

PROJETO DE SISTEMAS

**ENSINO A
DISTÂNCIA**



**Ficha catalográfica elaborada na fonte pela Biblioteca do
Centro Universitário Avantis - UNIAVAN
Maria Helena Mafioletti Sampaio. CRB 14 – 276**

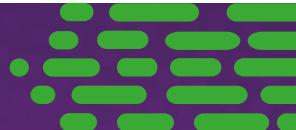
Bombasar, James Roberto.
B699p Projeto de sistemas. /EAD/ [Caderno pedagógico].
James Roberto Bombasar. Balneário Camboriú: Faculdade
Avantis, 2020.
109 p. il.

Inclui Índice
ISBN: 978-65-87252-26-1
ISBN: 978-65-87252-25-4

1. Ciências da computação – Projeto de Sistemas. 2.
Técnicas de Modelagem-UML – Aplicação. 3. Ferramentas
CASE. 4. Métricas de software. 5. Pontos de função –
Princípios. 3. Projeto de Sistema - Ensino a Distância. I.
Centro Universitário Avantis - UNIAVAN. II. Título.

CDD 21ª ed.
004.21 – Ciências da computação – Projeto de Sistemas.

PLANO DE ESTUDOS



EMENTA

Análise e Projeto de Sistemas: Fundamentos e Implementação. Aplicação de Técnicas de Modelagem – UML: Fundamentos, Visão e Diagramas. Ferramentas CASE. Princípios de Pontos de Função. Métricas de Software.

OBJETIVOS DA DISCIPLINA

- Conhecer as principais atividades que compõem as etapas de análise e projeto de sistemas;
- Adquirir habilidades para analisar e projetar sistemas em diferentes domínios de aplicação;
- Inteirar-se da Linguagem de Modelagem Unificada - UML;
- Compreender a visão proporcionada por cada um dos diagramas UML;
- Habilitar-se para a construção de diagramas UML;
- Aprender a definição e classificação das ferramentas CASE;
- Assimilar a importância das ferramentas CASE para o processo de desenvolvimento de sistemas;
- Estar apto para seleção de ferramentas CASE em diferentes cenários de desenvolvimento de sistemas;
- Entender o método de análise de pontos de função;
- Aplicar o método de análise de pontos de função para a medição do tamanho funcional de sistemas.

O PAPEL DA DISCIPLINA PARA A FORMAÇÃO DO ACADÊMICO

No processo de desenvolvimento de qualquer sistema, a análise e o projeto são atividades fundamentais para a entrega de um produto de qualidade. A disciplina de Projeto de Sistemas explora um arcabouço de métodos, técnicas e ferramentas na realização destas duas importantes atividades. Assim, o Caderno de Estudos de Projeto de Sistemas pretende ser seu aliado no desenvolvimento de habilidades e competências, necessárias para analisar e projetar sistemas, em diferentes domínios de aplicação, de maneira eficiente e eficaz.

Você terá a oportunidade de conhecer as principais atividades que compõem a análise e o projeto de um sistema, incluindo o levantamento de requisitos e o design dos diversos aspectos dele, tais como sua arquitetura e sua interface.

O papel do Analista de Desenvolvimento de Sistemas é tão importante quanto de qualquer outro profissional da área, pois é ele o principal responsável por garantir que o projeto de um sistema reflita exatamente no que os clientes e usuários esperam. Por isso, leve o estudo de nosso caderno a sério.

Bons estudos!

PROFESSOR



APRESENTAÇÃO DO AUTOR

James Roberto Bombasar

Olá! Meu nome é James Roberto Bombasar, mas sou também conhecido como Beto, devido ao meu segundo nome. Sou bacharel em Sistemas de Informação, Especialista em Ensino a Distância e Mestre em Computação Aplicada.

Já atuei como programador, conteudista e analista de sistemas. Atualmente, trabalho no Centro Universitário Avantis (UNIAVAN), como docente no Ensino Superior.

Estou muito grato em poder compartilhar com você os conhecimentos que adquiri, ao longo de minha trajetória acadêmica e profissional.

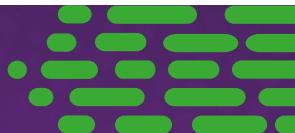
Pronto para começar?



Curriculum Lattes: <http://lattes.cnpq.br/4943270066505311>

E-mail: james.bombasar@uniavan.edu.br

SUMÁRIO



UNIDADE 1 - ANÁLISE E PROJETOS DE SISTEMAS: FUNDAMENTOS E IMPLEMENTAÇÃO	11
INTRODUÇÃO À UNIDADE.....	12
1 ANÁLISE E PROJETOS DE SISTEMAS: FUNDAMENTOS E IMPLEMENTAÇÃO	12
1.1 FASE DE PLANEJAMENTO	13
1.1.1 Análise de viabilidade	14
1.1.2 Seleção do modelo de processo de desenvolvimento	16
1.2 FASE DE ANÁLISE.....	18
1.2.1 Determinação dos requisitos.....	19
1.2.2 Análise de casos de uso.....	23
1.2.3 Modelagem de dados.....	27
1.3 FASE DE PROJETO	29
1.3.1 Design de arquitetura	30
1.3.2 Design de interface com o usuário	32
1.3.3 Design de armazenamento de dados	34
1.4 FÓRUM	36
CONSIDERAÇÕES FINAIS	37
EXERCÍCIO FINAL	38
REFERÊNCIAS.....	40
UNIDADE 2 - APLICAÇÃO DE TÉCNICAS DE MODELAGEM – UML: FUNDA- MENTOS, VISÃO E DIAGRAMAS.....	41
INTRODUÇÃO À UNIDADE.....	42

2 APLICAÇÃO DE TÉCNICAS DE MODELAGEM - UML: FUNDAMENTOS, VISÃO E DIAGRAMAS	42
2.1 BLOCOS DE CONSTRUÇÃO DA UML.....	43
2.1.1 Itens	44
2.1.2 Relacionamentos.....	45
2.1.3 Diagramas	46
2.2 VISÕES DA UML.....	47
2.3 DIAGRAMAS ESTRUTURAIS.....	48
2.3.1 Diagrama de classes	48
2.3.2 Diagrama de estrutura composta	49
2.3.3 Diagrama de objetos.....	50
2.3.4 Diagrama de pacotes.....	51
2.3.5 Diagrama de implantação	52
2.3.6 Diagrama de perfil	53
2.3.7 Diagrama de componentes	53
2.4 DIAGRAMAS COMPORTAMENTAIS.....	54
2.4.1 Diagrama de casos de uso	55
2.4.2 Diagrama de atividades.....	56
2.4.3 Diagrama de máquina de estados.....	57
2.4.4 Diagrama de sequência.....	57
2.4.5 Diagrama de comunicação.....	59
2.4.6 Diagrama de visão geral de interação.....	60
2.4.7 Diagrama de tempo	61
2.5 FÓRUM	62
CONSIDERAÇÕES FINAIS	63
EXERCÍCIO FINAL	64
REFERÊNCIAS.....	66

UNIDADE 3 - FERRAMENTAS CASE.....	67
INTRODUÇÃO À UNIDADE.....	68
3 FERRAMENTAS CASE.....	68
3.1 HISTÓRICO.....	69
3.2 ARQUITETURA GENÉRICA.....	70
3.3 CLASSIFICAÇÃO.....	71
3.3.1 Perspectiva de processo.....	71
3.3.2 Perspectiva de integração.....	72
3.3.3 Perspectiva funcional.....	73
3.4 PRINCIPAIS CARACTERÍSTICAS.....	75
3.5 VANTAGENS E PROBLEMAS DO USO DE FERRAMENTAS CASE	78
3.6 AVALIAÇÃO E SELEÇÃO DE FERRAMENTAS CASE.....	79
3.7 FÓRUM	81
CONSIDERAÇÕES FINAIS	83
EXERCÍCIO FINAL	84
REFERÊNCIAS.....	85

UNIDADE 4 - MÉTRICAS DE SOFTWARE, PRINCÍPIOS DE PONTOS DE FUNÇÃO.....	87
INTRODUÇÃO À UNIDADE.....	88
4 MÉTRICAS DE SOFTWARE	88
4.1 MEDIDAS, MÉTRICAS E INDICADORES.....	89
4.2 CLASSIFICAÇÕES DAS MÉTRICAS DE SOFTWARE.....	90
4.3 ANÁLISE DE PONTOS DE FUNÇÃO.....	95
4.3.1 Tipos de função	95
4.3.2 Parâmetros de complexidade e regras de contagem.....	97
4.3.3 Determinação da contribuição das funções	100
4.3.4 Cálculo do tamanho funcional.....	101

4.3.5 Ajuste da complexidade.....	104
4.4 FÓRUM	105
CONSIDERAÇÕES FINAIS.....	106
EXERCÍCIO FINAL	107
REFERÊNCIAS.....	109

unidade

1

ANÁLISE E PROJETOS
DE SISTEMAS:
FUNDAMENTOS E
IMPLEMENTAÇÃO

INTRODUÇÃO À UNIDADE

Conhecer as principais atividades que compõem as fases de análise e projeto de sistemas é o primeiro passo, para que você seja capaz de atuar como analista no processo de desenvolvimento de um sistema. Quando uso a expressão “principais atividades”, não quero dizer que as atividades a serem estudadas a partir de agora são essenciais em qualquer cenário de desenvolvimento de sistema, mas sim que elas representam um rol clássico de atividades, o qual pode ser adaptado, conforme as necessidades de cada projeto.

Você já se perguntou quais são as atividades desempenhadas por um Analista de Desenvolvimento de Sistemas? Como as atividades se inter-relacionam? Quais são os artefatos produzidos por estas atividades? São alguns exemplos de questionamentos que serão respondidos ao longo das próximas seções.

Assim, destacamos como objetivos de aprendizagem, específicos da Unidade I: Conhecer as principais atividades, que compõem as fases de análise e projeto de sistemas, e adquirir habilidades para desempenhar estas atividades em diferentes cenários de desenvolvimento.

Para que nosso estudo fique ainda mais rico, ao longo desta e das próximas unidades, o desafio será analisar e projetar um sistema de comércio eletrônico, chamado de “Loja ADS” (Acrônimo de Analista de Desenvolvimento de Sistemas). Será que o nome ficou legal? O que você acha?

1 ANÁLISE E PROJETOS DE SISTEMAS: FUNDAMENTOS E IMPLEMENTAÇÃO

Você já refletiu sobre o significado dos termos “análise” e “projeto”? Pois bem, vamos consultar nosso bom e velho Dicionário Aurélio: de acordo com Ferreira (2001, p. 48) o termo “Análise” pode ser definido como o “exame de cada parte de um todo para conhecer-lhe a natureza, as funções etc.”, e o termo “Projeto”, como o “Plano, intento”.

Deste modo, quando dizemos que vamos “analisar e projetar um sistema”, estamos, em outras palavras, indicando que realizaremos o exame detalhado das informações pertinentes ao sistema a ser desenvolvido, para que, então, possamos formalizar um plano de construção do sistema, por meio de um projeto. De forma análoga, quando um engenheiro é contratado para realizar a obra de um edifício, ele examina a natureza do edifício que será construído e, depois, elabora um plano de construção, o qual é

formalizado por projetos estruturais, hidráulicos, elétricos, e assim por diante.

A disciplina de Engenharia de Software nos ensina que o ciclo de vida clássico do processo de software é composto pelas atividades de requisitos, análise, projeto, implementação, teste e operação e manutenção. Na disciplina de Projeto de Sistemas, vamos “enxergar” o processo de desenvolvimento de sistemas, através da visão macro apresentada por Dennis, Wixom e Roth (2014), que divide o ciclo de vida do desenvolvimento de sistemas em quatro fases fundamentais: planejamento, análise, projeto e implementação (Figura 1).

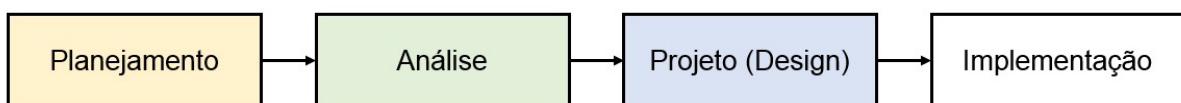


Figura 1: Fases fundamentais do ciclo de desenvolvimento de sistemas.

Fonte: Adaptado pelo autor, a partir de Dennis, Wixom e Roth (2014).

Entende-se ser esta uma visão adequada para a disciplina de Projeto de Sistemas, uma vez que ela inclui a fase de planejamento, possuindo atividades, nas quais o analista de desenvolvimento de sistemas desempenha um importante papel junto aos gestores de projetos.

Logo, nas próximas seções, faremos o estudo detalhado das principais atividades que compõem as fases de planejamento, análise e projeto.

1.1 FASE DE PLANEJAMENTO

A demanda pelo desenvolvimento de um sistema pode surgir em diferentes cenários, por exemplo: (I) uma empresa A deseja que o seu departamento de TI (Tecnologia da Informação) desenvolva um sistema para outro departamento interno (departamento de marketing); (II) uma empresa B deseja que uma empresa C, especializada em desenvolvimento de software, desenvolva um sistema para o seu negócio; (III) um acadêmico precisa desenvolver um sistema de coleta de dados para sua pesquisa.

Nos dois primeiros exemplos apresentados, a demanda pelo desenvolvimento do sistema, normalmente, chega ao departamento de TI na forma de uma solicitação ou requisição, a qual traz um breve resumo da necessidade. Vamos imaginar que a nossa empresa de software acabou de receber o e-mail da Figura 2.

Empresa A <contato@empresaa.com.br>
Para: Empresa B <contato@empresab.com.br>

01 de abril de 2020 10:24

Caro desenvolvedor,

Minha empresa precisa lançar um sistema de comércio eletrônico para aumentar suas vendas no mercado. Ela vende produtos de diversas categorias, como roupas, perfumes e eletrônicos, e aceita pagamento com cartão de crédito e boleto bancário. A loja fará a entrega dos itens solicitados no endereço do cliente e enviará, por e-mail, a nota fiscal eletrônica.

Aguardo análise.

Jorge.

Figura 2: Demanda de desenvolvimento de sistema de comércio eletrônico.

Fonte: Adaptado pelo autor, a partir de Bacelo e Yamaguti (2017).

Trata-se de uma empresa que deseja lançar um sistema de comércio eletrônico para vender seus produtos. Uma vez que o e-mail não detalha muitos aspectos do sistema, o próximo passo será trabalhar junto com a empresa que gerou a solicitação, para analisar a viabilidade do projeto sob aspectos técnicos, econômicos e organizacionais. Vejamos como isso funciona nas próximas subseções.

1.1.1 Análise de viabilidade

De acordo com Dennis, Wixom e Roth (2014), a análise de viabilidade de um projeto de sistema envolve as três perguntas, apresentadas na Figura 3.

A análise de **viabilidade técnica** busca identificar os riscos capazes de impedir que a equipe de desenvolvimento conclua o projeto com sucesso. Quanto maior for a familiaridade da equipe de desenvolvimento com o tipo de sistema que ela deverá desenvolver, e com as tecnologias necessárias para tal, menor será o risco.

O tamanho do projeto é outro ponto a ser considerado. Será que existem pessoas suficientes na equipe, para atender a demanda em tempo hábil? O projeto não é muito complexo? Estas e outras perguntas devem ser feitas durante a análise de viabilidade técnica. Uma abordagem bastante comum para analisar a viabilidade técnica de um projeto é compará-lo com outros projetos já desenvolvidos pela equipe.

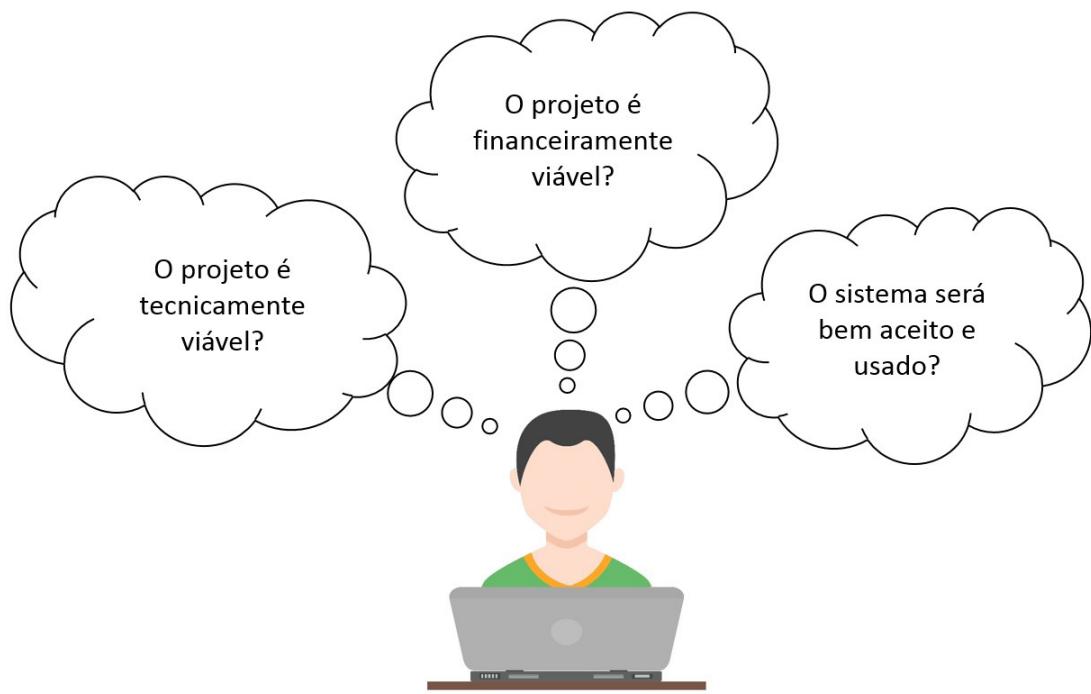


Figura 3: Dimensões da análise de viabilidade.

Fonte: Adaptado pelo autor, a partir de Dennis, Wixom e Roth (2014) e imagens de <https://elements.envato.com/50-professionals-flat-multicolor-icons-SXSHQQ>. Acesso em 13 setembro de 2019.

A análise de **viabilidade econômica** busca avaliar a famosa relação custo-benefício. É necessário atribuir valores aos custos de desenvolvimento, como salários, honorários e treinamentos, e aos custos operacionais, como licenças de software e gastos com comunicações. Para uma empresa de desenvolvimento de software, o benefício mais tangível certamente é o lucro, mas outros benefícios, como um maior reconhecimento da marca no mercado, também podem ser considerados na análise. Por outro lado, a empresa que utilizará o sistema pode identificar muitos benefícios tangíveis, como o aumento de vendas e redução de pessoal e estoque.

A análise de **viabilidade organizacional** busca avaliar se o sistema é passível de ser incorporado ao cotidiano da empresa e se será bem aceito e usado. A implantação de um sistema normalmente impacta na forma como os processos das empresas são executados. Por isso, é necessário realizar uma análise do grupo de pessoas que poderá ser afeto (ou será afetado) pela implantação do novo sistema.

Por fim, os resultados da análise de viabilidade devem ser apresentados para a pessoa, ou comitê gestor responsável pela aprovação do projeto. Uma vez aprovado o projeto, ele entra em processo de desenvolvimento.

É certo que a maior parte das empresas de software possui um modelo de processo de desenvolvimento de sistema bem definido, principalmente as empresas especializadas

no desenvolvimento de um tipo de sistema específico. Nos demais casos, é necessário selecionar (ou adaptar) um modelo de processo, levando em consideração as características do sistema e da equipe que irá desenvolvê-lo.

Vamos selecionar um modelo de processo para o desenvolvimento do sistema Loja ADS?

1.1.2 Seleção do modelo de processo de desenvolvimento

Para que seja possível selecionar um modelo de processo, adequado ao desenvolvimento do sistema, vamos analisar os principais modelos, fornecidos pela disciplina de Engenharia de Software, considerando que a equipe de desenvolvimento possui apenas duas pessoas (eu e você). As características de cada modelo são demonstradas no Quadro 1.

Modelo de processo	Características
Em cascata (ROYCE, 1970)	Sugere uma abordagem sequencial e sistemática, que inicia com os requisitos do cliente e avança pelas atividades de análise, projeto, implementação, teste e operação, culminando com a manutenção contínua do software concluído.
Iterativo e Incremental	É utilizado para desenvolver um software, de forma iterativa (na qual as atividades se repetem) e incremental (em que cada repetição realiza entregas), permitindo o rápido fornecimento de um determinado conjunto de funcionalidades ao cliente.
Prototipação	Pode ser utilizado como um modelo de processo isolado, que evolui de uma solução simplificada até alcançar um produto acabado, entretanto é mais utilizado, no contexto de qualquer um dos modelos de processo, como uma técnica de auxílio para as atividades de requisitos e projeto.
Em espiral (BOEHM, 1988)	Sugere a aplicação de uma sequência de atividades metodológicas, que são executadas de forma iterativa, para produzir versões cada vez mais refinadas do software, ou seja, reúne características dos modelos em cascata, iterativo e prototipação.
Baseado em componentes (MCILROY, 1968)	Propõe o desenvolvimento com reuso. Inicia com a especificação de requisitos e segue com a busca e análise de componentes, modificação dos requisitos (em função dos componentes encontrados), projeto do sistema com reuso, desenvolvimento e integração dos componentes e validação de sistema.

Processo Unificado (RUP - Rational Unified Process) (IBM, 1998)	Propõe a implementação dos princípios do desenvolvimento ágil, aproveitando as melhores características dos modelos de processo genéricos. Tanto uma atividade quanto o conjunto de atividades podem ser executados de forma iterativa e incremental.
---	---

Quadro 1 – Modelos de processo de desenvolvimento de software.

Fonte: Adaptado pelo autor, a partir de Pressman e Maxim (2016) e Sommerville (2011).

O modelo em cascata sugere uma abordagem sequencial e sistemática. Com isto, ele não é apropriado para projetos de software que estão constantemente sujeitos a mudanças, mas pode servir para projetos de curta duração, cujos requisitos sejam inicialmente bem compreendidos e pouco suscetíveis a mudanças. Os sistemas do tipo comércio eletrônico apresentam funcionalidades fáceis de serem compreendidas, como cadastrar produtos e realizar compras. Assim, consideraremos o modelo em cascata como a nossa primeira opção.

O desenvolvimento iterativo e incremental é uma abordagem comum para o desenvolvimento de sistemas com metodologias ágeis, pois permite entregar a parte essencial (mais urgente ou importante) do sistema, para que o cliente possa avaliar os requisitos básicos já nos estágios iniciais do desenvolvimento. Desse modo, podemos pensar em liberar as funcionalidades de cadastro de produtos, no primeiro incremento, o carrinho de compras no segundo e a geração de relatórios no terceiro, isto é, enquanto trabalhamos no carrinho de compras (segundo incremento), o cliente já pode utilizar o primeiro incremento para cadastrar o estoque. Então, vamos considerar este modelo como a nossa segunda opção.

A prototipação pode ser usada para identificar omissões e erros nos requisitos, bem como para compreender o que deve ser construído. Um protótipo também pode ser utilizado para apoiar o projeto de interface de usuário, principalmente no que se refere ao modelo de interação humano-computador. Conforme já discutimos, os sistemas do tipo comércio eletrônico apresentam funcionalidades bem compreendidas. Assim, podemos descartar o uso da prototipação, como ferramenta de apoio à atividade de requisitos, mas podemos pensar em usá-la para apoiar o projeto de interface, já que a navegação e a usabilidade são fatores importantes ao sucesso de qualquer sistema de comércio eletrônico.

O modelo espiral é uma abordagem realista para o desenvolvimento em larga escala. Por exemplo, a primeira iteração pode ser usada para a elaboração de um projeto de conceitos, enquanto as iterações subsequentes, para o desenvolvimento de versões cada vez mais evoluídas do software, permitindo aos desenvolvedores e clientes compreenderem melhor os riscos em cada nível evolucionário. Como não estamos diante de um desenvolvimento em larga escala, vamos descartar esta opção.

O modelo de desenvolvimento baseado em componentes incorpora as características, evolucionária e iterativa, do modelo espiral e proporciona uma série de benefícios mensuráveis, tais como a redução dos custos e do tempo de execução do projeto. Se adotarmos este modelo, ele pode levar a um software, o qual não atende às reais necessidades dos usuários, e à perda do controle sobre a evolução do sistema, já que as novas versões dos componentes não estão sob o nosso domínio.

O Processo Unificado (RUP) descreve o processo de desenvolvimento em termos de atividades, artefatos, pessoas e fluxos de trabalho (*workflows*). Existem seis *workflows* centrais e três de apoio. Todos os *workflows* podem estar ativos, em todas as fases, no entanto os *workflows* de modelagem, requisitos e análise tendem a estar mais ativos nas fases iniciais, enquanto os demais *workflows*, nas fases finais. É um modelo amplamente adotado por empresas. No entanto, sabendo que a nossa equipe de desenvolvimento é pequena, e que os *workflows* são muitos, o RUP não é o modelo mais adequado ao nosso caso.

Assim, para o desenvolvimento do sistema Loja ADS, a adoção do modelo de processo iterativo e incremental e o uso da prototipação para a atividade de design de interface nos parecem boas opções.



SAIBA MAIS

Segundo Sommerville (2011), o desenvolvimento incremental de software é uma abordagem comum ao desenvolvimento de sistemas, além de ser uma parte fundamental das metodologias ágeis de desenvolvimento de software.

Realize uma pesquisa sobre as metodologias ágeis Scrum e XP (eXtreme Programming) e elabore um descritivo de cada uma delas. Podem ser utilizados sites na Internet.

1.2 FASE DE ANÁLISE

Na fase de análise, o analista de sistemas deve trabalhar exaustivamente com as partes interessadas, identificando as necessidades a serem atendidas pelo sistema, de acordo com os objetivos do negócio. É necessário eliciar os requisitos do sistema e elaborar os

modelos de casos de uso e de dados, para que o cliente esteja ciente sobre o produto que será entregue, tendo o analista informações suficientes para elaborar o projeto técnico do sistema.

1.2.1 Determinação dos requisitos

A engenharia de requisitos é uma subárea da engenharia de software que faz uso de técnicas (por exemplo, entrevistas, questionários e seminários) para eliciar, analisar, documentar, validar e manter os requisitos de um software, de maneira completa, clara e precisa (PAULA FILHO, 2009; VAZQUEZ; SIMÕES, 2016). Trata-se de um processo iterativo (pois as atividades de elicitação, especificação e validação se repetem em ciclos) e, ao mesmo tempo, evolucionário, porque, a partir dos requisitos de negócio, são realizados sucessivos refinamentos, até que se atinjam os requisitos de sistema, conforme mostra a Figura 4.



Figura 4: Visão em espiral do processo de engenharia de requisitos.

Fonte: Adaptado pelo autor, a partir de Sommerville (2011).

Segundo Vazquez e Simões (2016), os requisitos de negócio são declarações que descrevem a motivação do projeto, suas metas, bem como as métricas que serão utilizadas para aferir o seu sucesso. No caso do sistema Loja ADS, nosso cliente declarou que o seu principal requisito de negócio é aumentar as vendas da empresa (Figura 2).

Os requisitos de usuário são as declarações em linguagem natural das necessidades de informação, para que as partes interessadas possam desempenhar suas tarefas. Considera-se parte interessada qualquer pessoa que possa afetar ou ser afetada pela solução, a qual está sendo desenvolvida.

A elicitação de requisitos é uma atividade desafiadora. O analista de sistemas deve trabalhar com os clientes, usuários e, até mesmo, especialistas da área de aplicação, para determinar os limites do sistema que será desenvolvido. Para tal, ele pode utilizar uma variedade de técnicas, como mapeamento de processos, questionários e entrevistas que realizam as mesmas perguntas, de diversas maneiras.

Imagine que, após vários encontros com as partes interessadas, na Empresa A, conseguimos extrair os requisitos de usuário do sistema Loja ADS, e que enviamos a especificação para o cliente validar (Figura 5).

Empresa B < contato@empresab.com.br> Para: Empresa A < contato@empresaa.com.br>	04 de abril de 2020 11:05
Caro cliente, aguardamos a validação da seguinte especificação de requisitos:	
1. Cadastro de produtos: cada produto deve ser cadastrado com sua descrição, preço de venda, quantidade em estoque e respectiva categoria.	
2. Cadastro de clientes: a) cada cliente tem que se cadastrar no sistema indicando seu nome, endereço, e-mail e CPF (se for individual) ou CNPJ (se for corporativo).	
3. Compras: a) o cliente cadastrado pode realizar um pedido de compra dos produtos em estoque na quantidade que desejar; b) as formas de pagamento são cartão de crédito e boleto bancário; c) o cliente escolhe uma forma de pagamento disponível e recebe, por e-mail, o número do pedido e as informações do status do pedido; d) após a confirmação do pagamento, a loja realiza a entrega dos itens solicitados no endereço do cliente e o sistema envia, por e-mail, a nota fiscal eletrônica.	
4. Relatórios: o sistema deve emitir relatório de vendas por produto, categoria de produto, região e período.	
5. O sistema deverá ser compatível com os navegadores Google Chrome, Internet Explorer, Safari e Mozilla Firefox, e deverá permitir o acesso através de dispositivos móveis. O tempo de carregamento das páginas não pode ser maior que 2 segundos.	
James	

Figura 5: Requisitos de usuário.
Fonte: Adaptado pelo autor, a partir de Bacelo e Yamaguti (2017).

Olhando novamente para a Figura 4, podemos verificar que a próxima atividade da engenharia de requisitos seria a prototipação. No entanto, você deve lembrar que na Seção 1.1.2 nós decidimos fazer uso da prototipação, apenas para apoiar o projeto de interface, pois os sistemas do tipo comércio eletrônico apresentam funcionalidades bem compreendidas. Assim, podemos descartar a etapa de prototipação e partir para a elicitação dos requisitos de sistema, os quais são compostos por **requisitos funcionais** e **requisitos não funcionais**.

Enquanto os requisitos funcionais descrevem o software, sob a perspectiva das interações dos seus usuários (por exemplo, cadastrar clientes e cadastrar produtos), os requisitos não funcionais se encarregam em descrever as propriedades do software, tais como usabilidade, desempenho, plataformas, linguagens de programação e sistemas de gerenciamento de banco de dados (VAZQUEZ; SIMÕES, 2016). A Figura 6 apresenta a taxonomia de requisitos não funcionais, proposta por Sommerville (2011).



Figura 6: Tipos de requisitos não funcionais.

Fonte: Adaptado pelo autor, a partir de Sommerville (2011).

De acordo com Sommerville (2011), os requisitos de produto referem-se ao comportamento do software, por exemplo: (I) todas as informações deverão ser acessadas com, no máximo, três cliques (usabilidade); (II) o tempo de carregamento das páginas não poderá ser maior que 2 segundos (eficiência); (III) a disponibilidade do sistema deverá ser de, no mínimo, 99% (confiança); (IV) a base de dados deverá ser replicada a cada 12h (proteção).

Os requisitos organizacionais (derivados das políticas de uma empresa, corporação) contemplam tanto os requisitos organizacionais da empresa que está adquirindo

o software (o cliente) quanto os da empresa que está fornecendo o software (o desenvolvedor): (I) o sistema deverá ser executado na nuvem (ambiente); (II) os relatórios deverão estar disponíveis somente das 8h às 18h (operação) e (III) o sistema deverá ser desenvolvido, utilizando a linguagem de programação PHP (desenvolvimento).

Por fim, os requisitos externos são todos os que derivam de fatores externos ao sistema e seu processo de desenvolvimento, tais como normativas para a emissão de declarações e certificados e o sigilo dos dados cadastrais dos clientes.

O Quadro 2 apresenta a especificação de requisitos de sistema da Loja ADS, elaborada a partir do refinamento dos requisitos de usuário, verificados na Figura 5.

Loja ADS – Especificação de Requisitos de Sistema
Requisitos funcionais
RF01. O sistema deverá permitir ao administrador cadastrar categorias de produtos. RF02. O sistema deverá permitir ao administrador cadastrar produtos. RF03. O sistema deverá permitir ao cliente realizar o seu cadastro. RF04. O sistema deverá permitir ao cliente realizar pedidos de compra com cartão de crédito ou boleto bancário e, após a confirmação do pedido, receber um e-mail, com o número do pedido e as informações do status do pedido. RF05. O sistema deverá gerar a nota fiscal eletrônica de cada pedido, após a confirmação do pagamento, e enviá-la para o e-mail do cliente. RF06. O sistema deverá permitir ao administrador gerar relatórios de vendas por produto, categoria de produto, região e período.
Requisitos não funcionais
RNF01. O sistema deverá ser responsivo, permitindo o acesso através de dispositivos móveis. RNF02. O tempo de carregamento das páginas do sistema não pode ser maior que 2 segundos. RNF03. O sistema deverá ser compatível com os navegadores Google Chrome, Internet Explorer, Safari e Mozilla Firefox. RNF04. O sistema deverá ser desenvolvido, utilizando a linguagem de programação PHP. RNF05. O banco de dados deve ser desenvolvido no MySQL.
Regras de negócio
RN01. Um cliente pode somente realizar pedidos de compra de produtos disponíveis no estoque.

Quadro 2 – Lista de requisitos de sistema da Loja ADS.

Fonte: Elaborado pelo autor (2020).

Nesse último nível de refinamento de requisitos, o documento de especificação deve ser capaz de: (I) proporcionar aos desenvolvedores o entendimento de como o cliente quer que o sistema funcione; (II) informar ao projetista quais funcionalidades e características o sistema deve ter e (III) informar à equipe de testes o que demonstrar, para convencer o cliente de que o sistema está sendo entregue, de acordo com o que foi solicitado.

Cada requisito funcional deve descrever uma tarefa, a qual será realizada no sistema.

Perceba que o requisito de usuário “cada produto deve ser cadastrado com sua descrição, preço de venda, quantidade em estoque e respectiva categoria” (Figura 5) implica a especificação dos requisitos funcionais “cadastrar categorias” (RF01) e “cadastrar produtos” (RF02), tendo em vista que os usuários não digitarão o valor do atributo categoria, durante o cadastro do produto, mas sim o selecionarão, a partir de um cadastro prévio de categorias.

No requisito funcional “realizar pedido” (RF04), há outra situação que merece atenção. As tarefas de realizar pagamento com cartão de crédito ou boleto bancário e de enviar e-mail serão sempre executadas como subtarefas de uma tarefa maior, a de realizar um pedido, e nunca serão executadas de forma isolada. Por este motivo, elas estão descritas dentro do requisito funcional “realizar pedido”.

Na descrição de um requisito funcional é importante informar quem será o ator da tarefa, especialmente quando o sistema opera com mais de um perfil de usuário. Perceba que, na especificação apresentada no Quadro 2, estamos informando que a tarefa “cadastrar produto” (RF02) será realizada pelo usuário do tipo administrador, enquanto a tarefa “realizar pedido” (RF04) será executada pelo usuário do tipo cliente.

Observe que a lista de requisitos não funcionais da especificação, apresentada no Quadro 2, possui um requisito de usabilidade (RNFO1), um requisito de eficiência (RNFO2) e três requisitos de desenvolvimento (RNFO3, RNFO4 e RNFO5). Assim, as regras de negócio completam a especificação dos requisitos de sistema, descrevendo políticas, condições ou restrições que devem ser consideradas na execução dos processos existentes na Loja ADS.

1.2.2 Análise de casos de uso

Na seção anterior, você viu que um documento de especificação de requisitos revela quais são as tarefas, realizadas pelos usuários de um sistema, por meio da descrição dos requisitos funcionais dele. No entanto, a especificação de requisitos, por si só, não revela detalhes de como ocorrem as interações dos usuários com o sistema, durante a realização destas tarefas. Segundo Dennis, Wixom e Roth (2014), o objetivo da análise de casos de uso é a criação de um conjunto de casos de uso que descreva as tarefas a serem executadas no sistema, sob o ponto de vista dos seus usuários.

Os principais produtos da análise de casos de uso são os diagramas de casos de uso, que também podem ser utilizados como ferramenta de comunicação com as partes interessadas, durante a atividade elicitação de requisitos (estudada na seção anterior), e

um documento de detalhamento dos casos de uso, o qual descreve como as interações dos usuários com o sistema ocorrem, durante a execução de cada caso de uso. A Figura 7 apresenta os casos de uso do sistema Loja ADS, gerados a partir da lista de requisitos funcionais, que se pode observar no Quadro 2.

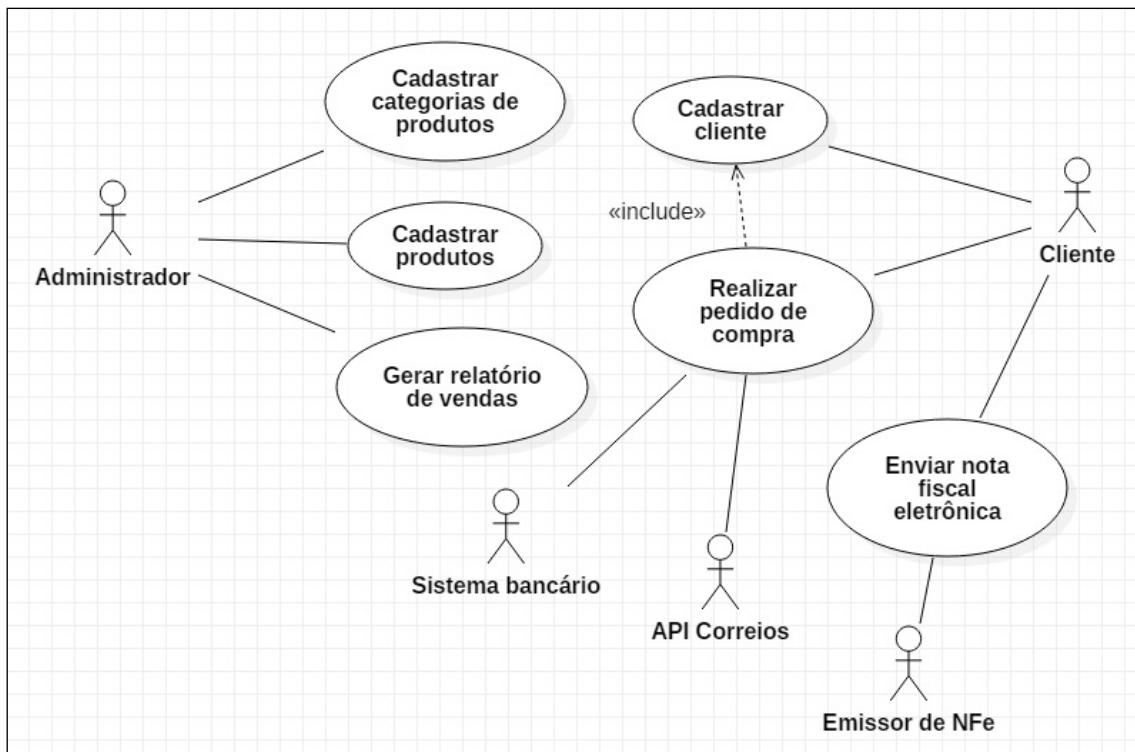


Figura 7: Casos de uso do sistema Loja ADS.

Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

As elipses representam os casos de uso do sistema, os bonecos de palito representam os atores, e as linhas contínuas representam as associações. Um ator representa um papel, desempenhado por usuário ou qualquer outra entidade externa, tal como um hardware ou outro sistema. A execução de um caso de uso pode envolver a participação de um ou mais atores.

Por exemplo, a execução do caso de uso “Realizar pedido de compra” envolve a participação dos atores “Cliente”, “Sistema bancário” e “API Correios”, pois para a realização de um pedido de compra ocorrer, é necessário que o sistema da Loja ADS se comunique com a API dos Correios, calculando o frete, e com o sistema bancário, a fim de que este autorize pagamentos com cartão de crédito.

Outro exemplo: a execução do caso de uso “Enviar nota fiscal eletrônica” envolve a participação do ator “Emissor de NFe”, que é responsável por gerar a nota fiscal, e do ator “Cliente”, que recebe a nota fiscal em seu e-mail.

Um diagrama de casos de uso também pode conter relações de inclusão (include) e

extensão (extend) entre casos de uso. A relação de inclusão é utilizada, quando a execução de um caso de uso inclui a execução de outro caso de uso, como subcaso. No diagrama da Figura 7, a anotação “<<include>>” indica que a execução do caso de uso “Realizar pedido de compra” inclui, obrigatoriamente, a execução do caso de uso “Cadastrar cliente”, ou seja, o cliente não pode realizar um pedido de compra, caso não esteja cadastrado no sistema.

Na Unidade 2, faremos o estudo detalhado da linguagem de modelagem UML (Unified Modeling Language), por meio da qual são construídos diagramas, tal qual o apresentado na Figura 7. Agora, vamos entender como é elaborado um documento de detalhamento dos casos de uso (Quadro 3), exemplificando o caso de uso “Realizar pedido de compra”.

CU01 - Realizar pedido de compra	
Descrição	Permite ao cliente realizar um pedido de compra dos produtos em estoque na quantidade que desejar.
Atores	Cliente. Sistema bancário. API Correios.
Requisitos	<p>RF04. O sistema deverá permitir ao cliente realizar pedidos de compra, com cartão de crédito ou boleto bancário, além de, após a confirmação do pedido, receber um e-mail, com o número do pedido e as informações do status do pedido.</p> <p>RNF01. O sistema deverá ser responsivo, permitindo o acesso através de dispositivos móveis.</p> <p>RNF02. O tempo de carregamento das páginas do sistema não pode ser maior que 2 segundos.</p> <p>RNF03. O sistema deverá ser compatível com os navegadores Google Chrome, Internet Explorer, Safari e Mozilla Firefox.</p> <p>RN01. Um cliente somente pode realizar pedidos de compra de produtos disponíveis no estoque.</p>
Pré-condições	O cliente deve estar cadastrado no sistema. O cliente deve estar autenticado no sistema.
Pós-condições	Um novo pedido de compra foi realizado.
Fluxo principal	

1. O cliente navega pelas páginas da loja, por meio do menu de categorias ou da ferramenta de busca por palavras-chave;
2. O cliente clica sobre o botão "Adicionar ao carrinho", a cada produto que deseja comprar, informando a quantidade do produto em cada operação;
3. O cliente seleciona a opção "Carrinho de compras";
4. O sistema exibe os produtos do carrinho de compras na tela;
5. O cliente seleciona a opção "Continuar";
6. O sistema exibe as configurações do pedido na tela;
7. O cliente confirma o endereço de entrega;
8. O sistema solicita o cálculo do frete à API dos Correios e exibe o valor total do pedido;
9. O cliente seleciona a bandeira do cartão de crédito e informa os dados do cartão;
10. O cliente seleciona a opção "Finalizar pedido";
11. O sistema solicita ao sistema financeiro que o pagamento seja autorizado;
12. O sistema exibe uma mensagem de confirmação do pedido na tela;
13. O sistema envia um e-mail para o cliente, com o número do pedido e seu status.

Fluxos alternativos

FA01 – Alterar pedido: no passo 4, caso deseje alterar o seu pedido:

O cliente seleciona os produtos a serem removidos do carrinho de compras;

O cliente seleciona a opção "Remover";

O sistema atualiza a lista de produtos do carrinho de compras, e o caso de uso continua no passo 5.

FA02 – Novo endereço: no passo 7, caso deseje alterar o endereço:

O cliente seleciona a opção "Alterar endereço";

O sistema exibe o formulário de cadastro de endereço;

O cliente informa os dados do novo endereço;

O cliente seleciona a opção "Alterar", e o caso de uso continua no passo 8.

FA03 – Pagamento com boleto: no passo 9, caso deseje pagar por meio de boleto bancário:

O cliente seleciona a opção "Boleto bancário";

O cliente seleciona a opção "Gerar boleto" e faz o download do arquivo PDF;

O cliente clica em "Finalizar pedido", e o caso de uso continua no passo 12.

Exceções

E01 - Campos obrigatórios não informados: no passo 2.3 do FA02, caso não sejam informados todos os dados obrigatórios, o sistema apresenta mensagem "Preencha todos os campos obrigatórios", e o caso de uso continua no passo 2.3.

E02 – Transação não autorizada: no passo 11 do fluxo principal, caso o pagamento não seja autorizado pelo sistema bancário, o sistema apresenta mensagem "Pagamento não autorizado", e o caso de uso continua no passo 9, ou é encerrado.

Quadro 3 – Detalhamento do caso de uso "Realizar pedido de compra".

Fonte: Elaborado pelo autor (2019).

Inicialmente, é fornecida uma breve descrição do caso de uso. Em seguida, são apresentados os atores, requisitos, pré-condições e pós-condições do caso de uso. Se houver alguma regra de negócio, relacionada com a execução do caso de uso, ela também deve ser apresentada no campo "Requisitos".

Todo caso de uso possui um fluxo principal, que envolve as interações entre os atores e o sistema, no cenário típico de execução, e pode possuir um ou mais fluxos alternativos, envolvendo interações opcionais ou interações que ocorrem, assim que determinadas condições são satisfeitas. As exceções descrevem o que acontece, quando algo inesperado ocorre na interação entre um ator e o sistema.

Um detalhamento de caso de uso também pode conter elementos que facilitam a compreensão e/ou revelam mais detalhes sobre a execução do caso de uso, tais como protótipos de tela e diagramas de atividades.

Você deve ter percebido que o detalhamento de casos de uso é uma atividade trabalhosa. Por isso, em muitos projetos de sistemas, é comum que os analistas optem por detalhar somente os casos de uso mais complexos. No caso da Loja ADS, o caso de uso “Realizar pedido de compra” é, certamente, o caso de uso mais importante do sistema e, ao mesmo tempo, o caso de uso que envolve mais interações, atores e fluxos alternativos, por isso optamos por detalhá-lo.



EXERCÍCIO

Com base no modelo, apresentado no Quadro 3, elabore o detalhamento do caso de uso “Cadastrar produto”, que se demonstrou na Figura 7.

1.2.3 Modelagem de dados

Nas seções anteriores, você deve ter percebido que as atividades de determinação de requisitos e análise de casos de uso fornecem ao analista o entendimento sobre como o sistema a ser desenvolvido irá operar, mas elas não relevam detalhes sobre como as informações, associadas à operação do sistema (por exemplo, os dados de um pedido de compra), serão criadas e usadas pelo sistema.

De acordo com Machado (2014), a modelagem de dados é um processo iterativo que inicia com a elaboração de um modelo conceitual, evolui para um modelo lógico e culmina com a elaboração de um modelo físico.

O modelo conceitual descreve os dados e seus relacionamentos, de forma a serem compreendidos pelos usuários do sistema no contexto de negócio, sem considerar qualquer aspecto da implementação do banco de dados, enquanto o modelo lógico

descreve os dados e seus relacionamentos, considerando os aspectos da abordagem que será utilizada para o armazenamento de dados (rede, relacional, orientada a objetos). Por fim, o modelo físico de dados descreve o banco de dados, em nível de implementação, ou seja, considerando os aspectos específicos do SGBD (Sistema de Gerenciamento de Banco de Dados) que será utilizado no armazenamento de dados, por exemplo, o MySQL (MACHADO, 2014).

Assim, podemos visualizar a modelagem de dados como um processo que inicia na fase de análise, com a elaboração de um modelo conceitual para a validação dos dados e seus relacionamentos, junto às partes interessadas (clientes, usuários do sistema), terminando na fase de projeto, com a elaboração de um modelo físico, para que os programadores possam implementar o sistema.

Vamos analisar o modelo lógico de dados da Loja ADS, exposto na Figura 8, o qual utiliza uma técnica de modelagem conceitual muito comum, denominada Diagrama de Entidade-Relacionamento (DER).

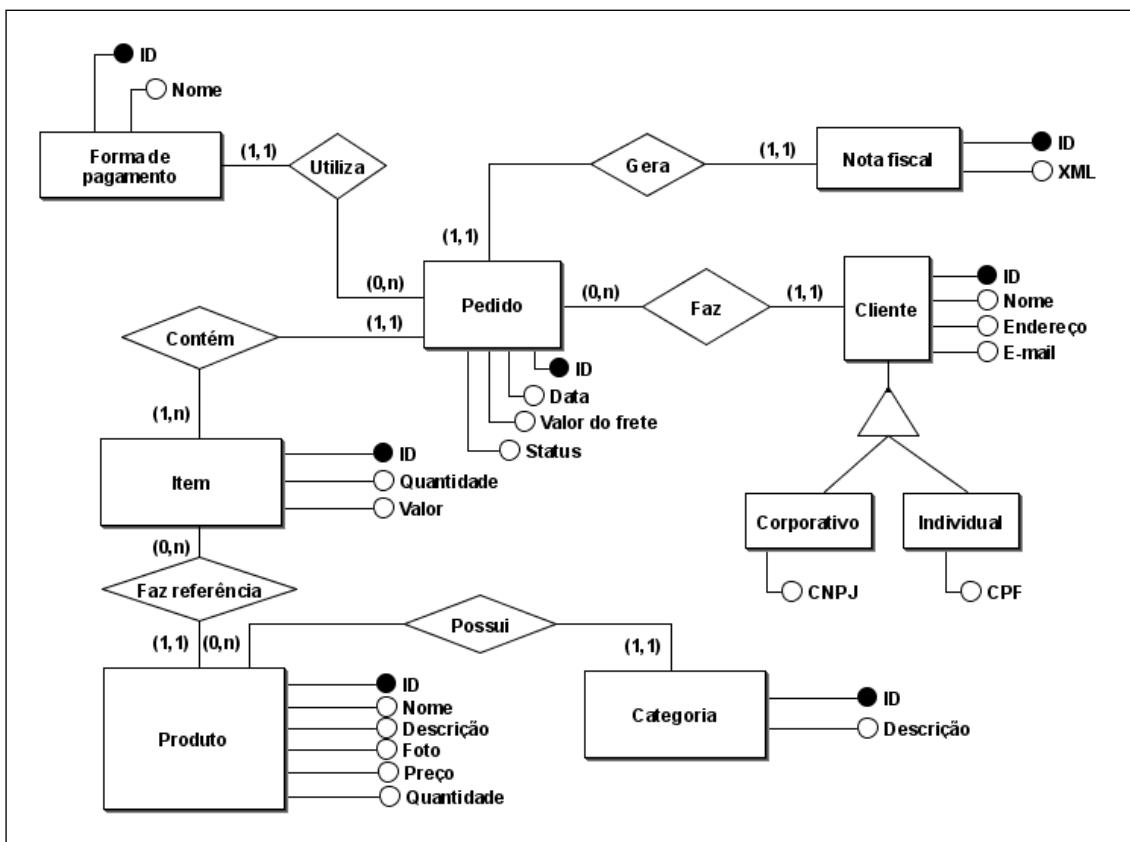


Figura 8: Modelo conceitual de dados do sistema Loja ADS.
Fonte: Elaborado pelo autor, com o uso da ferramenta brModelo (2020).

Os principais blocos de construção de um DER são: as entidades, os atributos e os relacionamentos. Uma entidade é representada por um retângulo e identifica um objeto,

sobre o qual se deseja manter informações no banco de dados. Cada entidade contém um conjunto de atributos, representados pelos círculos a ela associados. Os atributos de uma entidade indicam quais informações ela contém. Por exemplo, no DER da Loja ADS, a entidade “Produto” contém as informações “ID”, “Descrição”, “Preço” e “Quantidade”. Os círculos com preenchimento indicam que um atributo é do tipo identificador, isto é, que o atributo permite distinguir duas ocorrências de uma mesma entidade.

Um relacionamento é representado por um losango, o qual descreve como duas entidades se inter-relacionam, e pares de símbolos entre parênteses, que indicam a cardinalidade (multiplicidade) de cada entidade no relacionamento. Por exemplo, no diagrama ER, da Figura 8, a entidade “Cliente” possui o relacionamento “Faz” com a entidade “Pedido”, indicando que um cliente da Loja ADS pode fazer pedidos. Mas quantos pedidos um cliente pode realizar? Um mesmo pedido pode ser feito por mais de um cliente? As cardinalidades (0, n) e (1, 1) do relacionamento respondem a estas perguntas. A primeira indica que um cliente pode realizar no mínimo 0 e no máximo n (muitos) pedidos, e a segunda, que um pedido pode ser realizado por no mínimo 1 e no máximo 1 cliente.

Perceba que, no modelo conceitual, também podemos representar relações de herança. “Corporativo” e “Individual” são entidades filhas da entidade “Cliente”. Isto significa que, além de seus próprios atributos, as entidades “Corporativo” e “Individual” recebem (herdam) os atributos da entidade “Cliente”.

Muito bem! Agora que já temos informações suficientes, sobre como o sistema a ser desenvolvido deve operar, bem como as informações associadas à operação do sistema serão criadas e usadas, vamos iniciar a fase de projeto do sistema Loja ADS.

1.3 FASE DE PROJETO

Na fase de análise, nós descobrimos como o sistema Loja ADS irá funcionar, do ponto de vista das pessoas que o utilizarão. Agora, será necessário determinar como o sistema Loja ADS será construído e colocado em operação. Quais hardwares serão utilizados? Como será a infraestrutura de rede, servidores e banco de dados? Como os usuários navegarão pelo sistema? Essas e outras perguntas precisam ser respondidas, para que a equipe de programadores do sistema Loja ADS saiba como ela deve proceder na implementação do sistema. Então, vamos lá!

1.3.1 Design de arquitetura

De acordo com Dennis, Wixom e Roth (2014), os sistemas de software podem ser divididos sob dois pontos de vista: hardware e funções. Sob o ponto de vista do hardware, os principais componentes de um sistema são os computadores clientes, empregados pelos usuários (laptops, desktops, tablets, smartphones), os servidores, e a rede que os conecta. Sob o ponto de vista das funções, um sistema de software pode ser dividido em armazenamento de dados, lógica de acesso a dados, lógica de aplicativo e lógica de apresentação. O objetivo do design da arquitetura é determinar como as funções do sistema de software serão atribuídas aos componentes de hardware.

Na arquitetura cliente-servidor, o computador cliente é responsável pela lógica de apresentação, enquanto o servidor, pela lógica de acesso a dados e pelo armazenamento de dados. A lógica de aplicativo pode residir em somente um lado da arquitetura, ou estar distribuída entre o cliente e o servidor. Neste último caso, dizemos que o cliente é gordo, quando ele concentra a maior parte da lógica de aplicativo, e que ele é magro, quando a lógica de aplicativo está mais concentrada no servidor.

A lógica de acesso a dados e a lógica de aplicativo podem residir em diferentes servidores, formando uma arquitetura em 3 camadas. Outra possível estratégia é a de dividir a lógica de aplicativo entre dois ou mais servidores, formando o que chamamos de arquitetura de N camadas. A principal vantagem dessas arquiteturas, em relação à arquitetura cliente-servidor simples (com um único servidor) é a possibilidade de escalar apenas a parte do sistema que está sobrecarregada, substituindo o servidor por um mais poderoso, ou colocando vários servidores para compartilhar a carga.

Assim, as arquiteturas N camadas são adequadas para sistemas de comércio eletrônico, os quais podem ter que lidar com uma grande quantidade de acessos. A Figura 9 expõe o design da arquitetura do sistema Loja ADS.

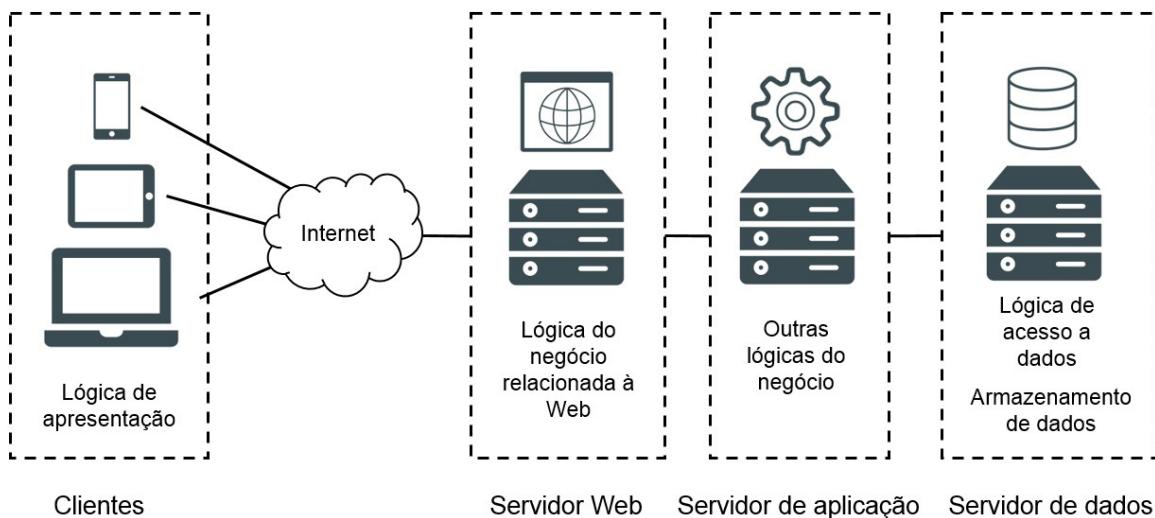


Figura 9: Design da arquitetura do sistema Loja ADS.

Fonte: Elaborado pelo autor, com imagens de <https://image.shutterstock.com/z/stock-vector-computer-icons-and-computer-accessories-icons-flat-vector-set-for-design-and-interface-392865640.jpg>.
Acesso em 09 abril de 2020.

O software do navegador (browser), nos computadores clientes, faz solicitações HTTP para exibir as páginas que são fornecidas pelo servidor Web. Assim que o usuário realiza um pedido de compra, o servidor de aplicação cuida de acrescentar itens ao carrinho de compras, determinar o valor do pedido e do frete, autorizar o pagamento, a fim de processar estas informações e enviá-las ao servidor de dados.



PARA REFLETIR

Conforme a arquitetura, expressa na Figura 9, o acesso ao sistema Loja ADS será através dos navegadores (browsers), instalados nos computadores clientes. No seu entendimento, estes navegadores serão clientes magros ou clientes gordos? Por quê?

1.3.2 Design de interface com o usuário

Basicamente, o design de interface com o usuário define a maneira como os usuários irão interagir com o sistema. A Interação Humano-Computador (IHC) é um campo de estudo que busca melhorar as interações entre usuários e computadores, por meio do estabelecimento de princípios, relacionados à estética, à experiência do usuário, à consistência e a outros aspectos que devem ser observados durante o processo de design de uma interface com o usuário (DENNIS; WIXOM; ROTH, 2014).

Sob o ponto de vista da estética, uma interface com o usuário deve possuir baixa densidade de informações, fontes legíveis e cores que não provoquem emoções muito intensas ou reduzidas, por exemplo ansiedade ou sono. Em outras palavras, ela deve ser convidativa ao uso.

O design de uma interface também precisa considerar o perfil dos usuários. De acordo com Dennis, Wixom e Roth (2014), normalmente, os usuários experientes estão mais preocupados com a rapidez com que eles poderão concluir suas tarefas, enquanto os usuários iniciantes estão mais preocupados com a rapidez com que eles poderão aprender a usar o sistema. No sistema da Loja ADS, a pessoa que fará o cadastro dos produtos no sistema é exemplo de um típico usuário experiente, e o cliente, o qual acessará o sistema para realizar um pedido de compra pela primeira vez, é exemplo de um típico usuário iniciante.

A consistência é um dos aspectos mais importantes de uma interface. Quando todas as partes de um mesmo sistema funcionam de maneira igual, o usuário pode “prever” o que acontecerá em uma parte do sistema, com base nas experiências que ele já teve com as outras partes dele. A consistência deve estar presente não só no modo como as tarefas são realizadas, mas também nas terminologias e simbologias, utilizadas em botões, títulos e outros elementos que compõem a interface.

Estando o projetista inseguro quanto ao modelo de IHC, ele poderá usar a prototipagem para testar e validar suas ideias. Observe na Figura 10:

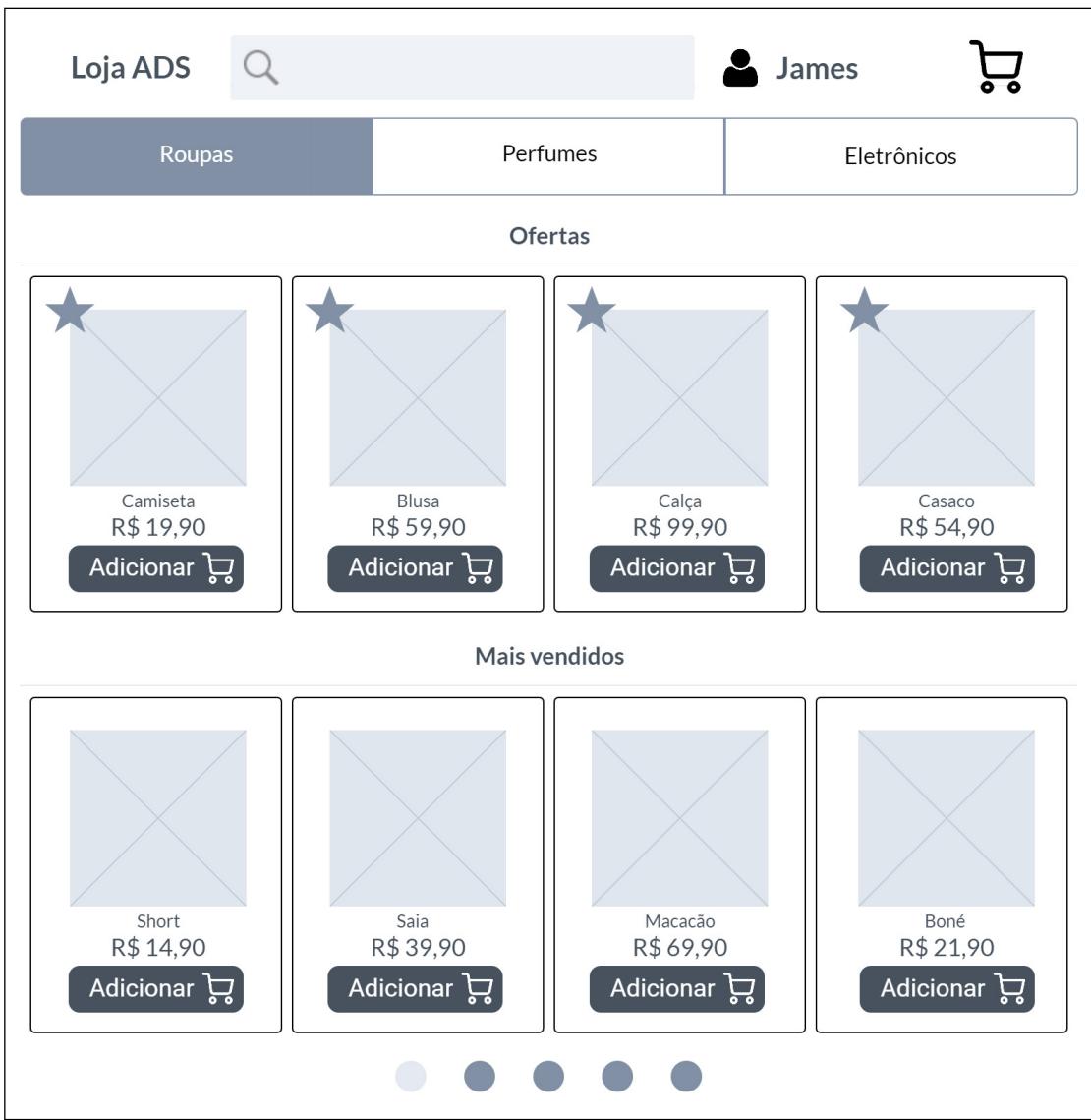


Figura 10: Protótipo de tela do sistema Loja ADS.

Fonte: Elaborado pelo autor, com o uso da ferramenta Marvel App (2020).

Um protótipo pode ser de baixa, média ou alta fidelidade. Os protótipos de baixa fidelidade, normalmente feitos em papel, são mais utilizados nas fases iniciais do projeto, para validar o conceito geral que será utilizado na construção das interfaces. Um protótipo de média fidelidade (como o apresentado na Figura 10) pode ser utilizado, para validar a arquitetura da informação e as interações dos usuários com os elementos da interface. Os protótipos de média fidelidade não têm preocupação com a estética e, em geral, são construídos com ferramentas do tipo “wireframe”.

Por fim, um protótipo de alta fidelidade fornece uma representação bem mais próxima do sistema. Ele permite simular o uso de todas as funcionalidades, mas sem salvar nada no banco de dados. Com um protótipo de alta fidelidade, é possível medir todos os aspectos da Interação Humano-Computador.

1.3.3 Design de armazenamento de dados

A primeira tarefa do design de armazenamento de dados é definir a forma como os dados serão armazenados, de modo a garantir que os programas, os quais compõem o sistema, possam acessar estes dados de maneira fácil e eficiente (DENNIS; WIXOM; ROTH, 2014).

Quanto ao formato de armazenamento, um banco de dados pode ser orientado a objetos, hierárquico, relacional, documental, multidimensional e assim por diante. A decisão, sobre qual formato utilizar, é tomada em nível de modelo lógico de dados, conforme estudamos na Seção 1.2.3. No projeto da loja ADS, utilizaremos o MySQL. Trata-se de um Sistema de Gerenciamento de Banco de Dados (SGBD) relacional, de código aberto e largamente utilizado no armazenamento de dados de aplicações Web.

O próximo passo é a construção do modelo físico de dados. Neste momento, é preciso lidar com questionamentos como: Qual será o tipo de dado de cada atributo? Quantos dígitos serão necessários para armazenar o preço dos produtos? As imagens dos produtos e as notas fiscais serão armazenadas no banco de dados?

A Figura 11 exibe o modelo físico de dados do sistema Loja ADS. O modelo foi construído com o uso da ferramenta MySQL Workbench, a partir da qual é possível gerar o código SQL, para a criação das tabelas no banco de dados MySQL. Vamos analisá-lo e compará-lo com o modelo conceitual, apresentado na Seção 1.2.3.

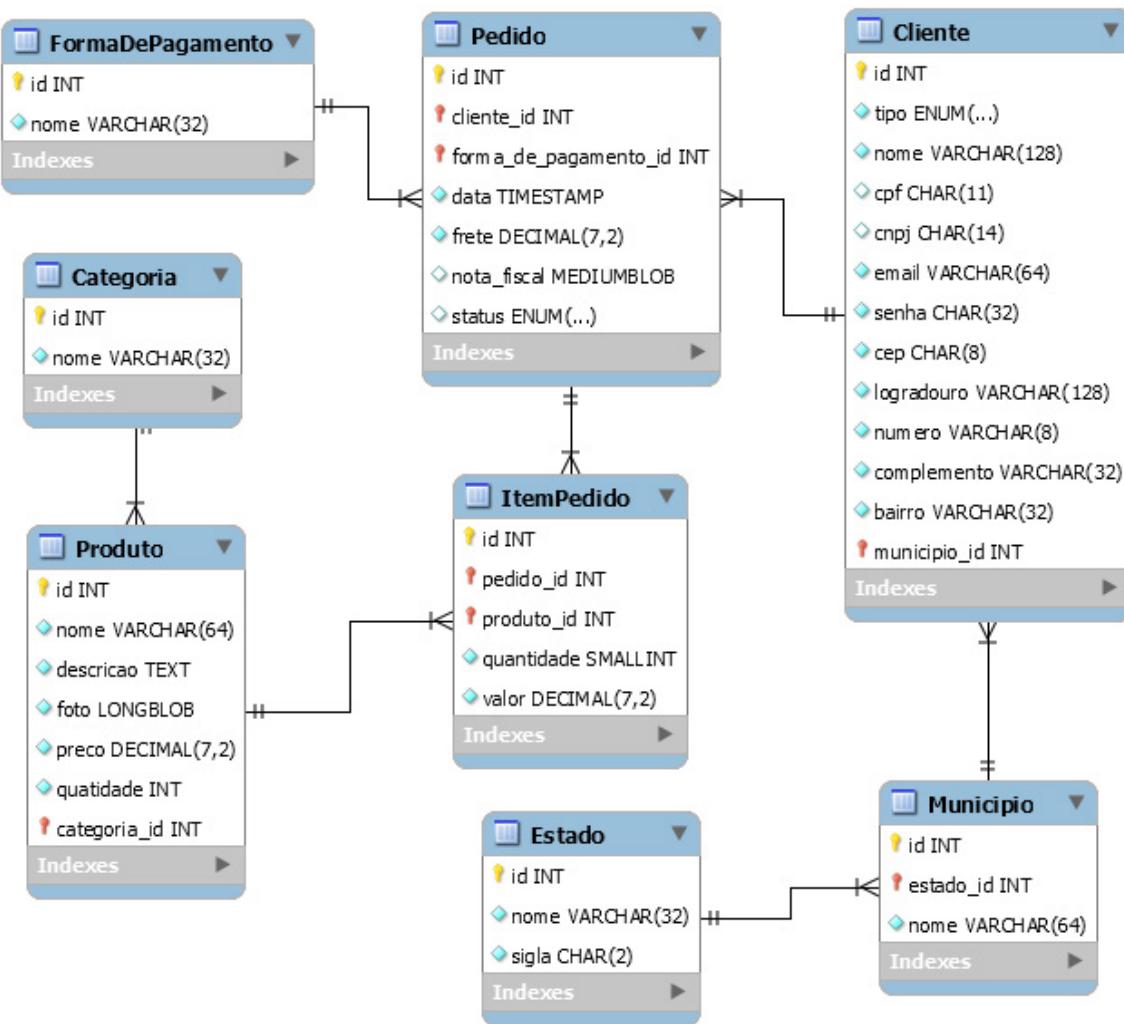


Figura 11: Modelo físico de dados do sistema Loja ADS.

Fonte: Elaborado pelo autor, com o uso da ferramenta MySQL Workbench (2020).

No modelo conceitual da Figura 8 (Seção 1.2.3), as entidades “Individual” e “Corporativo” herdam os atributos da entidade “Cliente”. Existem três principais estratégias para implementar um relacionamento de herança em banco de dados: (I) criar somente as tabelas das entidades filhas e adicionar os atributos da entidade pai nestas tabelas; (II) criar uma única tabela com os atributos de todas as entidades e um campo discriminador e (III) criar uma tabela para cada entidade e inserir chaves estrangeiras nas tabelas das entidades filhas, apontando para a tabela da entidade pai.

No modelo físico (Figura 11), adotamos a segunda estratégia para implementar a herança, porque as entidades filhas possuem poucos atributos específicos. Em cada registro da tabela “Cliente”, haverá apenas um campo (CPF ou CNPJ) com o valor nulo, e o atributo “tipo” será utilizado para informar se o registro é uma entidade “Individual” ou “Corporativo”. Para armazenar o endereço do cliente, foram criados os campos “CEP”,

“logradouro”, “número”, “complemento” e “bairro”, na tabela “Cliente”, e duas tabelas “Município” e “Estado”, a fim de garantir a normalização dos dados.

O tipo de dado DECIMAL (7,2), utilizado nos campos, os quais armazenam valores monetários, define que tais campos poderão armazenar valores com sete dígitos e duas casas decimais. Com isso, estamos definindo que o valor máximo de um pedido, um produto ou um frete será de R\$ 99.999,99. Observe que as fotos dos produtos e as notas fiscais dos pedidos serão armazenadas em campos do tipo BLOB (Binary Large Object), permitindo-se armazenar qualquer tipo de informação em formato binário.

Ufa! Enfim, terminamos o projeto do sistema da Loja ADS, passando pelas clássicas atividades de design de arquitetura, design de interface com o usuário e design de armazenamento de dados. Agora, é hora de conversarmos um pouco com os colegas de disciplina sobre o trabalho realizado até aqui.

1.4 FÓRUM

Utilize o fórum on-line da disciplina, para propor uma discussão mais aprofundada sobre os artefatos, produzidos ao longo desta unidade. converse sobre a importância do detalhamento de casos de uso, do modelo conceitual de dados e da prototipagem, para o processo de desenvolvimento de um sistema.

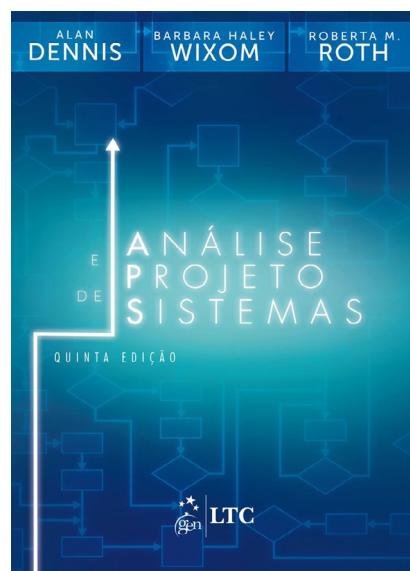


SUGESTÃO DE LIVRO

DENNIS, Alan; WIXOM, Barbara Haley;
ROTH, Roberta M. **Análise e projeto de sistemas**. 5. ed. Rio de Janeiro: LTC, 2014.

A proposta pedagógica de “Análise e Projeto de Sistemas” é apresentar a estudantes da área, e de cursos afins, o conteúdo da disciplina, tal como ela é aplicada no mundo real. São contempladas todas as etapas e a sequência, adotadas na ordem do ciclo de vida do desenvolvimento de sistemas.

O objetivo é fornecer aos estudantes proficiência, prática



e familiarização, relacionadas às diferentes situações e necessidades que se apresentam no exercício profissional cotidiano. Os capítulos foram agrupados em quatro partes, correspondentes às fases comuns aos projetos: Planejamento, Análise, Projeto e Implementação.



SUGESTÃO DE FILME

A Rede Social (2010)

A Rede Social (em inglês: The Social Network) é um filme estadunidense, de 2010, do gênero drama biográfico, dirigido por David Fincher, com roteiro de Aaron Sorkin e Ben Mezrich, baseado no livro de não ficção *The Accidental Billionaires*, de Ben Mezrich.

O filme conta a história da criação da rede social Facebook, bem como seus desdobramentos. Em uma noite de outono, em 2003, o estudante de Harvard e gênio da computação Mark Zuckerberg se senta ao seu computador e começa a trabalhar em um novo conceito, o qual acaba se transformando em uma rede social global. Seis anos e 500 milhões de amigos mais tarde, Zuckerberg se torna o mais jovem bilionário da história, devido ao sucesso de sua rede social.

CONSIDERAÇÕES FINAIS

Nesta unidade, você viu que as fases de análise e projeto de um sistema de software envolvem a execução de um conjunto de atividades inter-relacionadas, que permitem ao analista de sistemas compreender as reais necessidades dos usuários, traduzindo-as em modelos capazes de fornecer informações suficientes, para que a equipe de desenvolvimento possa implementar o sistema.

É importante você ter em mente que o objetivo maior de um analista de sistemas não é a criação de um sistema maravilhoso, mas sim a criação de um sistema que permita à empresa executar melhor o seu trabalho e, com isto, obter melhores resultados. Muitos projetos de sistemas de software são descontinuados, porque o analista dedica muito tempo para criar um sistema perfeito, sem entender claramente como ele auxiliará a empresa a atingir os objetivos propostos por ela.

Continue levando a sério o estudo deste caderno. Na próxima unidade, vamos conhecer a UML (Unified Modeling Language), linguagem de modelagem que fornece diversos diagramas para projetar e documentar sistemas de software.

EXERCÍCIO FINAL

1. A avaliação da viabilidade de um projeto de sistema envolve três tipos de análise: (I) análise de viabilidade técnica, que busca identificar riscos, capazes de impedir que a equipe de desenvolvimento conclua o projeto com sucesso; (II) análise de viabilidade econômica, que busca avaliar a relação custo-benefício e (III) análise de viabilidade organizacional, a qual busca avaliar se o sistema é passível de ser incorporado ao cotidiano da empresa e se será bem aceito e usado.

Em relação aos três tipos de análise de viabilidade apresentados, avalie as afirmações a seguir:

I- A familiaridade da equipe de desenvolvimento com as tecnologias necessárias para a implementação de um sistema é um fator a ser considerado durante a análise de viabilidade técnica;

II- Custos como salários, honorários e treinamentos para o desenvolvimento de um sistema são fatores a serem considerados durante a análise de viabilidade econômica;

III- O impacto que a implantação de um sistema terá na forma como os processos de uma empresa são executados é um fator a ser considerado na análise de viabilidade organizacional.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) I e III, apenas.
- (E) I, II e III.

2. De acordo com a taxonomia proposta por Sommerville (2011), os requisitos não funcionais de um sistema podem ser classificados em: requisitos de produto, requisitos organizacionais e requisitos externos. Os requisitos não funcionais, do tipo requisitos de produto, podem estar relacionados com a usabilidade, a eficiência, a confiança ou a proteção do sistema.

Com base na sistemática apresentada no texto, avalie os requisitos não funcionais a seguir:

I- Todas as informações do sistema deverão ser acessadas com, no máximo, três cliques;

II- O tempo de carregamento do sistema não poderá ser maior que 2 segundos;

III- A disponibilidade do sistema deverá ser de, no mínimo, 99%.

As sentenças I, II e III referem-se, respectivamente, a quais tipos de requisitos de produto?

- (A) Usabilidade, Confiança e Proteção.
- (B) Usabilidade, Eficiência e Confiança.
- (C) Eficiência, Usabilidade e Proteção.
- (D) Eficiência, Confiança e Proteção.
- (E) Proteção, Usabilidade e Confiança.

3. De acordo com Machado (2014), a modelagem de dados é um processo iterativo que inicia com a elaboração de um modelo conceitual, evolui para um modelo lógico e culmina com a elaboração de um modelo físico. O modelo conceitual descreve os dados e seus relacionamentos, de forma que eles possam ser compreendidos pelos usuários do sistema no contexto de negócio. Por sua vez, o modelo lógico descreve os dados e seus relacionamentos, considerando os aspectos da abordagem que será utilizada no armazenamento de dados. Por fim, o modelo físico de dados descreve o banco de dados em nível de implementação.

Em relação aos diferentes níveis de modelos de dados apresentados no texto, avalie as afirmações a seguir:

I- O modelo lógico considera os aspectos de um Sistema de Gerenciamento de Banco de Dados específico;

II- O modelo conceitual não considera os aspectos do Sistema de Gerenciamento de Banco de Dados específico;

III- O modelo físico não considera os aspectos do Sistema de Gerenciamento de Banco de Dados específico.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) I e III, apenas.
- (E) I, II e III.

REFERÊNCIAS

- BACELO, Ana Paula Terra; YAMAGUTI, Marcelo Hideki (Org.). **ENADE comentado: sistemas de informação 2014.** [Recurso Eletrônico]. Porto Alegre: EDIPUCRS, 2017.
- BOEHM, B. W. A Spiral Model of Software Development and Enhancement, **IEEE Computer**, 21(5), pp. 61-72, 1988.
- DENNIS, Alan; WIXOM, Barbara Haley; ROTH, Roberta M. **Análise e projeto de sistemas.** [Recurso Eletrônico]. 5. ed. Rio de Janeiro: LTC, 2014.
- FERREIRA, Aurélio Buarque de Holanda. **Miniaurélio Século XXI: O minidicionário da língua portuguesa.** 5. ed. Rio de Janeiro: Nova Fronteira, 2001.
- IBM. **Rational Unified Process: best practices for software development teams.** Rational Software White Paper, 1998. Disponível em: <https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TPo26B.pdf>. Acesso em: 19 set 2019.
- MACHADO, Felipe Nery Rodrigues. **Projeto e implementação de banco de dados.** 3. ed. São Paulo: Érica, 2014.
- MCILROY, M. D. **Mass produced software components.** In Naur, P. and B. Randell, editors, Report on a Conference of the NATO Science Committee, pp 138–150, 1968.
- PAULA FILHO, Wilson de Pádua. **Engenharia de software: fundamentos, métodos e padrões.** 3.ed. Rio de Janeiro: LTC, 2009. ISBN 978-85-216-1650-4.
- PRESSMAN, Roger S; MAXIM, Bruce R. **Engenharia de Software: uma abordagem profissional.** [Recurso Eletrônico]. 8. ed. Porto Alegre: AMGH, 2016.
- ROYCE W. W. **Managing the Development of Large Software Systems.** Proceedings of the IEEE WESCON. Los Angeles: IEEE; 1970:1–9.
- SOMMERVILLE, Ian. **Engenharia de software.** 9. ed. São Paulo: Pearson Prentice Hall, 2011.
- VAZQUEZ, C. E.; SIMÕES, G. S. **Engenharia de requisitos: software orientado ao negócio.** Rio de Janeiro: Brasport, 2016.

unidade



.....

APLICAÇÃO DE
TÉCNICAS DE
MODELAGEM – UML:
FUNDAMENTOS, VISÃO E
DIAGRAMAS



INTRODUÇÃO À UNIDADE

Na primeira unidade, você conheceu as principais atividades que compõem as fases de análise e projeto de sistemas. Nosso foco de estudo foi compreender como estas atividades se inter-relacionam, tendo como exemplo o desenvolvimento do sistema Loja ADS. Durante a fase de análise, elicitamos os requisitos do sistema, realizamos a análise de casos de uso e elaboramos um modelo conceitual de dados. Já na fase de projeto, elaboramos um desenho da arquitetura do sistema, um protótipo da interface com o usuário e, por fim, um modelo físico do armazenamento de dados.

Na Unidade 2, complementaremos a fase de projeto do sistema Loja ADS com os diagramas, fornecidos pela Linguagem de Modelagem Unificada ou UML (do inglês, Unified Modeling Language). Os diagramas da UML permitem representar diversos aspectos estruturais e dinâmicos de um sistema de software orientado a objeto.

Assim, os objetivos específicos desta unidade são: conhecer a Linguagem de Modelagem Unificada - UML; compreender a visão, proporcionada por cada um dos diagramas UML e adquirir habilidades para a construção de diagramas UML.

2 APLICAÇÃO DE TÉCNICAS DE MODELAGEM – UML: FUNDAMENTOS, VISÃO E DIAGRAMAS

A UML é uma linguagem visual, utilizada para modelar e documentar sistemas de software, por meio do paradigma de orientação a objetos. Dizemos que a UML é uma linguagem visual, pois ela utiliza uma notação diagramática, composta de figuras, símbolos e textos que permitem aos engenheiros de software representar diversas características de um software, tais como sua estrutura, seu comportamento e, até mesmo, suas necessidades físicas (GUEDES, 2014; LARMAN, 2007).

O conjunto de diagramas da UML abrange todas as visões necessárias ao desenvolvimento e implementação de um sistema de software. Em outras palavras, a UML fornece recursos suficientes, para que uma modelagem de software possa ser considerada completa (SILVA, 2008). Neste caderno de estudos, utilizaremos a UML versão 2.5.1, cuja especificação foi publicada pela OMG (Object Management Group), em dezembro de 2017 (OMG, 2017).

O primeiro passo, para podermos utilizar qualquer linguagem, é conhecer o seu

vocabulário e as suas regras de formação, ou seja, a sua gramática. Sendo assim, na próxima seção, conhiceremos os blocos de construção da UML.



SAIBA MAIS

O Modelo e Notação de Processos de Negócio, ou BPMN (do inglês, Business Process Model and Notation), é um padrão de notação, também criado pela OMG. Ele possui elementos semelhantes à UML, no tocante à modelagem de processos, porém foi criado com objetivo diferente. Pesquise sobre a notação BPMN. Descreva suas principais características e suas diferenças, em relação à UML. Podem ser utilizados sites da Internet.

2.1 BLOCOS DE CONSTRUÇÃO DA UML

De acordo com Booch, Rumbaugh e Jacobson (2005), o vocabulário da UML abrange os três tipos de blocos de construção, apresentados na Figura 12. Os itens são os blocos de “primeira classe” em um modelo. Os relacionamentos conectam os itens, e os diagramas agrupam coleções de itens e relacionamentos, com o objetivo de comunicar algo quanto ao sistema de software.

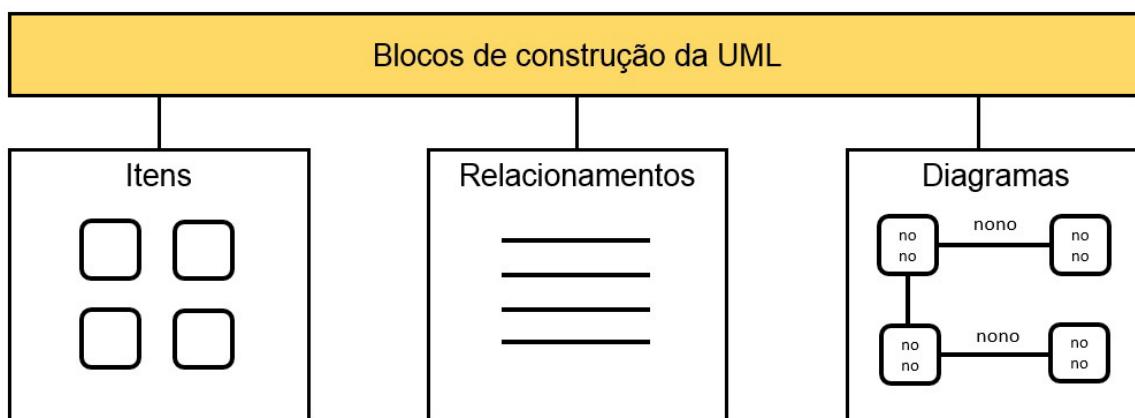


Figura 12: Blocos de construção da UML.
Fonte: Adaptado pelo autor, a partir de Booch, Rumbaugh e Jacobson (2005).

Como você pode perceber, na Figura 12, construir um diagrama UML é nada mais que reunir um conjunto de itens, relacionamentos e textos em um documento do tipo diagrama. Cada tipo de diagrama UML suporta um determinado subconjunto de

itens e relacionamentos do vocabulário UML. Nas próximas subseções, estudaremos detalhadamente cada um dos três blocos de construção da UML.

2.1.1 Itens

Existem quatro tipos de itens na UML: estruturais, comportamentais, de agrupamento e anotacionais.

Os itens estruturais são os **substantivos**, utilizados nos modelos UML. Eles representam os elementos físicos ou conceituais do modelo, tais como classes, casos de uso, componentes e nós.

Os itens comportamentais são os **verbos**, utilizados nos modelos UML, os quais representam as partes dinâmicas dos modelos, no tempo e no espaço (BOOCH; RUMBAUGH; JACOBSON, 2005).

A UML possui um tipo de item de agrupamento chamado pacote. Os pacotes são utilizados para organizar os modelos UML. Por exemplo, quando se trabalha com modelos complexos, contendo dezenas ou até centenas de itens, podemos organizar os itens, agrupando-os, de acordo com o módulo do sistema ao qual eles pertencem ou outro critério qualquer.

Por fim, os itens anotacionais são utilizados para esclarecer, ou fazer uma observação sobre algum elemento do modelo.

A Figura 13, a seguir, traz exemplos destes tipos de itens.

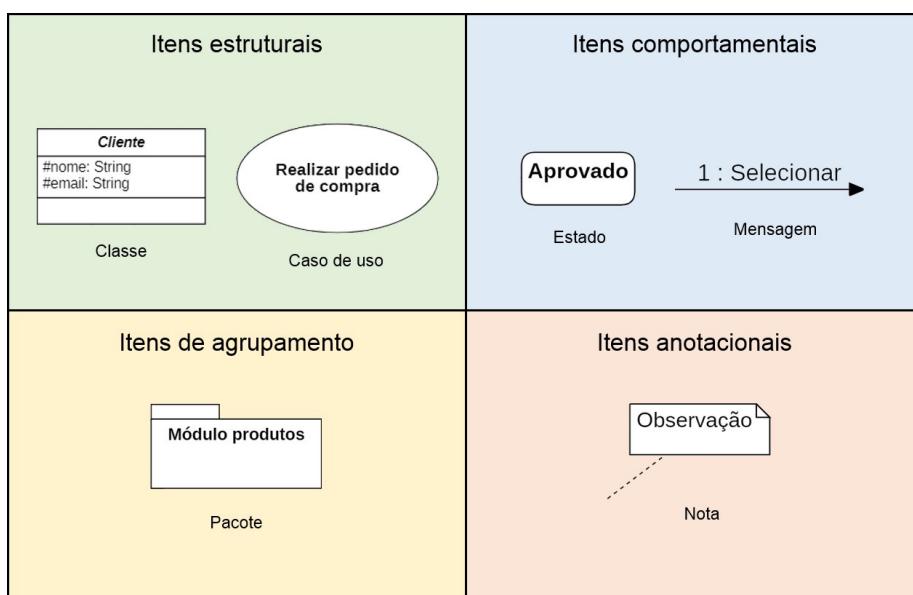


Figura 13: Tipos de itens da UML.

Fonte: Adaptado pelo autor, a partir de Booch, Rumbaugh e Jacobson (2005).

2.1.2 Relacionamentos

Existem quatro tipos de relacionamento na UML: dependência, associação, generalização e realização. A dependência é um relacionamento, representado por uma linha tracejada. Ele indica que a alteração de um item (para o qual a seta aponta) pode afetar outro item (do qual a seta parte). No exemplo, exposto através da Figura 14, a classe “Pedido” depende da classe “CalculoDeFrete”, para calcular o frete do pedido. Se a classe “CalculoDeFrete” sofrer alguma alteração, a função de cálculo do frete, que é executada internamente na classe “Pedido”, pode deixar de funcionar.

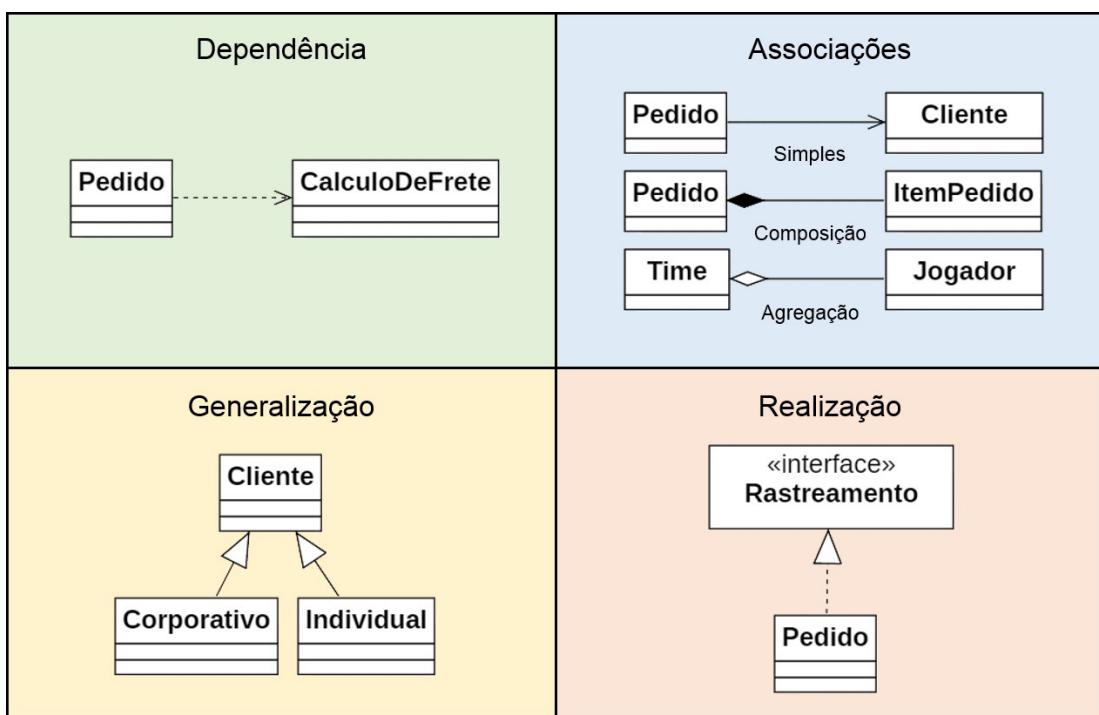


Figura 14: Tipos de relacionamentos da UML.
Fonte: Adaptado pelo autor, a partir de Booch, Rumbaugh e Jacobson (2005).

A associação é uma notação que indica um relacionamento estrutural entre dois itens. A **composição** e a **agregação** são associações do tipo **todo/parte**, ou seja, fornecem a ideia de que um item deve estar associado a um ou mais itens para estar completo (SILVA, 2008).

No exemplo da Figura 14, para um pedido estar completo, ele deve estar associado a um ou mais itens de pedido, assim como para um time estar completo, ele precisa estar associado a jogadores. A diferença entre a agregação e a composição é que, no segundo caso, o item que representa a parte não pode ser compartilhado, e a eliminação do item que representa o todo, implica também a eliminação do item que representa a parte (SILVA, 2008). Nos demais casos de associação, em que não há o conceito de todo/parte, a associação simples é utilizada.

A generalização é uma notação que indica um relacionamento hierárquico, no qual um item compartilha sua estrutura e seu comportamento com outro item. No exemplo ilustrado pela Figura 14, as classes “Corporativo” e “Individual” herdam os atributos e métodos da classe “Cliente”. Por fim, a realização é um relacionamento em que um item especifica uma responsabilidade a ser implementada, e o outro item incorpora a obrigação de implementá-la (SILVA, 2008).

2.1.3 Diagramas

A UML versão 2.5.1 fornece catorze diagramas. Sete deles são voltados à modelagem estrutural e sete, à modelagem comportamental (dinâmica), conforme a taxonomia retratada na Figura 15. Os diagramas estruturais mostram a estrutura estática dos elementos de um sistema, isto é, descrevem os elementos em uma especificação independente de tempo, enquanto os diagramas comportamentais mostram o comportamento dinâmico destes elementos, incluindo seus métodos, atividades, colaborações e histórico de estados (OMG, 2017).

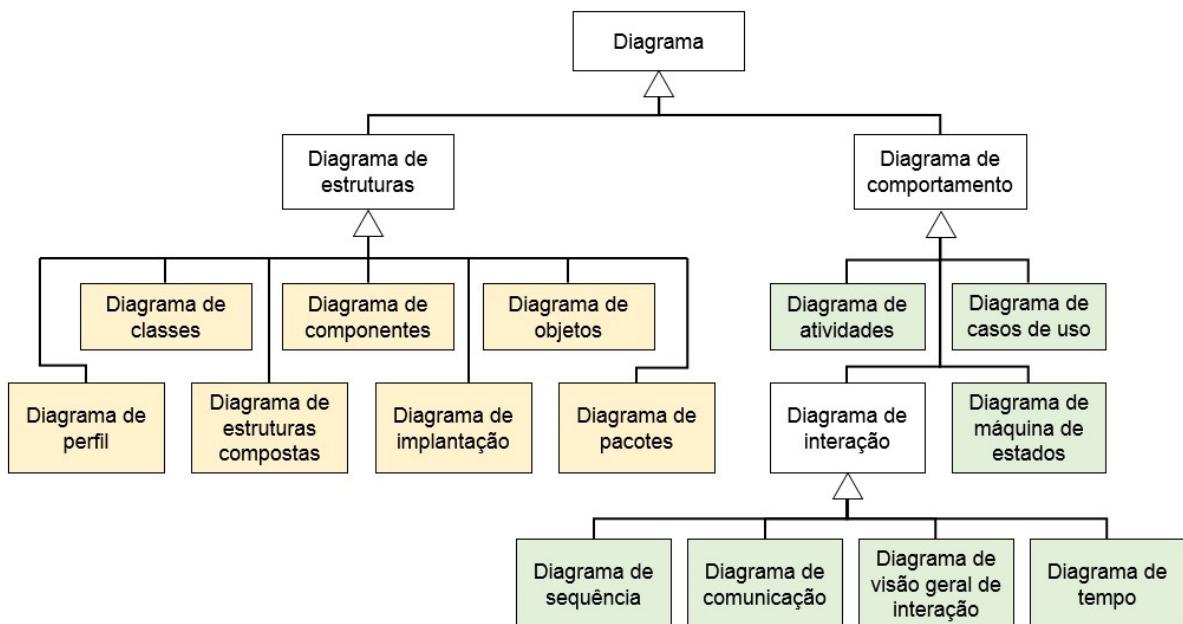


Figura 15: Diagramas da UML versão 2.5.1.
Fonte: Adaptado pelo autor, a partir de OMG (2017).

Neste momento, você pode estar se perguntando: por que tantos diagramas para modelar um sistema? Mas se pensar na diversidade de documentos que podem ser

gerados, quando, por exemplo, uma casa é projetada (planta baixa, projeto estrutural, projeto elétrico, projeto hidráulico, memorial descritivo dos materiais, dentre muitos outros), você perceberá que não são tantos diagramas assim, pois cada um possui uma finalidade bem definida, segundo veremos na próxima seção.

2.2 VISÕES DA UML

Uma modelagem de sistema de software deve permitir que as pessoas, as quais desenvolverão o sistema, tenham a possibilidade de examiná-lo e estudá-lo, a partir de diferentes perspectivas, denominadas visões UML (BEZERRA, 2015). A Figura 16 demonstra as cinco visões de modelagem, comumente propostas na literatura.

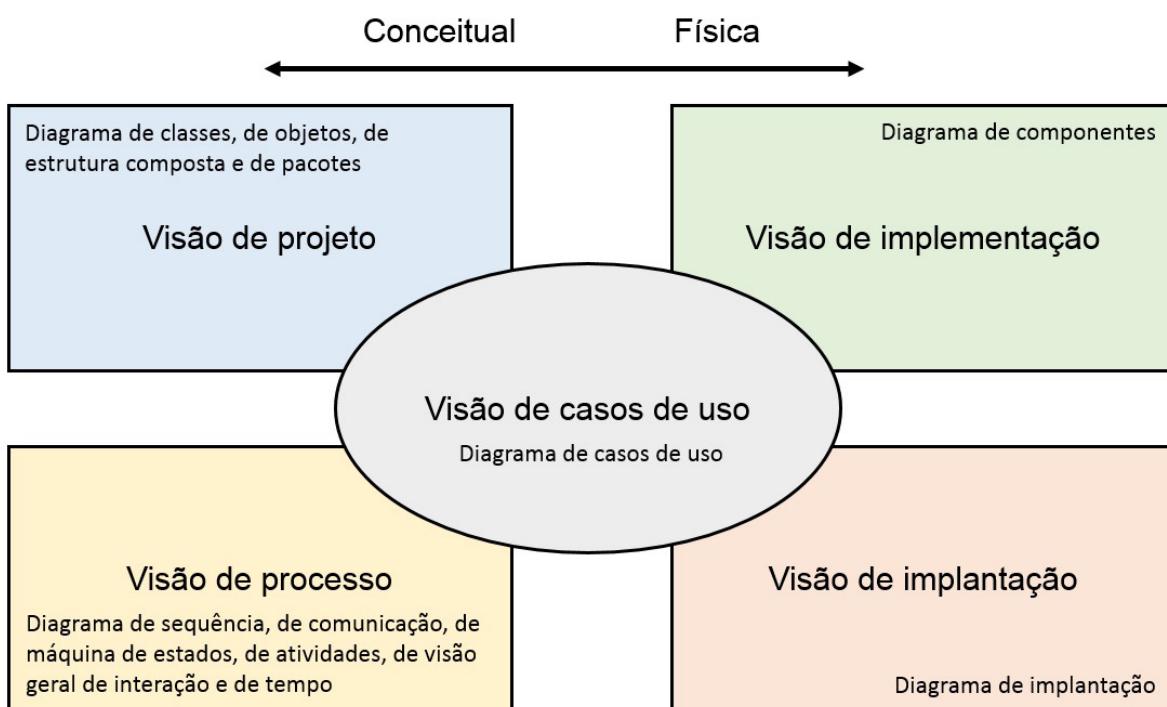


Figura 16: Visões da UML.
Fonte: Adaptado pelo autor, a partir de Bezerra (2015).

A visão de casos de uso descreve o sistema sob a perspectiva das interações entre o sistema e os agentes externos a ele, tais como os usuários e outros sistemas. Ela é obtida a partir do diagrama de casos de uso, que é construído nos estágios iniciais do desenvolvimento, para direcionar a modelagem das demais visões do sistema.

Os diagramas, relacionados no lado esquerdo da Figura 16, enfatizam o aspecto conceitual do sistema. Por exemplo, um diagrama de classes fornece uma visão de projeto, que permite enxergar como o sistema e seus objetos estão estruturados; ou um diagrama de sequência fornece uma visão de processo, a qual revela como os objetos do sistema interagem durante a execução de um caso de uso.

Os diagramas, relacionados no lado direito da Figura 16, enfatizam o aspecto físico do sistema. O diagrama de componentes fornece a visão de implementação, que revela como componentes e/ou subsistemas devem ser combinados para formar um sistema. Por fim, o diagrama de implantação fornece a visão de implantação, permitindo visualizar como as funções do sistema serão atribuídas aos componentes de hardware.

2.3 DIAGRAMAS ESTRUTURAIS

Na Seção 2.1.3, você viu que a UML versão 2.5.1 fornece catorze diagramas. Sete deles são voltados à modelagem estrutural e sete à modelagem comportamental. A partir de agora, vamos estudar, detalhadamente, cada um destes diagramas, a começar pelos diagramas estruturais, que descrevem os elementos de um sistema independente de tempo.

2.3.1 Diagrama de classes

De acordo com Dennis, Wixom e Roth (2014), uma classe é o modelo usado para definir e criar objetos. No projeto do sistema Loja ADS, por exemplo, podemos definir que cada pedido, realizado no sistema, será um objeto da classe “Pedido”, assim como cada cliente, que se cadastrar no sistema, será um objeto da classe “Corporativo” ou “Individual”. Em outras palavras, uma classe pode ser entendida como uma “planta” para a construção de um determinado tipo de objeto. É ela que determina quais serão as características e comportamentos dos objetos, construídos a partir dela.

Basicamente, um diagrama de classes descreve o conjunto de classes do sistema e seus relacionamentos (Figura 17). Cada item do tipo classe deve conter um identificador, podendo conter um conjunto de atributos, com seus respectivos tipos, e um conjunto de métodos, que podem ser invocados durante a execução do sistema. Para descrever os relacionamentos, são utilizadas as notações já estudadas na Seção 2.1.2. Perceba que, assim como no modelo conceitual de dados que elaboramos na Seção 1.2.3, no diagrama de classes também é possível informar a cardinalidade das relações.

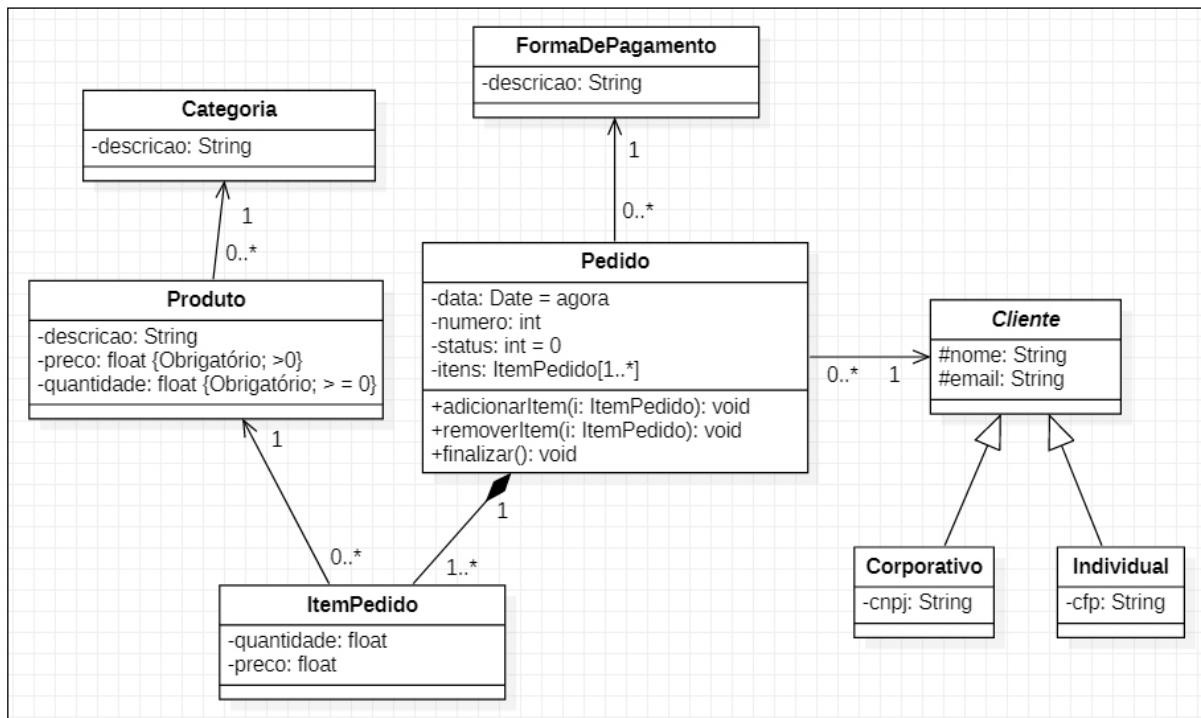


Figura 17: Exemplo de diagrama de classes.

Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

O diagrama de classes é um dos mais utilizados na modelagem de sistemas de software, pois ele fornece a visão estrutural, tanto do sistema como das classes. Uma boa estratégia para a construção deste diagrama é a de iniciar o trabalho com a modelagem de alto nível, descrevendo os identificadores das classes e os relacionamentos, e, depois, realizar o refinamento estrutural, descrevendo os atributos e métodos de cada classe.



PARA REFLETIR

No diagrama de classes apresentado na Figura 17, apenas os métodos da classe “Pedido” foram descritos. Pense sobre quais métodos poderiam ser adicionados nas classes “Cliente” e “Produto” e apresente alguns exemplos.

2.3.2 Diagrama de estrutura composta

O diagrama de estrutura composta desempenha um papel semelhante ao diagrama de classes, com a diferença de que ele permite detalhar as partes internas das classes.

Por exemplo, vamos imaginar a criação de uma nova classe, para o sistema da loja ADS, chamada “GerenciadorDeEstoque”. Esta classe terá que se relacionar com diversas classes do sistema, a fim de atingir seus objetivos. Por exemplo: (I) com a classe “Produto”, para saber a quantidade disponível em estoque de cada produto; (II) com a classe “ItemPedido”, para saber a quantidade de produtos que foram vendidos. A Figura 18 mostra como esta nova classe poderia ser representada em um diagrama de estrutura composta.

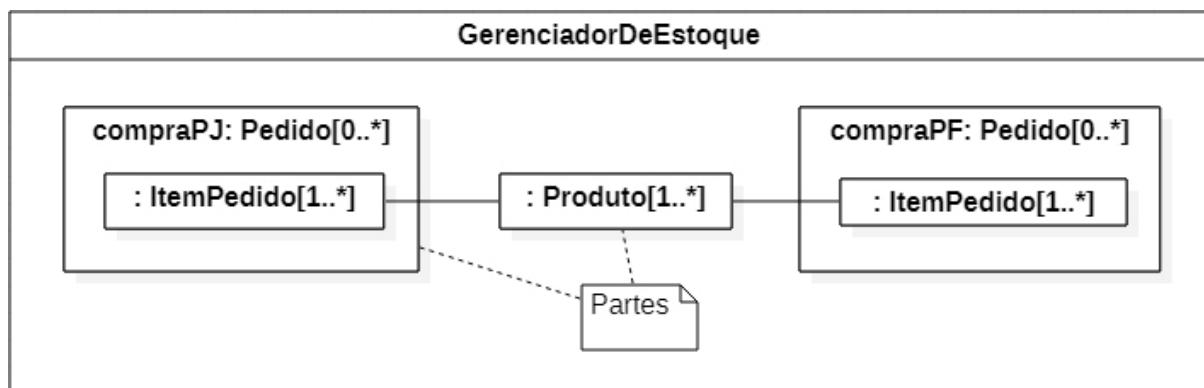


Figura 18: Exemplo de diagrama de estrutura composta.
Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

Os itens do tipo “Parte” mostram as partes internas da classe. Perceba que a classe “ItemPedido” não está diretamente contida na classe “GerenciadorDeEstoque”, ela é uma parte da classe “Pedido”. No entanto, podemos mostrar a sua relação com a classe “Produto”, a qual está contida diretamente na classe “GerenciadorDeEstoque”.

2.3.3 Diagrama de objetos

Na Seção 2.3.1, você viu que um diagrama de classes permite enxergar como o sistema e seus objetos estão estruturados. No entanto, ao olhar novamente para a Figura 17, você saberia responder se o CPF dos clientes individuais será armazenado no formato “ooo.ooo.ooo-oo” ou no formato “oooooooooooo”? Pois bem, o diagrama de objetos é capaz de responder a estes e outros questionamentos, conforme demonstra a Figura 19.

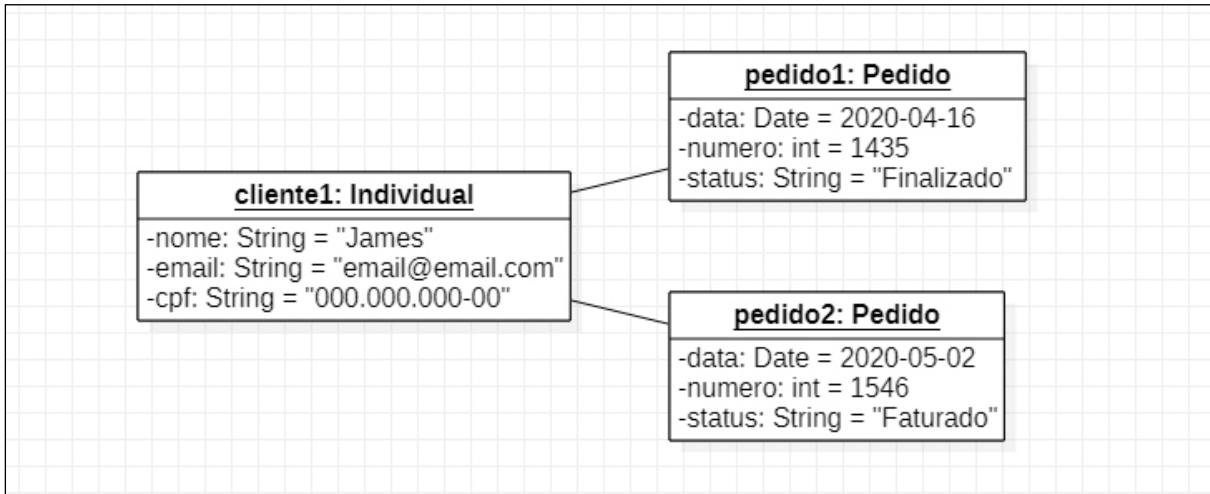


Figura 19: Exemplo de diagrama de objetos.

Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

Segundo Silva (2009), o diagrama de objetos é uma variação do diagrama de classes, que mostra as instâncias das classes, em algum momento da execução do sistema. Assim, o diagrama de objetos, também chamado de diagrama de instâncias, pode ser visto como uma “captura” do perfil do sistema, que mostra não só os valores das instâncias dos objetos, mas também como elas se relacionam entre si.

2.3.4 Diagrama de pacotes

Quando uma modelagem UML atinge uma grande quantidade de itens, como classes ou componentes, torna-se boa ideia agrupar estes itens em pacotes, a fim de facilitar a leitura da modelagem. De acordo com Fowler (2005, p. 96), um pacote é uma “construção de agrupamento que permite a você pegar qualquer construção UML e agrupar seus elementos em unidades de nível mais alto”.

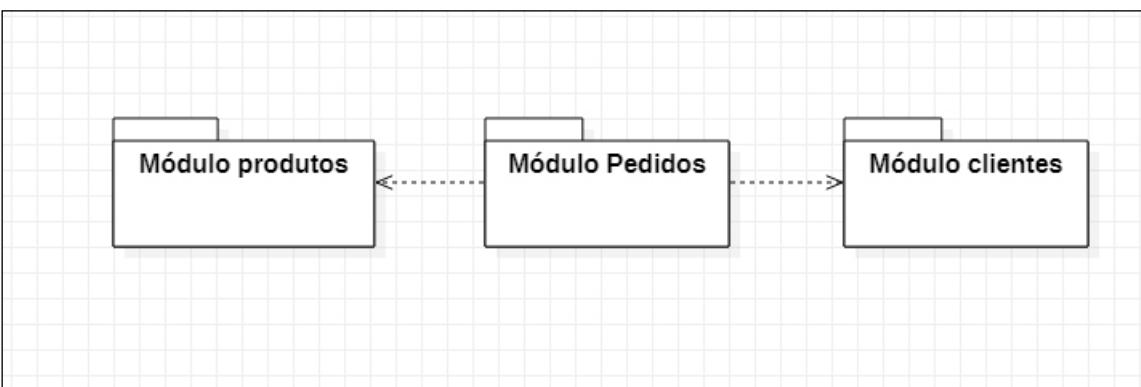


Figura 20: Exemplo de diagrama de pacotes.

Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

Diversos critérios de agrupamento de construções UML podem ser utilizados. O exemplo da Figura 20 indica uma possível separação das classes do sistema Loja ADS em três módulos: produtos, pedidos e clientes. Observe que o diagrama de pacotes permite mostrar dependências. Em nosso exemplo, temos que o pacote “Módulo pedidos” depende dos pacotes “Módulo produtos” e “Módulo clientes”.

2.3.5 Diagrama de implantação

Na Seção 1.3.1, utilizamos uma ferramenta de desenho comum, para esboçar o design de arquitetura do sistema Loja ADS. No entanto, veja, na Figura 21, que a UML também fornece um diagrama para esta finalidade.

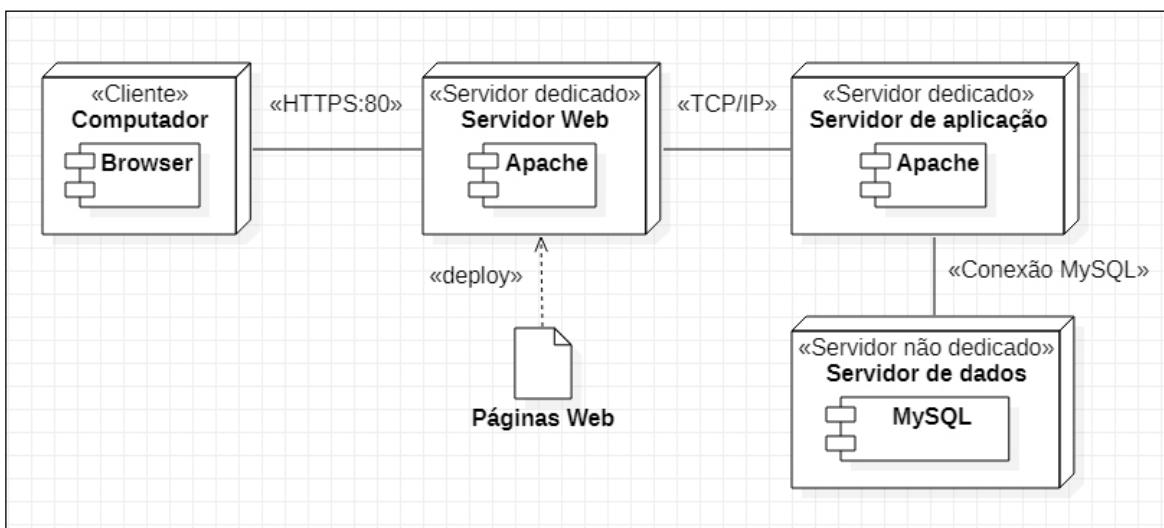


Figura 21: Exemplo de diagrama de implantação.
Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

O diagrama de implantação, também chamado de diagrama de instalação, mostra o layout físico de um sistema, revelando como as partes de software são distribuídas entre as partes de hardware. Os principais itens do diagrama são os nós e os caminhos de comunicação que conectam os nós. De acordo com Fowler (2005), um nó pode aparecer como um dispositivo de hardware, que contém algum software (browser, apache, MySQL), ou como um software, o qual contém a si mesmo. Os artefatos são os itens que representam as manifestações físicas dos softwares, por exemplo, os arquivos das páginas Web.

2.3.6 Diagrama de perfil

De acordo com Guedes (2014), o diagrama de perfil fornece um conjunto de elementos, os quais permitem estender a semântica dos elementos pré-existentes na UML. Com isso, podemos incluir estruturas customizadas às necessidades específicas de cada projeto. Se você olhar novamente para o diagrama de implantação da Figura 21, perceberá que, na parte superior de cada nó, foi adicionado um estereótipo. Tais estereótipos foram definidos no diagrama de perfil, apresentado na Figura 22 a seguir.

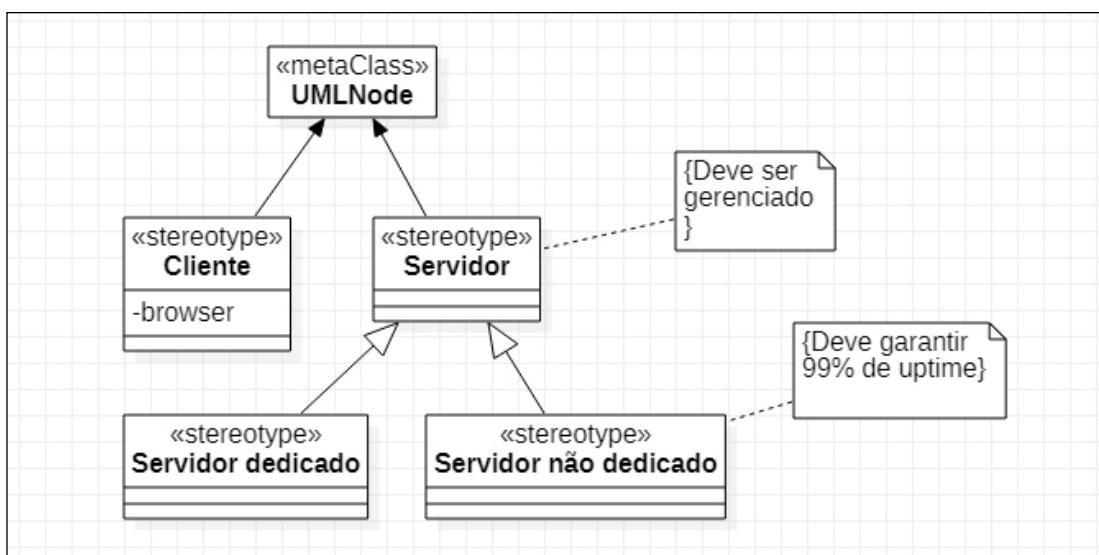


Figura 22: Exemplo de diagrama de perfil.

Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

Os estereótipos “Cliente” e “Servidor” são extensões da metaclasses “Node” da UML. Por sua vez, os estereótipos “Servidor dedicado” e “Servidor não dedicado” são nós especializados, que herdam as características do nó “Servidor”. As anotações associadas aos nós, do tipo “Servidor” e “Servidor não dedicado”, representam as restrições que devem ser observadas na implantação do sistema.

2.3.7 Diagrama de componentes

O diagrama de componentes é voltado à modelagem de sistemas de software, baseados em componentes, ou seja, sistemas que fazem o reúso de software (SILVA, 2009).

No caso do projeto Loja ADS, não estamos diante de um desenvolvimento baseado em

componentes. No entanto, podemos pensar no sistema de comércio eletrônico como um grande componente, utilizando um diagrama de componentes para mostrar como será a sua integração com os sistemas externos, conforme a Figura 23.

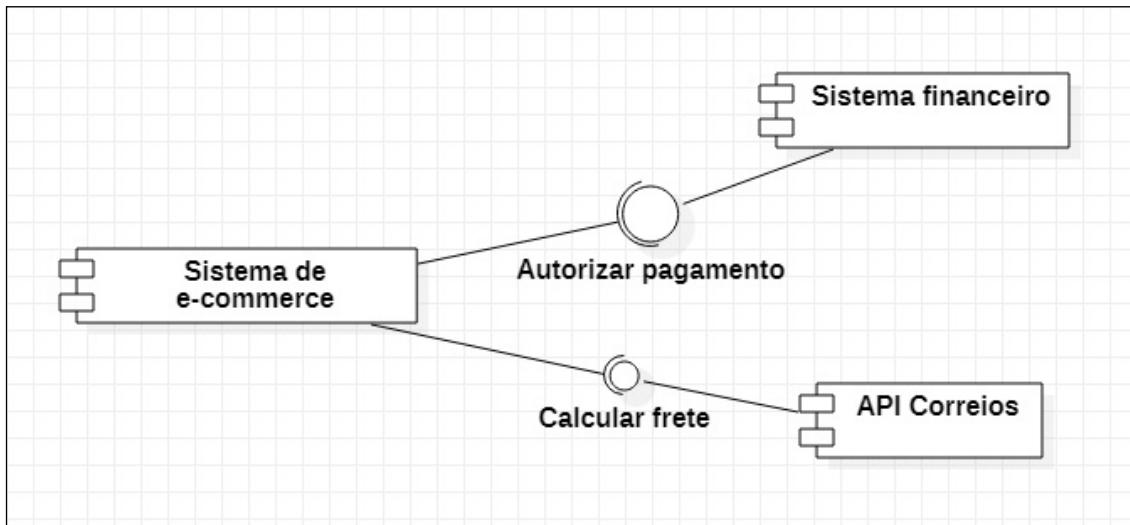


Figura 23: Exemplo de diagrama de componentes.
Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

No diagrama acima, cada componente é representado por um retângulo, com dois retângulos menores no lado esquerdo. Um componente pode apresentar interfaces requeridas, indicadas por semicírculos, e interfaces fornecidas, indicadas por círculos completos. No caso do sistema Loja ADS, o sistema bancário fornece uma interface “Autorizar pagamento” e a API Correios fornece uma interface “Calcular frete”. O sistema de e-commerce requer estas duas interfaces, para que um pedido de compra possa ter seu valor calculado e seu pagamento aprovado.

2.4 DIAGRAMAS COMPORTAMENTAIS

Nesta seção, iniciaremos o estudo dos diagramas comportamentais da UML versão 2.5.1, de acordo com a taxonomia apresentada na Seção 2.1.3.

Os diagramas comportamentais da UML são aqueles que mostram o comportamento dinâmico dos elementos de um sistema, incluindo seus métodos, atividades, colaborações e histórico de estados.

2.4.1 Diagrama de casos de uso

Sommerville (2011) explica que os casos de uso são uma técnica de elicitação de requisitos, a qual busca identificar as interações entre o sistema e seus usuários e/ou outros sistemas. Na UML, os casos de uso são documentados pelo diagrama de casos de uso, que dá nome às interações e identifica os atores envolvidos com cada interação. Observemos o exemplo da Figura 24.

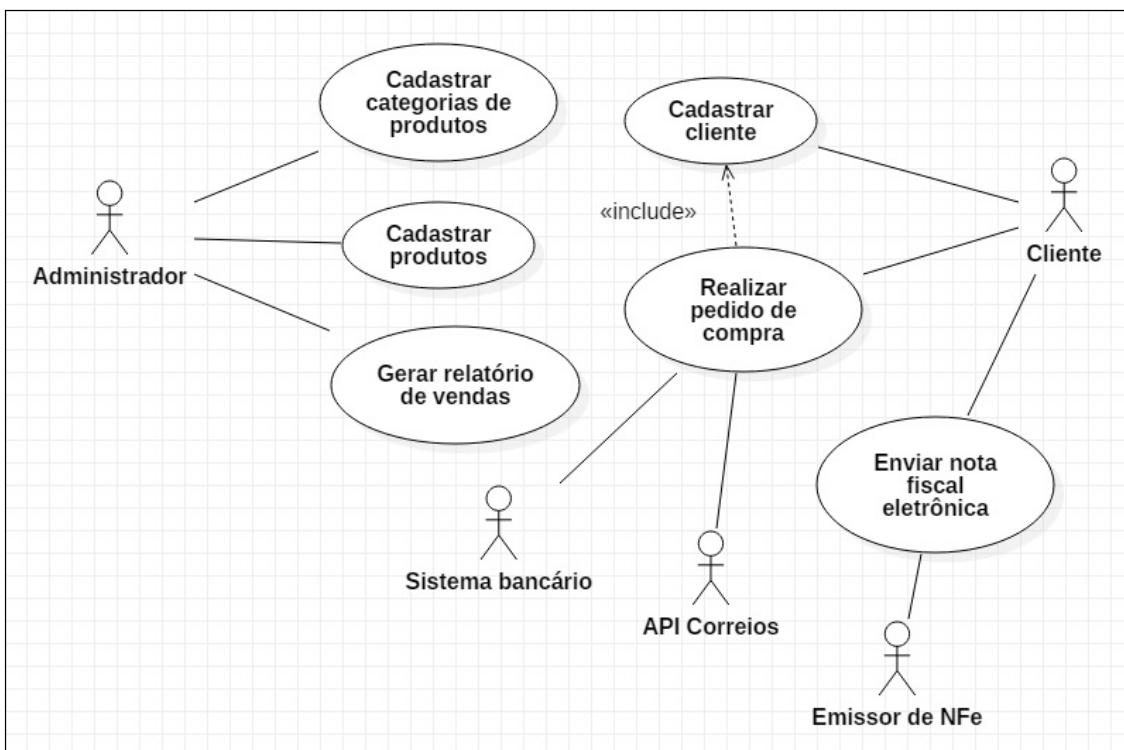


Figura 24: Exemplo de diagrama de casos de uso.

Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

As elipses representam os casos de uso, os bonecos de palito representam os atores, e as linhas contínuas representam as associações. O diagrama pode conter relações de inclusão («include») e extensão («extends») entre as elipses. A relação de inclusão é utilizada, quando a execução de um caso de uso inclui a execução de outro caso de uso como subcaso, e a notação de extensão é utilizada para indicar um comportamento excepcional, disparado por alguma condição. Os detalhes sobre como cada iteração ocorre são apresentados em um documento de detalhamento de casos de uso.

2.4.2 Diagrama de atividades

O diagrama de atividades fornece a visão dinâmica de um processo de sistema, por meio da descrição das atividades que compõem o processo e do fluxo de controle de uma atividade para a outra (SOMMERVILLE, 2011). Por exemplo, as atividades do caso de uso “Realizar pedido de compra” (Figura 24) estão descritas no diagrama de atividades, ilustrado na Figura 25.

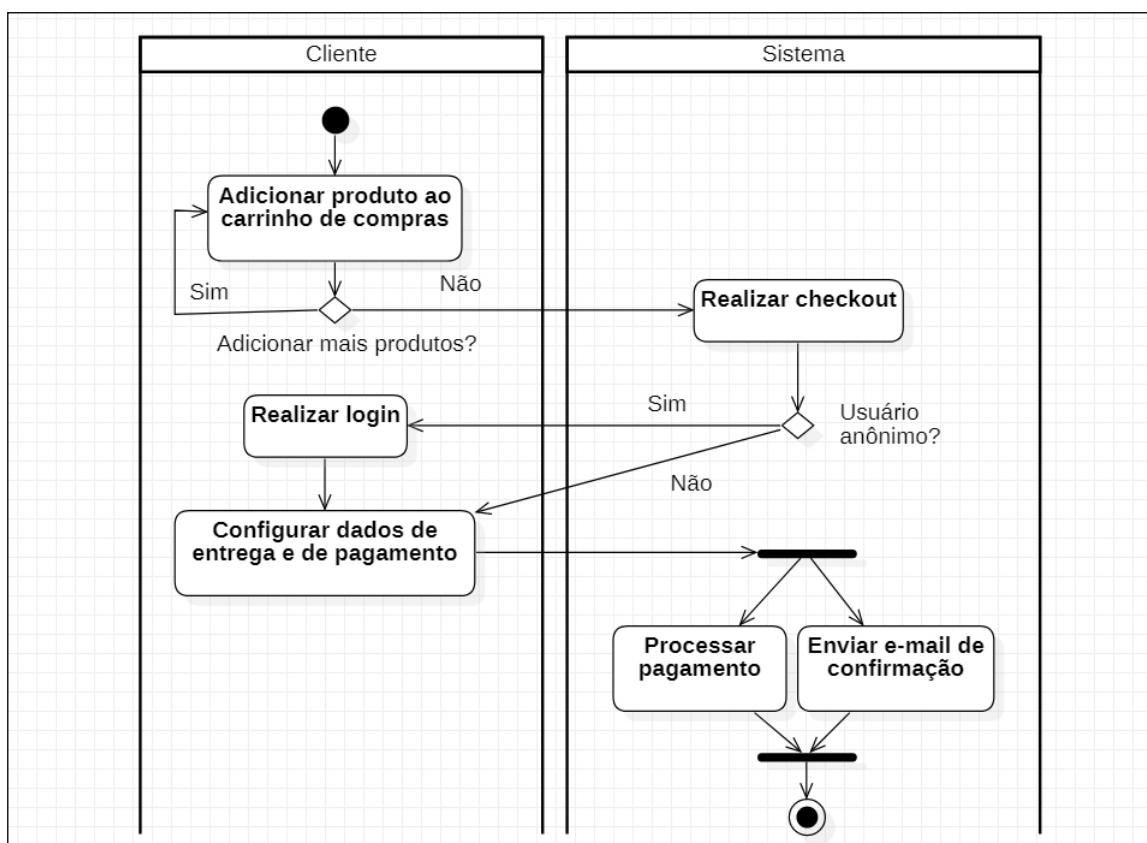


Figura 25: Exemplo de diagrama de atividades.

Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

Os retângulos com cantos arredondados indicam as atividades que compõem o processo. As setas são usadas para demonstrar o fluxo de atividades, que inicia no ponto indicado por um círculo preenchido e termina no ponto indicado por um círculo preenchido dentro de outro círculo. O losango é o símbolo usado para indicar tanto uma decisão como uma fusão. O que determina ser o símbolo uma decisão ou fusão é a quantidade de entradas e saídas. Enquanto uma decisão contém uma entrada e duas saídas, uma fusão pode conter duas ou mais entradas e apenas uma saída.

2.4.3 Diagrama de máquina de estados

Existem situações, no decorrer da execução de um processo, em que os objetos de um sistema podem assumir diferentes estados. Por exemplo, vamos imaginar que, no sistema Loja ADS, um pedido de compra possa assumir os estados “Criado”, “Aprovado”, “Faturado”, “Enviado”, “Finalizado” e “Cancelado”. Durante a fase de implementação do sistema, é muito provável que surjam perguntas do tipo: o pedido pode ser cancelado depois de faturado? Qual evento coloca o pedido no estado “Aprovado”? O diagrama de máquina de estados, da Figura 26, responde a estes questionamentos.

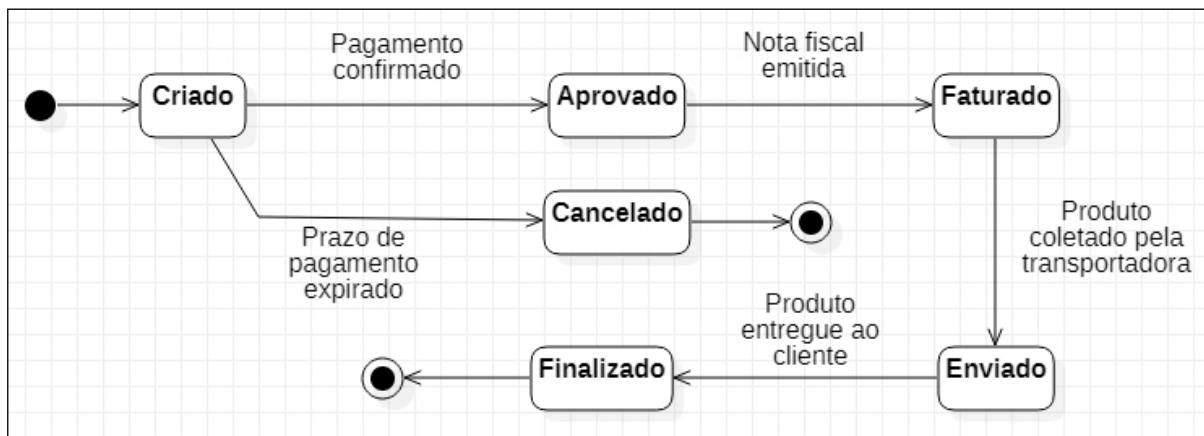


Figura 26: Exemplo de diagrama de máquina de estados.
Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

O diagrama inicia com um pseudo-estado (indicado por um círculo preenchido), o qual aponta para o estado inicial. As linhas com setas indicam as possíveis transições entre os estados. Associadas às linhas, estão as descrições dos gatilhos, os quais fazem com que as transições ocorram. O círculo preenchido dentro de outro círculo indica um estado final. Neste ponto, a máquina de estados está completa e o objeto que controla os estados pode ser destruído (FOWLER, 2005).

2.4.4 Diagrama de sequência

Na Seção 2.4.2, vimos que um diagrama de atividades descreve as atividades que compõem um processo de sistema e o fluxo de controle de uma atividade para a outra. No

entanto, um diagrama de atividades não revela como os objetos do sistema interagem, durante a execução de cada atividade.

Observando apenas o diagrama da Figura 25, você saberia me dizer quais objetos precisam ser criados durante a execução da atividade “Adicionar produto ao carrinho de compras”? E como estes objetos interagem?

Muito provavelmente não, pois para responder a tais perguntas, é necessário analisarmos o diagrama da Figura 27. Segundo Pressman e Maxim (2016), um diagrama de sequência é utilizado para mostrar como as mensagens são enviadas entre os objetos, durante a execução de um caso de uso de um sistema de software.

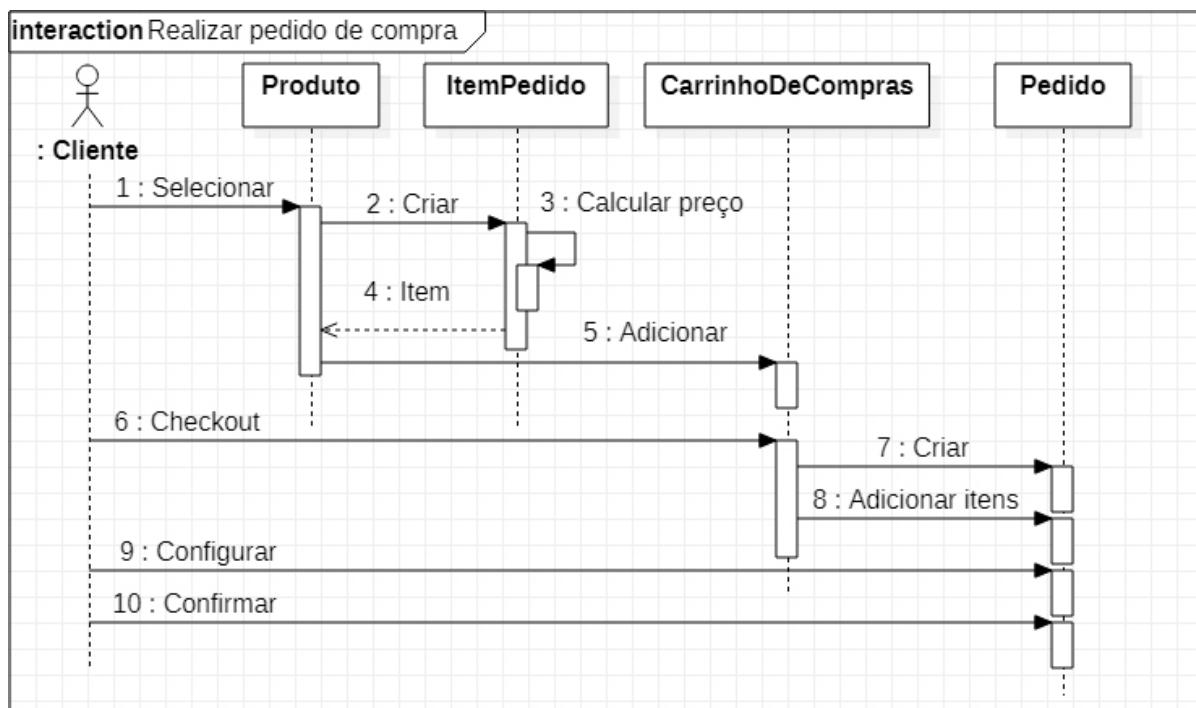


Figura 27: Exemplo de diagrama de sequência.

Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

Cada elemento, no topo do diagrama, corresponde a um objeto. Abaixo de cada objeto, há uma linha tracejada, chamada de linha de vida do objeto. O eixo vertical do diagrama corresponde ao tempo, o qual aumenta, à medida que se desloca para baixo. Um retângulo sobre a linha de vida de um objeto indica o período de tempo em que uma interação do objeto permanece ativa. No exemplo da Figura 27, o caso de uso inicia com o objeto “Cliente”, enviando a mensagem “Selecionar” para o objeto “Produto”, que por sua vez interage com o objeto “ItemPedido” e permanece ativo, até um novo item ser adicionado ao carrinho de compras.

2.4.5 Diagrama de comunicação

Na subseção anterior, você viu que o diagrama de sequência é utilizado para modelar a interação entre objetos, durante a execução de um caso de uso. O diagrama de comunicação tem a mesma finalidade, contudo ele apresenta maior ênfase na ligação entre os objetos do que na ordem em que as mensagens ocorrem. Observe, por exemplo, que é mais fácil identificar com quais objetos o objeto “Cliente” interage – através do diagrama de comunicação, exibido na Figura 28 – a olhar para o diagrama de sequência da Figura 27.

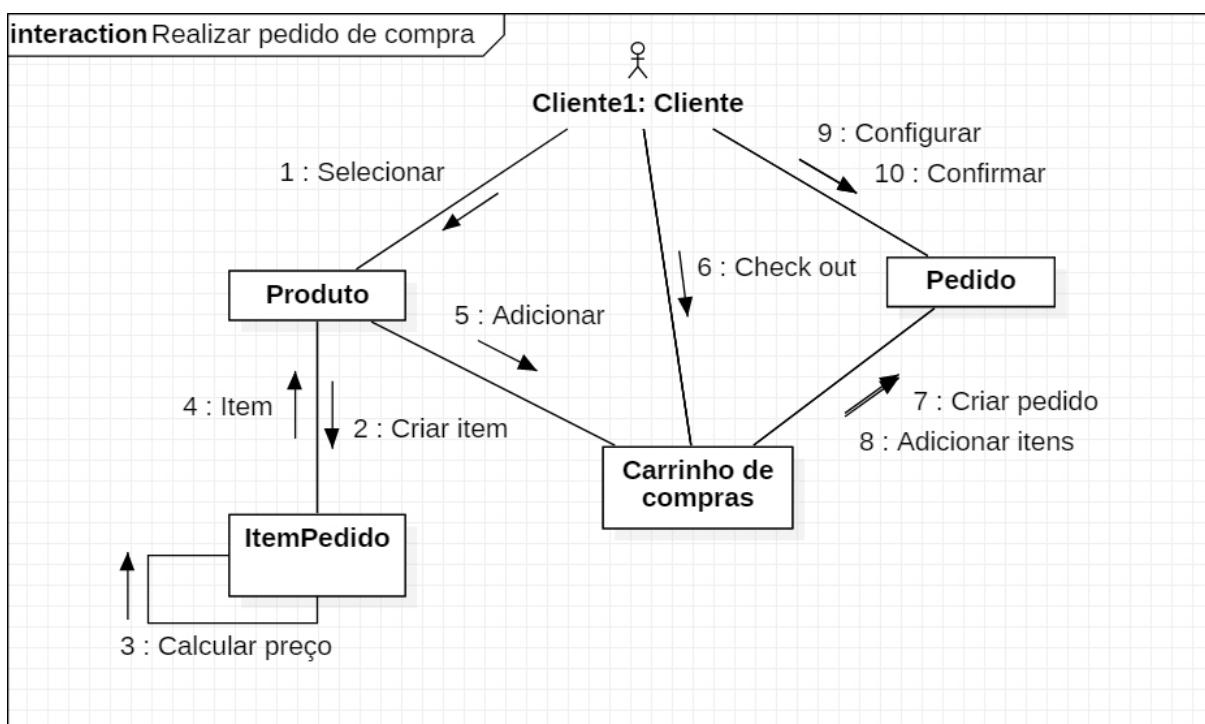


Figura 28: Exemplo de diagrama de comunicação.
Fonte: Elaborado pelo autor, com o uso da ferramenta StarUML (2020).

De acordo com Fowler (2005), na UML 1.x, o diagrama de comunicação era chamado de diagrama de colaboração. Por este motivo, você ainda poderá encontrar o termo “diagrama de colaboração” sendo usado em materiais de estudo.

2.4.6 Diagrama de visão geral de interação

Nas Seções 2.4.2 e 2.4.4, estudamos que o diagrama de atividades descreve as atividades que compõem um processo de sistema, enquanto o diagrama de sequência descreve a interação entre os objetos, durante a execução do processo.

O diagrama de visão geral de interação é uma variação do diagrama de atividades, em que interações substituem as atividades (SILVA, 2009). Ou seja, é um diagrama de atividades que revela como os objetos do sistema interagem, durante cada atividade, por meio de diagramas de sequência, anexados ou referenciados. Observe a Figura 29:

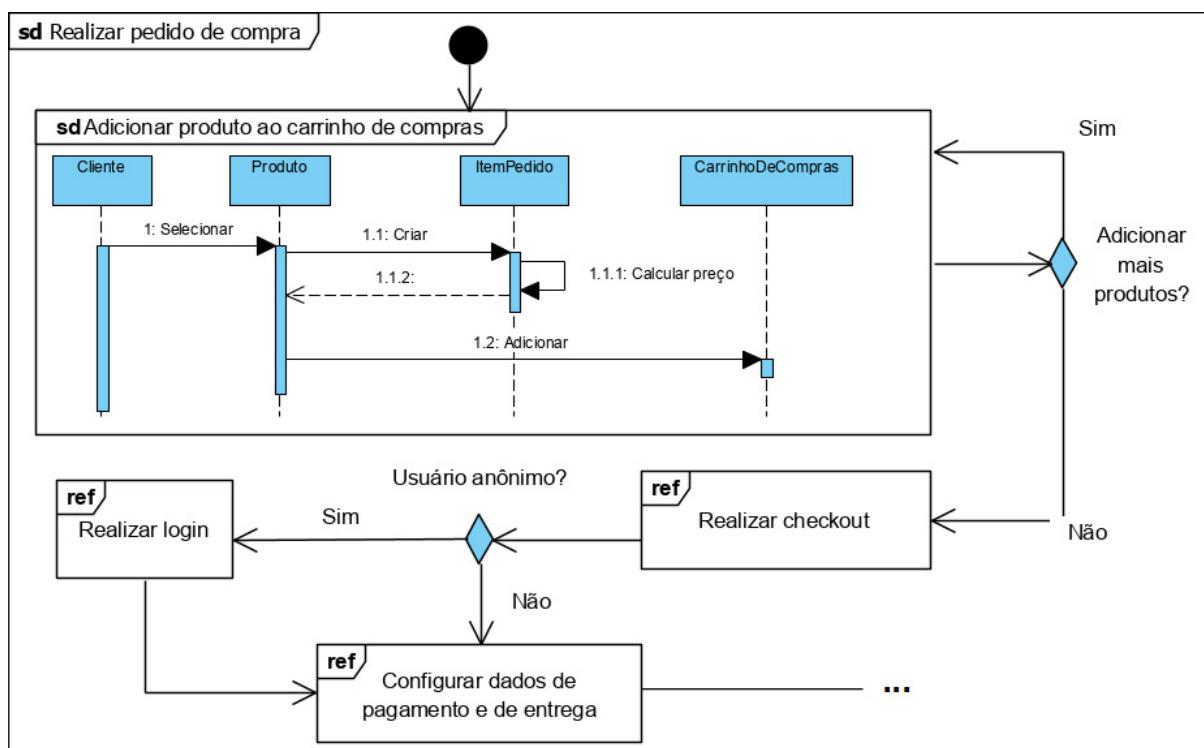


Figura 29: Exemplo de diagrama de visão geral de interação.

Fonte: Elaborado pelo autor, com o uso da ferramenta Visual Paradigm Enterprise (2020).

O nó de um diagrama de visão geral de interação pode ser apresentado como um quadro de interação (sinalizado com "sd"), mostrando qualquer tipo de diagrama de interação da UML, ou como um quadro de ocorrência (sinalizado com "ref"), que mostra apenas a referência a um diagrama de interação, sem detalhá-lo.

2.4.7 Diagrama de tempo

Um diagrama de tempo, também chamado de diagrama de temporização, é voltado para a modelagem das restrições temporais de um sistema (SILVA, 2009). Vamos imaginar que, na modelagem do sistema Loja ADS, precisamos deixar claro que uma consulta do banco de dados não pode demorar mais que 500ms para ser processada pelo servidor de banco de dados, e que uma página de produtos do sistema de comércio eletrônico não pode demorar mais que 3000ms (ou 3 segundos) para ser exibida ao usuário. Podemos representar tais restrições temporais, a partir do diagrama, exposto na Figura 30.

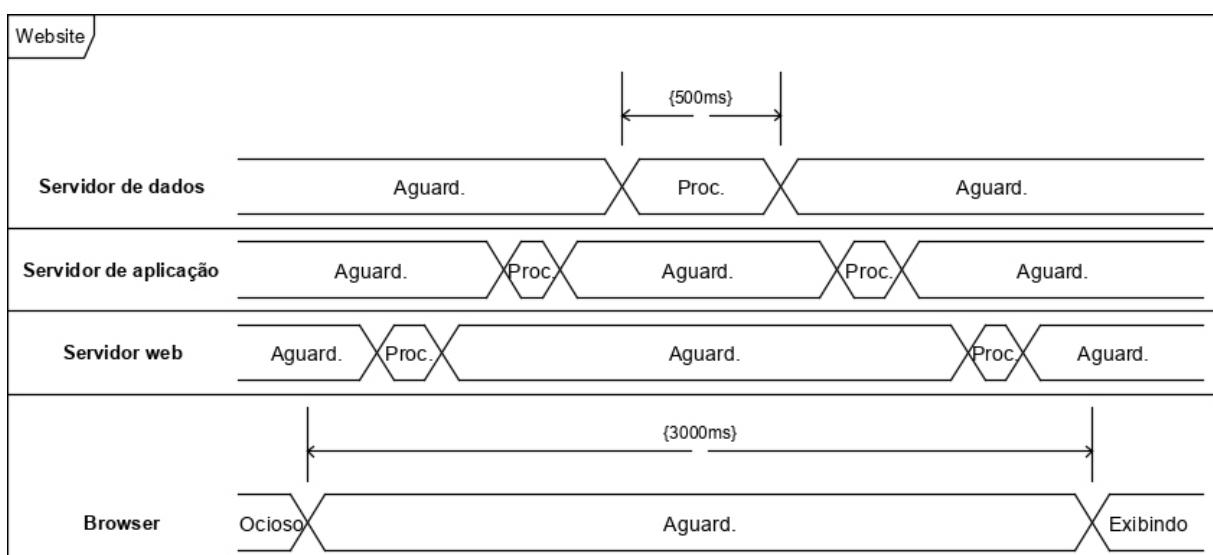


Figura 30: Exemplo de diagrama de tempo.

Fonte: Elaborado pelo autor, com o uso da ferramenta Visual Paradigm Enterprise (2020).

O diagrama de tempo também permite modelar a evolução dos estados de cada objeto ao longo do tempo. Perceba que o objeto “Browser” inicia sua execução no estado “Ocioso”. Quando uma requisição é feita ao servidor Web, ele entra no estado “Aguardando” e, ao receber a resposta, no estado “Exibindo”.

Ufa! Chegamos ao final do estudo dos diagramas da UML versão 2.5.1. A UML é um assunto extenso e suficiente para um livro com centenas de páginas. Por isso, o foco das Seções 2.3 e 2.4 foi apresentar a você uma visão geral de cada diagrama. À medida que for praticando a modelagem de sistemas, você descobrirá outros elementos do vocabulário UML. Ao final desta unidade, há uma sugestão de livro para que possa ampliar os seus conhecimentos, quando necessário.



EXERCÍCIO

Com base nos estudos realizados até agora, descreva brevemente qual a finalidade de cada um dos diagramas da UML 2.5.1, listados a seguir.

Diagrama	Finalidade
Classes	
Estrutura composta	
Objetos	
Pacotes	
Implantação	
Perfil	
Componentes	
Casos de uso	
Atividades	
Máquina de estados	
Sequência	
Comunicação	
Visão geral de interação	
Tempo	

2.5 FÓRUM

Nesta unidade, utilizamos os 14 diagramas da UML 2.5.1 para enriquecer o projeto do sistema Loja ADS. Utilize o fórum on-line da disciplina, para discutir a importância dos diagramas apresentados ao projeto do sistema Loja ADS.

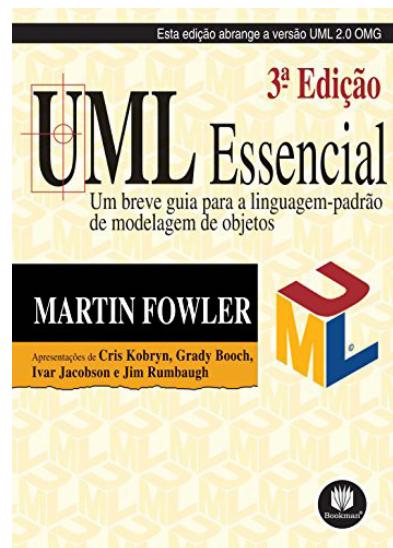
Para tal, considere um cenário real de desenvolvimento, em que normalmente os projetos de software são pressionados pelos fatores tempo e custo de desenvolvimento. Quais diagramas UML poderiam deixar de ser elaborados?



SUGESTÃO DE LIVRO

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

O livro "UML essencial" descreve a maioria dos tipos de diagramas UML, sua utilização e a notação básica, envolvida na criação. Os exemplos são claros e as explicações chegam ao fundamento lógico do projeto.



SUGESTÃO DE FILME

Urbanized (2011)

O documentário "Urbanized", dirigido por Gary Hustwit, introduz a problemática do rápido crescimento demográfico e a concentração das populações nas grandes cidades, apresentando conversas com arquitetos e urbanistas, como Norman Foster, Rem Koolhaas, Oscar Niemeyer e Alejandro Aravena. O documentário desperta a reflexão sobre a importância de se projetar um sistema, pensando na sua evolução.

CONSIDERAÇÕES FINAIS

Na segunda unidade de nosso estudo, você entendeu que a UML é uma linguagem utilizada para modelar e documentar sistemas de software, por meio do paradigma de orientação a objetos. Na sua versão 2.5.1, a UML fornece um conjunto de catorze diagramas para modelagem das características de um sistema. Sete deles são voltados à modelagem estrutural e sete, à modelagem de comportamento, sendo que cada diagrama é capaz de fornecer uma diferente visão sobre o sistema em desenvolvimento.

Dependendo da complexidade e das características do sistema, nem todos os

diagramas precisam ser construídos. Por exemplo, se o sistema tiver que ser executado em um ambiente computacional não distribuído, o diagrama de implantação torna-se desnecessário; caso o sistema seja constituído de processos com poucas interações entre objetos, os diagramas de sequência e de visão geral de interação tornam-se menos importantes. Além disso, o grau de detalhamento de cada diagrama pode ser maior ou menor, dependendo do grau de compreensão necessário para o desenvolvimento do sistema.

Em nossa próxima unidade, vamos conhecer algumas ferramentas que auxiliam os desenvolvedores nas atividades de engenharia de software.

Até lá!

EXERCÍCIO FINAL

1. De acordo com Booch, Rumbaugh e Jacobson (2005), o vocabulário da UML abrange os três tipos de blocos de construção: itens, relacionamentos e diagramas. Assim, construir um diagrama UML é nada mais que reunir um conjunto de itens, relacionamentos e textos, em um diagrama, sendo que cada tipo de diagrama suporta um determinado subconjunto de itens e relacionamentos do vocabulário UML.

Em relação aos blocos de construção, citados no texto, avalie as afirmações a seguir:

- I. Os estados e as mensagens são relacionamentos de dependência;
- II. As classes, casos de uso, componentes e nós são itens estruturais;
- III. A agregação e a composição são itens de agrupamento.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) I e III, apenas.
- (E) I, II e III.

2. Uma modelagem de sistema de software deve permitir que as pessoas, as quais desenvolverão o sistema, tenham a possibilidade de examiná-lo e estudá-lo, a partir de diferentes perspectivas, denominadas visões UML. Bezerra (2015) apresenta os cinco

tipos de visões de modelagem, a saber: visão de casos de uso, visão de projeto, visão de implementação, visão de processo e visão de implantação.

Em relação aos diagramas UML que podem ser utilizados para proporcionar as visões apresentadas no texto, avalie as afirmações a seguir:

- I. O diagrama de classes proporciona uma visão de projeto;
- II. O diagrama de sequência proporciona uma visão de processo;
- III. O diagrama de componentes proporciona uma visão de casos de uso.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) I e III, apenas.
- (E) I, II e III.

3. A UML (*Unified Modeling Language*) ou Linguagem de Modelagem Unificada é uma notação utilizada para modelar sistemas computacionais, por meio do paradigma de orientação a objetos. A versão 2.5.1 da UML fornece um conjunto de 14 diagramas para modelagem das características de um software, sendo 7 deles voltados à modelagem estrutural e 7, à modelagem de comportamento.

Em relação ao conjunto de diagramas da UML 2.5.1, analise as seguintes afirmações:

- I. Os diagramas estruturais mostram a estrutura estática dos elementos de um sistema;
- II. Os diagramas de comportamento são usados para descrever como os elementos do sistema se comportam e interagem em tempo de execução;
- III. O diagrama de classes é um diagrama estrutural, e o diagrama de casos de uso é um diagrama de comportamento.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) I e III, apenas.
- (E) I, II e III.

REFERÊNCIAS

- BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. [Recurso Eletrônico]. 3. ed. Rio de Janeiro: Elsevier, 2015.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML: Guia do usuário**. Rio de Janeiro: Elsevier, 2005.
- DENNIS, Alan; WIXOM, Barbara Haley; ROTH, Roberta M. **Análise e projeto de sistemas**. [Recurso Eletrônico]. 5. ed. Rio de Janeiro: LTC, 2014.
- FOWLER, Martin. **UML Essencial: Um breve guia para a linguagem-padrão de modelagem de objetos**. 3. ed. Porto Alegre: Bookman, 2005.
- GUEDES, Gilleanes T. A. **UML 2: Guia prático**. 2. ed. São Paulo: Novatec, 2014.
- LARMAN, Craig. **Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo**. [Recurso Eletrônico]. 3. ed. São Paulo: Bookman, 2007.
- OMG - OBJECT MANAGEMENT GROUP. **Unified Modeling Language version 2.5.1**. 2017. Disponível em: <<https://www.omg.org/spec/UML/2.5.1/PDF>>. Acesso em: 15 abr 2020.
- PRESSMAN, Roger S; MAXIM, Bruce R. **Engenharia de Software: uma abordagem profissional**. [Recurso Eletrônico]. 8. ed. Porto Alegre: AMGH, 2016.
- SILVA, Ricardo Pereira e. **UML 2 – Modelagem Orientada a Objetos**. São Paulo: Visual Books: 2008.
- SILVA, Ricardo Pereira e. **Como modelar com UML**. Florianópolis: Visual Books, 2009.
- SOMMERVILLE, Ian. **Engenharia de software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

unidade



.....

3

FERRAMENTAS CASE

INTRODUÇÃO À UNIDADE

A disciplina de Engenharia de Software nos ensina que o desenvolvimento de *software* abrange: (I) um processo, composto de atividades que variam, conforme o modelo de processo adotado (em cascata, incremental, espiral); (II) um conjunto de métodos para a execução dessas atividades e (III) um conjunto de **ferramentas**, o qual fornece suporte automatizado ou semiautomatizado, tanto ao processo quanto aos métodos.

Nas unidades anteriores, você deve ter percebido que aplicamos diversas ferramentas nas atividades de análise e projeto do sistema loja ADS. Por exemplo, a ferramenta MySQL Workbench foi utilizada no design do modelo físico de dados e a ferramenta StarUML, na modelagem do sistema. Nesse sentido, caso implementássemos o sistema loja ADS, é certo que mais ferramentas seriam aplicadas.

Mas afinal, existe alguma denominação para esse conjunto de ferramentas que auxilia a engenharia de software? Quais são os tipos de ferramentas que podem estar presentes no conjunto? Estas e outras questões serão respondidas, ao longo da Unidade 3, cujos objetivos específicos são: (I) Conhecer a definição e classificação das ferramentas CASE; (II) Compreender a importância das ferramentas CASE para o processo de desenvolvimento de sistemas e (III) Adquirir habilidades para selecionar ferramentas CASE em diferentes cenários de desenvolvimento de sistemas.

3 FERRAMENTAS CASE

Em um projeto de software, a equipe pode utilizar diversas ferramentas para agilizar e documentar suas atividades em todas as fases de desenvolvimento. Quando a utilização das ferramentas é feita de forma **integrada**, isto é, podendo a informação produzida por uma ferramenta ser utilizada por outra, tem-se o cenário chamado de **Engenharia de Software Auxiliada por Computador** ou **CASE** (Computer-Aided Software Engineering) (BEZERRA, 2015; SBROCCO; MACEDO, 2012).

Assim, as ferramentas de software, utilizadas para apoiar as atividades da engenharia de software, são chamadas de **ferramentas CASE** (SOMMERVILLE, 2011). As ferramentas CASE podem auxiliar uma equipe, tanto nas atividades de desenvolvimento do software em si quanto nas atividades relacionadas aos aspectos organizacionais do projeto, tais como a elaboração de planos e especificações (SCHACH, 2010). Muito bem, mas quando surgiram as ferramentas CASE?

3.1 HISTÓRICO

No século passado, o desenvolvimento de aplicações já necessitava do auxílio de diferentes ferramentas, tais como: (I) os editores de texto, utilizados para a escrita do código da aplicação; (II) os interpretadores e compiladores, usados para traduzir o código, escrito em uma linguagem de programação de alto nível para uma linguagem comprehensível pelo computador e (III) os linkers, aplicados durante o processo de compilação, para combinar diversos arquivos de código em um único arquivo executável (Figura 31).

Silva e Videira (2001) relatam que as primeiras ferramentas, consideradas como sendo do universo CASE, surgiram no início da década de 1980. Elas tinham como principal preocupação o suporte a atividades de análise, modelagem, design e documentação. Neste período, surgiram ferramentas para a construção de Diagramas de Fluxo de Dados (DFD's), Diagramas Entidade-Relacionamento (ER's) e esquemas de Bancos de Dados (BD's).

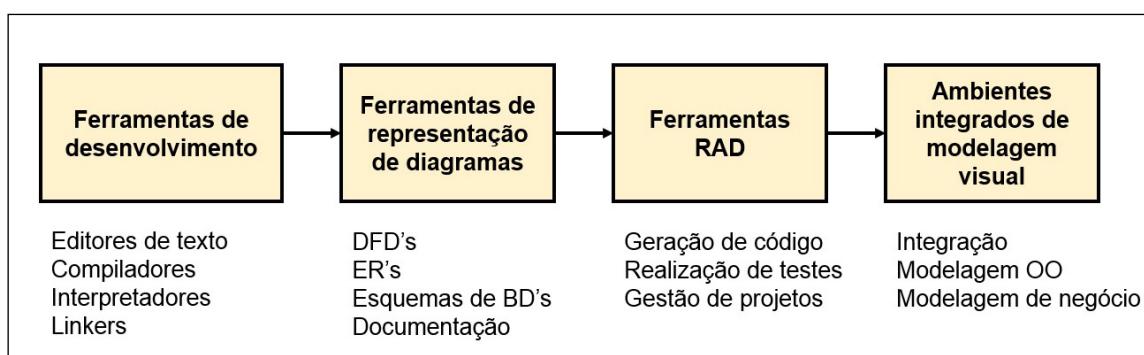


Figura 31: Evolução das ferramentas CASE.
Fonte: Adaptado pelo autor, a partir de Silva e Videira (2001).

Com a crescente preocupação em aumentar o ritmo de desenvolvimento de software, no início da década de 1990 muitas das ferramentas CASE passaram a se chamar ferramentas RAD (Rapid Application Development). Nessa fase do histórico, o foco do desenvolvimento de novas ferramentas estava no suporte a atividades de geração de código, realização de testes e gestão de projetos.

Em meados da década de 1990, muitos autores passaram a se referir às ferramentas CASE como ambientes integrados de modelagem, devido à crescente importância do paradigma de programação orientada a objetos. A partir daí, houve a necessidade de revolucionar o mercado CASE, com a introdução de ferramentas de modelagem de negócio e de modelagem orientada a objetos, bem como de técnicas de integração das ferramentas com as abordagens já existentes.

Foi nesse contexto que surgiu a noção de ambiente CASE, ou seja, um conjunto de ferramentas capaz de fornecer suporte para todas, ou para grande parte das atividades do processo de desenvolvimento de software, conforme veremos a seguir.

3.2 ARQUITETURA GENÉRICA

A arquitetura de um ambiente CASE é constituída, genericamente, por um conjunto de ferramentas CASE que auxiliam a engenharia de software em atividades e modelagem, desenvolvimento, testes e gestão (Figura 32). As ferramentas colaboram entre si, por meio de um repositório centralizado de artefatos, tais como templates (modelos genéricos), bibliotecas, diagramas, códigos fonte e quaisquer outros tipos de artefatos, consumidos ou produzidos pelas ferramentas que compõem a arquitetura (SILVA; VIDEIRA, 2001).

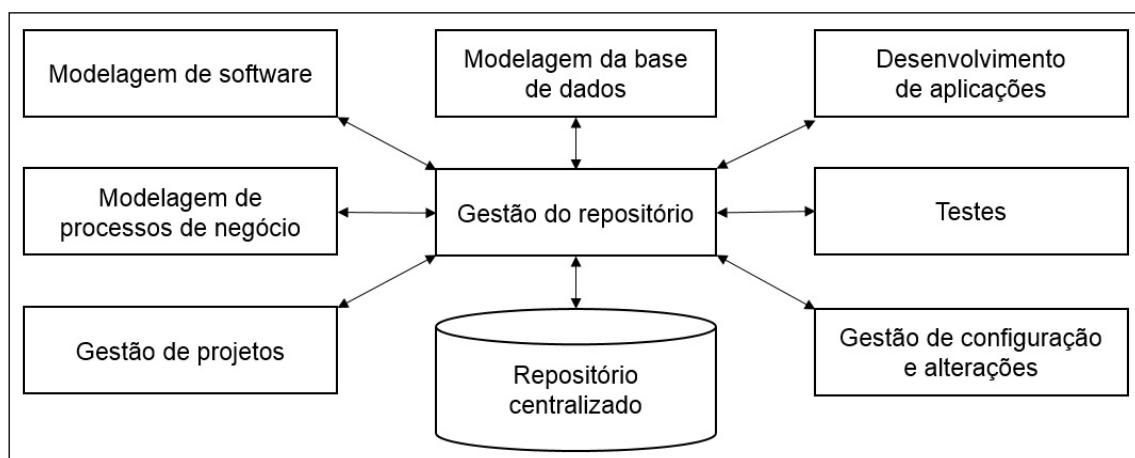


Figura 32: Arquitetura genérica de um ambiente CASE.
Fonte: Adaptado pelo autor, a partir de Silva e Videira (2001).

O repositório é um componente crítico de um ambiente CASE, pois ele precisa garantir o compartilhamento de artefatos, de forma concorrente, e sem oferecer riscos à integridade dos dados (SILVA; VIDEIRA, 2001). Normalmente, esse cenário ideal é mais factível, quando todas as ferramentas que compõem o ambiente CASE são de um mesmo fornecedor. Nos demais casos, é necessário desenvolver estratégias próprias de integração.

3.3 CLASSIFICAÇÃO

A maioria das ferramentas CASE especializa-se em uma atividade específica do processo de desenvolvimento de software (AMUI, 2015). A ferramenta Astah (astah.net), por exemplo, fornece recursos de modelagem que são mais utilizados nas fases de análise e projeto do software. Outro exemplo é um típico Ambiente de Desenvolvimento Integrado, ou IDE (do inglês Integrated Development Environment), que fornece recursos para agilizar e facilitar a fase de implementação (ou codificação) do software.

Com isso, podemos classificar uma ferramenta CASE sob três perspectivas: (I) quanto às fases que ela apoia (perspectiva de processo); (II) quanto à quantidade de atividades que ela apoia (perspectiva de integração) e (III) quanto aos tipos de atividades que ela apoia (perspectiva funcional). Vamos estudar cada uma destas perspectivas, detalhadamente, nas subseções a seguir.

3.3.1 Perspectiva de processo

Quanto às fases do processo de desenvolvimento de software que uma ferramenta CASE apoia, podemos classificá-la como **upperCASE**, **lowerCASE** ou **Integrated CASE**, conforme indicado na Figura 33. As ferramentas upperCASE ou Front End são aquelas que apoiam o desenvolvedor, nas etapas iniciais do processo de software, em atividades de planejamento, análise e projeto (SBROCCO; MACEDO, 2012). Um software para a criação de diagramas UML é um típico exemplo de ferramenta upperCASE.

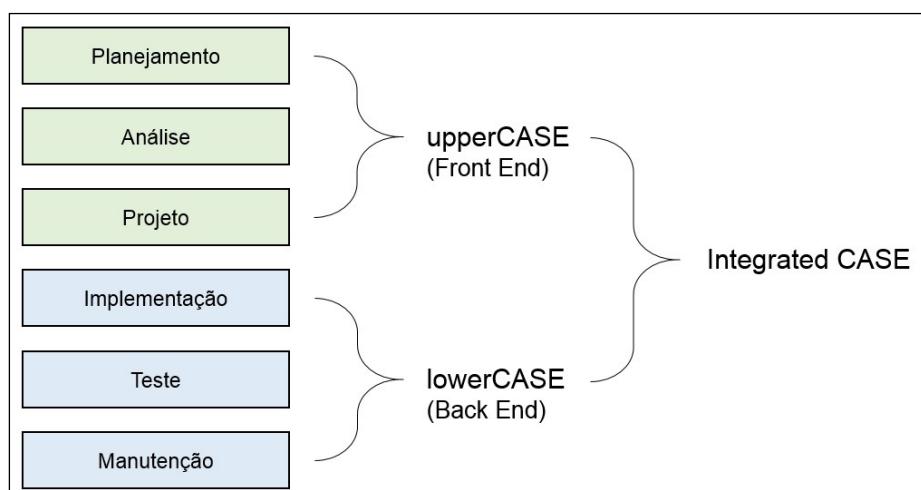


Figura 33: Classificação das ferramentas CASE, sob a perspectiva de processo.

Fonte: Adaptado pelo autor, a partir de Sbrocco e Macedo (2012).

Por sua vez, as ferramentas lowerCASE ou Back End são aquelas que apoiam o desenvolvedor, nas etapas finais do processo de software, em atividades de implementação, teste e manutenção (SBROCCO; MACEDO, 2012). Dentre as ferramentas lowerCASE estão os Ambientes Integrados de Desenvolvimento, utilizados para geração de código, bem como as ferramentas usadas para apoiar a manutenção pós-entrega do software. Por fim, são classificadas como Integrated CASE ou I-Case as ferramentas que apoiam todo o ciclo de vida do software, desde o planejamento até à manutenção.

3.3.2 Perspectiva de integração

Quanto à quantidade de atividades que uma tecnologia CASE apoia, podemos classificá-la como: **ferramenta**, **bancada de trabalho** ou **ambiente** (Figura 34). Em sua forma mais simples, uma tecnologia CASE se apresenta como uma ferramenta que suporta apenas uma tarefa específica do desenvolvimento de software, caso em que é chamada simplesmente de ferramenta (SCHACH, 2010).

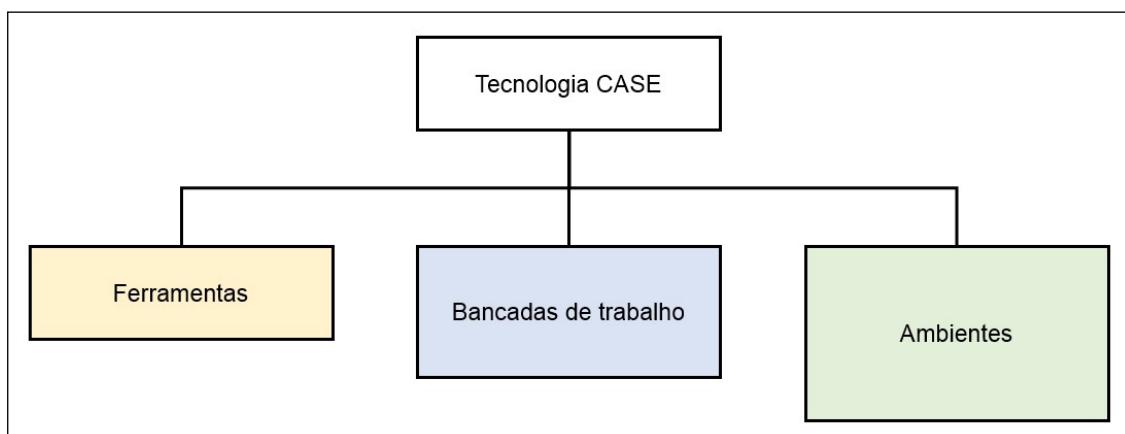


Figura 34: Classificação das ferramentas CASE sob a perspectiva de integração.

Fonte: Adaptado pelo autor, a partir de Schach (2010).

Quando uma tecnologia CASE se apresenta como um conjunto de ferramentas que suporta uma ou algumas das atividades do processo de desenvolvimento de software, dizemos que ela é uma bancada de trabalho. Por fim, um ambiente CASE é uma tecnologia que suporta todas, ou grande parte das atividades do processo de desenvolvimento de software (SCHACH, 2010).

3.3.3 Perspectiva funcional

Sob a perspectiva funcional, diversas classificações (SCHACH, 2010; AMUI, 2015; SILVA; VIDEIRA, 2001) para as ferramentas CASE já foram propostas. A Figura 35 apresenta uma visão genérica destas classificações.

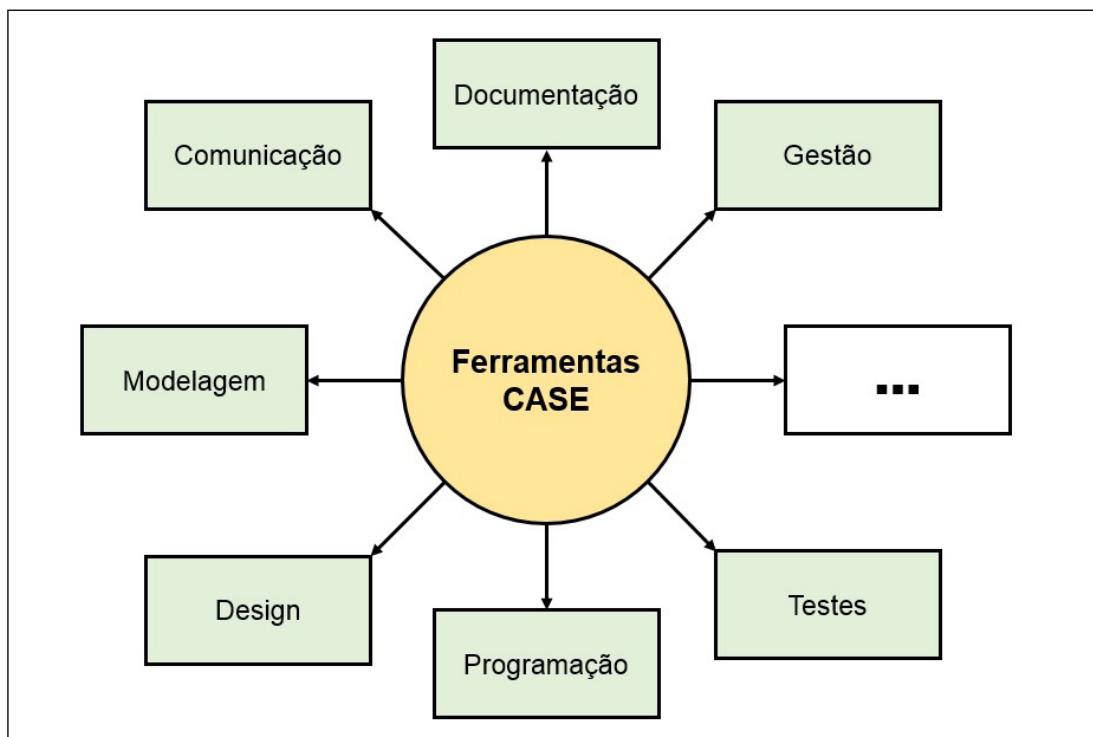


Figura 35: Taxonomia das ferramentas CASE sob a perspectiva funcional.

Fonte: Adaptado pelo autor, a partir de Amui (2015), Schach (2010) e Silva e Videira (2001).

Em um primeiro plano, estão as ferramentas de gestão de projetos, as quais apoiam a estimativa e o planejamento de prazos, recursos e custos, bem como a coordenação e o controle das atividades do projeto de software (AMUI, 2015), por exemplo: Trello, Microsoft Project e Redmine. Na categoria de gestão, encontram-se, também, as ferramentas de gestão de configuração, as quais apoiam o controle das mudanças que são incorporadas em cada nova versão do software (SOMMERVILLE, 2011), por exemplo: GitHub, BitBucket e Jira.

Você já vivenciou algum projeto que não utilizou ferramentas de documentação e comunicação? Pois é, difícil lembrar, não é mesmo? Processadores de texto, planilhas eletrônicas e e-mails são ferramentas quase sempre indispensáveis para qualquer atividade. Na engenharia de software, as ferramentas de documentação são utilizadas na

elaboração de diversos tipos de documentos, tais como dicionário de dados, detalhamento de casos de uso, especificação de requisitos e casos de teste. Já as ferramentas de comunicação são utilizadas para manter os membros da equipe informados sobre eventos, como uma alteração no status do projeto e uma nova demanda.

Na categoria de modelagem, há três subcategorias principais de ferramentas: modelagem de processos de negócio, modelagem de dados e modelagem de sistema. Apesar de não ser uma atividade específica da Engenharia de Software, a modelagem de processos de negócio pode ser realizada nas fases iniciais de um projeto, a fim de que os processos, a serem implementados no sistema, sejam mais bem compreendidos. Para tal, são utilizadas ferramentas de diagramas BPMN (Business Process Model and Notation).

As ferramentas de modelagem de dados são aquelas que apoiam a elaboração dos diagramas das estruturas lógica e física do banco de dados do sistema (AMUI, 2015), por exemplo: MySQL Workbench, brModelo e DBDesigner. As ferramentas de modelagem de sistema, por sua vez, são aquelas que apoiam a elaboração de diagramas, os quais servem de base para a implementação do sistema (SILVA; VIDEIRA, 2001). Uma ferramenta de criação de diagramas UML é uma típica amostra de ferramenta de modelagem de sistema. Por exemplo: Astah e StarUML.

Na categoria de ferramentas de design, encontram-se as ferramentas para a criação dos elementos visuais do sistema. Um exemplo é uma ferramenta de prototipagem, muito utilizada em engenharia de software. Ela pode apoiar tanto a atividade de requisitos, para uma melhor compreensão do que deve ser construído, quanto a atividade de projeto, para a definição do modelo de interação humano-computador (PRESSMAN; MAXIM, 2016; SOMMERVILLE, 2011). Por exemplo: Marvel Prototyping e Justinmind.

As ferramentas de programação são aquelas que normalmente se apresentam em um único pacote, chamado Ambiente de desenvolvimento Integrado, ou IDE (do inglês, Integrated Development Environment), que possui funcionalidades para edição, interpretação, compilação e depuração de código-fonte (AMUI, 2015). Por exemplo: PhpStorm, NetBeans e Eclipse.

Por fim, a categoria de teste comprehende ferramentas, as quais permitem a criação de cenários de teste, incluindo a definição de dados e regras teste, geração de scripts de teste e a obtenção de estatísticas relacionadas à execução dos testes (SILVA; VIDEIRA, 2001). Por exemplo: TestLink, Zephyr e PHPUnit.

Vale lembrar que o conjunto de categorias apresentado na Figura 35 não é exaustivo, e que diferentes classificações de ferramentas têm sido propostas na literatura.



EXERCÍCIO

Ao longo desta seção, foram apresentados alguns exemplos de ferramentas CASE sob a perspectiva funcional. Agora é a sua vez! Forneça exemplos de ferramentas CASE sob as perspectivas apresentadas no quadro a seguir. Podem ser realizadas pesquisas na Internet.

Perspectiva	Classificação	Exemplo de ferramenta CASE
De processo	upperCASE	
	lowerCASE	
	I-CASE	
De integração	Ferramenta	
	Bancada de trabalho	
	Ambiente	

3.4 PRINCIPAIS CARACTERÍSTICAS

É certo que as características mais importantes para uma ferramenta CASE estão diretamente relacionadas à sua classificação sob a perspectiva funcional. Por exemplo, uma característica importante para as ferramentas de gestão de projetos é a possibilidade de representar a linha de tempo do projeto e atribuir atividades aos recursos humanos; como outro exemplo, espera-se que os ambientes de programação forneçam depuradores, para que o programador possa monitorar a execução do programa.

No entanto, Silva e Videira (2001) apresentam uma lista de características que são comuns à grande parte das ferramentas CASE:

- Mecanismo de login/logout;
- Definição de grupos e perfis de usuários;
- Suporte ao trabalho em equipes;
- Possibilidade de implementar a noção de projeto;
- Possibilidade de reutilizar artefatos de outros projetos;
- Histórico de alterações, associado ao controle de versões.
- Essas características apoiam tanto o controle do acesso à informação quanto à gestão do projeto.

Outra característica importante para grande parte das ferramentas CASE é o suporte à automação de tarefas. Denomina-se engenharia *round-trip* a capacidade de uma ferramenta CASE de interagir com o código-fonte do sistema que está sendo desenvolvido (Figura 36).

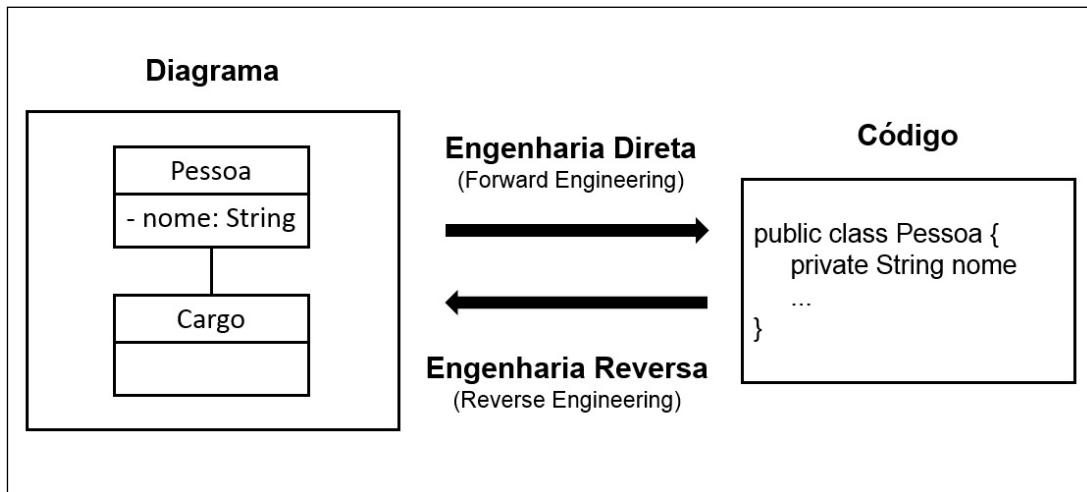


Figura 36: Engenharia Round-Trip.
Fonte: Elaborado pelo autor (2020).

Há dois tipos de engenharia *round-trip*: engenharia direta (em inglês, forward engineering) e engenharia reversa (em inglês, reverse engineering). A engenharia direta é aquela que gera o código-fonte do sistema, a partir de diagramas, enquanto a engenharia reversa realiza o processo inverso, ou seja, gera os diagramas do sistema, a partir do seu código-fonte (BEZERRA, 2015).

A engenharia direta é uma característica muito encontrada em ferramentas de modelagem UML (Unified Modeling Language), demonstrado na Figura 37. A ideia é que os programadores possam gerar o código-fonte do sistema, na linguagem de programação desejada, a partir do diagrama de classes, desenvolvido na fase de projeto. No entanto, para que um diagrama de classes seja útil na geração de código, a notação utilizada na identificação das classes, atributos e métodos deve obedecer às regras de sintaxe da linguagem de programação alvo.

É certo que um diagrama de classes não possui informações suficientes para a geração automática de todo o código do sistema, pois ele é um diagrama estrutural, e não comportamental. Normalmente, um diagrama de classes é utilizado para gerar os arquivos das classes, contendo a declaração dos atributos e o corpo dos métodos, o que já representa uma significativa economia de tempo aos programadores.

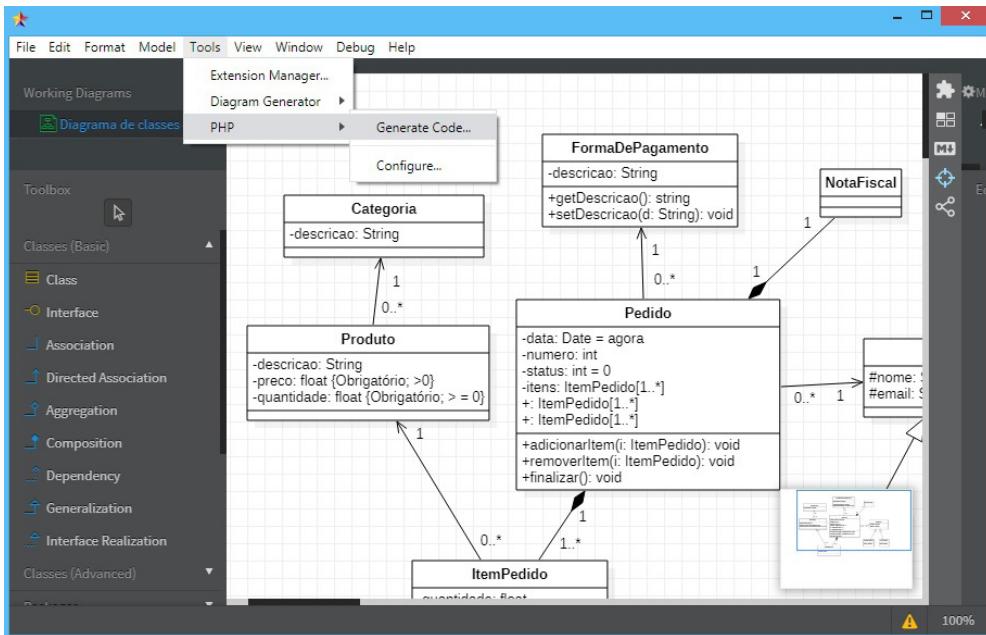


Figura 37: Geração de código com a ferramenta StarUML.

Fonte: Print screen, produzido pelo autor (2020).

Outra aplicação típica da engenharia direta (forward engineering) é a geração de instruções SQL de criação de banco de dados, a partir de diagramas ER (Entidade-Relacionamento), conforme exemplificado na Figura 38.

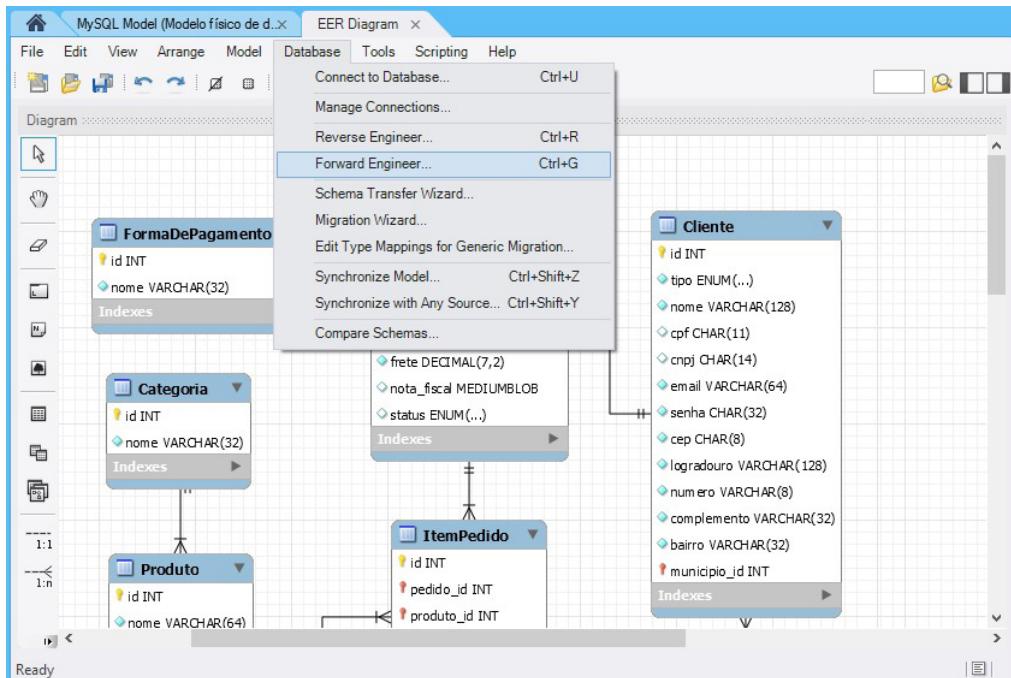


Figura 38: Engenharia direta com a ferramenta MySQL Workbench.

Fonte: Print screen, produzido pelo autor (2020).

A engenharia reversa (reverse engineering), por sua vez, é muito útil, quando se trabalha com a manutenção de sistemas legados, desenvolvidos por terceiros, e cuja documentação não existe ou não está acessível. Neste cenário, o engenheiro de software pode realizar uma engenharia reversa para obter o diagrama ER do banco de dados do sistema, facilitando-se a análise de como os dados estão armazenados e estruturados.

3.5 VANTAGENS E PROBLEMAS DO USO DE FERRAMENTAS CASE

A utilização de ferramentas CASE pode apresentar vantagens e problemas. As vantagens são numerosas, podendo variar, de acordo com as características de cada projeto. A seguir, é apresentada uma lista compilada das vantagens, identificadas por Dennis, Wixom e Roth (2014) e Silva e Videira (2001):

- **Diminuição do tempo de desenvolvimento:** a centralização das informações, o reúso de artefatos ao longo do projeto e a geração automatizada de código e de documentação promovem o aumento da produtividade.
- **Melhora na qualidade do software:** a uniformização das atividades realizadas, e dos artefatos produzidos, bem como a integração de ferramentas impõem uma disciplina que conduz a um processo de desenvolvimento mais estruturado.
- **Redução do esforço/custo de manutenção:** as informações, ilustradas por meio de diagramas e outros formatos padronizados, facilitam a compreensão do software que foi desenvolvido e, consequentemente, minimizam o esforço necessário para realizar alterações após a sua entrega.

Silva e Videira (2001) consideram que o fato de uma equipe adotar as ferramentas CASE, por si só, já é um aspecto positivo, pois pressupõe uma predisposição para a aplicação de regras e padrões ao processo de desenvolvimento. Vejamos, agora, quais são os principais problemas que podem surgir com a utilização de ferramentas CASE:

- **Necessidade de treinamento para utilização:** tirar o melhor proveito de uma ferramenta CASE pode exigir um elevado tempo de aprendizagem, o que, muitas vezes, vem de encontro às exigências das organizações por resultados rápidos.
- **Incompatibilidade de ferramentas:** quando as ferramentas que compõem um ambiente CASE são incompatíveis, ou seja, não conseguem interagir entre si, é necessário desenvolver estratégias de integração entre elas.

Apesar desses e de outros problemas que podem surgir, é certo que a tecnologia CASE está cada vez mais presente nos processos de desenvolvimento de software da atualidade, pois muitas são as vantagens proporcionadas pela sua adoção. Sem contar que existem diversas ferramentas CASE, disponibilizadas de forma gratuita na Internet.



PARA REFLETIR

Nesta seção, citamos a diminuição do tempo de desenvolvimento como uma das vantagens do uso de ferramentas CASE. No seu entendimento, quais são os fatores que podem fazer com que duas equipes de desenvolvimento, que utilizam a mesma tecnologia CASE, apresentem resultados de produtividade diferentes?

3.6 AVALIAÇÃO E SELEÇÃO DE FERRAMENTAS CASE

Nas seções anteriores, você viu que as ferramentas CASE representam uma parte importante das tecnologias de suporte, utilizadas para desenvolver e manter sistemas de software. Por isso, a escolha das ferramentas CASE que serão adotadas por uma equipe de desenvolvimento de software deve ser feita de forma cuidadosa, considerando requisitos técnicos e de gerenciamento.

A norma ISO/IEC 14102:2008 define um conjunto de processos para avaliação e seleção de ferramentas CASE, no que se refere aos requisitos técnicos. O objetivo desses processos é formar um conjunto estruturado de características que permita realizar uma avaliação técnica quantitativa de um conjunto de ferramentas candidatas, para indicar até que ponto cada ferramenta atende aos requisitos estabelecidos pelo usuário e às funcionalidades reivindicadas (ISO/IEC, 2008a). Pois bem, vamos analisar o conjunto de processos definidos na norma ISO/IEC 14102:2008. Observe a a Figura 39.

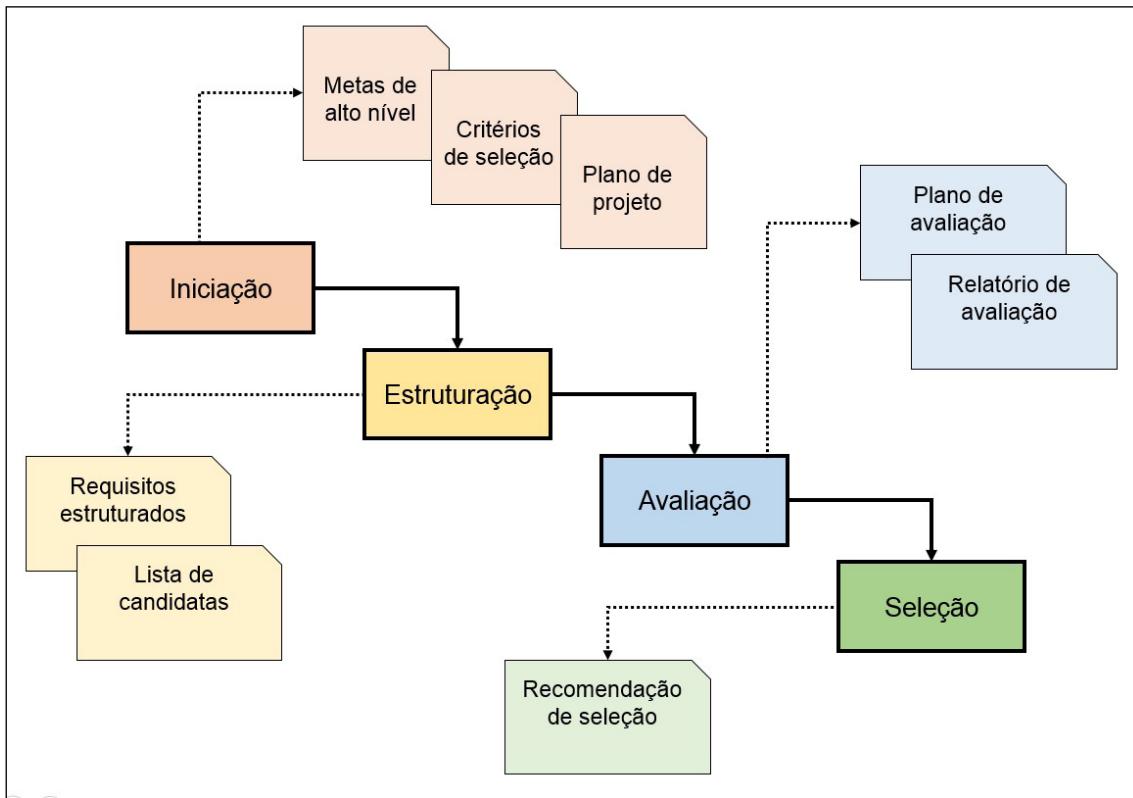


Figura 39: Avaliação e seleção de ferramentas CASE.

Fonte: Adaptado pelo autor, a partir de ISO/IEC (2008b).

A avaliação de seleção de ferramentas passa por quatro processos: iniciação, estruturação, avaliação e seleção. A primeira saída do processo de iniciação são as metas de alto nível, que têm como objetivo justificar a introdução (ou aperfeiçoamento) da ferramenta CASE. A partir das metas, são definidos os critérios de seleção das ferramentas candidatas. As metas de alto nível e os critérios de seleção são utilizados para a elaboração de um plano de projeto, contendo informações gerais sobre o projeto e mecanismos de controle dos critérios de seleção.

Com base nas saídas do processo de iniciação, o processo de estruturação define os requisitos da ferramenta CASE e passa a levantar informações sobre as ferramentas CASE disponíveis no mercado, para que seja possível definir quais serão as ferramentas candidatas. Assim, as saídas do processo de estruturação são uma lista de ferramentas candidatas, e os requisitos são estruturados, a partir da comparação das funcionalidades CASE, tendo em vista as funcionalidades reivindicadas pelos usuários.

Com os requisitos estruturados e a lista de ferramentas candidatas em mãos, o processo de avaliação gera um plano de avaliação das ferramentas candidatas. A execução do plano gera um relatório de avaliação de cada ferramenta, contendo informações, como: nome da ferramenta, versão, fornecedor, custo, funcionalidades, ambiente de execução, atividades que serão apoiadas pela ferramenta e público alvo.

Por fim, o processo de seleção identifica a ferramenta adequada, certificando-se de que as funcionalidades que ela oferece estão de acordo com as funcionalidades reivindicadas pelos usuários.



SAIBA MAIS

Para a aquisição de um conjunto de tecnologias CASE que forneça suporte para todo o processo de desenvolvimento de software, uma empresa pode adotar duas estratégias: **best-of-breed** ou **best-of-suite**. Descubra o significado dos termos e descreva as principais vantagens e desvantagens de cada estratégia. Podem ser realizadas pesquisas na Internet.

3.7 FÓRUM

Nas primeiras unidades deste caderno de estudos, realizamos a análise e o projeto do sistema Loja ADS. Pois bem, vamos imaginar que chegou a hora de implementá-lo, utilizando a linguagem de programação PHP.

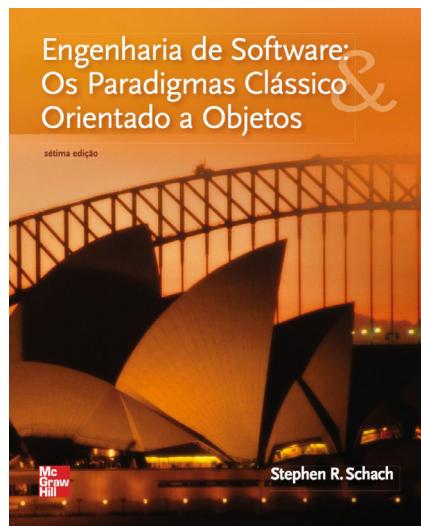
Utilize o fórum on-line da disciplina para propor uma ferramenta CASE a esta atividade, descrevendo suas principais funcionalidades. Ao longo das discussões, apresente alguma vantagem e/ou desvantagem da ferramenta proposta por você, em relação à outra ferramenta proposta por um de seus colegas.



SUGESTÃO DE LIVRO

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

Este livro possui um capítulo que trata das ferramentas de trabalho da engenharia de software, no qual são discutidos o escopo e a taxonomia de ferramentas CASE, incluindo descrições de ferramentas de controle de versões, ferramentas de controle de configurações e ferramentas de consolidação.



SUGESTÃO DE VÍDEO

O vídeo "Ferramentas que Utilizamos no Dia a Dia // Vlog #69", disponível no canal do Código Fonte TV do YouTube, fala sobre alguns exemplos de ferramentas CASE, utilizadas no dia a dia de uma empresa de desenvolvimento de sistemas.

CONSIDERAÇÕES FINAIS

Em nossa terceira unidade, entendemos que as ferramentas CASE são ferramentas de software, as quais fornecem suporte automatizado ou semiautomatizado às atividades de engenharia de software. Estudou-se um ambiente CASE, uma tecnologia que suporta todas, ou grande parte das atividades do processo de desenvolvimento de software, tais como gestão de projeto, gestão de configuração, modelagem de dados, modelagem de sistema, desenvolvimento e testes.

Assim, em um projeto de software, os computadores podem ser utilizados para apoiar, tanto as atividades associadas ao desenvolvimento do software em si quanto as atividades de criação e organização de planos, especificações e outros tipos de documentos. Além disso, as ferramentas CASE podem ser importantes aliadas dos gestores de projetos, no melhor alcance do trinômio tempo-custo-qualidade, desde que elas sejam selecionadas, com critérios adequados, e introduzidas, com planejamento e treinamento da equipe.

Na próxima unidade, conheceremos uma técnica para a medição de projetos de desenvolvimento de software.

Até lá!

EXERCÍCIO FINAL

1. Em um projeto de software, diversas ferramentas podem ser utilizadas para agilizar e documentar atividades em todas as fases de desenvolvimento. Quando a utilização dessas ferramentas permite que a informação produzida por uma ferramenta possa ser utilizada por outra, tem-se o cenário chamado de Engenharia de Software Auxiliada por Computador ou CASE (Computer-Aided Software Engineering).

Em relação ao conceito de CASE, avalie as afirmações a seguir:

- I. Um ambiente CASE apoia todas, ou grande parte das atividades do processo de desenvolvimento de software;
- II. Uma ferramenta CASE pode apoiar tanto as atividades de desenvolvimento do software em si, como as atividades relacionadas aos aspectos organizacionais do projeto;
- III. O repositório é um componente do ambiente CASE.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) I e III, apenas.
- (E) I, II e III.

2. Uma ferramenta CASE pode ser classificada sob diferentes perspectivas. Quanto às fases do processo de desenvolvimento de software que ela apoia, podemos classificá-la como upperCASE, lowerCASE ou I-CASE e, quanto à quantidade de atividades que ela apoia, como ferramenta, bancada de trabalho ou ambiente.

Em relação a essas classificações, avalie as afirmações a seguir:

- I. São classificadas como upperCASE as ferramentas que apoiam as etapas finais do processo de software;
- II. São classificadas como I-CASE as ferramentas que apoiam todo o processo de software;
- III. Uma bancada de trabalho se apresenta como um conjunto de ferramentas que suporta uma ou algumas das atividades do processo de desenvolvimento de software.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) II e III, apenas.
- (E) I, II e III.

3. A norma ISO/IEC 14102 define o seguinte conjunto de processos para avaliação e seleção de ferramentas CASE: **iniciação, estruturação, avaliação e seleção**. O objetivo desse conjunto de processos é formar um conjunto estruturado de características, que permita selecionar uma ou mais ferramentas CASE, com base na avaliação técnica de um conjunto de ferramentas candidatas.

Em relação aos quatro processos da norma ISO/IEC 14102, avalie as afirmações a seguir:

- I. Os critérios de seleção são definidos no processo de iniciação;
- II. O processo de estruturação gera uma lista de ferramentas candidatas;
- III. A saída do processo de avaliação é a recomendação de seleção.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) I e III, apenas.
- (E) I, II e III.

REFERÊNCIAS

AMUI, Saulo. **Processos de desenvolvimento de software**. Rio de Janeiro: SESES, 2015.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. [Recurso Eletrônico]. 3. ed. Rio de Janeiro: Elsevier, 2015.

DENNIS, Alan; WIXOM, Barbara Haley; ROTH, Roberta M. **Análise e projeto de sistemas**. [Recurso Eletrônico]. 5. ed. Rio de Janeiro: LTC, 2014.

ISO/IEC. Information technology – Guideline for the evaluation and selection of CASE tools. 2008a. Disponível em: <<https://www.iso.org/obp/ui/#iso:std:iso-iec:14102:ed-2:v1:en>>. Acesso em: 30 abr 2020.

_____. **Information technology – Guideline for the evaluation and selection of CASE tools.** 2008b. Disponível em: <<https://www.sis.se/api/document/preview/910371/>>. Acesso em: 30 abr 2020.

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

SBROCCO, José Henrique Teixeira de Carvalho; MACEDO, Paulo Cesar de. **Metodologias ágeis:** engenharia de software sob medida. [Recurso Eletrônico]. 1. ed. São Paulo: Erica, 2012.

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

SILVA, Alberto Manuel Rodrigues da; VIDEIRA, Carlos Alberto Escaleira. **UML, Metodologias e Ferramentas CASE.** Lisboa: Centro Atlântico, 2001.

SOMMERVILLE, Ian. **Engenharia de software.** 9. ed. São Paulo: Pearson Prentice Hall, 2011.

unidade



MÉTRICAS DE
SOFTWARE. PRINCÍPIOS
DE PONTOS DE FUNÇÃO



INTRODUÇÃO À UNIDADE

Suponha que você seja um programador e tenha sido procurado para desenvolver um sistema, semelhante ao Loja ADS, projetado nas duas primeiras unidades deste caderno de estudos. Você saberia estimar o tempo necessário para a entrega do sistema ao cliente? Quanto você deveria cobrar para realizar o serviço? Pois então, estimativas como estas estão sempre rodeadas de incertezas, não é mesmo?

É comum que desenvolvedores de sistemas façam estimativas de esforço e custo de maneira empírica, principalmente quando o sistema a ser desenvolvido é simples, ou semelhante a outro sistema já desenvolvido, o que diminui o risco de uma estimativa estar muito otimista ou pessimista. No entanto, há situações em que o desenvolvedor está diante de um projeto complexo, seja pelo tamanho do sistema ou pela diversidade de funcionalidades. Uma boa ideia é utilizar alguma métrica para estimar o esforço necessário no desenvolvimento do sistema.

Na Unidade 4, estudaremos os tipos de métricas, utilizadas em engenharia de software, em especial a métrica de pontos de função, a qual pode ser usada para responder às perguntas, apresentadas no início deste texto introdutório.

Assim, os objetivos específicos desta unidade são: (I) conhecer os tipos de métricas de software; (II) compreender o método de análise de pontos de função e (III) aplicar o método de análise de pontos de função, para a medição do tamanho funcional de sistemas.

4 MÉTRICAS DE SOFTWARE

Frequentemente, as pessoas fazem uso de medidas, para nortear as escolhas, com as quais elas precisam lidar no cotidiano. Por exemplo: uma medida de consumo de combustível pode ser usada, ao decidir entre comprar ou não um carro novo que se propõe a ser mais econômico; uma medida de área quadrada, para decidir entre comprar ou não um imóvel, ou uma medida de peso, para decidir qual bicicleta comprar.

Em um projeto de software, diversas medidas também são usadas para nortear decisões. No entanto, a obtenção de medidas de software nem sempre é uma tarefa simples. Primeiro, porque muitas medidas só podem ser obtidas de forma indireta, ou seja, por meio do relacionamento entre medidas; segundo, porque existem muitos

atributos mensuráveis, tanto do produto de software quanto do processo utilizado para desenvolvê-lo (SCHACH, 2010; SOMMERVILLE, 2011).

No início de um projeto de software, o gestor do projeto pode estar interessado em medir o tamanho do software a ser desenvolvido, para dimensionar a sua equipe e calcular os custos de desenvolvimento; no decorrer do projeto, em medir o número de linhas de código, produzidas até então, para avaliar a produtividade da equipe ou das ferramentas utilizadas; e, no final do projeto, o gestor pode estar interessado em medir o número de defeitos, para avaliar a qualidade do software desenvolvido.

Desse modo, a medição de software é o processo de anexar números aos atributos de um produto ou processo de software, a fim de que eles sejam mais bem compreendidos e possam servir de alerta para problemas em potencial (PRESSMAN; MAXIM, 2016; SCHACH, 2010).

Perceba que o título desta unidade sugere que deveríamos estar falando em métricas, e até então utilizamos somente o termo “medidas”. Mas não se preocupe, chegou a hora de compreendermos a diferença entre medidas, métricas e indicadores.

4.1 MEDIDAS, MÉTRICAS E INDICADORES

De acordo com Pressman e Maxim (2016), as medidas usadas em engenharia de software fornecem indicações quantitativas de atributos dos produtos ou processos de software. Elas surgem da medição, por meio da coleta de um ou mais pontos de dados da unidade em investigação (Figura 40).

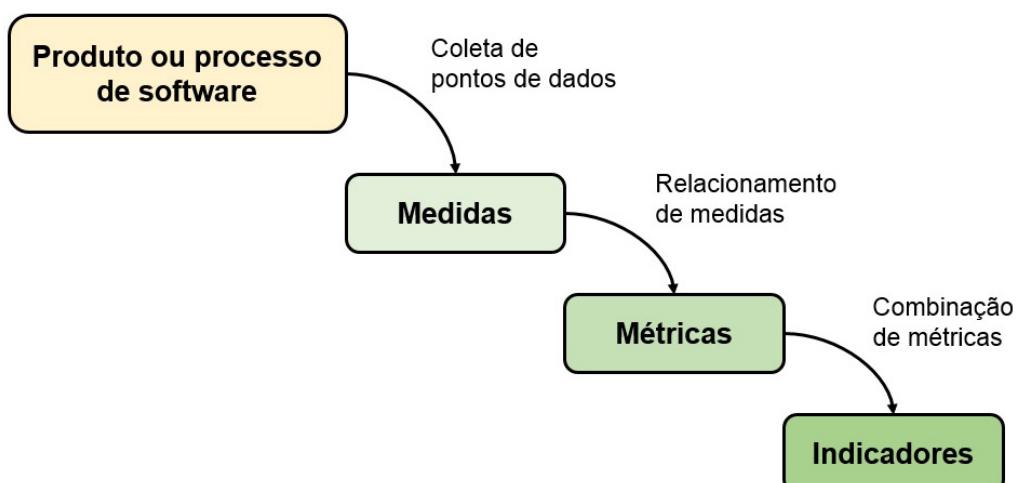


Figura 40: Medidas, métricas e indicadores.
Fonte: Adaptado pelo autor, a partir de Pressman e Maxim (2016).

É certo que qualquer produto de software possui diversos atributos, os quais podem ser mensurados quantitativamente, tais como: número de linhas de código, velocidade de execução, consumo de memória e número de defeitos. Mas será que a medição de um único atributo do software é suficiente para nos fornecer alguma informação útil? Por exemplo, vamos supor que existam 10 defeitos em um determinado software. Com base nesta única informação, você saberia avaliar se o software possui muitos defeitos?

Imagino que, antes de realizar qualquer avaliação, você perguntaria quantas linhas de código o software possui, tendo em vista que o valor 10 pode representar um número baixo de defeitos para um software, com um milhão de linhas de código, e, ao mesmo tempo, um número alto de defeitos para outro software, com apenas mil linhas de código.

É daí que surge o conceito de métrica de software. Uma métrica nada mais é do que o relacionamento entre duas medidas de grandezas iguais ou diferentes, como no exemplo que acabamos de apresentar, no qual se busca avaliar a qualidade de um software com base na métrica DEFEITOS (medida 1) / LINHAS DE CÓDIGO (medida 2). Outro exemplo: poderíamos pensar em avaliar a produtividade de uma equipe de desenvolvimento, com base na métrica LINHAS DE CÓDIGO / PESSOAS-MÊS.

Por fim, um indicador é uma combinação de métricas. Os indicadores fornecem informações que permitem incluir melhorias no processo ou produto de software. Por exemplo, um gestor de projetos pode estar interessado em comparar a métrica de produtividade de um projeto – desenvolvido com um modelo de processo A – com a métrica de produtividade de um projeto – desenvolvido com um modelo de processo B – para, então, decidir qual dos modelos de processo (A ou B) será utilizado nos próximos projetos.

Muito bem, conhecendo as definições de medida, métrica e indicador, vamos descobrir algumas métricas de software e como elas podem ser classificadas sob diferentes perspectivas.

4.2 CLASSIFICAÇÕES DAS MÉTRICAS DE SOFTWARE

Diversos atributos dos produtos ou processos de software permitem que se tenha certeza sobre os valores que lhes são atribuídos, em razão da forma como estes valores são obtidos. Por exemplo, a quantidade de linhas de código é um atributo, cuja medição pode ser realizada diretamente sobre o código-fonte do software, inclusive com o auxílio das ferramentas CASE, utilizadas na fase de implementação.

Por outro lado, existem diversos atributos, cuja medição está sempre sujeita a interpretações e incertezas. Vejamos: você saberia determinar o valor exato da qualidade

de um software? Acredito que a sua resposta seja algo como “sim, mas para determinar a qualidade, é necessário medir diversos atributos do software, tais como quantidade de defeitos, velocidade de execução e consumo de memória”.

Perceba, então, que a qualidade de software é uma métrica, cuja obtenção do valor depende da obtenção do valor de outras métricas, enquanto a quantidade de linhas de código é uma métrica que não depende de nenhuma outra métrica. Assim, sob uma primeira perspectiva, as métricas de software podem ser classificadas em diretas e indiretas. Observe na Figura 41.

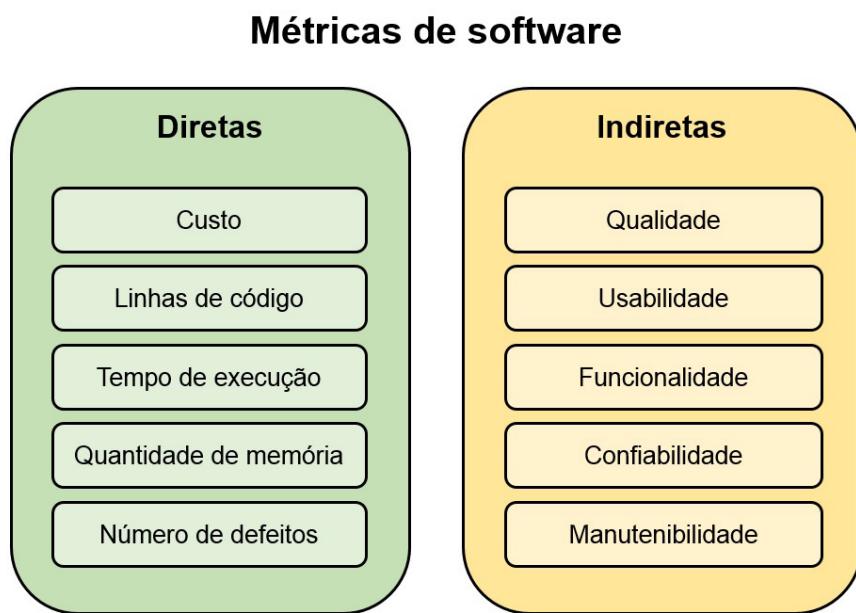


Figura 41: Exemplos de métricas diretas e indiretas.
Fonte: Adaptado pelo autor, a partir de Wazlawick (2013).

As **métricas diretas** são aquelas que podem ser medidas de modo direto, sem necessidade de interpretação ou incerteza, tais como: custo, linhas de código, tempo de execução, quantidade de memória e número de defeitos. Já as **métricas indiretas** são as obtidas a partir de outras métricas, cuja interpretação nem sempre é um consenso (VETORAZZO, 2018; WAZLAWICK, 2013). São exemplos de métricas indiretas: qualidade, usabilidade, funcionalidade e confiabilidade.

Na Seção 4, vimos que medições podem ser realizadas, tanto nos atributos de um produto de software quanto nos atributos do processo utilizado para desenvolvê-lo, sendo outra perspectiva sob a qual podemos classificar as métricas de software, conforme mostra a Figura 42.

Métricas de software

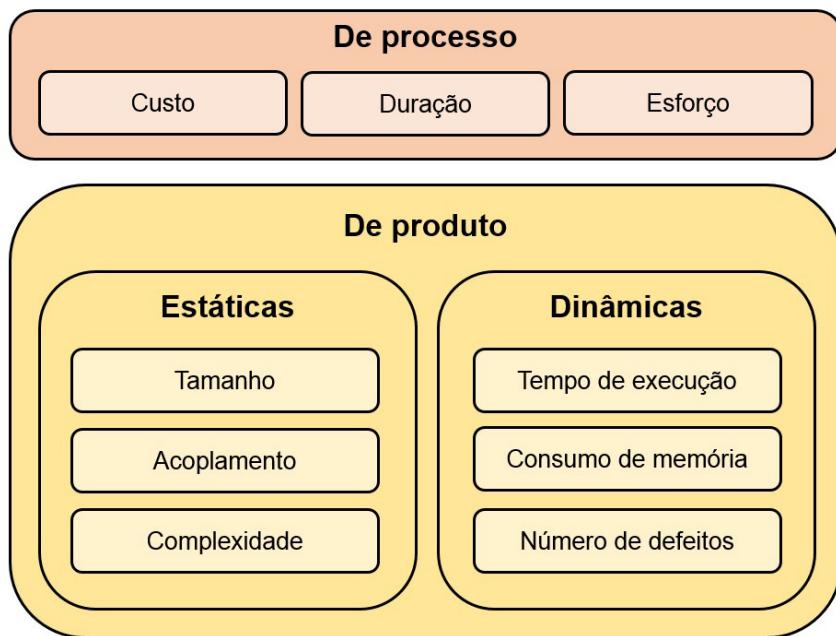


Figura 42: Exemplos de métricas de processo e de produto.
Fonte: Adaptado pelo autor, a partir de Sommerville (2011) e Schach (2010).

As **métricas de produto** medem aspectos do próprio produto, como seu tamanho ou sua complexidade, enquanto as **métricas de processo** são usadas para deduzir informações sobre o processo de software, tais como custo e duração (SCHACH, 2010). As métricas de produto podem ser classificadas em dinâmicas e estáticas. As **métricas dinâmicas** são aquelas coletadas por meio de medições, realizadas durante a execução do software, enquanto as **métricas estáticas** são aquelas coletadas por meio de medições, realizadas em representações do software, tais como o seu projeto ou a sua documentação (SOMMERVILLE, 2011).

Outra perspectiva de classificação das métricas de software está diretamente relacionada à gestão do processo de software, exemplificado na Figura 43. As **métricas de produtividade** são aquelas que permitem ao gestor de projetos avaliar a velocidade de desenvolvimento do produto e software, tendo como foco as saídas do processo de engenharia de software (WAZLAWICK, 2013).

Métricas de software

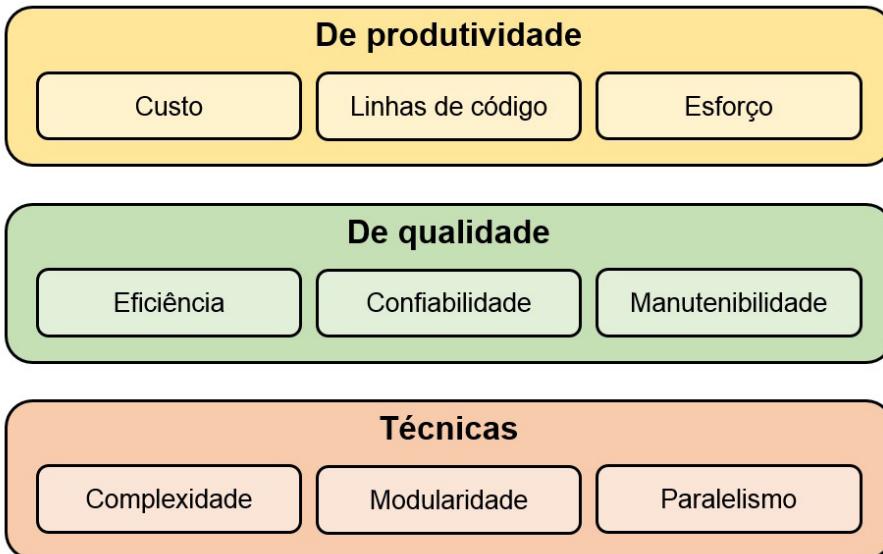


Figura 43: Exemplos de métricas técnicas, de qualidade e de produtividade.

Fonte: Adaptado pelo autor, a partir de Wazlawick (2013).

As **métricas de qualidade**, por sua vez, permitem ao gestor de projetos avaliar o quanto o produto de software atende às exigências do cliente. Elas também revelam as ineficiências, ocorridas durante o processo de desenvolvimento de software. Por fim, as **métricas técnicas** são aquelas usadas para avaliar o produto de software, em si, e não o processo, utilizado para desenvolvê-lo (WAZLAWICK, 2013).

A próxima e última perspectiva de classificação das métricas de software que estudaremos é, comumente, adotada por literaturas especializadas em técnicas para a medição do tamanho de softwares. Ela divide as métricas de software em métricas orientadas ao tamanho e métricas orientadas à função, de acordo com o exposto na Figura 44.

Métricas de software

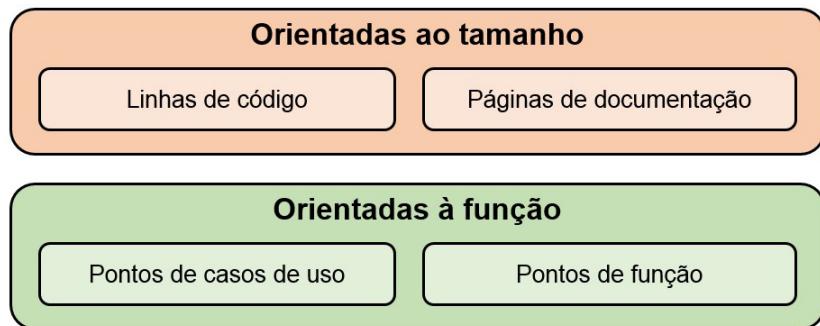


Figura 44: Exemplos de métricas orientadas ao tamanho e orientadas à função.

Fonte: Adaptado pelo autor, a partir de Vetorazzo (2018).

Ambos os tipos de métricas permitem avaliar a complexidade de um software. Contudo, enquanto as **métricas orientadas ao tamanho** o fazem, com base nos artefatos do software (código-fonte, documentação), as **métricas orientadas à função** o fazem, com base na visão dos usuários do software (casos de uso, funções) (VETORAZZO, 2018). As métricas de complexidade podem ser utilizadas para estimar a quantidade de trabalho, a ser executada no desenvolvimento de um software, e para estabelecer os prazos e custos do projeto (CALAZANS; PALDÊS; MARIANO, 2015).

Para Vetorazzo (2018), embora a contagem de artefatos seja uma tarefa simples, as métricas orientadas ao tamanho não são universalmente aceitas, pois elas podem penalizar programas bem projetados e/ou que utilizam linguagens de programação mais expressivas. Assim, as métricas orientadas à função, também chamadas **métricas funcionais**, são as mais utilizadas para medir a complexidade de softwares, inclusive no Brasil, onde existem normas que indicam a necessidade de utilização de métricas funcionais para a contratação, pelo governo, de serviços de desenvolvimento de software (CALAZANS; PALDÊS; MARIANO, 2015).

A métrica de **pontos de caso de uso** baseia-se na contagem dos atores e casos de uso do software, enquanto a métrica de **pontos de função** se baseia na contagem dos componentes funcionais do software. A métrica de pontos de função será o nosso objeto de estudo, na próxima seção.

4.3 ANÁLISE DE PONTOS DE FUNÇÃO

A Análise de Pontos de Função (APF) é uma técnica para a medição das funcionalidades, fornecidas por um software do ponto de vista dos seus usuários, ou seja, ela busca medir um software pelo que ele é capaz de fazer, independente das tecnologias que ele utiliza (VAZQUEZ, 2013). Foi desenvolvida, em 1979, por Allen Albrecht, da IBM (DENNIS; WIXOM; ROTH, 2014).

De acordo com Pressman e Maxim (2016), a APF pode ser utilizada para: (I) estimar o custo, ou trabalho, necessário para desenvolver o software; (II) prever o número de erros que serão encontrados nas atividades de teste e (III) prever o número de componentes, e/ou linhas de código, os quais serão implementados.

4.3.1 Tipos de função

Basicamente, o processo de análise de pontos de função (APF) inicia-se com a decomposição do projeto de software em elementos, denominados componentes funcionais básicos ou funções. Existem cinco tipos de funções, sendo três do grupo de funções de transição e duas do grupo de funções de dados. Observemos a Figura 45 a seguir.

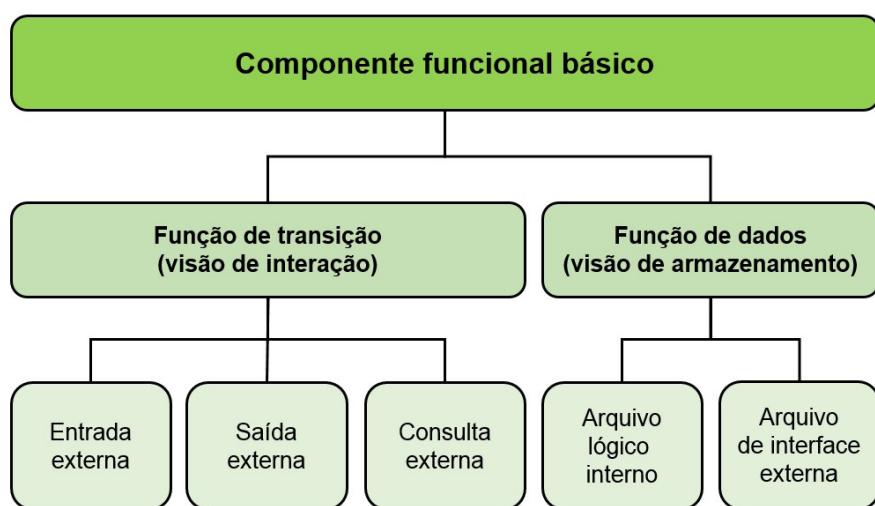


Figura 45: Visão geral da análise de pontos de função.
Fonte: Adaptado pelo autor, a partir de Vazquez (2013).

As **funções de transição** permitem medir a complexidade do software, do ponto de vista das suas interações com o usuário, enquanto as **funções de dados** permitem medir a complexidade do software, do ponto de vista dos dados que ele armazena. A soma destas medições é que permite determinar a complexidade do software como um todo. O Quadro 4 apresenta a sigla e a definição de cada um dos tipos de função, demonstrados na Figura 45.

Tipo de função	Sigla	Definição
Entrada externa	EE	Transação, na qual dados cruzam a fronteira da aplicação de fora para dentro.
Saída externa	SE	Transação, na qual dados derivados cruzam a fronteira da aplicação de dentro para fora.
Consulta externa	CE	Transação que resulta na simples recuperação de dados de um ou mais arquivos lógicos.
Arquivo lógico interno	ALI	Grupo de dados logicamente relacionados, identificável pelo usuário, que é mantido dentro da fronteira da aplicação.
Arquivo de interface externa	AIE	Grupo de dados logicamente relacionados, identificável pelo usuário, que é mantido fora da fronteira da aplicação.

Quadro 4 – Tipos de função.

Fonte: Adaptado pelo autor, a partir de Paula Filho (2009).

Uma **entrada externa** (EE) é uma transação, na qual dados cruzam a fronteira da aplicação de fora para dentro (PAULA FILHO, 2009). Em um sistema de comércio eletrônico, por exemplo, a função “incluir produto” é uma EE, pois ela permite que o usuário introduza os dados de um produto para dentro da aplicação, através de um formulário.

Nas transações que operam no sentido inverso, ou seja, nas quais os dados cruzam a fronteira da aplicação de dentro para fora, existem duas situações: I) se os dados são simplesmente recuperados de um arquivo lógico (por exemplo, uma tabela de banco de dados) e apresentados ao usuário, a função é considerada uma **consulta externa** (CE); II) caso ocorra algum processamento com os dados, antes de serem apresentados ao usuário, a função é considerada uma **saída externa** (SE) (PAULA FILHO, 2009).

Os relatórios fornecidos por uma aplicação são típicos exemplos de SEs, pois o processo de geração de um relatório quase sempre requer que sejam realizadas operações matemáticas sobre os dados armazenados nos arquivos lógicos e, como exemplo de CE, podemos considerar uma função que apenas apresenta os dados de um determinado produto, sem realizar nenhuma transformação de dados.

No grupo de funções de dados, temos os **arquivos lógicos internos** (ALIs) e os **arquivos de interface externa** (AIEs). Do ponto de vista da APF, um arquivo é um conjunto de dados

logicamente relacionados, o qual é utilizado por uma ou mais funções de transação do software. Quando um arquivo é mantido (adicionado, modificado ou excluído) dentro da fronteira da aplicação, ele é considerado um ALI; quando ele é mantido fora da fronteira da aplicação, sendo apenas referenciado pelas funções de transação, é considerado um AIE (PAULA FILHO, 2009).

Em geral, os arquivos de uma aplicação são implementados como tabelas de banco de dados. Neste ponto, é importante perceber que, apesar de o usuário não conhecer os detalhes da implementação física de um ALI ou AIE, ele consegue identificar a existência destes arquivos, por meio dos dados que são apresentados na interface da aplicação.

Os componentes funcionais de um sistema podem apresentar diferentes níveis de complexidade de implementação. Por exemplo, implementar uma função, para incluir dados em um arquivo, quase sempre exige mais esforço que implementar uma função para excluir estes dados. Assim, durante o processo de APF, é necessário medir a complexidade de cada componente funcional, conforme veremos a seguir.

4.3.2 Parâmetros de complexidade e regras de contagem

A complexidade de um tipo de função é determinada pela contagem dos seus respectivos tipos de elementos. Existem três tipos: elementos de dados, elementos de arquivos referenciados e elementos de registro. O Quadro 5 expressa a sigla e a definição de cada um deles:

Parâmetro de contagem	Sigla	Definição
Tipos de elementos de dados	TEDs	Número de campos distintos e não repetidos, identificáveis pelo usuário.
Tipos de arquivos referenciados	TARs	Número de arquivos lógicos internos (ALIs) e arquivos de interface externa (AIEs), referenciados por uma transação.
Tipos de elementos de registro	TERs	Número de subgrupos de dados, identificáveis pelo usuário, componentes de um arquivo lógico interno ou de um arquivo de interface externa.

Quadro 5 – Parâmetros de contagem.

Fonte: Adaptado pelo autor, a partir de Paula Filho (2009).

O parâmetro TEDs é usado na determinação da complexidade de todos os tipos de função, apresentados no Quadro 4 (EE, CE, SE, ALI e AIE). Ao realizar a contagem dos

TEDs, você deve ter em mente que está sendo contada não a quantidade de campos, mas sim a quantidade de campos distintos.

Perceba que, apesar de os dois formulários, apresentados na parte superior da Figura 46, possuírem 3 campos, o formulário à esquerda tem 3 TEDs (telefone residencial, telefone comercial e telefone celular), enquanto o formulário à direita tem apenas 1 TED (telefones). Por sua vez, a listagem de telefones, apresentada na parte inferior da Figura 46, possui 2 TEDs (tipo de telefone e número).

3 TEDs	1 TED												
<table border="1"> <thead> <tr> <th>Contatos</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>Telefone residencial</td> <td></td> </tr> <tr> <td>Telefone comercial</td> <td></td> </tr> <tr> <td>Telefone celular</td> <td></td> </tr> </tbody> </table>	Contatos	X	Telefone residencial		Telefone comercial		Telefone celular		<table border="1"> <thead> <tr> <th>Contatos</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>Telefones</td> <td></td> </tr> </tbody> </table>	Contatos	X	Telefones	
Contatos	X												
Telefone residencial													
Telefone comercial													
Telefone celular													
Contatos	X												
Telefones													
2 TEDs													
	<table border="1"> <thead> <tr> <th>Contatos</th> <th>X</th> </tr> </thead> <tbody> <tr> <th colspan="2"> <table border="1"> <thead> <tr> <th>Tipo de telefone</th> <th>Número</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> <tr> <td></td> <td></td> </tr> <tr> <td></td> <td></td> </tr> </tbody> </table> </th> </tr> </tbody> </table>	Contatos	X	<table border="1"> <thead> <tr> <th>Tipo de telefone</th> <th>Número</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> <tr> <td></td> <td></td> </tr> <tr> <td></td> <td></td> </tr> </tbody> </table>		Tipo de telefone	Número						
Contatos	X												
<table border="1"> <thead> <tr> <th>Tipo de telefone</th> <th>Número</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> <tr> <td></td> <td></td> </tr> <tr> <td></td> <td></td> </tr> </tbody> </table>		Tipo de telefone	Número										
Tipo de telefone	Número												

Figura 46: Exemplos de contagem de TEDs.
Fonte: Adaptado pelo autor, a partir de Vazquez (2013).

O parâmetro TARs é usado na determinação da complexidade de funções do tipo EE, CE e SE. A contagem de TARs verifica quantos ALIs ou AIEs estão envolvidos com o componente funcional em análise. Devem ser considerados os AIEs referenciados e os ALIs lidos e/ou mantidos (adicionados, modificados ou excluídos) durante uma transação (VAZQUEZ, 2013).

Por exemplo, se uma função envolve a modificação dos dados de dois arquivos (ALIs ou AIEs), a quantidade de TARs desta função será 2. Outro exemplo: se uma função envolve a leitura de um arquivo, antes de os dados serem adicionados no mesmo arquivo, a quantidade de TARs desta função será 1, pois a contagem está interessada nos tipos de arquivos, e não na quantidade de vezes que eles são lidos, referenciados ou mantidos.

Por fim, o parâmetro TERs é usado para determinar a complexidade de funções do tipo ALI e AIE. Em muitos casos, os dados de um arquivo estão organizados em grupos. Para facilitar o entendimento deste conceito, vamos considerar, novamente, o sistema de comércio eletrônico, projetado nas duas primeiras unidades deste caderno de estudos.

Podemos afirmar que o pedido de compra é um ALI, pois ele: (I) constitui-se de um grupo de dados logicamente relacionados; (II) é identificável pelo usuário e (III) é mantido dentro da fronteira da aplicação. Para implementarmos o arquivo em banco de dados, teremos que criar uma tabela “pedido”, armazenando-se os dados do pedido, e outra tabela “item_pedido”, armazenando-se os dados dos itens adicionados ao pedido.

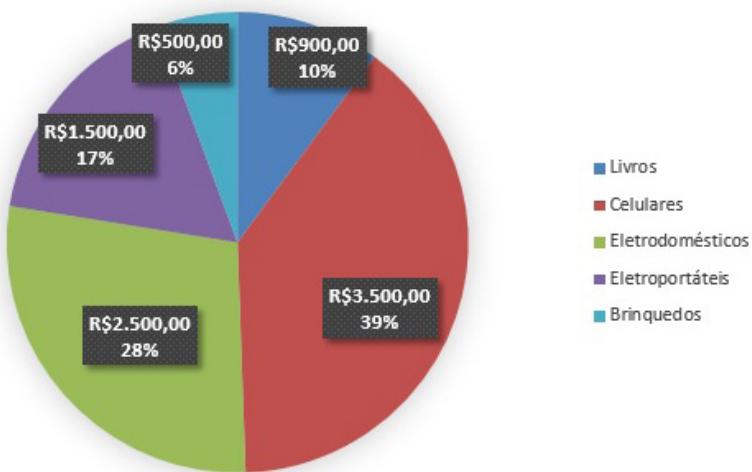
O parâmetro TEDs nos permite medir a complexidade do ALI “pedido”, sob o ponto de vista de seus atributos (cliente, data, valor, forma de pagamento e assim por diante). No entanto, ele não mede a complexidade do ALI “pedido” sob o ponto de vista do agrupamento destes dados. É aí que entra em cena o parâmetro TERs. No caso do ALI “pedido”, o número de TERs é 2, uma vez que temos dois grupos de dados: os dados do pedido e os dados dos itens do pedido.



PARA REFLETIR

Nesta seção, você viu que o número de TEDs de uma função é dado pelo número de campos distintos e não repetidos, identificáveis pelo usuário. Imagine que uma das funções a serem implementadas em um sistema de comércio eletrônico é uma SE que fornece o gráfico apresentado na imagem a seguir. Quais são os TEDs que você consegue identificar nesta função?

Faturamento por produto



4.3.3 Determinação da contribuição das funções

Sabendo-se quais são os tipos de função (EE, SE, CE, ALI e AIE) e quais são os parâmetros (TEDs, TARs e TEDs) de complexidade usados na APF, veremos como funciona o cálculo da contribuição de cada função.

O cálculo é realizado em duas etapas. Primeiro, é necessário determinar o valor de complexidade (baixa, média ou alta) de cada função. A complexidade das funções de dados (ALIs e AIEs) é determinada, com base na tabela do Quadro 6, que considera o número de TEDs e TERs.

		Número de TEDs		
		<20	20-50	>50
Número de TERs	1	Baixa	Baixa	Média
	2-5	Baixa	Média	Alta
	>5	Média	Alta	Alta

Quadro 6 – Determinação da complexidade de ALIs e AIEs.

Fonte: Adaptado pelo autor, a partir de Vazquez (2013).

Por exemplo, um ALI, com 4 TERs e 40 TEDs, é uma função de complexidade média, enquanto um AIE, com 4 TERs e 10 TEDs, é uma função de complexidade baixa.

A complexidade das funções de transição é determinada, baseando-se no número de TARs e TEDs. Neste caso, são utilizadas duas tabelas, uma para as EEs (Quadro 7) e outra para as SEs e CEs (Quadro 8).

		Número de TEDs		
		<5	5-15	>15
Número de TARs	<2	Baixa	Baixa	Média
	2	Baixa	Média	Alta
	>2	Média	Alta	Alta

Quadro 7 – Determinação da complexidade de EEs.

Fonte: Adaptado pelo autor, a partir de Vazquez (2013).

		Número de TEDs		
		<6	6-19	>19
Número de TARs	<2	Baixa	Baixa	Média
	2-3	Baixa	Média	Alta
	>3	Média	Alta	Alta

Quadro 8 – Determinação da complexidade de SEs e CEs.

Fonte: Adaptado pelo autor, a partir de Vazquez (2013).

Em seguida, os valores de complexidade (baixa, média e alta) são convertidos para pontos de função (PF), embasados na tabela do Quadro 9. Perceba que as EEs e CEs de complexidade baixa são as funções que apresentam o menor peso, pois contribuem com somente 3 pontos de função. Já os ALIs de complexidade alta são as funções que apresentam o maior peso, pois contribuem com 15 pontos de função.

Tipo de função	Complexidade		
	Baixa	Média	Alta
Entrada externa (EE)	3 PF	4 PF	6 PF
Saída externa (SE)	4 PF	5 PF	7 PF
Consulta externa (CE)	3 PF	4 PF	6 PF
Arquivo lógico interno (ALI)	7 PF	10 PF	15 PF
Arquivo de interface externa (AIE)	5 PF	7 PF	10 PF

Quadro 9 – Determinação do peso das funções.

Fonte: Adaptado pelo autor, a partir de Paula Filho (2009).

Após a determinação da contribuição das funções, é chegada a hora de realizar o cálculo do tamanho funcional do sistema. Observemos a seguir.

4.3.4 Cálculo do tamanho funcional

A fim de que você possa compreender melhor os conceitos estudados nas seções anteriores, vamos imaginar um sistema de gestão de estoque, o qual permite manter (incluir, alterar e excluir) produtos, registrar entradas e saídas de produtos e emitir relatórios de estoque. Os dados do produto são: nome e descrição, e os dados da

movimentação de estoque são: produto, tipo de movimentação (entrada ou saída), data e quantidade, sendo todos os dados mantidos dentro da fronteira da aplicação.

A princípio, vamos determinar a contribuição das funções de dados. O sistema mantém dois ALIs, um para armazenar os dados dos produtos, e outro para armazenar os dados das movimentações de estoque. O primeiro arquivo possui 2 TEDs (nome e descrição) e o segundo arquivo 4 TEDs (produto, tipo, data e quantidade). Ambos os ALIs possuem 1 TER, pois eles não mantêm subgrupos de dados, isto é, utilizam uma única tabela de banco de dados. Assim, com base no Quadro 6 e no Quadro 9, podemos determinar que ambos os ALIs contribuem com 7 pontos de função.

Agora, vamos determinar a contribuição das funções de transição. Existem 4 EEs, ou seja, 4 transações, nas quais os dados cruzam a fronteira da aplicação de fora para dentro. São elas: incluir produto, alterar produto, excluir produto e registrar movimentação. Na contagem dos TEDs de uma EE, os botões utilizados para receber os comandos dos usuários (por exemplo, o botão “Salvar”), bem como as mensagens exibidas na tela (por exemplo, “Produto cadastrado com sucesso”), também devem ser contabilizadas como TEDs.

Desse modo, vamos considerar que: as funções incluir produto e alterar produto possuem 4 TEDs (nome, descrição, botão de comando e mensagem); a função excluir produto possui 2 TEDs (botão de comando e mensagem), e a função registrar movimentação possui 6 TEDs (produto, tipo, data e quantidade, botão de comando e mensagem). Em todas as transações, o número de TARs é 1, pois somente um arquivo lógico é mantido. A partir dos Quadros 7 e 9, podemos determinar que cada uma das EEs em análise contribui com 3 pontos de função.

Por fim, resta a análise da função que emite relatórios de estoque. Para gerar um relatório de estoque, o sistema realizará cálculos, a partir dos dados de entrada e saída de cada produto. Com isso, podemos concluir tratar-se de uma função do tipo SE. Supondo-se que, no relatório, sejam apresentados os seguintes dados: produto, descrição e quantidade em estoque. Então, temos uma SE, a qual possui 2 TARs, pois utiliza dados de 2 arquivos (produto e movimentação), e 3 TEDs. Fundamentados nos Quadros 8 e 9, podemos determinar que ela contribui com 4 pontos de função.

O Quadro 10 apresenta os dados da APF que acabamos de realizar. Somando as contribuições de cada componente funcional, temos o tamanho funcional do sistema de gestão de estoque com 30 pontos de função.

Nome	Tipo	TARs/TERs	TEDs	Complexidade	PF
Produto	ALI	1	2	Baixa	7
Movimentação	ALI	1	4	Baixa	7
Incluir produto	EE	1	4	Baixa	3
Alterar produto	EE	1	4	Baixa	3
Excluir produto	EE	1	2	Baixa	3
Registrar movimentação	EE	1	6	Baixa	3
Relatório de estoque	SE	2	3	Baixa	4
				Total	30

Quadro 10 – Exemplo de cálculo do tamanho funcional de um sistema.

Fonte: Elaborado pelo autor (2020).

O valor que acabamos de obter é chamado **pontos de função não ajustados**, visto que ele mede o tamanho funcional do sistema, sem considerar os seus requisitos não funcionais, os quais podem facilitar ou dificultar o projeto. Assim, a última atividade de uma APF é o ajuste da complexidade, a ser abordado na próxima seção.



EXERCÍCIO

Determine a complexidade e o peso (PF) de cada uma das funções, listadas a seguir.

Nome	Tipo	TARs/TERs	TEDs	Complexidade	PF
Fornecedor	ALI	1	6		
Inclusão de fornecedor	EE	2	8		
Alteração de fornecedor	EE	2	8		
Exclusão de fornecedor	EE	2	3		
Pesquisa de fornecedor	CE	2	8		
Relatório de fornecedores	SE	2	7		

4.3.5 Ajuste da complexidade

De acordo com Paula Filho (2019), o ajuste da complexidade de um sistema é feito por meio da atribuição de um nível de influência, avaliado em uma escala de 0 (nenhuma influência) a 5 (influência forte), para cada uma das 14 características gerais, demonstradas no Quadro 11.

1. Comunicação de dados 2. Processamento distribuído 3. Desempenho 4. Configuração altamente utilizada 5. Volume de transações 6. Entrada de dados on-line 7. Usabilidade	8. Atualização on-line 9. Complexidade de processamento 10. Reutilização de código 11. Facilidade de implantação 12. Facilidade de operação 13. Operação em múltiplos locais 14. Facilidade de manutenção
---	---

Quadro 11 – Características gerais para ajuste da complexidade.

Fonte: Adaptado pelo autor, a partir de Vazquez (2013) e Paula Filho (2019).

Uma vez determinados os níveis de influência das 14 características gerais, o fator de ajuste é calculado com a seguinte fórmula: $FA = (SNI \times 0,01) + 0,65$, onde FA é o fator de ajuste e SNI é o somatório dos 14 níveis de influência das características gerais (VAZQUEZ, 2013).

Por exemplo, vamos supor que o valor do somatório dos níveis de influência do sistema de gestão de estoque tenha sido 28. Então, temos que $FA = (28 \times 0,01) + 0,65 = 0,93$. Por fim, multiplicamos o valor 30 (que obtivemos na seção anterior) por 0,93 e temos como resultado o valor 27,9, o qual representa o tamanho do sistema em **pontos de função ajustados**.

Finalmente, o tamanho do sistema em pontos de função pode ser utilizado pelo gestor de projetos, a fim de estimar o esforço e o tempo necessários para a conclusão do projeto de software. Para tal, são utilizadas métricas, estudadas na disciplina de Gestão de Projetos.

É importante destacar que o ajuste da complexidade não é um padrão ISO, isto é, tratar-se de um cálculo opcional. Pelo fato de a contagem de características não ser uma tarefa de fácil padronização, muitos gestores de projetos preferem utilizar a quantidade **pontos de função não ajustados** como métrica de tamanho funcional.



SAIBA MAIS

Ao longo da Seção 4.3, utilizamos o método de **contagem detalhada** de pontos de função. Este método consiste em: (I) identificar todas as funções do sistema (EEs, SEs, CEs, ALIs e AIEs); (II) determinar a contribuição de cada função identificada e (III) calcular o total de pontos de função.

Existem mais dois métodos de contagem de pontos de função chamados **contagem indicativa** e **contagem estimada**. Descubra como são realizadas estas contagens. Você pode pesquisar na Internet.

4.4 FÓRUM

Utilize o fórum on-line da disciplina, para propor uma discussão sobre a técnica de Análise de Pontos de Função (APF). Converse com seus colegas sobre as vantagens e desvantagens que a técnica apresenta.



SUGESTÃO DE LIVRO

VAZQUEZ, Carlos Eduardo; SIMÕES, Guiherme Siqueira; ALBERT, Renato Machado.

Análise de pontos de função: medição, estimativas e gerenciamento de projetos de software. 13. ed. São Paulo: Érica, 2013.

O livro aborda a importância da medição, em projetos de software, e o uso da APF para gerenciar escopo de projetos de software.





SUGESTÃO DE VÍDEO

O vídeo "ISD BRASIL - o que é o Nível 1 de Maturidade do CMMI", disponível no canal do ISDOnlineTV do YouTube, é uma descrição lúdica de uma empresa de desenvolvimento de software que está no nível 1 de maturidade do CMMI (Capability Maturity Model Integration). O vídeo mostra os principais problemas, enfrentados por uma organização com baixa maturidade de processos, e desperta a reflexão sobre a importância de se mensurar corretamente o tamanho de um software.

CONSIDERAÇÕES FINAIS

Em nossa última unidade, você aprendeu que a medição de software é o processo de anexar números aos atributos de um produto ou processo de software, para que eles sejam mais bem compreendidos e possam nortear decisões em um projeto de software. Estudou-se, também, a Análise de Pontos de Função (APF), uma técnica que busca medir um software pelo que ele é capaz de fazer, podendo ser utilizada para estimar o custo ou trabalho, necessários para desenvolver um software.

Você deve ter percebido que a obtenção de medidas de software nem sempre é uma tarefa trivial. Muitas medidas só podem ser obtidas de forma indireta, ou seja, através do relacionamento entre medidas. Por este motivo, elas quase sempre estão abertas ao debate. Além disso, a coleta de pontos de dados necessários para o cálculo das métricas de software, na maior parte dos casos, envolve algum custo.

Mas, então, até que ponto os atributos de um produto ou processo de software devem ser medidos? Esta é uma decisão que precisa levar em conta as necessidades e os riscos associados ao projeto de software.

Espero ter contribuído com seu conhecimento. Desejo sucesso na aplicação prática dos estudos abordados em nosso material.

EXERCÍCIO FINAL

1. As medidas usadas em engenharia de software fornecem indicações quantitativas de atributos dos produtos ou processos de software. Elas surgem da medição, por meio da coleta de um ou mais pontos de dados da unidade em investigação. O relacionamento entre duas medidas de grandezas iguais ou diferentes forma uma métrica de software, que pode ser classificada sob diferentes perspectivas.

Em relação às classificações das métricas de software, avalie as afirmações a seguir:

- I- As **métricas indiretas** são aquelas obtidas a partir de outras métricas, e cuja interpretação nem sempre é um consenso;
- II- As **métricas dinâmicas** são aquelas obtidas a partir de medições, realizadas nas representações do software;
- III- As **métricas de produtividade** são aquelas que permitem ao gestor avaliar a velocidade de desenvolvimento do produto e software, tendo como foco as saídas do processo de engenharia de software.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) I e III, apenas.
- (E) I, II e III.

2. O processo de análise de pontos de função (APF) inicia-se com a decomposição do projeto de software em elementos, denominados componentes funcionais básicos ou funções. Existem cinco tipos de funções que são identificados pela APF: Entrada Externa (EE), Saída Externa (SE), Consulta Externa (CE), Arquivo Lógico Interno (ALI) e Arquivo de Interface Externa (AIE).

Em relação aos tipos de função apresentados, avalie as afirmações a seguir:

- I- Uma EE é uma transação, na qual dados cruzam a fronteira da aplicação de fora para dentro;
- II- Uma CE é uma transação que resulta na simples recuperação de dados de um ou mais arquivos lógicos;
- III- Um AIE é um grupo de dados, logicamente relacionados, identificável pelo usuário, que é mantido dentro da fronteira da aplicação.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) I e III, apenas.
- (E) I, II e III.

3. Na Análise de Pontos de Função (APF), a complexidade das funções de um sistema é determinada por três tipos de parâmetros de contagem: tipos de elementos de dados (TEDs), tipos de arquivos referenciados (TARs) e tipos de elementos de registro (TERs). A complexidade das funções de dados está baseada no número de TEDs e TARs, enquanto a complexidade das funções de transição está baseada no número de TEDs e TERs.

Em relação aos parâmetros de contagem apresentados, avalie as afirmações a seguir:

- I- TEDs: Número de campos distintos e não repetidos, identificáveis pelo usuário;
- II- TARs: Número de arquivos lógicos internos e arquivos de interface externa, referenciados por uma transação;
- III- TERs: Número de subgrupos de dados identificáveis pelo usuário, componentes de um arquivo lógico interno ou de um arquivo de interface externa.

É correto o que se afirma em:

- (A) II, apenas.
- (B) III, apenas.
- (C) I e II, apenas.
- (D) I e III, apenas.
- (E) I, II e III.

REFERÊNCIAS

CALAZANS, A. T. S.; PALDÊS, R. A.; MARIANO, A. M. Uma revisão sistemática da bibliografia sobre métricas funcionais de tamanho de software utilizando o enfoque metaanalítico. *Universitas. Gestão e Tecnologia*, [s. l.], v. 5, n. 2, p. 67–77, 2015. DOI 10.5102/un.gti.v5i2.3532.

DENNIS, Alan; WIXOM, Barbara Haley; ROTH, Roberta M. *Análise e projeto de sistemas*. [Recurso Eletrônico]. 5. ed. Rio de Janeiro: LTC, 2014.

PAULA FILHO, Wilson de Pádua. *Engenharia de software: fundamentos, métodos e padrões*. [Recurso Eletrônico]. 3.ed. Rio de Janeiro: LTC, 2009. ISBN 978-85-216-1650-4.

PAULA FILHO, Wilson de Pádua. *Engenharia de software: projetos e processos*. [Recurso Eletrônico]. 4. ed. Rio de Janeiro: LTC, 2019.

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

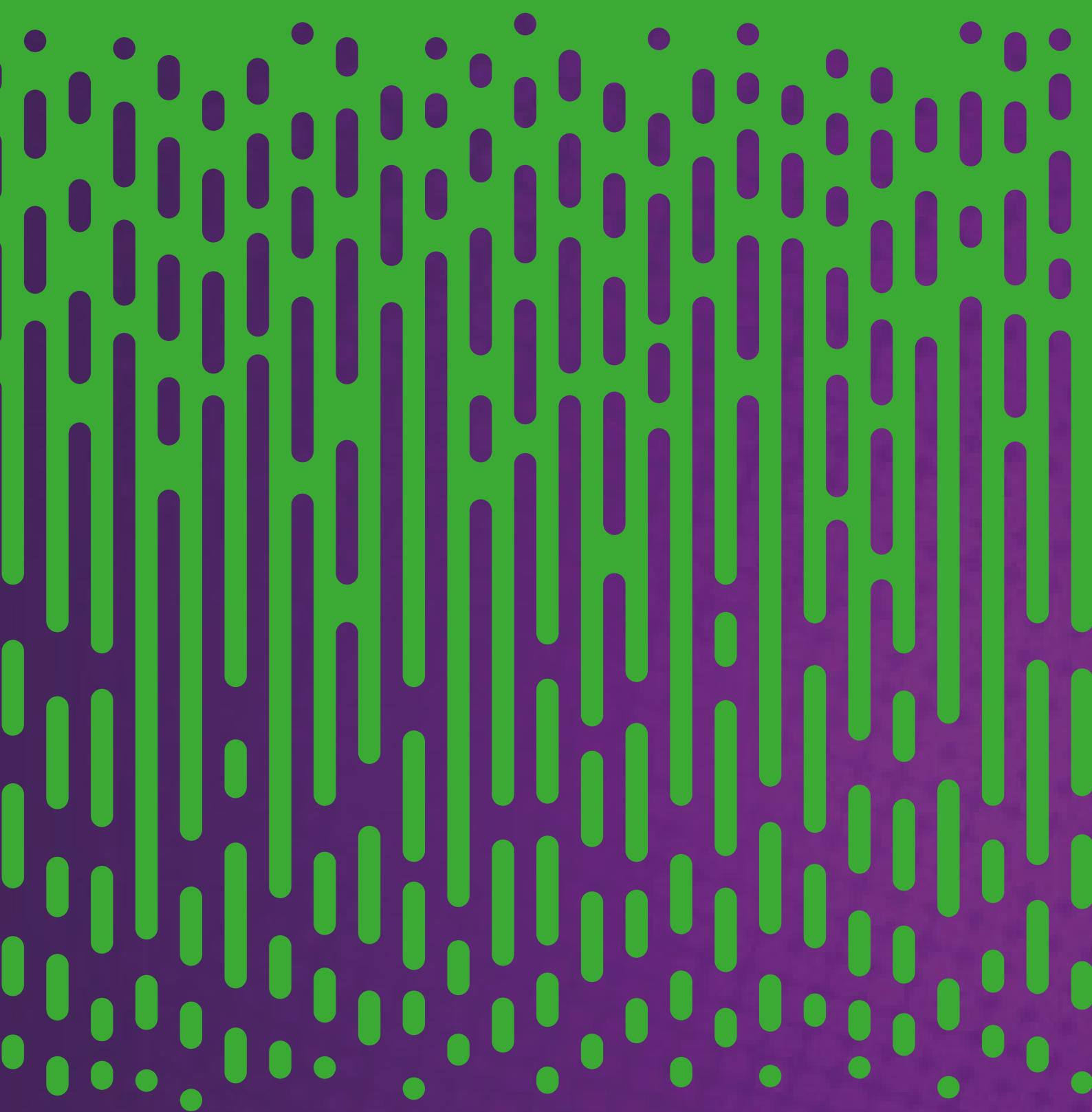
SCHACH, Stephen R. *Engenharia de software: os paradigmas clássico e orientado a objetos*. [Recurso Eletrônico]. 7. ed. Porto Alegre: AMGH, 2010.

SOMMERVILLE, Ian. *Engenharia de software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

VAZQUEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira; ALBERT, Renato Machado. *Análise de pontos de função: medição, estimativas e gerenciamento de projetos de software*. [Recurso Eletrônico]. 13. ed. São Paulo: Érica, 2013.

VETORAZZO, Adriana de Souza. *Engenharia de software*. [Recurso Eletrônico]. Porto Alegre: SAGAH, 2018.

WAZLAWICK, Raul Sidnei. *Engenharia de software: conceitos e práticas*. [Recurso Eletrônico]. Rio de Janeiro: Elsevier, 2013.



uniavan.edu.br