas Dresden OCL incluído no ArgoUML permite executar sintaxe e programar projetos em Java, C ++, C #, PHP4 e PHP5. Oferece o aplicativo “To Do List”, que permite fazer uma lista de coisas a fazer para o projeto.

ASTAH COMMUNITY

Astah Community é uma ferramenta gratuita para a modelagem de softwares orientados a objeto. Além do Astah Community, ele é distribuído em três outras versões: Astah UML, Astah Professional e Astah Share, cada uma delas disponibiliza funcionalidades diferentes e possui licença comercial.

Oferece ainda uma versão Acadêmica contemplando todos os recursos e é livre de licença, quando comprovada que sua utilização será puramente para a orientação e pesquisas acadêmicas. A ferramenta Astah Community é reconhecida por sua praticidade e simplicidade. Exporta os diagramas para os formatos PNG/JPEG/EMF/SVG.

A Figura 54 apresenta o ambiente de trabalho do Astah Community.

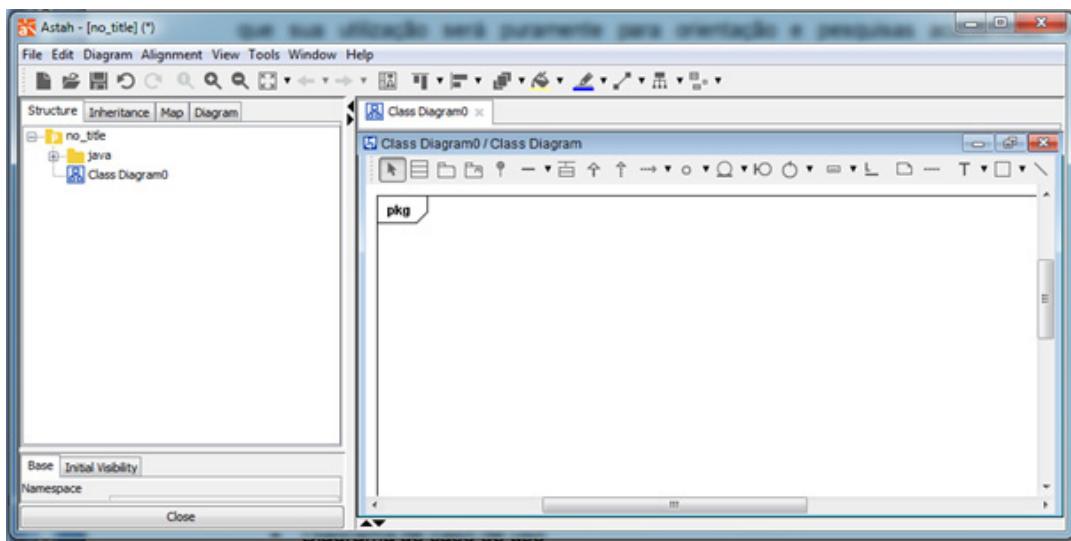


Figura 54: Ambiente de trabalho do Astah Community

Fonte: a autora.

Recursos oferecidos pelo Astah Community:

Modelagem UML:

- Diagrama de classes.
- Diagrama de caso de uso.
- Diagrama de sequência.
- Diagrama de Atividade.
- Diagrama de comunicação.
- Diagrama de estados.
- Diagrama de componente.

- Diagrama de implantação.
- Diagrama de estrutura de composição.
- Digrama de objetos.
- Diagrama de pacotes.

Plug-ins:

- Atlassian Confluence Plug-in
- Script Plug-in
- PHP Export
- Tweet-a-gram
- yUML Plug-in

Transformações:

- Java source code
- Java skeleton code
- C#, C++ skeleton code

Outros recursos:

- Documentação em HTML (*javadoc*) e RTF
- Exportação SQL
- Merge (projetos)

COMPARATIVO

| CRITÉRIO | RSA | EA | ARGOUML | ASTAH |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Licença | Proprietário. Versão Trial (30 dias) Versão Academic | Proprietário Versão Trial (30 dias) | Open Source | Proprietário Versão Trial (30 dias) Versão Community Versão Academic |
| Diagramas UML suportados | Diagramas de Casos de Uso Diagramas de Atividades Diagramas de Classes Diagramas de Interação Diagramas de Estrutura Composta Diagramas de Componentes Diagramas de Implementação Diagramas de Máquina de Estado | Diagramas de Casos de Uso Diagramas de Atividades Diagramas de Classes Diagramas de Interação Diagramas de Estrutura Composta Diagramas de Componentes Diagramas de Implementação Diagramas de Máquina de Estado | Diagramas de Casos de Uso Diagramas de Atividades Diagramas de Classes Diagramas de Interação Diagramas de Estrutura Composta Diagramas de Componentes Diagramas de Máquina de Estado | Diagrama de classes Diagrama de caso de uso Diagrama de sequência Diagrama de Atividade Diagrama de comunicação Diagrama de estado Diagrama de componente Diagrama de implantação Diagrama de estrutura de composição Digrama de objetos Diagrama de pacotes |
| Recursos adicionais | Design ágil Rastreabilidade e Análise Estática Modelando Topologias de Implementação Modelagem em Equipe Simulação do Comportamento do Modelo Transformações | Subversion Engenharia reversa Comparação de modelos Rastreabilidade | To do list | Documentação em HTML (javadoc) e RTF Exportação SQL Merge (projetos) |
| Plataforma | Windows | windows | multiplataforma | multiplataforma |

SAIBA MAIS



“Nossa obrigação de sobreviver e prosperar são devidos não apenas a nós mesmos, mas também ao cosmos, antigo e vasto, do qual surgimos.” Carl Sagan

Fonte: Oliveira (2015, online).

Como colocado no início desta unidade, várias outras ferramentas estão disponíveis para a modelagem de software, oferecendo mais ou menos recursos, com licença comercial ou distribuição gratuita. A seleção de uma em detrimento de outra deve considerar as características exclusivas de cada projeto de desenvolvimento de software e também as características organizacionais do ambiente de trabalho e cultural da organização.

CONSIDERAÇÕES FINAIS

Chegamos ao final de mais uma etapa! O objetivo desta unidade foi pontuar as vantagens da utilização de ferramentas CASE para a modelagem de software e apresentar algumas delas.

Podemos definir o termo CASE como um conjunto de ferramentas e técnicas com o propósito de auxiliar os desenvolvedores na construção de softwares e sistemas complexos, garantindo a diminuição dos esforços e a da complexidade. Também pode ser atribuída a uma ferramenta CASE a melhoria do controle do projeto pela aplicação de processos automatizados, produzindo produtos finais com qualidade. Quando se opta por utilizar uma ferramenta CASE, o grande objetivo a ser alcançado para a obtenção de todas as vantagens que podem oferecer é a implementação de um ambiente de desenvolvimento amplamente integrado que permita abordar desde a fase de concepção do sistema, implementação e posterior manutenção.

A utilização de ferramentas automatizadas garante inúmeros benefícios, como, por exemplo, aumento na qualidade do produto final, aumento da produtividade, redução das atividades de programação, agilidade no retrabalho, diminuição da manutenção e do esforço para isso, redução de custos.

Vimos que existem diversas ferramentas, desde soluções livres até ferramentas pagas. Dentre as ferramentas pagas que se destacam, foram apresentados o software *IBM® Rational® Architecture* e o *Enterprise Architect*, que são softwares proprietários, mas oferecem inúmeros recursos e garantias. Dentre as ferramentas livres, apresentamos ArgoUML e o Astah, ambas com versões gratuitas e acessíveis a qualquer indivíduo que queira fazer uso delas. Em comum, tanto as pagas como as gratuitas possuem as facilidades de ligação entre os diagramas e a geração de código básico baseado nos diagramas construídos.

Na próxima unidade será apresentado o processo de modelagem na prática, utilizando uma ferramenta CASE.

ATIVIDADES



1. Em relação às funcionalidades oferecidas por ferramentas CASE (*Computer-Aided Software Engineering*), marque V para a afirmação verdadeira e F para afirmação falsa e escolha a opção que contém a sequência correta.
 - () Padronização do processo de desenvolvimento.
 - () Reutilização de vários artefatos por diferentes projetos, permitindo o aumento da produtividade.
 - () Automação de atividades, tais como: a geração de código e de documentação.
 - () Modelagem de processos de negócio.

a) F, V, V, V.
b) V, F, V, V.
c) V, V, F, V.
d) V, V, V, V.
2. Analise a definição a seguir:

“CASE é o acrônimo de Computer-Aided Software Engineering, trata-se de um software que tem por objetivo auxiliar os profissionais do desenvolvimento de software. Um dos componentes indispensáveis de uma ferramenta CASE é a modelagem visual, isto é, permitir a representação, por meio de modelos gráficos, o que está sendo definido e, em particular, diagramas da análise orientada a objetos por meio da UML.”

Selecione a alternativa que possui dois exemplos de ferramentas.
 - a) ERwin e Delphi.
 - b) Dreamweaver e Astah
 - c) MagicDraw e JAVA.
 - d) Enterprise Architect e StarUML.
3. Assinale a alternativa que NÃO é uma vantagem do uso de ferramentas CASE:
 - a) Agilizar o tempo para a tomada de decisão.
 - b) Treinamento para a utilização.
 - c) Melhoria e redução de custos na manutenção.
 - d) Menor quantidade de códigos de programação.



UMA AVALIAÇÃO DE FERRAMENTAS DE MODELAGEM DE SOFTWARE

Ferramentas de modelagem servem para orientar e disciplinar o processo de desenvolvimento de software durante a fase de projeto. Entretanto, tais ferramentas não são totalmente exploradas do ponto de vista funcional, seja por complexidade do assunto abordado ou pela usabilidade das mesmas.

Este artigo se propõe a avaliar o uso de duas ferramentas de modelagem de software, sendo uma delas no formato online e outra em desktop, por meio de um experimento realizado na Universidade Federal de Minas Gerais com alunos de graduação e pós-graduação.

O objetivo do estudo foi avaliar a usabilidade das duas ferramentas open source, ArgoUml e Gliffy, que permitem a modelagem de software utilizando a notação da UML. São apresentados estudos referentes aos níveis de aceitação e de utilização das ferramentas investigadas.

Pelo modelo proposto por Wohlin et al. (2012), foi definido que este experimento possui por objetivo avaliar as ferramentas ArgoUml e Gliffy referente à usabilidade do ponto de vista dos alunos de graduação e pós-graduação no contexto da disciplina. O experimento foi executado em dois laboratórios, equipados com 20 máquinas idênticas em cada um, na Universidade Federal de Minas Gerais (UFMG).

O experimento foi executado com alunos da disciplina de Engenharia de Software Experimental da UFMG. Para a boa realização do experimento, foi feito um resumo para os participantes das principais notações dos diagramas UML que seriam usados. O objetivo é que todos os participantes estivessem no mesmo nível de conhecimento para a execução do experimento. O experimento foi executado em um só dia: 31/03/2014. A turma foi dividida em duplas, sendo um integrante do curso de graduação e o outro do curso de pós-graduação. A metade da turma utilizou a ferramenta ArgoUml e a outra metade a ferramenta Gliffy.

Podemos concluir, com base nos resultados preliminares deste experimento, que a ferramenta Gliffy possui melhor usabilidade que a ferramenta ArgoUml, visto que ela apresentou uma maior facilidade de aprendizado. Ou seja, os participantes rapidamente conseguiram explorar o sistema e elaborar os diagramas em um tempo menor que na ferramenta ArgoUml. Além disso, os usuários da ferramenta Gliffy acionaram menos vezes o grupo de avaliadores para tirar dúvidas. Além disso, nossa conclusão é de que este experimento deva ser realizado novamente, com um quantitativo maior de participantes e/ou de ferramentas UML para que possa trazer um resultado mais significativo e seguro. Uma métrica que gostaríamos de adicionar na próxima execução é a quantidade de cliques acionados pelos participantes para elaborar cada diagrama.

Fonte: Oliveira, Souza e Figueiredo (online).

MATERIAL COMPLEMENTAR



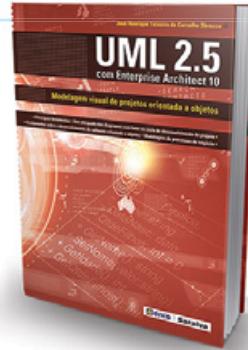
LIVRO

UML 2.5 com Enterprise Architect 10 - Modelagem Visual de Projetos Orientada a Objetos

Teixeira de Carvalho Sbrocco, José Henrique

Editora: Erica

Sinopse: Com exemplos inspirados no cenário corporativo, é uma leitura dirigida não apenas para estudantes de cursos relacionados à computação, mas também para profissionais já atuantes no mercado. Apresenta considerações sobre a versão 2.5 Beta 2 da UML, trazendo conceitos, objetivos e aplicações. Fornece uma visão geral do uso da UML durante um ciclo de desenvolvimento de software, abordando conceitos importantes de orientação a objetos. Destaca as principais ferramentas que implementam a UML, considerando aspectos de sua seleção e incluindo orientações sobre o uso do Enterprise Architect 10. Conceitua atores, entidades e outros aspectos do modelamento de processo de negócio. Apresenta os diagramas que podem ser utilizados para representar necessidades complementares, como o de estrutura composta e o de temporização. Descreve mecanismos de extensão proporcionados pela UML e aborda OCL e WAE. O livro segue a mesma estrutura didática que UML 2.3 – Teoria e Prática, mas foi atualizado e reestruturado com mais informações e exemplos relacionados à nova versão.



MODELAGEM ORIENTADA A OBJETO COM UML® E ASTAH COMMUNITY

UNIDADE

V

Objetivos de Aprendizagem

- Apresentar uma aplicação prática da modelagem de software.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Projeto de desenvolvimento de software

INTRODUÇÃO

Olá, aluno(a)! Chegamos à última etapa dos nossos estudos sobre modelagem de software. Acompanhando o contexto deste material, esta unidade irá trabalhar a aplicação prática dos conceitos apresentados. Por observação, percebi que normalmente os alunos não encontram dificuldades na interpretação dos conceitos, por exemplo, quando falamos de herança e sugerimos que Funcionário herda de Pessoa, a compreensão é passiva. A dificuldade fica para o momento de o aluno ter que abstrair essas informações do mundo real, de um problema real, no momento da prática.

Esta unidade tem o objetivo de apresentar uma aplicação prática da modelagem de software, tentando reproduzir da melhor maneira uma situação real de projeto de desenvolvimento de software que segue as especificações da engenharia de software, utilizando um estudo de caso.

Trabalharemos com a ferramenta CASE *Astah Community*. Foi escolhida por fornecer uma versão para distribuição gratuita e por prover todos os recursos necessários para os diagramas e notações da UML®. Não entraremos nos pormenores da produção de código, nosso objetivo é a elaboração da modelagem.

Primeiro, retomaremos conteúdos da disciplina de projeto e de engenharia de requisitos, na intenção de demonstrar a interdisciplinaridade dos conteúdos apresentados. Como já discutimos, na academia, separamos os conteúdos em disciplinas sequenciais para facilitar a didática, mas, na prática, eles estão interligados, eles dependem um do outro, eles coexistem.

Na sequência, entramos no mundo prático do desenvolvimento de software. A partir de uma solicitação para desenvolvimento de sistema fictício, iremos adotar um processo de desenvolvimento de software e seguir todas as suas fases, dando ênfase, é claro, à modelagem. Não definiremos interface, tampouco alcançaremos as fases de implementação e manutenção, nosso objetivo é focar a análise e modelagem dos requisitos, a definição das classes de projeto, a definição do diagrama de classes, de estados e outros que se fizerem necessários para a modelagem dos objetos mais complexos. Vamos lá!?

PROJETO DE DESENVOLVIMENTO DE SOFTWARE

Trabalharemos nesta unidade utilizando o estudo de caso apresentado por Deboni (2003, p. 140 e ss.).

Cenário: agilizar processo de venda em loja de departamentos, a fictícia *TemPraTodos*.

Cenário atual: um cliente escolhe os produtos que deseja comprar. Ele se encaminha a um caixa que deve finalizar a venda. Em geral, o caixa possui um terminal com um leitor de código de barras que obtém, com base em um código, o preço dos produtos. Alguns produtos podem ter descontos para pagamento à vista, ou parcelamento em duas ou três vezes, com ou sem juros.

Solicitação: Deseja-se um sistema de informação que, quando inserido o código do produto, ele informe o preço e as condições de pagamento deste produto. As condições de pagamento serão definidas em função do crédito que o cliente tem com a loja em uma regra de negócio pré-estabelecida.

A boa prática garante: não existe processo sem projeto. A primeira etapa para o desenvolvimento de qualquer sistema de informação, seja ele uma simples modificação até a elaboração de um sistema de informação complexo, é a definição do projeto de software. Não entraremos nos pormenores sobre a gestão de projetos, mas definiremos um Plano de Projeto para o desenvolvimento do sistema Automação da Condição de Pagamento.

O plano e demais artefatos apresentados nesta unidade são fictícios, possuem conotação acadêmica e não estão relacionados a nenhuma outra disciplina, são elaborações próprias com o objetivo de estabelecer a modelagem no contexto do desenvolvimento do software. Tampouco farei o registro de um processo completo de desenvolvimento de software, pretendo somente registrar as principais fases, novamente com o objetivo de situar a modelagem no contexto do desenvolvimento de software.

PROCESSO DE SOFTWARE

O processo de software é um conjunto de atividades que nos orienta durante o desenvolvimento do software. Um processo irá determinar as responsabilidades de cada um dos envolvidos, o prazo e quais ferramentas serão utilizadas.

Existem inúmeras propostas e sugestões para processo de softwares, todas baseadas em experiências bem ou mal sucedidas. Não existe um processo ideal, o que a literatura propõe são “boas práticas” que podem ser adaptadas para cada empresa ou profissional e, ainda assim, dentro da mesma empresa, para cada projeto, deve ser estabelecido seu próprio processo, considerando suas exigências e realidade comercial.

Qualquer modelo que você decida estudar, ainda que possua etapas e métodos diferentes, perceberá que apresenta os mesmos objetivos finais: cumprir o prazo, cumprir o custo, entregar um produto de qualidade e único; também exige as mesmas condições: pessoas treinadas e motivadas, processos claros e definidos e ferramentas adequadas.

Na prática, o que precisamos identificar é o processo de software que melhor se adapte à realidade do nosso negócio e que atenda às necessidades de produção do software a ser desenvolvido. Sommerville (2011, p. 19) me apoia nessa afirmação quando diz que: “não existe um processo ideal, a maioria das organizações desenvolve os próprios processos de desenvolvimento de software”. Para nosso projeto, partiremos do princípio de que a empresa já possua um modelo de processo de software padrão que considera de uma forma geral as seguintes fases: Levantamento de requisitos; Análise de Requisitos; Projeto; Implementação; Testes; Implantação.

Termo de abertura do projeto – TAP

| | | |
|------------------------------|-----------------------------|-----------------|
| NOME DO PROJETO: TEMPRATODOS | GERENTE : Guilherme Salgado | C. CUSTO 07.415 |
|------------------------------|-----------------------------|-----------------|

ESCOPO INICIAL

OBJETIVO: Desenvolver nova funcionalidade para um sistema em operação e promover a automação do processo de condições de pagamento a partir da informação do código do produto, em tres meses, com orçamento de R\$ 10.000,00.

METAS: Análise da arquitetura atual, análise da regra de negócios, projeto de desenvolvimento

PREMISSAS

- prioridade dos funcionários é para as rotinas internas da loja;
- Guilherme entrará em férias em 15 dias;

RESTRIÇÕES

- manter arquitetura atual

RISCOS

- Atraso no projeto pela falta de colaboração/tempo dos funcionários, durante levantamento de requisitos
- Atraso no projeto pela falta do gerente do projeto

PRAZO

| | |
|----------------------------------|--------------|
| Entrega do Projeto em 10/01/2016 | INVESTIMENTO |
|----------------------------------|--------------|

INVESTIMENTO

| |
|-----------|
| 50.000,00 |
|-----------|

| PRINCIPAIS FASES | DATAS DE ENTREGAS | CUSTOS |
|-------------------------------------------------------------|-------------------|--------|
| Definição do projeto de desenvolvimento e Plano de software | 01/10/2015 | |
| Avaliação técnica | 10/10/2015 | |
| Levantamento | 10/10/2015 | |
| Análise e projeto | 10/11/2015 | |
| Implementação e testes | 01/01/2016 | |
| Instalação | 10/01/2016 | |
| Testes | 10/02/2016 | |

PRINCIPAIS STAKEHOLDERS: Mariana Kaji(Gestor de Vendas), Fernando Castro(Gestor de Tecnologia), Lucilene Gomes(Gestor Financeiro).

COMENTÁRIOS:

Fonte: a autora.

AVALIAÇÃO TÉCNICA

Ambiente técnico

| SGBD | MICROSOFT SQL SERVER |
|---------------------------|-------------------------------|
| Linguagem de programação | Java |
| Servidor de aplicação web | Sugerido JBoss |
| Sistema operacional | Multiplataforma |
| Serviço de email | Java Mail |
| Interface Usuário | DHTML (HTML, CSS, JavaScript) |
| Páginas ativas | JSP |
| Arquitetura | Web-Centric |
| Patterns | Model View Controller (MVC) |

Quadro 1: Avaliação técnica

Fonte: a autora.

Arquitetura lógica

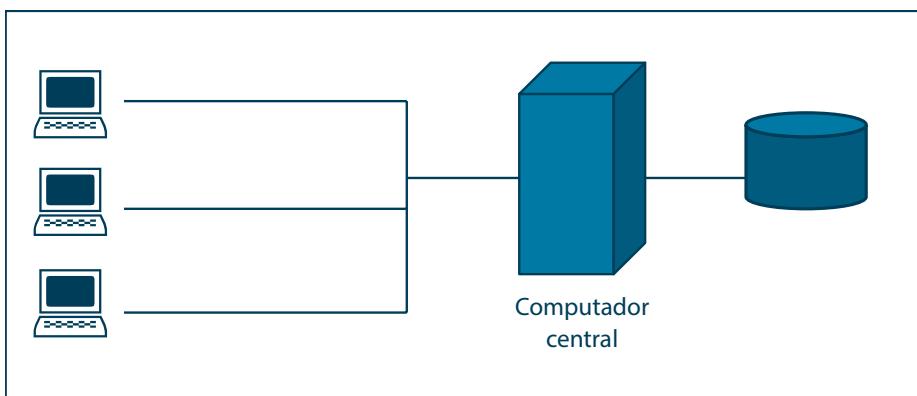


Figura 55: Esquema arquitetura do sistema

Fonte: adaptada de Deboni (2003, p. 73).

Deboni (2003, p. 74) relata as regras lógicas da arquitetura operante na loja:

O computador central armazena os dados dos produtos, dos clientes e do histórico de vendas em um banco de dados. As regras do negócio também são processadas no computador central. Os POS (acrônimo para Point of Sale – pontos de venda dentro da loja) implementam uma camada de apresentação e comunicação com o caixa e fazem as requisições ao computador central. Existe um pequeno poder de processamento local nos POS que pode ser utilizado, dependendo da aplicação e da estratégia de desenvolvimento escolhida.



REFLITA

"Existem duas maneiras de construir um projeto de software. Uma é fazê-lo tão simples que obviamente não há falhas. A outra é fazê-lo tão complicado que não existem falhas óbvias".

Fonte: C.A.R. Hoare.

LEVANTAMENTO

O levantamento de requisitos é o início da atividade de desenvolvimento de software, é o ir a campo e compreender o que o cliente deseja e como deverá ser desenhada a proposta para atender às expectativas dele. Nesse ponto, são aplicadas técnicas de levantamento de requisitos, como entrevistas, *brainstorms*, oficinas etc. Depois de levantados, os requisitos são analisados e processados pela equipe de desenvolvimento a fim de identificar incoerências, desacordos, repetições, para serem negociados junto ao cliente ou validados junto à equipe. Nessa etapa, também é produzida a matriz de rastreabilidade. Quando um projeto é desenvolvido por uma ferramenta CASE, a rastreabilidade é automática; nos casos da não utilização, é importante o seu desenvolvimento para manter uma documentação das conexões entre os requisitos e entre seus solicitantes. A rastreabilidade é utilizada nos processos de manutenções e de mudanças no software, auxilia a identificar o impacto que será causado na estrutura quando da implantação de novas funcionalidades ou alterações de uma já existente.

Nessa fase do projeto, casos de uso podem ser utilizados para auxiliar na definição dos relacionamentos entre os usuários e o sistema e também para a definição dos seus limites, aqui iniciamos a modelagem de contexto, considerando o ponto de vista externo. Para o nosso projeto, essa etapa foi realizada e dispomos dos artefatos resultantes dela: Documento de Requisitos, Modelo do contexto e Diagrama de caso de uso.

DOCUMENTO DE REQUISITO

DOCUMENTO DE REQUISITOS

| Histórico de Revisões | | | |
|-----------------------|--------|-------------------------|-------------------|
| Data | Versão | Descrição | Responsável |
| 01/01/2015 | 1.0 | Elaboração do documento | Guilherme Salgado |
| | | | |

1. OBJETIVO DO DOCUMENTO

Este documento tem por objetivo definir os requisitos funcionais e não funcionais do sistema, relacionar seus casos de uso, quando ocorrerem, junto com suas respectivas definições e servirá de base para a implementação, bem como servirá de acordo entre o cliente e a equipe de desenvolvimento. Qualquer necessidade de alteração nos itens aqui listados deverá ser registrada em documento específico para análise do impacto no escopo, no tempo e no financeiro do projeto.

2. ESCOPO GERAL

O processo de venda do cliente resume-se em três ações executadas pelo caixa por meio dos POS: (1) identificar o produto pelo seu código; (2) Identificar o crédito do cliente e seus dados pessoais pelo seu CNPJ; (3) Ao informar o número de parcelas, verificar se a venda parcelada foi aprovada. Todo cliente tem um crédito gerenciado pela loja e que pode ser elevado pelo gerente, conforme as novas compras efetuadas por ele. No entanto, o cliente não pode financiar compras acima do seu crédito. Se o saldo de uma compra parcelada for superior ao crédito do cliente, a venda não é aprovada. Para incentivar a venda para alguns clientes especiais identificados como Clientes VIP, a loja libera crédito dobrado, isto é, clientes VIP podem fazer dívidas duas vezes superiores ao valor do seu crédito. Produtos específicos podem entrar em Promoção e sofrer descontos ou condições especiais de venda a serem definidos sazonalmente (DEBOI, 2003, p. 75).

3. REQUISITOS

3.1 Requisitos funcionais

| RF | Descrição/Ação | Entrada | Saída |
|----|--------------------------------------------------------------|----------------------------------------------------------|------------------------|
| 01 | Acessar base de dados cliente | Código do cliente | Crédito do cliente |
| 02 | Processar compra do cliente | Código do produto/valor do produto/condição de pagamento | Saldo do cliente |
| 03 | Acessar base de dados produtos | Código do produto | Informações do produto |
| 04 | Identificar se o produto está sob condição especial de venda | Informações do produto | Condição especial |
| 05 | Aplicar regra da condição especial de venda | Condição especial | |
| 06 | Autorizar ou negar o parcelamento | Saldo do cliente | Autorização/Negação |

3.2 Requisitos não funcionais

| RF | Descrição/Ação |
|----|-----------------------------|
| 01 | Prover interface com os POS |

4. ESCOPO NÃO CONTEMPLADO

Não faz parte do escopo deste projeto:

- Alterações na interface padrão do sistema (atual).
- Geração de relatórios.

5. APROVAÇÃO

Mario Kitano

Diretor Comercial - TEMPRA TODOS

Guilherme Salgado

Gerente de Projeto

Modelo de contexto

Analisando a descrição do contexto do sistema, observa-se que o processo de venda envolve os seguintes elementos:

- O Cliente (VIP/Não VIP/Possui o Crédito/Não possui o crédito)
- O Caixa (processa o pedido – usuário final)
- POS (Interface com o sistema)
- A loja física (Regras de negócio)
- Produto (Preço/condição especial de venda)

Com base nessas informações, é possível definir o contexto do sistema, seus limites e o processo de venda. Sob a ótica da orientação a objetos, trata-se da definição dos objetos e da caracterização das mensagens que os objetos trocam entre si. O processo de venda então ocorre em três fases:

1. Identificação do produto
2. Identificação do cliente
3. Autorização do parcelamento

Os elementos envolvidos nesse processo são:

1. Caixa/POS
2. Produto
3. Cliente
4. Loja

A Figura 56 mostra uma abstração do esquema de comunicação entre os principais elementos da loja.

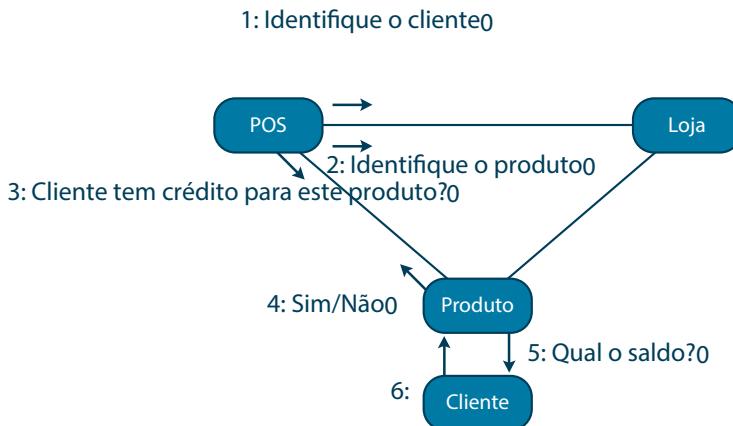


Figura 56: Comunicação entre os elementos da loja

Fonte: adaptada de Deboni (2003, p. 78).

Aplicando os conceitos de orientação a objeto, a primeira característica possível de se identificar na descrição do problema são os objetos. Na descrição e no esquema de comunicação, podemos observar os elementos utilizados, o POS, a Loja, o Cliente e o Produto. Vamos agora “encapsulá-los”!

No modelo, a Loja é uma entidade que se relaciona com os POS para prover informações para as vendas. O POS é uma interface de acesso às informações da Loja, e por isso pergunta a Loja o que precisa saber. Para obter as respostas necessárias (preço do Produto, ou crédito do Cliente), a Loja conta com outros objetos da própria Loja, uma listaDeClientes, ou uma listaDeProdutos. Quando questionada, a Loja busca na sua lista e devolve outro objeto oProduto. O mesmo ocorre quando for solicitado para a loja o Cliente, podemos identificar, portanto, duas classes: Loja e Cliente.

A lista de clientes e a lista de produtos da loja devem estar disponíveis para a pesquisa assim que o sistema inicia a operação. Para que isso seja possível, elas devem estar armazenadas em um banco de dados como Tabela de clientes e Tabela de Produtos. A Figura 57 demonstra o fluxo de dados.

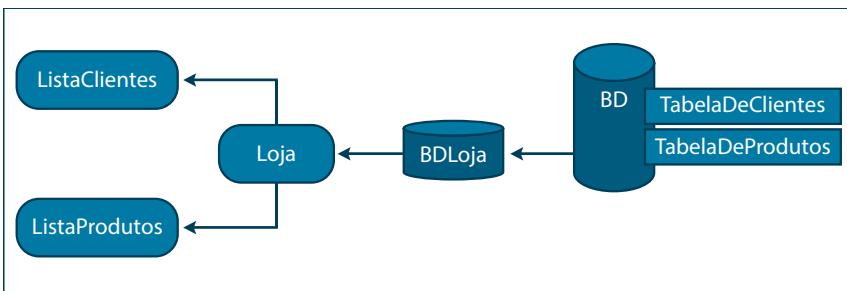


Figura 57: Fluxo de dados do BD

Fonte: adaptada de Deboni (2003, p. 82).



SAIBA MAIS

O Diagrama de classes é uma das técnicas mais utilizadas no desenvolvimento orientado a objetos. Ele descreve a estrutura dos objetos de um sistema e, para cada objeto, descreve a sua identidade, os seus relacionamentos com os outros objetos, os seus atributos e as suas operações. Por ser um diagrama de grande utilidade, esse artigo mostra o conceito dos elementos mais importantes e um estudo de caso detalhado, criado no Astah Professional.

O diagrama de classes é um dos diagramas mais utilizados da UML. Ele permite a visualização das classes que irão compor o sistema com seus respectivos atributos e métodos. Ele visa demonstrar como as classes do diagrama se relacionam e transmitem informações entre si. Ele é composto por suas classes e associações entre elas.

Leia mais em Trabalhando com os diagramas da UML – Parte 2, disponível em: <<http://www.devmedia.com.br/trabalhando-com-os-diagramas-da-uml-parte-2/33224#ixzz3mKtw7KDQ>>. Acesso em: 22 out. 2015.

Fonte: Trabalhando...(online).

Como estudamos, um objeto pode receber chamadas de mensagens de outros objetos por meio das funções. Uma mensagem pode levar com ela uma informação e receber outras como resposta. A interação entre o Caixa, que é uma pessoa física do mundo real, e o POS, que é um software, é realizada por troca de mensagens. A Figura 58 mostra um protótipo da interface entre o caixa e o POS.

| | | | |
|---------------|------|-----------------|--------------------|
| Código | 101 | PRODUTO | Guiana Solz Fender |
| CGC | 1020 | CLIENTE | Star Betz |
| N. Protocolo | 2 | PARCELAS | Aprovada |
| Limpar | | | |

Figura 58: Protótipo Interface entre caixa (usuário) e POS

Devemos considerar ainda da regra de negócios a característica do cliente poder ser VIP. Cabe aqui outro princípio da orientação a objeto, a herança. Um cliente VIP possui todas as propriedades de um cliente normal, mas em um determinado momento ele se especializa para um cliente VIP.

Analisando o escopo, percebemos que o sistema deverá estar integrado com outros sistemas de informação necessários para a realização das funções as quais está designado a desempenhar. Modelaremos cada um desses sistemas como pacotes que participarão na análise como subsistemas. A Figura 59 mostra a modelagem dos subsistemas relacionados ao sistema da loja e, na sequência, o diagrama do caso de uso ilustrado pela Figura 60.

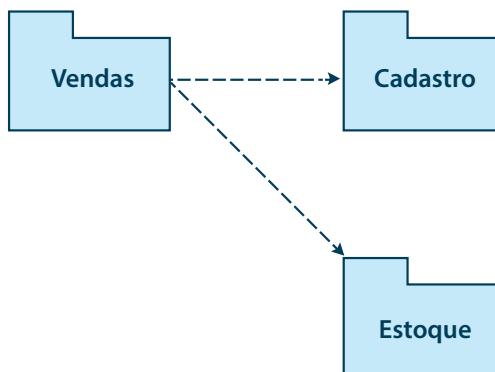


Figura 59: Subsistemas relacionados ao sistema da loja

Fonte: adaptada de Deboni (2003, p. 119).

DIAGRAMA DE CASO DE USO

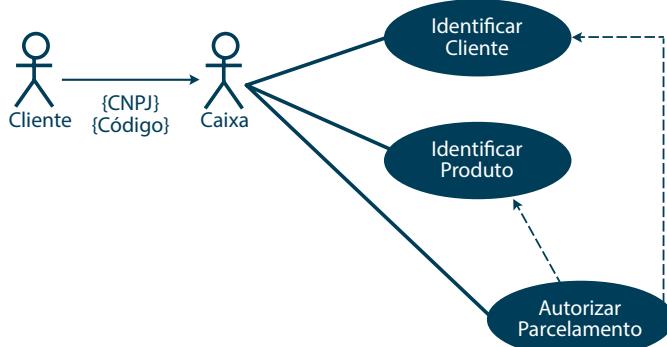


Figura 60: Diagrama de caso de uso dos processos de venda - autorizar parcelamento

Fonte: adaptada de Deboni (2003, p. 119).

Descrição do caso de uso:

1. **Identificar Cliente:** o caixa recebe o número do CNPJ do cliente. Caso o cliente exista no cadastro da loja, verificam-se os dados do cliente. Nome e Saldo de crédito.
2. **Identificar Produto:** o caixa faz a leitura do código do produto com leitor de código de barras ou digitando o código diretamente no console. Verifica dados do produto.
3. **Autorizar pagamento:** com as informações do cliente e do produto, a venda será realizada. Na compra parcelada, se o valor total do produto mais a dívida do cliente for menor que o saldo de crédito do cliente cadastrado na loja.

Esse é o primeiro marco do projeto (a primeira Iteração). Todos os artefatos gerados – Comunicação entre os elementos da loja; Fluxo de dados do BD; Protótipo Interface entre caixa (usuário) e POS; Subsistemas relacionados ao sistema da loja e Diagrama de caso de uso dos processos de venda - autorizar parcelamento – devem ser apresentados ao cliente em uma reunião formal na qual será verificado se o que foi especificado está de acordo com as suas expectativas.

Finalizando essa etapa do levantamento, onde definimos todos os requisitos, os principais elementos e os limites do sistema, passamos o foco para a definição da estrutura e comportamento.

Análise e projeto

Com base nas informações levantadas, o modelo conceitual é representado pela Figura 61.

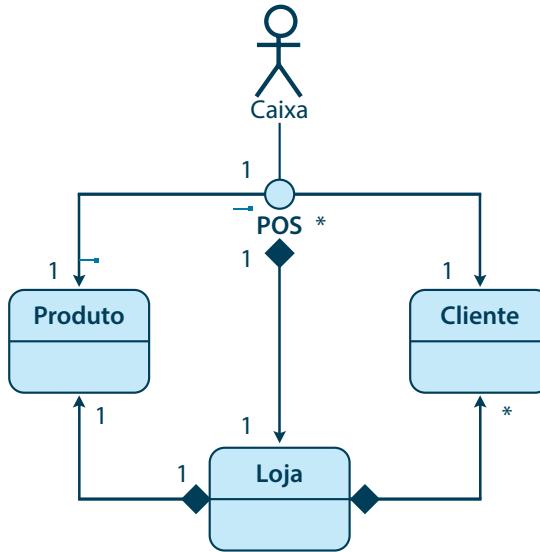


Figura 61: Diagrama conceitual

Fonte: adaptada de Deboni (2003, p. 276).

A partir do modelo conceitual é possível simular os processos dinâmicos envolvidos pelos objetos do sistema e distribuir as operações pelas classes. O diagrama de sequência descreve a série de mensagens que são trocadas entre os objetos do sistema.

O Diagrama de Sequência do processo autorização de vendas está modelado na Figura 62, e o último passo da análise, o Diagrama de classes, está ilustrado pela Figura 63.

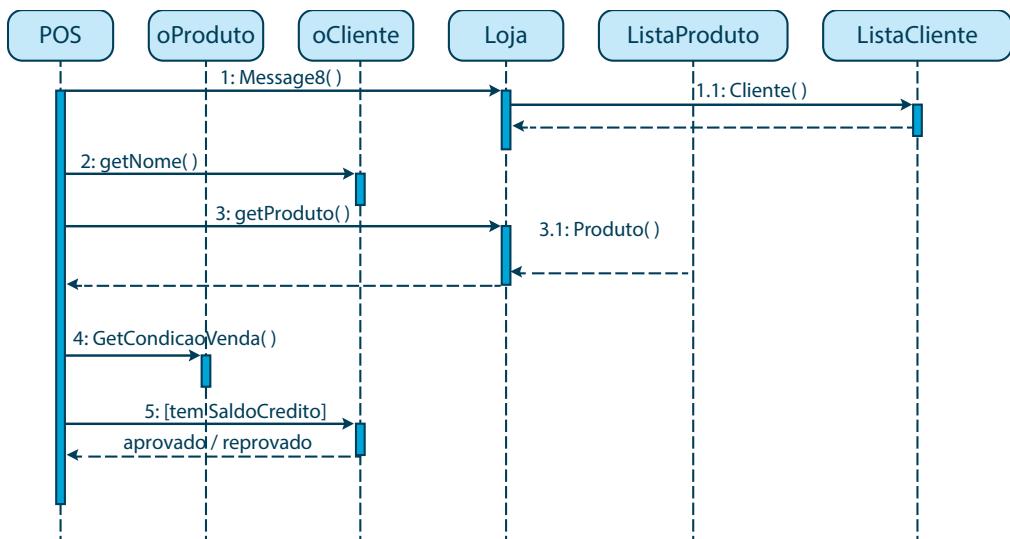


Figura 62: Diagrama de Sequência do processo autorização de vendas

Fonte: adaptada de Deboni (2013, p. 279).

Analisando as informações, vamos agora compor as classes para a modelagem da estrutura interna do sistema, estes são os principais pontos da análise e do projeto de software: a definição das classes e de seus atributos e métodos. Nesse ponto, já se tem uma ideia clara do sistema e da regra de negócio.

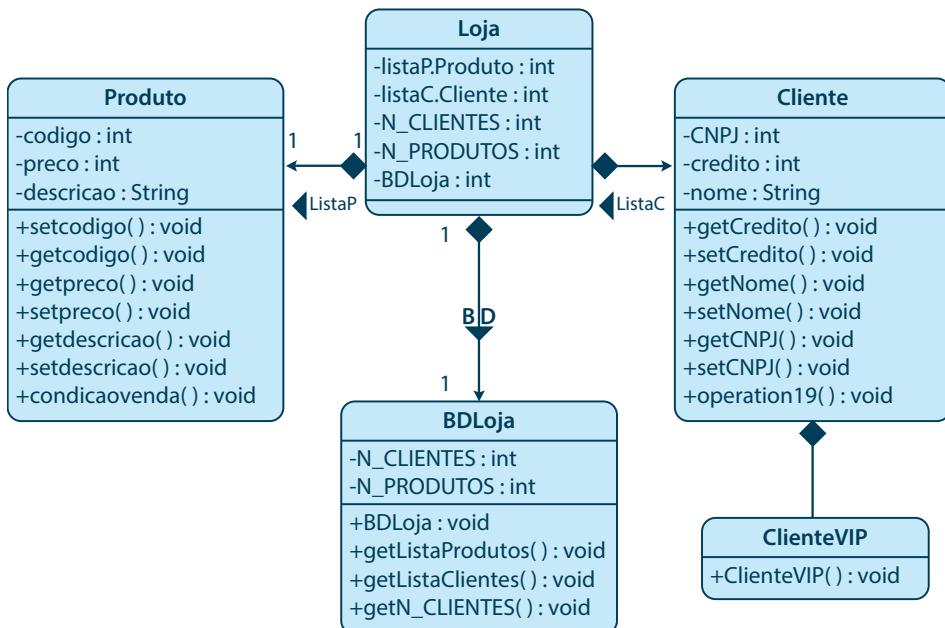


Figura 63: Diagrama de classes

Fonte: adaptada de Deboni (2003, p. 283).

Finalizamos nosso processo de modelagem nesse ponto. A partir daqui seguem as etapas de implementação (programação), implantação e testes, que são assuntos para outras disciplinas!

CONSIDERAÇÕES FINAIS

Enfim, chegamos ao final de nossa disciplina, nesta última unidade, aplicamos todos os conceitos aprendidos na prática. No meu ponto de vista, é válido afirmar que uma das principais metas da modelagem é a construção do diagrama de classes, pois ali estão contidas todas as informações necessárias para a programação pôr a mão na massa, para isso ser possível, é muito importante um detalhamento do processo eficiente. A engenharia de requisitos é fundamental para o processo de modelagem, pois ela capta as necessidades do cliente e as processa a fim de se produzirem resultados claros e precisos, as ferramentas de modelagem podem e devem ser utilizadas também nessa etapa.

O processo de desenvolvimento deve ser feito por incrementos. Estudamos como adquirir os requisitos, isto é, como determinar as necessidades do cliente e caminhamos até a modelagem das classes. Como afirmei, o processo deve ser incremental; no final da primeira iteração, apresentamos ao cliente um mapa conceitual, composto pela visão geral do sistema do ponto de vista do cliente. Partindo da aprovação, do mapa conceitual, a fase de análise define os modelos e a equipe de programação deve priorizar alguns requisitos estratégicos e implementá-los. Com isso, uma primeira versão do sistema é produzida e já deve ser apresentada para o cliente em uma segunda iteração. E, assim por diante, até a entrega do produto final. Gerando esses pequenos pacotes, cria-se um envolvimento maior com o cliente e, consequentemente, o seu comprometimento com o projeto.

O estudo de caso utilizado foi retirado do livro “Modelagem orientada a objeto com técnicas de análise, documentação e projeto de sistema”, da editora Futura. Caso tenha interesse em se aprofundar no assunto, o autor apresenta mais detalhes de implementação exemplificados em Java, os quais não foram apresentados aqui, pois nosso foco nessa disciplina é o processo de modelagem de software.

ATIVIDADES



1. Dentre os diagramas que auxiliam o projeto de software, um deles é muito útil para representar lógica comportamental, que é uma excelente ferramenta para modelagem de fluxos de trabalho e de processos. A notação em questão é o diagrama de:
 - a) Máquina de estados.
 - b) Atividades.
 - c) Classes.
 - d) Estruturas compostas.
 - e) Implementação.

2. Sobre a UML, identifique os itens corretos e, na sequência, assinale a alternativa que os representa.
 - I. A multiplicidade pode ocorrer tanto na associação entre duas classes, no diagrama de classes, quanto na associação entre ator e caso de uso, no diagrama de caso de uso.
 - II. Caso seja necessário modelar visões de um conjunto de instâncias que cooperam entre si para executar uma função específica, o diagrama de interação geral será mais adequado que o diagrama de estrutura composta.
 - III. A UML é usada para facilitar a compreensão de aspectos complexos inerentes aos softwares e oferece um conjunto de notações gráficas e diagramas que ajudam na descrição e captura de diferentes visões de um software.
 - a) Somente a afirmativa I está correta.
 - b) Somente a afirmativa II está correta.
 - c) Somente a afirmativa III está correta.
 - d) Somente as afirmativas I e II estão corretas.
 - e) Todas as afirmativas estão corretas.

ATIVIDADES



3. O diagrama que é utilizado pela UML para modelar as interações funcionais entre os usuários e o sistema é denominado de:
- a) Componentes.
 - b) Pacotes.
 - c) Implantação
 - d) Caso de uso.
 - e) Interação.
4. O diagrama de Atividade da UML tem como finalidade apresentar o sistema por meio da visão que modela os fluxos de controle. Essa visão possui uma característica:
- a) Estática.
 - b) Dinâmica.
 - c) Particionada.
 - d) Imparcial.
 - e) Parcial
5. Assinale a alternativa que apresenta o diagrama UML que mostra um conjunto de classes e seus relacionamentos.
- a) Diagrama de Estados.
 - b) Diagrama de Interação.
 - c) Diagrama de Casos de Uso.
 - d) Diagrama de Colaboração.
 - e) Diagrama de Classes.



PROJETO DE SOFTWARE UTILIZANDO UML

Muito se fala sobre a curva de aprendizado associada à orientação a objetos. Essa curva de aprendizado é fruto da necessidade e troca de paradigma ao deixar de projetar e desenvolver estruturado para fazê-lo seguindo os princípios da orientação a objetos. Em alguns aspectos, a mudança pode ser até fácil. Aqueles que não trabalharam por muito tempo com o paradigma estruturado tendem a ter mais facilidades. Vale deixar claro aqui que não estamos questionando o quanto difícil é programar em uma linguagem orientada a objetos, mas sim aprender a tirar proveito das vantagens que essas linguagens fornecem.

Nesse contexto, situamos a UML, Linguagem de Modelagem Unificada. Até certo ponto, podemos dizer que ela foi projetada para ajudar as pessoas a focarem as vantagens provenientes do uso do paradigma orientado a objetos. UML é utilizada para visualizar, especificar, construir e documentar artefatos de software. Veremos agora o que significa cada um desses contextos de utilização.

- Visualizar: para muitos programadores, a distância entre pensar em uma solução para o problema e transformá-la em código é próxima de zero. Ele cria a solução e ele mesmo a desenvolve. Ainda assim, ele, de alguma forma, está modelando mentalmente o sistema que irá construir. Entretanto, existem sérios problemas com essa abordagem. Primeiro, comunicar o modelo criado mentalmente para outros desenvolvedores é uma tarefa cujo risco de perda de informação durante a comunicação é alto. E, segundo, imagine que o projeto em questão é grande e a equipe envolvida não se restringe a um ou dois programadores. Teríamos sérias dificuldades na construção do sistema. Isso sem falar que não existiria documentação para o software e sua manutenção no futuro traria dor de cabeça, com certeza. Assim, o uso da UML provê uma notação comum para o entendimento compartilhado sobre o software que se está construindo.
- Especificar: a UML permite a construção de modelos precisos, não ambíguos e completos.
- Construir: os modelos construídos utilizando a UML podem ser conectados a uma série de linguagens de programação, permitindo uma tradução entre os modelos construídos e o código. Esse mapeamento permite também a engenharia reversa na qual os modelos são gerados a partir do código fonte. Vale uma ressalva aqui, UML não é uma linguagem visual de programação.
- Documentar: neste caso, os modelos criados durante o desenvolvimento fazem parte da documentação do software.

Fonte: Spinola (2007, online)



MATERIAL COMPLEMENTAR



NA WEB

A página oficial do projeto ArgoUML, mantida pela Tigris.org, oferece um conjunto de tutoriais e exemplos que podem ser acessados por qualquer interessado em saber mais sobre a modelagem de software e a linguagem UML®. Ali, é possível também fazer o download das realeseas da ferramenta e se manter informado sobre as novidades de cada versão.

Aprofunde-se no assunto, acessando a página da Tigris.org pelo endereço disponível em: <<http://argouml.tigris.org/tours/>>. Acesso em: 22 out. 2015.



CONCLUSÃO

Chegamos ao final de mais uma etapa!

Não tenho preocupação em me equivocar ao afirmar que desenvolver software é uma atividade complexa e que grande parte dessa complexidade é inherente a sua própria natureza. Softwares são desenvolvidos por pessoas e exigem, para o funcionamento, um grande número de “estados” possíveis, além disso, estão sujeitos às pressões constantes por mudanças, seja por solicitação do cliente ou por exigência da própria evolução tecnológica. Nesse sentido, percebemos a importância do projeto para o desenvolvimento do software, e a modelagem é parte essencial, senão fundamental, para qualquer projeto de software.

Vimos na unidade I os principais conceitos referentes à atividade de modelagem de software e observamos que a modelagem deve estar relacionada com o processo de engenharia de requisitos, garantindo uma ponte entre as etapas de definição do sistema e projeto, que representam as atividades mais importantes do processo de desenvolvimento de software.

A unidade II propôs a análise dos principais tipos de modelos utilizados nos projetos de desenvolvimento de software. Percebemos que diferentes tipos de modelos são necessários porque existem várias formas de se observar o problema em desenvolvimento; o conjunto de visões tem o objetivo de propiciar formas diferentes de se observar a mesma coisa, sempre considerando a ótica do interessado ou da necessidade naquele momento.

A orientação a objetos está intimamente ligada ao desenvolvimento de software, por isso estudamos também os principais conceitos desse paradigma na unidade III, juntamente com a apresentação da linguagem UML®, em que observamos suas especificações e diagramas utilizados na modelagem de software.

A utilização de ferramentas automatizadas garante inúmeros benefícios, como aumento na qualidade do produto final, aumento da produtividade, redução das atividades de programação, agilidade no retrabalho, diminuição da manutenção e do esforço para isso e redução de custos. O objetivo da unidade IV foi pontuar as vantagens da utilização de ferramentas CASE para a modelagem de software e apresentar algumas delas.

Encerramos nossos estudos aplicando todos os conceitos em um estudo de caso. O estudo de caso utilizado foi retirado do livro Modelagem orientada a objeto com técnicas de análise, documentação e projeto de sistema, da editora Futura, e nos garantiu uma visão prática dos conceitos apresentados.



REFERÊNCIAS

- ATALLA, F. Por que os Projetos de TI Fracassam? **Instituto de Educação Tecnológica** [online]. Disponível em: <http://www.techoje.com.br/site/techoje/categoria/detalhe_artigo/1532>. Acesso em: 04 set. 2015.
- BARCELAR, R. R. Modelagem de Sistemas Orientada a Objetos com UML. Disponível em: <http://ricardobarcelar.com.br/aulas/eng_sw/mod3-uml.pdf>. Acesso em: 22 out. 2015.
- BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. RJ: Elsevier, 2014.
- BRAZ JUNIOR, G. **Estudo de Caso:** Sistema de Caixa Automático. Departamento de Informática. UFMA. Disponível em: <<http://www.deinf.ufma.br/~geraldob/14Es-tudoCaso.pdf>>. Acesso em: 23 out. 2015.
- CAELUM. Apostila Java e orientação a objetos. Capítulo 10 – Interfaces. Disponível em: <<http://www.caelum.com.br/apostila-java-orientacao-objetos/interfaces/>>. Acesso em: 15 set. 2015.
- CARDOSO, C. **UML na prática:** do problema ao sistema. Rio de Janeiro: Ciência Moderna Ltda., 2003.
- CELEPAR INFORMÁTICA DO PARANÁ. Guia para Modelagem de Interações Metodologia CELEPAR. 2009. Disponível em: <<file:///C:/Users/Usuario/Documents/UNICESUMAR/Modelagem%20de%20Software/Material%20de%20apoio%20MS/guia-ModelagemInteracoes.pdf>>. Acesso em: 06 set. 2015.
- CORREIA, C.; TAFNER, M. **Análise orientada a objetos**. 2. ed. Florianópolis: Visual Books, 2006.
- COSTA, C. A. A aplicação da Linguagem de Modelagem Unificada (UML) para o suporte ao projeto de sistemas computacionais dentro de um modelo de referência. **Gestão e Produção**, Departamento de Engenharia Mecânica, Universidade de Caxias do Sul (UCS), v. 8, n. 1, p. 19-36, abr. 2001.
- DEBONI, J. E. Z. **Modelagem orientada a objeto com UML**. Técnicas de análise, documentação e projeto de sistema. São Paulo: Futura, 2003.
- DIAGRAMA de pacotes. UFPR. Disponível em: <<http://www.inf.ufpr.br/silvia/ES/UML/Diagramadepacotes.pdf>>. Acesso em: 23 out. 2015.
- ENGHOLM JUNIOR, H. **Engenharia de software na prática**. São Paulo: Novatec, 2010.
- ESPÍNDOLA, E. C. A importância da modelagem de objetos no desenvolvimento de sistemas. **Linha de Código**. Disponível em: <<http://www.linhadecodigo.com.br/artigo/1293/a-importancia-do-modelagem-de-objetos-no-desenvolvimento-de-sistemas.aspx#ixzz3kn4dKfnA>>. Acesso em: 04 set. 2015.

REFERÊNCIAS

- FERRAMENTAS para Modelagem. **Portal Educação**. 2013. Disponível em: <<https://www.portaleducacao.com.br/informatica/artigos/27187/ferramentas-para-modelagem>>. Acesso em: 14 set. 2015.
- GASPAROTTO, H. M. Os 4 pilares da Programação Orientada a Objetos. **Devmedia**. 2014. Disponível em: <<http://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264#ixzz3lxbe0htF>>. Acesso em: 16 set. 2015.
- IBM Knowledge Center. Rational Software Architect RealTime Edition 9.1.2. Disponível em: <http://www-01.ibm.com/support/knowledgecenter/SS5JSH_9.1.2/com.ibm.xtools.modeler.doc/topics/cinterfc.html?cp=SS5JSH_9.1.2%2F7-3-1-6-3-0&lang=pt-br>. Acesso em: 16 set. 2015.
- JUNGTHON, G.; GOULART, C. M. **Paradigmas de Programação**. Artigo Publicado pela Faculdade de Informática de Taquara (FIT) – Faculdades Integradas de Taquara - FACCAT. Taquara/RS. Disponível em: <https://fit.faccat.br/~guto/artigos/Artigo_Paradigmas_de_Programacao.pdf>. Acesso em: 15 set. 2015.
- MACHADO, G. M.; OLIVEIRA, J. P. M. de. Modelos de contexto para sistemas adaptativos baseados em modelos de usuário e localização. **Cadernos de Informática**, v. 7, n. 1 (2013). Disponível em: <<http://seer.ufrgs.br/cadernosdeinformatica/article/view/v7n1p1-13>>. Acesso em: 05 set. 2015.
- MACORATTI, J. C. Programação Orientada a Objetos em 10 lições práticas – Parte 07. **Imasters**, 2014. Disponível em: <<http://imasters.com.br/desenvolvimento/visual-basic/programacao-orientada-objetos-em-10-licoes-praticas-parte-07/>>. Acesso em: 23 out. 2015.
- MODELAR. In: Dicionário Michaelis online. Disponível em: <<http://michaelis.uol.com.br/moderno/portugues/index.php?lingua=portugues=portugues&palavra=modelar>>. Acesso em: 07 out. 2015.
- MODELAR. In: Dicionário online de português. Disponível em: <<http://www.dicio.com.br/modelar/>>. Acesso em: 07 out. 2015.
- MORAIS, L. Modelagem em uma Visão Ágil. **Revista Engenharia de Software**, n. 32. DevMedia. Rio de Janeiro, 2011. Disponível em: <http://www.devmedia.com.br/websys.5/webreader.asp?cat=48&artigo=3211&revista=esmagazine_32#a-3211>. Acesso em: 04 set. 2015.
- NAGAO, F. Programação orientada a eventos e lambda function em ASP/VBScript. **Revista Imaster**, São Paulo, 2009. Disponível em: <http://imasters.com.br/artigo/11514/asp/programacao_orientada_a_eventos_e_lambda_function_em_asp-vbscript/>. Acesso em: 05 set. 2015.
- NELSON, R. R. Project retrospectives: Evaluating project success, failure, and everything in between. **MIS Quarterly Executive**, [S.I.], v. 4, n. 3, p. 361-372, set. 2005. Disponível em: <<http://misqe.org/ojs2/index.php/misqe/article/view/85>>. Acesso em: 04 set. 2015.



REFERÊNCIAS

- NUNES, M.; O'NEILL, H. **Fundamental de UML**. 7. ed. Lisboa: Editora FCA, 2000.
- OBJECTS BY DESIGN. UML Products by Product. Disponível em: <http://www.objectsbydesign.com/tools/umltools_byProduct.html>. Acesso em: 18 set. 2015.
- OLIVEIRA, F. G. de; SEABRA, J. M. P. Metodologias de desenvolvimento de software: uma análise no desenvolvimento de sistemas na web. **Periódico Científico**, v. 6, n. 1, 2015. Disponível em: <<http://revista.faculdadeprojecao.edu.br/index.php/Projecao4/article/view/497>>. Acesso em: 18 set. 2015.
- OLIVEIRA, J. A. de; SOUZA, P. P. de; FIGUEIREDO, E. **Uma Avaliação de Ferramentas de Modelagem de Software**. Laboratório de Engenharia de Software (LabSoft) - Universidade Federal de Minas Gerais (UFMG), Belo Horizonte-MG . Disponível em: <<http://labsoft.dcc.ufmg.br/lib/exe/fetch.php?media=smes.pdf>>. Acesso em: 18 set. 2015.
- OLIVEIRA, J. de. 12 reflexões que vão te introduzir ao pensamento de Carl Sagan. **Revista Galileu** [online]. Disponível em: <<http://revistagalileu.globo.com/Ciencia/Espaco/noticia/2015/03/12-reflexoes-que-vao-te-introduzir-ao-pensamento-de-carl-sagan.html>>. Acesso em: 23 out. 2015.
- OMG Unified Modeling Language™ (OMG UML). Version 2.5. OMG Document Number formal/2015-03-0. Disponível em: <<http://www.omg.org/spec/UML/2.5>>. Acesso em: 15 set. 2015.
- PARADIGMA. In: Dicionário Michaelis online. Disponível em: <<http://michaelis.uol.com.br/moderno/portugues/index.php?lingua=portugues=-portugues&palavra-paradigma>>. Acesso em: 23 out. 2015.
- PIMENTEL, A. R. **Projeto de Software Usando a UML** (Apostila). Universidade Federal do Paraná. Disponível em: <<http://www.inf.ufpr.br/andrey/ci221/apostilaUml.pdf>>. Acesso em: 15 set. 2015.
- PRESMAN, R. S. **Engenharia de Software**. Porto Alegre: McGrawHill, 2010.
- RODRIGUES, A. F. **Um guia para a criação de modelos de desenho de software no Praxis Synergia**. 2007. 140 f. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal de Minas Gerais. Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, Belo Horizonte. Disponível em: <<https://www.dcc.ufmg.br/pos/cursos/defesas/871M.PDF>>. Acesso em: 29 ago. 2015.
- SILVA, C. F. da. **Construção e Realidade nas Imagens dos Livros Didáticos de Física: implicações e aplicações no ensino da física**. 2008. 118 f. Dissertação (Mestrado em Ensino de Física) – Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte. Disponível em: <http://www.biblioteca.pucminas.br/teses/EnCiMat_SilvaCF_1.pdf>. Acesso em: 03 set. 2015.

REFERÊNCIAS

- SILVA, P. C. B. da. Utilizando UML: Diagrama de Atividade. SQL Magazine, 66. Disponível em:<<http://www.devmedia.com.br/artigo-sql-magazine-66-utilizando-uml-diagrama-de-atividade/13577>>. Acesso em: 23 out. 2015.
- SOMMERVILLE, I. **Engenharia de Software**. São Paulo: Pearson Prentice Hall, 2011.
- SOUZA, D. Introdução ao IBM Rational Software Architect. **Devmedia**, 2012. Disponível em: <<http://www.devmedia.com.br/introducao-ao-ibm-rational-software-architect/26748>>. Acesso em: 18 set. 2015.
- SPARX SYSTEMS. Creating Strategic Models with Enterprise Architect. 2010. Disponível em: <http://www.sparxsystems.com.au/downloads/quick/strategic_modeling_with_enterprise_architect.pdf>. Acesso em: 23 out. 2015.
- SPINOLA, R. O. Projeto de software utilizando UML. **SQL Magazine**: 2007. Versão online disponível em: <<http://www.devmedia.com.br/artigo-sql-magazine-13-projeto-de-software-utilizando-uml/5640#ixzz3mLRWZXYz>>. Acesso em: 19 set. 2015.
- THE UNIFIED Modeling Language. Uml-diagrams. Disponível em: <<http://www.uml-diagrams.org/>>. Acesso em: 18 set. 2015.
- TUCKER, A. B.; NOONAN, R. E. **Linguagens de Programação: Princípios e Paradigmas**. São Paulo: MacGraw Hill, 2008.
- UFCG. Exemplo. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/uml/diagramas/classes/images/especificacao_exemplo_banco.GIF>. Acesso em: 23 out. 2015.
- UFCG. Diagrama de Atividades. UML. Disponível em: <http://www.dsc.ufcg.edu.br/~sampaio/cursos/2007.1/Graduacao/SI-II/Uml/diagramas/atividades/diag_atividades.htm>.
- UML – Linguagem de Modelagem Unificada. Disponível em: <<http://pt.slideshare.net/Ridlo/uml-1858376>>. Acesso em: 23 out. 2015.
- UML – Linguagem de Modelagem Unificada. ETELG. Disponível em: <http://www.etelg.com.br/paginaete/downloads/informatica/apostila_uml.pdf>. Acesso em: 23 out. 2015.
- VALLE, F. A. Ferramentas CASE e qualidade dos dados: O paradigma da boa modelagem. **Devmedia**, 2007. Disponível em: <<http://www.devmedia.com.br/ferramentas-case-e-qualidade-dos-dados-o-paradigma-da-boa-modelagem/6905#ixzz3mDrVBv4DevMedia>>. Acesso em: 15 set. 2015.
- VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. **Modelos e processos para o desenvolvimento de sistemas sensíveis ao contexto**. Biblioteca Digital da Universidade Federal da Bahia. Disponível em: <http://homes.dcc.ufba.br/~vaninha/context/2009_TextoJAI_Final.pdf>. Acesso em: 03 set. 2015.



GABARITO

UNIDADE I

1. Modelagem de software é a atividade de construir modelos que representam a estrutura e o comportamento de um software.
2. - Descrever o que o cliente exige; - Estabelecer a base para a criação de um projeto de software; - Definir um conjunto de requisitos que possam ser validados quando o software for construído.
3. É a atividade de ignorar alguns detalhes, dando foco nos aspectos essenciais para uma determinada visão ou análise.

UNIDADE II

1. O cliente é uma peça importante no desenvolvimento de software, pois é ele quem irá usar o novo sistema e ninguém melhor que ele para saber os requisitos do software, mesmo que não entenda de engenharia de software. Com o cliente participando ativamente do desenvolvimento, fica mais fácil se por acaso os requisitos mudarem, e a fase de treinamento ficará mais curta, já que o cliente já sabe como o software funcionará.
2. Software farmacêutico, Software de caixa eletrônico. Visão Interação - Casos de Uso: conjunto de interações entre o sistema e os agentes externos. Visão estrutural - Projeto: visa construir um sistema que atenta os Casos de Uso.
3. B
4. A

UNIDADE III

1. Generalização é usar uma ideia específica para representar um espectro mais amplo de ideias. Por exemplo, mamífero é um conceito específico que representa uma classe bem maior de definições, já que existem inúmeras espécies de mamíferos. Especialização é especificar um caso geral de maneira mais detalhada, até chegar ao conceito mais restrito. Por exemplo, os cachorros são mamíferos.
2. C
3. D
4. B
5. A
6. C
7. B

GABARITO

UNIDADE IV

1. D
2. D
3. B

UNIDADE V

6. B
7. C
8. D
9. B
10. E