MEDIOtec

Curso Técnico em Informática

Análise e Projeto de Sistemas

Angélica Toffano Sedel Calazans

Presidência da República Federativa do Brasil
Ministro da Educação
Secretaria de Educação Profissional e Tecnológica
Instituto Federal de Brasília – IFB

Reitora
Luciana Miyoko Massukado

Pró-Reitora de Ensino – PREN
Yvonete Bazbuz da Silva Santos

Diretora de Educação a Distância – DEaD
Jennifer de Carvalho Medeiros

Apoio Administrativo
Cláudia Sabino Fernandes
Joscélia Moreira de Azevedo
Noeme César Gonçalves

Coordenador Geral - DEaD
Hênio Delfino Ferreira de Oliveira

Coordenadora Adjunta Administrativo
Cláudia Sabino Fernandes

Coordenadora Adjunta Ensino
Luciana Brandão Dourado

Coordenadora Adjunta Produção de Conteúdos para EaD
Sylvana Karla da Silva de L. Santos

Coordenador Adjunto Tecnologia
Hugo Silva Faria

Orientador de Ensino e Aprendizagem
Jefferson Amauri Leite de Oliveira

Coordenador Curso Técnico em Informática
Heitor Barros

Equipe de Elaboração

Conteudista
Angélica Toffano Sedel Calazans

Revisora de Conteúdo
Lidiane Szerwinsk Camargos

Ilustrador e Produtor de Vídeos Animados
Márlon Cavalcanti Lima

Diagramador
Fábio Lucas Vieira

Gestora de Ambiente Virtual de Aprendizagem
Daniele Cadête de Araujo Lima

Catalogação na fonte pela Biblioteca do instituto Federal de Educação, Ciência e Tecnologia - Brasília

Sumário

1.1 Projeto de software

Neste tópico, estudaremos o que é e o que compõe um projeto de software. Esse é um conceito-chave, pois é base de tudo que veremos nesta unidade.

Contudo, talvez seja mais fácil ir por parte

O que é um projeto?

Segundo o PMI (2018), “projeto é o esforço temporário empreendido para criar um produto, serviço (...)”. Quando vamos construir uma casa temos um projeto. Temos o esforço de várias pessoas – arquiteto, engenheiro, pedreiro, servente – que vão executar, durante um tempo, uma série de atividades para construir o produto que é a casa.

Fonte: Disponível em: < https://pixabay.com/pt/software-cd-dvd-digital-disco-871026/>.

Bora Refletir!

E quando decidimos estudar informática, temos também um projeto? Quem está envolvido nesse projeto? Quais as suas atividades? Qual o produto final?

O que é um software?

Para Sommerville (2011), um software são "programas de computador e a documentação associada". No seu celular, existem uma série de softwares ou app. Você consegue identificar alguns deles?

Agora ficou fácil definir projeto de software.

"Um projeto de software é um conjunto de atividades e resultados associados que produz um produto de software." (SOMMERVILLE, 2011)

Esclarecendo: quando falamos em projeto de software ou sistema estamos nos referindo também ao processo de desenvolvimento de software adotado para construção daquele software. Existem muitos processos e abordagens que se propõem à construção de um software.

Vamos ver um exemplo? Estamos desenvolvendo um Sistema de Biblioteca, ou seja, um projeto de software de biblioteca. Esse sistema, para ser desenvolvido, necessita seguir algum processo de desenvolvimento de software. Posso desenvolver esse sistema seguindo a Abordagem de Orientação a Objeto ou posso desenvolver esse sistema seguindo uma Abordagem de Métodos ágeis.

Não existe um processo ideal ou perfeito para a construção de um determinado software. Os processos de desenvolvimento evoluíram e estão constantemente evoluindo por vários motivos: o avanço da tecnologia, a capacidade das pessoas, o tipo de sistema a ser desenvolvido, tipo de organização que vai desenvolver ou utilizar o produto (Medeiros, 2004). Assim, embora existam muitos processos de sistema, algumas atividades são comuns a todos eles, conforme você pode visualizar na Figura 1.

Figura 1 - Atividades comuns ao processo de desenvolvimento de sistema
Fonte: adaptada de Castro et al, 2014.

Essas atividades podem ser executadas em diferentes sequências e agrupadas em diferentes etapas de acordo com a metodologia adotada. A literatura apresenta uma série de modelos de processo de desenvolvimento de sistema, tais como: Cascata, Prototipação, Processo Unificado (Orientação a objetos), Métodos Ágeis (XP, SCRUM) e outros. Detalharemos esses modelos na Unidade 3.

Vamos agora detalhar melhor o que acontece na atividade relacionada aos requisitos.

1.2 A importância da Engenharia de Requisitos no projeto de software

Antes de verificarmos a importância da Engenharia de requisitos, vamos compreender o que é requisito de software.

Requisitos são descrições dos serviços oferecidos pelo software (SOMMERVILLE, 2011).

Ou

Requisito de software é uma AÇÃO que o software deve executar e que possui características e condições próprias para automatizar um processo ou necessidade do cliente (CASTRO et al, 2014).

Bora Refletir!

Gosto sempre de fazer a comparação com a construção de uma casa. Quando vamos construir, temos que definir a metragem da casa, a quantidade de quartos, portas, janelas etc. Com relação ao desenvolvimento de software, temos que identificar o que o cliente quer. Por exemplo: se fôssemos desenvolver um aplicativo ou software de comércio eletrônico de roupas, ele precisaria de uma listagem de todos os produtos? Se sim, essa listagem deve ter um filtro para que o cliente possa filtrar o tipo de roupa que está procurando? Um filtro por preço? Tudo isso são os requisitos do sistema. Você consegue imaginar mais alguns requisitos com relação a este contexto? Procure na internet, entre em sites de comércio eletrônico e veja que tipo de consultas eles proporcionam. Cada consulta identificada é um requisito que foi definido.

Veja a tirinha do Dilbert sobre requisitos. Retrata bem a dificuldade de elicitar requisitos do cliente.

Figura 2: Tirinha sobre Requisitos
Fonte: Disponível em: <https://esdepre.files.wordpress.com/2012/03/dilbert-requisitos-de-sistema.jpg>.

Assim, podemos afirmar que Engenharia de Requisitos (ER) visa definir as formas para elucidar, organizar e documentar os requisitos de maneira sistêmica. Para tanto, a compreensão completa do problema, a definição dos requisitos do software e sua especificação detalhada é fundamental para a obtenção de um software com alta qualidade.

Não importa quão bem projetado ou codificado esteja um programa, se os requisitos forem mal analisados e especificados, futuramente poderá trazer problemas para o cliente e o desenvolvedor. Por isso, a Engenharia de Requisitos (ER) é importante para a compreensão da Engenharia de Software (ES).

O processo de Engenharia de requisitos é composto de várias etapas conforme a figura 3. A seguir, vamos detalhá-las para que você compreenda e consiga trabalhar com requisitos de forma eficaz.

Figura 3 – Principais etapas do processo de Engenharia de requisitos
Fonte: (IEEE, 2014)

Na fase de Elicitação se tenta obter o máximo de informações possíveis para o conhecimento do sistema. Busca-se, nesta fase, responder questões do tipo: quais os problemas do cliente? Quem são os atores envolvidos? Quais as necessidades com relação ao sistema? Qual o escopo do sistema? Estaremos detalhando melhor essa fase no próximo tópico.

Nessa etapa, se você estiver utilizando a abordagem Orientada a Objeto, você provavelmente fará um documento de visão ou mesmo um caso de uso relativo ao documento de visão detalhando essas informações; se estiver utilizando uma abordagem ágil, poderá ter que criar um cartão de visão (no caso da abordagem extreme programming ou XP).

Na fase de análise, aprofunda-se sobre o conhecimento obtido na fase Elicitação e começa-se a identificar as funcionalidades do software a ser desenvolvido. Para atender às necessidades do cliente, quais as funções ou módulos o software deve ter?

Na fase de especificação, já conhecendo as funções ou módulos, iniciamos a escrita dos requisitos. Para isso, temos que conhecer os principais tipos de requisitos. No tópico classificação de requisitos detalharemos melhor esse assunto.

Lembra que existem vários tipos de metodologias para desenvolver o produto de software?

A literatura apresenta uma série de modelos e/ou abordagens de processo de desenvolvimento de software, tais como Cascata, Espiral, Orientado a objeto, RUP, Métodos Ágeis.

Assim, se estivermos desenvolvendo seguindo a abordagem Orientada a objeto no momento da especificação, estaremos desenhando os diagramas de caso de uso e descrevendo os casos de uso. Se utilizarmos uma abordagem de Métodos ágeis, por exemplo o SCRUM, estaremos escrevendo história de usuários ou talvez casos de uso, dependendo de como a organização define os requisitos no modelo ágil. Detalharemos essas duas técnicas, casos de uso e história de usuário, na próxima unidade.

Na fase de validação se obtém o aceite do cliente sob determinado artefato que foi gerado. Esses artefatos podem ser os documentos de visão, diagramas, os casos de uso, as histórias de usuário. Esta atividade significa aprovar junto ao cliente as restrições, escopo, os diagramas de caso de uso e as descrições de caso de uso (se estiver utilizando a abordagem OO) ou história de uso (se estiver utilizando metodologia ágil) etc. Lembre-se de que o cliente deve registrar a validação realizada. Isso é realizado por meio de atas de reunião validadas, e-mail e outros documentos.

Todas essas tarefas são complexas, dependem do entendimento, da comunicação entre pessoas que têm visões diferentes (negócios vs técnica). São resultados de muitas horas de trabalho. Para entender melhor essas dificuldades vamos ler o texto a seguir.

Saiba mais!

Elicitar e analisar requisitos não são tarefas fáceis. Leia o texto “A difícil arte de Analisar requisitos” (Disponível em: https://esdepre.wordpress.com/2012/03/26/a-dificil-arte-de-analisar-requisitos/). Ele demonstra as principais dificuldades para realizar essas atividades.

Vamos agora entender melhor o que é feito na etapa Elicitação de requisitos.

1.3 Principais conceitos: atores (organizações, sistemas, clientes), stakeholders, fronteiras, restrições

Na etapa de Elicitação, conforme já vimos, temos que identificar as principais necessidades do cliente com relação ao sistema, os atores envolvidos (cliente, organizações, equipamentos, sistemas etc.), as fronteiras e restrições.

Aqui é importante envolver e ouvir o cliente. O sistema deve atender às suas necessidades.

Antes de entendermos as perguntas que deverão ser respondidas, nesta etapa, é interessante entender o conceito de stakeholders, atores, fronteiras e restrições.

O que são atores (organizações, equipamentos, sistemas)?

Quando falamos em atores, na Engenharia de Requisitos, estamos identificando todos os personagens que vão interagir com projeto do sistema a ser construído. Bezerra (2006) acredita que o ator possui um papel no sistema e indica algumas categorias de atores: cargos (empregado, cliente, gerente etc.), organizações (Administradora de cartões, Correios etc.), outros sistemas (CADIN, SERASA, Receita Federal etc.), equipamentos (leitora de código de barras, sensor etc.). Atores representam o "papel exercido por uma pessoa que interage com o sistema" (Bezerra, 2006).

Mas o que são stakeholders?

Segundo Sommerville (2011), o termo stakeholder é utilizado para "se referir a qualquer pessoa ou grupo afetado pelo sistema, direta ou indiretamente". Podem ser os usuários finais, engenheiros que estão desenvolvendo sistemas relacionados, gerentes de negócio, especialistas daquele domínio. Podem ou não ser atores do sistema. Se interagirem com o sistema serão atores com a denominação correta: cliente, fornecedor, gerente etc.

Para identificar os stakeholders algumas perguntas podem ser feitas: Quem definirá as necessidades, o escopo do sistema? Quem vai utilizar o sistema? Quem será afetado pelas saídas do sistema? Quem vai avaliar e aprovar o sistema quando ele for entregue?

Assim, é muito importante identificar corretamente os stakeholders do seu projeto. Já pensou se você se esquecer de identificar algum stakeholder? Isso pode impactar no seu projeto, você não acha? Podem ser esquecidos alguns requisitos importantes.

Fique Ligado!

E os desenvolvedores, são atores? Não. Os desenvolvedores ou analistas participam da criação do sistema. Participam do processo de desenvolvimento do sistema. Se eles depois vierem a utilizar o sistema com o papel de cliente, aí sim eles terão esse papel, pois estarão interagindo com o sistema na condição de cliente.

Ficou claro?

Outro conceito importante são as fronteiras do sistema. A fronteira é o limite de atuação do sistema. Na definição da fronteira estamos identificando o que estará dentro e o que estará fora dos objetivos do sistema.

Imagine que vamos desenvolver um sistema de biblioteca. O nosso sistema inicialmente só terá como objetivos a manutenção do acervo (livros) e dos clientes (empréstimo, devolução e reserva de livros). Poderíamos identificar então como atores: o cliente e a bibliotecária.

Não estão nessa fronteira a compra de livros (departamento financeiro), a contabilidade da biblioteca (departamento contábil) etc. E, por isso, não são demonstrados como atores. E se mais adiante se identificar a necessidade de uma ligação com o sistema financeiro (para a compra de novos livros)? Então teremos mais um ator representado pelo sistema financeiro.

A Figura 4 apresenta como poderíamos representar a fronteira desse sistema na versão inicial.

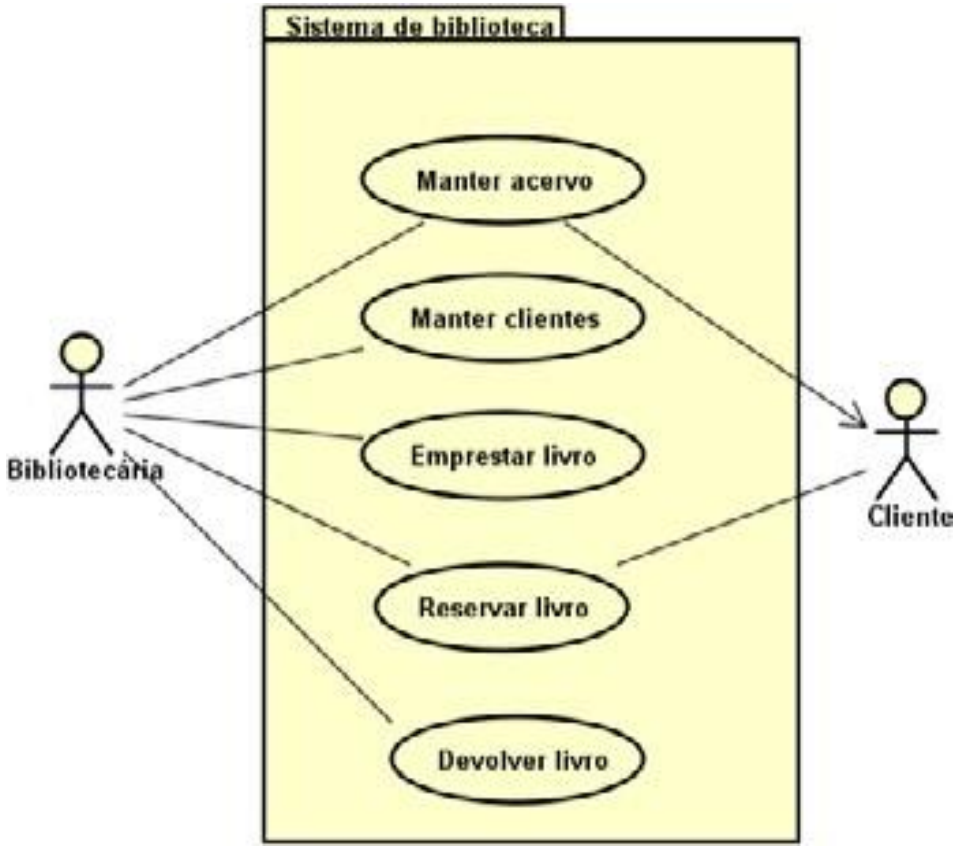


Figura 4 – Exemplo de Caso de uso identificando a fronteira do Sistema de biblioteca
Fonte: CASTRO et al, 2014.

Finalmente, vamos para o último conceito importante que são as restrições.

O que são restrições?

Restrições são as limitações internas ou externas ao projeto de sistema. Elas existem para definir os limites do projeto e impactam o projeto. Algumas delas podem ser constatações de que ações podem ser realizadas para diminuir ou eliminar essas restrições.

Alguns exemplos:

Vamos pensar que estamos construindo um sistema para o lançamento de um e-commerce para a venda de produtos de natal (árvores de natal, bolas, enfeites). Concorda que a restrição seria o sistema estar pronto até antes do Natal? Essa seria uma restrição com relação ao prazo.

Os computadores de determinada empresa recebem manutenção todos os finais de semana. Você concorda que uma restrição é o sistema a ser desenvolvido não estar disponível todos os dias? Um sistema disponível todos os dias é identificado como 7/24 (sete dias com 24 horas). Assim, o sistema a ser desenvolvido seria 5/24 (cinco dias com 24 horas? Esse é um exemplo de restrição tecnológica.

E com relação a custo? Você consegue pensar em algum tipo de restrição possível?

Se o seu stakeholder que define o sistema junto à equipe de desenvolvimento só pode estar disponível 1 dia por semana, isso pode ser considerado uma restrição? Por quê?

Se o sistema precisa de 50 GB no servidor, que atualmente só tem 30 GB, você considera isso um exemplo de restrição? Por quê?

Saiba mais!

Se quiser entender melhor as restrições, esse texto da equipe AEVO detalha mais esse assunto. Ele também aborda o conceito de premissas, que é muito interessante.

http://blog.aevo.com.br/premissas-e-restricoes-em-projetos-o-que-eu-preciso-saber/

Vamos voltar agora às atividades executadas durante a Elicitação. Agora que já entendemos alguns conceitos como stakeholders, atores, fronteiras, restrições é importante lembrar de algumas questões que devem ser respondidas nessa etapa, tais como (SBROCCO, 2012):

- Quais os problemas que estamos tentando resolver?
- Quais os objetivos do sistema que vamos desenvolver?
- Quem são os stakeholders?
- Quais são as fronteiras?
- Quais são as restrições?

Como já dissemos, dependendo a metodologia adotada no projeto de desenvolvimento de sistemas, essas informações comporão um documento de visão (Orientação a objeto), um cartão de visão (Metodologia ágil). O mais importante aqui não é o documento em que as informações são inseridas, mas o conteúdo dessas informações ser o mais aderente às necessidades dos stakeholders, de forma a garantir a completa elicitação das informações iniciais necessárias para a construção de um sistema.

Alguns autores citam o Método 5W2H para facilitar e complementar esse trabalho de elicitação. É o que veremos a seguir.

1.4 Método 5W2H (What, Who, Why, When, Where, How, How much)

O método 5W2H facilita a identificação de forma organizada das informações em qualquer projeto, inclusive projetos de TI. Representa as palavras em inglês *What* (O quê), *Who* (Quem), *Why* (Por quê), *When* (Quando), *Where* (onde), *How* (como) e *How much* (Quanto). Essas perguntas correspondem às perguntas básicas relacionadas a requisitos (SBROCCO, 2012).

Vamos detalhar?

What - O que o sistema tem que fazer? Quais os seus os problemas? Quais os objetivos? Quais as funções ou módulos? Quais as entradas? Quais as saídas?

Who - Quem são os atores envolvidos? Quem são os stakeholders? Quem vai validar as entregas do sistema?

Why - Por que esse processo existe? Por que estamos desenvolvendo esse produto?

When - Quando será executado? Quando será avaliado?

Where - Onde será executado? Onde será avaliado?

How - Como será executado?

How much – Quanto irá custar?

Bora Refletir!

Vamos voltar ao nosso Sistema de biblioteca:
Você consegue responder a algumas dessas perguntas, outras você necessitaria discutir mais com o stakeholder. Você consegue identificar quais as perguntas você responderia e quais seriam necessários mais detalhamento?

Saiba mais!

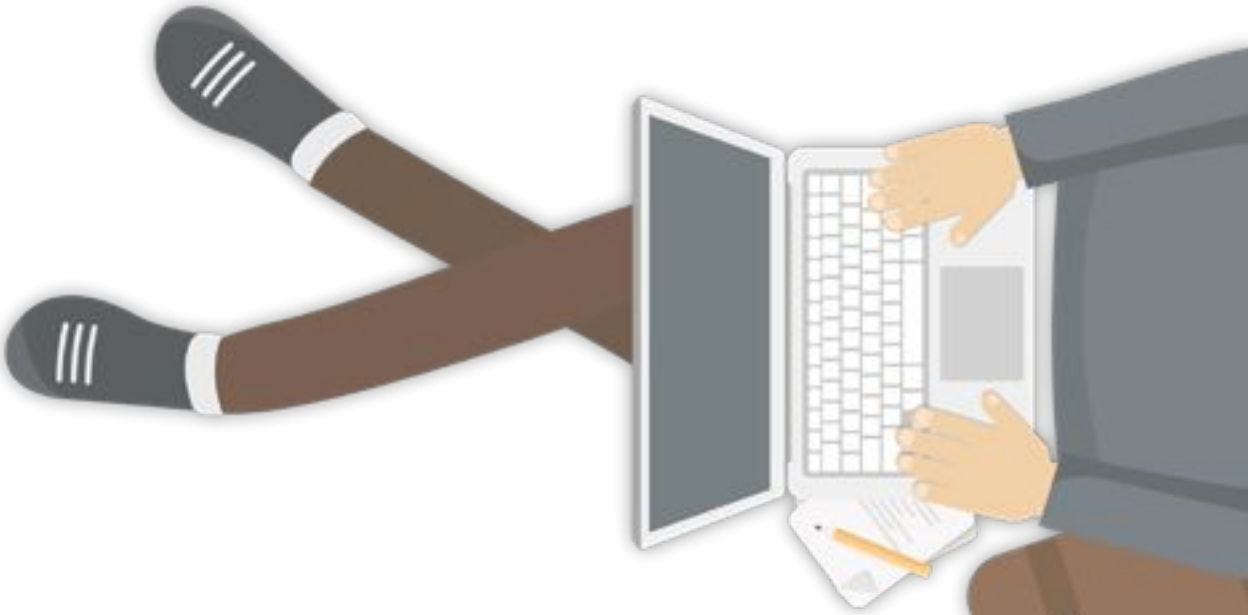
Veja o vídeo de Suedilson Filho sobre o 5W2H; ele complementa as informações sobre o método.
https://www.youtube.com/watch?v=o1nqbxTzqMU

A seguir, veremos alguns conceitos sobre a classificação dos requisitos. Lembrando: após a Elicitação, temos a Análise e a Especificação dos requisitos. E para isso, precisamos entender que existem tipos de requisitos diferentes. Esse assunto será muito importante para a próxima unidade, na qual exercitaremos a escrita de requisitos.

1.5 Classificação de requisitos

Os requisitos são normalmente classificados em requisitos funcionais, não funcionais e de domínio (SOMMERVILLE, 2011).

Requisitos funcionais definem o que o sistema deve fazer, como o próprio nome diz representam as funcionalidades do sistema. Descrevem que necessidades de negócio devem atender, os cadastros, os relatórios, as funcionalidades tangíveis ao usuário. No caso de utilizar a abordagem Ágil (SCRUM), os requisitos funcionais são cadastrados no "Product Backlog" e posteriormente se transformam em "estórias". No caso da abordagem Orientada a Objeto são definidos por meio de casos de uso.



Alguns exemplos de requisitos funcionais em descrição de caso de uso considerando o exemplo do sistema de biblioteca:

. este caso de uso destina-se a possibilitar uma bibliotecária realizar a manutenção do cadastro dos clientes através das operações de inclusão, alteração, exclusão e consulta;

. este caso de uso destina-se a possibilitar uma bibliotecária realizar a manutenção do acervo dos livros através das operações de inclusão, alteração, exclusão e consulta.

Os requisitos não funcionais são aqueles não relacionados diretamente ao objetivo do sistema, mas que determinam "como" e "onde" o sistema deve funcionar. Por exemplo, critérios de segurança e criptografia, escalabilidade, disponibilidade, compatibilidade, portabilidade, etc. Além disso, os requisitos não funcionais também são cadastrados como itens do product backlog.

Alguns exemplos de requisitos não funcionais em linguagem natural considerando o exemplo do sistema de biblioteca:

. as funcionalidades de consulta ao acervo devem funcionar tanto em Firefox como Internet Explorer (característica de portabilidade);

. todas as funcionalidades do sistema devem seguir o padrão visual da instituição (cores, layouts, fontes) referente à Característica de conformidade.

Ambos os requisitos funcionais e não funcionais fazem parte do escopo do produto. E tanto um como outro são muito importantes para garantir a eficácia do sistema. Todos os requisitos devem ser testáveis, ou seja, devem ser escritos de forma a que seja possível testá-los.

Saiba mais!

https://www.youtube.com/watch?v=V74qIKo-OqI

Os requisitos de domínio são derivados do domínio do sistema. Podem ser funcionais ou não funcionais. Podem ser requisitos técnicos relacionados a padrões de criptografia, de acesso, velocidade, interfaces padrões etc. Podem também ser requisitos específicos de negócio, cálculos específicos do domínio negócio, necessários para que o sistema funcione. Podem ser restrições.

Alguns exemplos de requisitos de domínio em linguagem natural considerando o exemplo do sistema de biblioteca.

A interface do usuário com o banco de dados deve ser baseada no padrão Z39.50.

A codificação do acervo de livros deve obedecer ao padrão Classificação Decimal de Dewey (CDD).

Figura 5 -Tirinha
Fonte: imagem disponível em:
https://oblogdotarcisio.wordpress.com/tag/requisitos/

Encerramento

Caro estudante, nesta unidade você estudou os principais conceitos relacionados ao Projeto de Sistema. Entendeu que quando falamos de projeto de sistema estamos nos referindo também ao processo de desenvolvimento de sistema. Que existem muitos processos/abordagens que se propõem a construção de um sistema. Não existe um processo ideal, é importante escolher, de acordo com o projeto, as necessidades, a arquitetura e o processo mais adequado ao seu sistema.
Independente do processo escolhido, a atividade de requisitos existe em todos os processos.

Estudamos os conceitos de requisitos e as etapas da Engenharia de Requisitos. Entendemos os principais conceitos de atores, stakeholders, fronteiras e restrições. Vimos ainda as informações essenciais para a elicitação das necessidades dos clientes para a construção do sistema. Estudamos que o método 5W2H pode ajudar a obter mais informações para a etapa de Elicitação. Por fim, estudamos os requisitos funcionais, não funcionais e de domínio.

Espero que tenha gostado e assimilado estes conhecimentos de forma a aplicá-los na sua vida profissional.

Nas próximas unidades detalharemos melhor esses conceitos e aprenderemos outros. Vamos lá?

Unidade 2

Modelagem, especificação, validação e verificação de requisitos

2.1 – A importância da modelagem e suas ferramentas

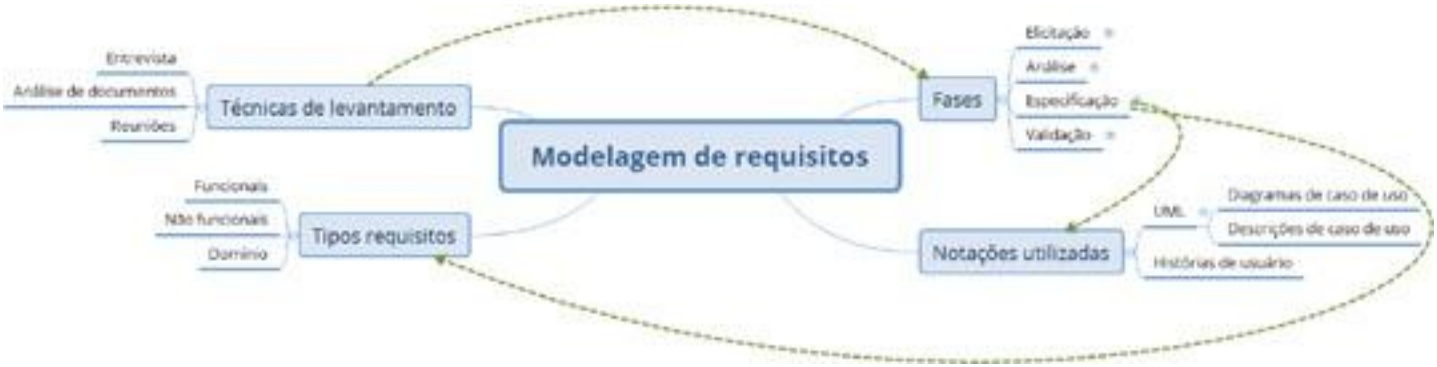


Figura 1 – Conceitos de modelagem de requisitos.
Fonte: elaborado pela autora.

Neste tópico, estudaremos a importância da modelagem de requisitos e suas ferramentas. A Figura 1 apresenta os principais conceitos relacionados à modelagem de requisitos. Alguns desses conceitos já estudamos na unidade anterior, como as fases da Engenharia de requisitos (Elicitação, Análise, Especificação e Validação). Também vimos classificação de requisitos em Requisitos funcionais, não funcionais e de domínio. Lembra? Todos esses conceitos estão bastante interligados.

Analisando a figura 1, você pode verificar o link que existe entre a fase de especificação, as notações utilizadas para representar os requisitos e os tipos de requisitos. Lembre-se de que na fase de especificação você deve começar a escrever os requisitos (funcionais, não funcionais e de domínio). A forma dessa escrita vai depender da notação adotada pelo projeto de sistemas, UML, histórias do usuário etc.

Vamos nos aprofundar nessa discussão. Inicialmente, vamos ver o conceito de modelagem de requisitos.

O que é modelagem de requisitos?

A modelagem de requisitos tem como objetivo a construção de modelos que representem os requisitos do sistema. Normalmente, utiliza-se algum tipo de notação ou ferramenta para modelar os requisitos.

A modelagem de requisitos é realizada com o objetivo de simplificar a complexidade para representar as necessidades do sistema (MEDEIROS, 2004). Não temos a facilidade de entender e gravar todas as informações sobre tudo que o software precisa realizar, por isso utilizamos modelos. Normalmente é mais fácil compreender modelos desenhados do que textos escritos. Você concorda?

Veja os exemplos a seguir:

Figura 2. Emojis utilizados em aplicativos para conversas
Fonte: Whatsapp.

Figura 3. Imagem do Campus Brasília do IFB.
Fonte: retirada do Google Maps.

Mas, que notações existem para a modelagem de requisitos?

No caso de Sistemas Orientados a objetos, a UML – Linguagem de Modelagem Unificada (Unified Modeling Language) – é a linguagem utilizada para modelar esses sistemas. A UML auxilia os desenvolvedores a definir as características do software, seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos. A modelagem de requisitos nesse contexto utiliza diagramas de casos de uso e descrições de casos de uso, lista de requisitos não funcionais para representar os requisitos não funcionais e de domínio (se necessário).

No caso de sistemas construídos com metodologias ágeis, a notação utilizada para os requisitos são normalmente histórias de usuário, casos de uso, lista de requisitos não funcionais etc. Isso depende da metodologia adotada (XP, SCRUM ou outra) e dos padrões da organização.

Vamos conhecer um pouco mais de UML? Vamos ver um vídeo muito interessante sobre a UML: https://www.devmedia.com.br/uml/8579 .

Quer esclarecer melhor esse assunto? Leia o texto sobre o conceito de modelagem e diagramação.

Leitura complementar

http://spaceprogrammer.com/uml/introduzindo-o-conceito-de-modelagem-e-diagramacao/

Qualquer que seja a notação utilizada, UML ou outra, é importante entender que a modelagem de um sistema é uma forma de documentá-lo. Mas, para que documentar?

Os sistemas estão em constante mudança. Os clientes desejam melhorias, o mercado força a adoção de estratégias para que as empresas se tornem mais competitivas, as leis são modificadas com o passar dos anos, os sistemas podem ter falhas. Todos esses fatores e outros podem repercutir em mudanças no sistema que foi desenvolvido ou mesmo que está sendo desenvolvido. E uma documentação completa vai ajudar para que a manutenção do sistema ocorra de maneira correta, sem gerar novos erros ao corrigir os antigos.

Agora vamos conhecer algumas das técnicas de levantamento de requisitos. Qualquer que seja a notação utilizada, UML ou outra, é importante entender que a modelagem de um sistema é uma forma de documentá-lo. Mas, para que documentar?

Os sistemas estão em constante mudança. Os clientes desejam melhorias, o mercado força a adoção de estratégias para que as empresas se tornem mais competitivas, as leis são modificadas com o passar dos anos, os sistemas podem ter falhas. Todos esses fatores e outros podem repercutir em mudanças no sistema que foi desenvolvido ou mesmo que está sendo desenvolvido. E uma documentação completa vai ajudar para que a manutenção do sistema ocorra de maneira correta, sem gerar novos erros ao corrigir os antigos.

Agora vamos conhecer algumas das técnicas de levantamento de requisitos.

2.2 – Técnicas de levantamento de requisitos

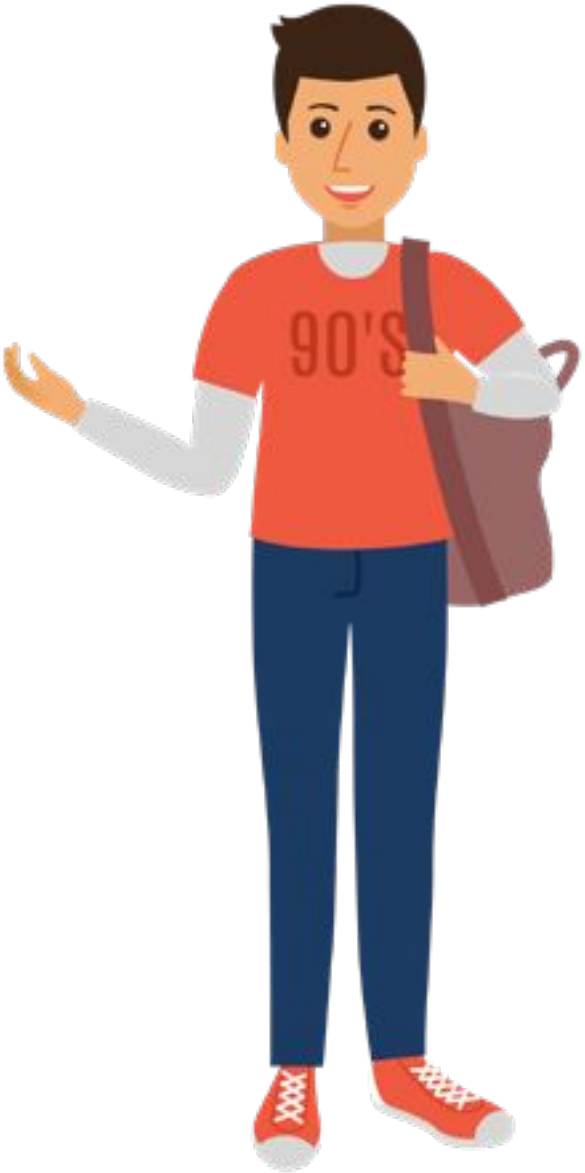
São muitas as técnicas que podem ser utilizadas para o levantamento de requisitos. Agora, estudaremos as mais utilizadas: análise de documentos, entrevistas, workshop/reunião ou oficina, prototipação (CASTRO et al, 2014).

A análise de documentos é um passo muito importante na elicitação de requisitos. É necessário conhecer a organização, as áreas envolvidas, o sistema existente (se for uma manutenção) ou as metas e objetivos organizacionais com relação ao sistema a ser desenvolvido. É importante para o desenvolvedor ir à primeira entrevista já com um conhecimento prévio do ambiente, do sistema, da organização. Essas informações podem ser obtidas na internet (site da organização) ou mesmo em documentos organizacionais que sejam entregues pelos stakeholders para a equipe.

A entrevista é a técnica de elicitação mais utilizada e de simples implementação. Isso se deve ao fato de que a comunicação utilizada nas entrevistas é mais natural, simples e pode ser utilizada em várias situações.

O objetivo da entrevista é a obtenção de informações, fatos vivenciados, tendências e experiências sobre a empresa e suas atividades. Fornece uma visão concreta da realidade. Os requisitos são derivados das respostas a essas questões.

Segundo Sommerville (2011), as entrevistas podem ser fechadas e abertas. Fechadas quando o desenvolvedor leva uma série de questões pré-definidas a serem respondidas; e abertas quando não existe um roteiro pré-definido de questões e a equipe de desenvolvedores explora diversos assuntos objetivando o entendimento do sistema.



Fique Ligado!

Para realizar uma entrevista, é necessário que o desenvolvedor se prepare, considerando os seguintes pontos (Castro et al, 2014):

- Identificar ideias gerais sobre a organização, o tipo de produto, o domínio do problema;
- Estudar os documentos organizacionais;
- Identificar as pessoas a serem entrevistadas;
- Definir objetivos, escopo e duração da entrevista (previsão do tempo da entrevista);
- Preparar um roteiro;
- Providenciar material necessário para a entrevista.

Além disso, é importante que ao iniciar a entrevista, o entrevistador apresente a equipe, os objetivos da entrevista, a previsão do tempo a ser gasto. E, se for gravar, que peça autorização. Ao término desta, ele não pode se esquecer de agradecer, fazer um breve resumo do que foi discutido, informar como essas informações serão validadas pelo cliente (ata, e-mail etc.) e, se necessário, marcar a próxima entrevista.

Saiba mais!

Reforce essas etapas assistindo ao vídeo sobre essa técnica.
https://www.youtube.com/watch?v=XqnSRGGBDuU

A técnica de workshop, também chamada de oficina, tem por objetivo juntar os stakeholders de um projeto por um determinado período de tempo para discutirem os aspectos mais importantes para o desenvolvimento da aplicação. Trata-se de uma técnica de elicitação em grupo usada em uma reunião estruturada. Fazem parte do grupo os desenvolvedores e uma seleção dos stakeholders que melhor representam o contexto em que o sistema será usado, possibilitando a obtenção de um conjunto de requisitos bem definidos. O workshop tem o objetivo de incentivar o trabalho em equipe, o consenso e a concordância em um curto espaço de tempo (CASTRO et al, 2014).

O workshop traz alguns benefícios, tais como: envolvimento de todos os interessados, agiliza o entendimento do projeto do Sistema, facilita a obtenção de consenso, pois agrega as partes interessadas.

Fique Ligado!

O workshop necessita também de uma preparação detalhada que envolve:

definição de participantes, definição de agenda, logística, distribuição do material com antecedência para que todos os participantes cheguem à reunião informados sobre o que será discutido e o que se espera como resultado e, por fim, sensibilização envolvendo os benefícios da sessão.

Os workshops são, muitas vezes, realizados em uma série de sessões, com duração de 3 ou 4 horas cada. Estas sessões são realizadas duas ou três vezes por semana e, em média, possuem 4 a 12 participantes. Isto inclui tanto o domínio do negócio como da solução (CASTRO et al, 2014).

Antes dos workshops, normalmente, são realizadas entrevistas para que os desenvolvedores entendam o domínio do negócio, a política, o problema e os objetivos do projeto.

A técnica de Prototipação (BEZERRA, 2006) serve de complemento na análise de requisitos. Protótipos são esboços e podem ser criados para telas de entrada, de saída, consultas ou mesmo o sistema inteiro.

Na Engenharia de software, após a elicitação de requisitos, o protótipo é construído para ser validado junto aos usuários. Essa técnica assegura que os requisitos foram bem entendidos e que começaremos a produzir o produto correto.

As definições de requisitos são conceitos abstratos. Retratam o que o cliente pretende que o software faça. Retratam um pensamento. Os protótipos são coisas concretas, telas de entrada, de saída. Para o usuário trabalhar com coisas concretas é mais fácil; e para os desenvolvedores é a garantia de que o que foi entendido no processo de comunicação estava correto.

Vamos nos aprofundar nesse assunto? Existem vários tipos de prototipação...

Saiba mais!

Vamos assistir ao vídeo sobre a importância da prototipação:
https://www.youtube.com/watch?v=RI1XH64NC0Q

Vamos melhorar nossos conhecimentos? Leia sobre a importância e os tipos da prototipação e sobre as ferramentas que existem para construir protótipos.

Saiba mais!

Para complementar seus conhecimentos, acesse:
http://dextra.com.br/pt/blog/prototipacao-e-sua-importancia-no-desenvolvimento-de-software/

2.3 – Técnicas de especificação de requisitos - Orientação a objetos e métodos ágeis

Acabamos de estudar várias técnicas para levantar requisitos. O levantamento ou elicitação de requisitos não é uma tarefa fácil. E a utilização de técnicas ajuda a entender melhor as necessidades do cliente. A seguir, veremos técnicas de especificação de requisitos com foco na Orientação a objetos e nos Métodos ágeis.

Fonte: Imagem disponível em: <http://sosvip.blogspot.com/2010/05/fonte-dos-desejos-cuidado-com-o-que.html>.

Neste tópico vamos detalhar sobre como especificar requisitos considerando duas abordagens: a Orientação a objetos e os Métodos Ágeis.

Você recorda o que vimos na Unidade 1?

Fique Ligado!

A literatura apresenta uma série de modelos de processo de desenvolvimento de sistema, tais como: Cascata, Prototipação, Processo unificado (Orientação a objetos), Métodos ágeis (XP, SCRUM) e outros. Detalharemos esses modelos na Unidade III.

Na abordagem Orientação a objetos, as etapas de elicitação, análise e especificação, os requisitos podem utilizar uma ou mais das técnicas de levantamento de requisitos do tópico II. Muitas empresas utilizam entrevista e prototipação. Outras empresas, somente entrevistas. O mais importante é obter todas as informações sobre as necessidades do cliente com relação ao sistema. É identificar o escopo, as fronteiras, os atores e stakeholders.

Essas informações, na maior parte das empresas, são inseridas em um documento de visão. Sobre isso, Medeiros (2004) apresenta um modelo de documento de visão bem detalhado, mas sugere que cada organização customize o seu as suas necessidades. Nem todas as informações são necessárias para todos os sistemas. Veja no box abaixo:

Modelo de Documento Visão - Detalhado segundo Medeiros (2004, p.23-24)

1. **Introdução**

Reserve esta seção para uma descrição geral do trabalho esperado.

2. **Escopo**

Relacione os principais módulos que serão detalhados em perspectivas do produto (Veja os itens 8.1, 8.2 etc.).

3. **Definições Acrônimos e Abreviaturas**

Faça o possível para não utilizar termos relativos à área de informática, se isso for imprescindível, então forneça o significado desses termos nessa seção.

4. **Referências**

Relate a fonte na qual esse documento se baseou e os participantes desses eventos. Nomeie as pessoas, seus cargos e localidades.

5. **Oportunidades de Negócio**

Descreva qual é a oportunidade de negócio que está em apreciação como motivação para o surgimento desse software.

5.1 Problema a Ser Solucionado

Descreva qual é o problema que está sendo solucionado com esse software.

6. Descrições de Stakeholder e Usuários

Relacione os stakeholders com seus cargos, nomes, telefones, e-mails; e faça o mesmo com os usuários principais. Isso será uma boa referência para futuros participantes desse software.

1. **Stakeholder**
2. **Usuários**
3. **Ambiente Atual dos Clientes**

   Descreva o ambiente atual de informática relativo a esse software que será construído, como sistema operacional, tecnologias e outros que considerar importante.

7. Observação

Agregue qualquer informação que achar importante.

8. Módulos

1. Perspectiva do Produto: Nome do Módulo I
2. Perspectiva do Produto: Nome do Módulo II

...

8.N Perspectiva do Produto: Nome do Módulo N

...Descreva nesses tópicos os detalhes que cada módulo atenderá. Procure descrever o que o módulo faz, para que serve e como se relacionará com outros módulos, se isso for importante. Cada módulo será um componente no diagrama de "Caso de Uso" que representará o <sistema>>.

9. Precedência e Prioridades

Indique qual é a prioridade dos módulos e se ela será a mesma ordem de precedência para o desenvolvimento. Explique por que adotou essa estratégia e, principalmente, por que deixou de usar outras. Explicar isso pode ser importante para futuras reuniões, nas quais podem perguntar: "Mas, por que não fizemos isso dessa maneira?"

10. Requisitos Não Funcionais

Relate aqui as dependências de softwares externos, particularidades de performance que são esperadas, grandes necessidades de treinamento e outras que achar necessário.

11. Requisitos de Sistemas e Ambientes

Descreva o sistema operacional elegido, a linguagem de programação e os bancos de dados que serão utilizados. Nessa seção, informe se será utilizado um servidor de aplicação, qual é esse servidor, assim como o servidor de internet e outros softwares dos quais ele dependerá.

1. **Sistema Operacional**
2. **Linguagens de Desenvolvimento**
3. **Banco de Dados**

12. Requisitos de Documentação

Descreva em quais partes a documentação será dividida, quais são essas partes e o que o cliente pode esperar de cada uma.

13. Visão Geral UML do Sistema - Modelo Conceitual

Insira uma figura que represente o diagrama de Caso de Uso baseado neste documento.

Em relação aos requisitos funcionais, estes são especificados por meio de diagramas de casos de uso e descrições de caso de uso. Os requisitos não funcionais, normalmente, são definidos em listas de requisitos ou no documento de visão.

Um diagrama de caso de uso é um diagrama comportamental da UML que evidencia a visão externa do sistema a ser desenvolvido. Ele mostra quais são as funcionalidades desse sistema e quem interage com elas (BOOCH et al, 2004).

O diagrama de caso de uso mostra o que o sistema deve fazer sem se fixar no como, apresenta de forma gráfica as funcionalidades, possui convenções simples e de fácil entendimento para o usuário final; sua documentação pode ser detalhada em versões futuras.

Vamos voltar ao caso de uso do sistema de biblioteca? Vamos entender melhor a Figura 4, considerando as notações da Tabela 1?

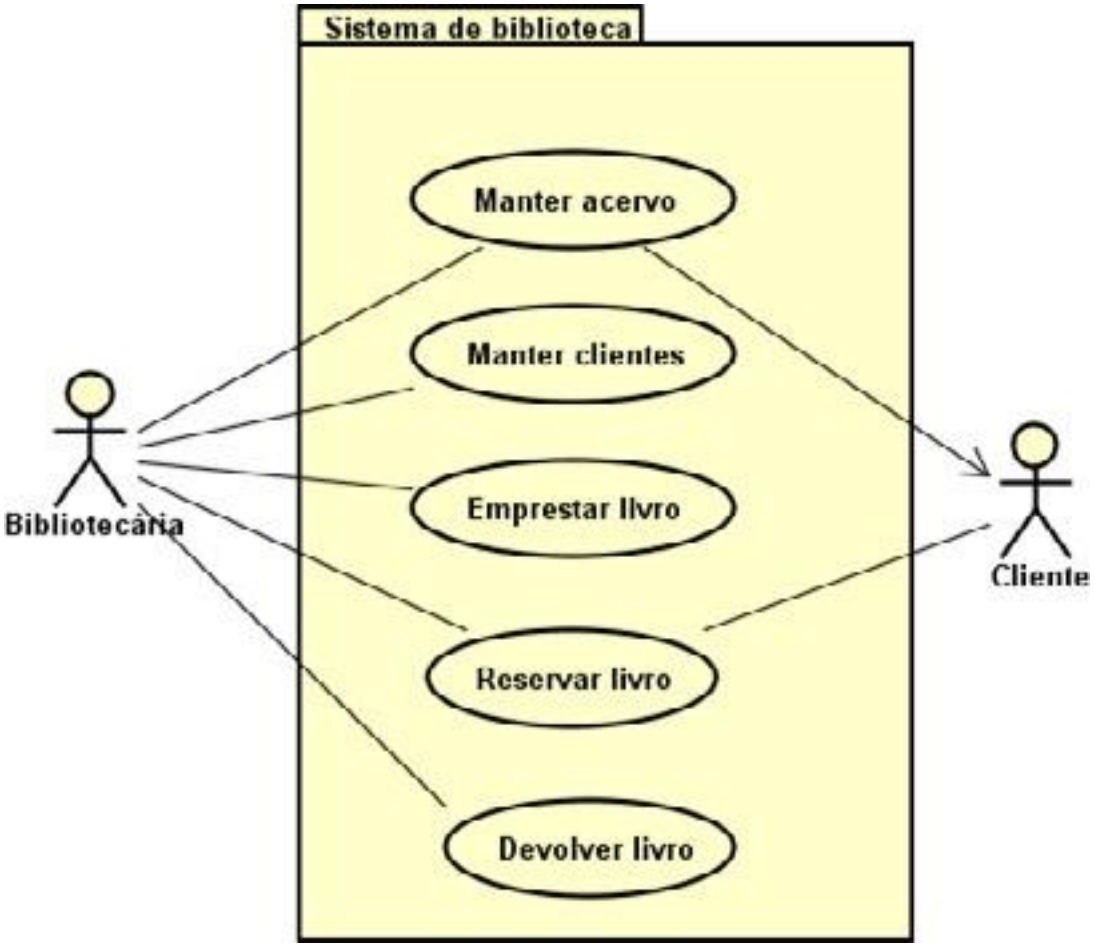


Figura 4 – Caso de uso Sistema de biblioteca
Fonte: CASTRO et al, 2014.

As principais notações estão apresentadas na Tabela 1 (adaptado de CASTRO et al, 2014):

ELEMENTO	SIMBOLOGIA	SIGNIFICADO
ATOR		É uma entidade externa que interage com o caso de uso. Representa um papel perante o sistema que pode ser de desempenhado por um grupo de pessoas, outro sistema, uma organização ou parte dela, um equipamento.
CASO DE USO		Representa uma funcionalidade que atende a um ou mais requisitos do cliente. Como nome, é suficiente usar um verbo no infinitivo + um complemento.
Associação ou (Relacionamento de Comunicação)		Representa um relacionamento entre o ator que utiliza o caso de uso. A ausência de uma seta indica que a comunicação se processa nos dois sentidos. Só use uma seta se a comunicação se processa em um sentido preferencial.

Tabela 1 – Principais notações do diagrama de caso de uso
Fonte: Adaptado Castro et al, 2014.

Fique Ligado!

Atores e os casos de uso são identificados a partir de informações coletadas no levantamento de requisitos (BEZERRA, 2006). Algumas perguntas são interessantes para identificar atores, tais como: que órgãos, empresas ou pessoas (cargos) irão utilizar o sistema? Que outros sistemas irão se comunicar com o software?

Outras perguntas ajudam a identificar os casos de uso, tais como: quais são as necessidades e objetivos de cada ator em relação ao sistema? Que informações o sistema deve produzir?

Além das notações citadas na Tabela 1, os seguintes tipos especiais de relacionamento podem ser usados destacando a dependência entre casos de uso: Generalização / Especialização; Extensão e Inclusão.

TIPO	SIMBOLOGIA	SIGNIFICADO	DIREÇÃO DA SETA
Generalização / Especialização		Depois de especificado um caso de uso, esse trabalho pode ser especializado. O caso de uso especializado (ou derivado) herda as características do caso de uso geral (ou base).	O caso de uso geral (ou base) recebe a ponta da seta.
EXTENSÃO		É a chamada opcional de um caso de uso por outro caso de uso. Ou seja, ao executar o primeiro caso uso, o segundo caso de uso pode ou não ser executado.	A ponta da seta vai para quem chama.
INCLUSÃO		É continuação obrigatória de um caso de uso em outro. Ou seja, ao se executar o primeiro caso de uso, o segundo caso de uso será executado.	A ponta da seta vai na direção da continuação

Tabela 2 – Outras notações de dependência
Fonte: CASTRO et al, 2014.

Vamos entender melhor com um exemplo de cada.

Exemplo de Generalização/Especialização – Vamos imaginar um sistema de um barzinho. Este sistema possui um caso de uso geral que representa a ação de realizar pagamento. O ator cliente precisa realizar pagamento, mas este pagamento pode ser realizado de duas maneiras: com cartão de crédito e com dinheiro. A representação da Figura 5 apresenta os casos de usos especializados herdando as características do geral. A notação mostra o caso de uso geral "Realizar Pagamento" recebendo a ponta da seta.

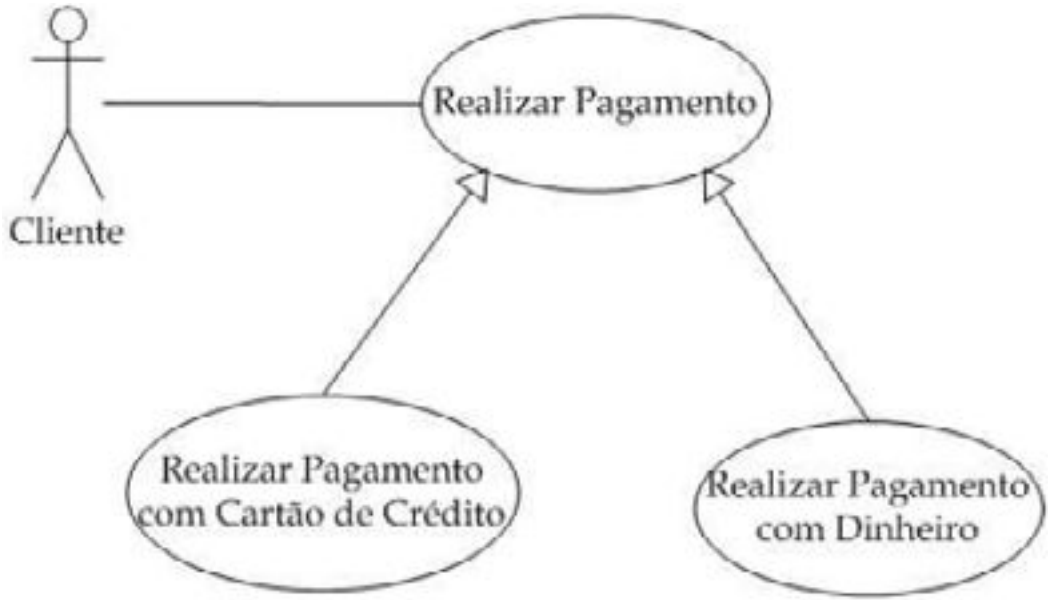


Figura 5 – Exemplo de Especialização
Fonte: BEZERRA, 2006.

Exemplo de Extensão - Vamos imaginar um sistema de edição de uma gráfica. Esse sistema possui como ator o escritor que se relaciona com o caso de uso editar documento. Em alguns momentos pode ser necessário o caso de uso substituir texto ou corrigir ortografia. A extensão nesse caso é representada pela chamada de um caso por outro. Lembrando que a ponta da seta vai para quem chama (Figura 6).

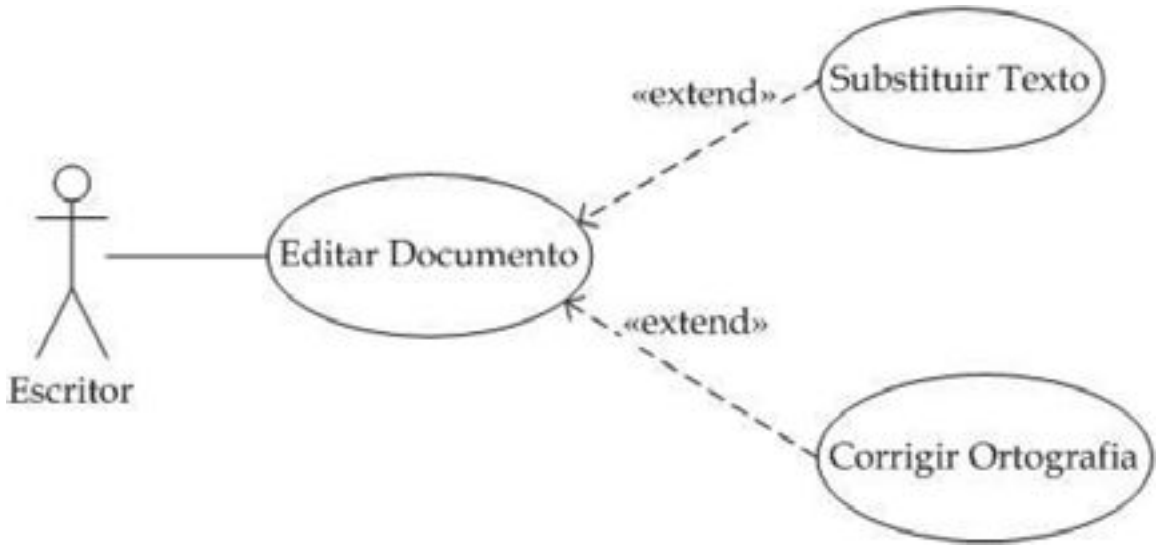


Figura 6 – Exemplo de Extensão
Fonte: BEZERRA, 2006.

Exemplo de Include – Vamos imaginar um sistema bancário, no qual o cliente se relaciona aos casos de uso obter extrato, realizar saque e realizar transferência. Todos esses casos de uso são continuação do caso de uso fornecer identificação. Assim, foi criado o include e a seta vai na direção da continuação (Figura 7).

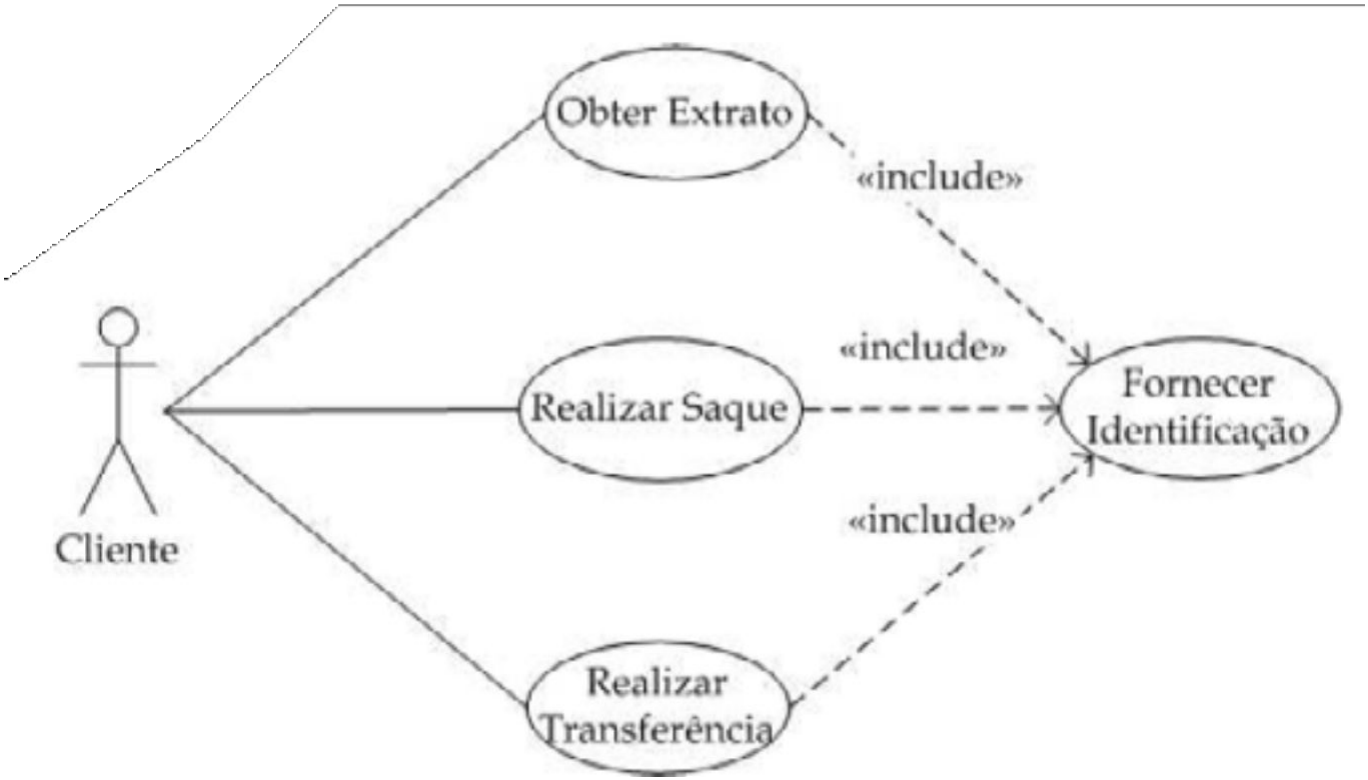


Figura 7: – Exemplo de include
Fonte: BEZERRA, 2006

Fique Ligado!

Uma descrição de cada ator deve ser incluída ao modelo de casos de uso, por exemplo, “Aluno – representa que fazem curso dentro da universidade”.

Com relação à descrição dos casos de Uso, a UML não define um padrão de descrição textual. A equipe e/ou a organização deve definir quais informações são importantes. Mas algumas informações são mais utilizadas: sumário, atores, fluxo principal, fluxos alternativos e referências cruzadas (BEZERRA, 2006).

Saiba mais!

Você sabia que a comunidade Astah disponibiliza ferramenta free para modelagem UML? Acesse o link e baixe a ferramenta. Ela possibilita construir casos de uso utilizando as notações da UML.
http://astah.net/editions/uml-new

E com relação aos Modelos ágeis? Como são tratados os requisitos?

Vamos entender um pouco o SCRUM como um todo?

Saiba mais!

Vamos assistir ao vídeo:
https://www.youtube.com/watch?v=vg1S1WYZa6o

O SCRUM é uma das metodologias ágeis mais utilizadas. Conforme vimos no vídeo, temos também uma visão (pode ser bem sucinta) com uma frase do objetivo do sistema. No levantamento são identificadas as incertezas técnicas, financeiras etc. São mitigadas as incertezas, definidas as estratégias, priorizadas as funcionalidades. E a partir daí são definidas as histórias de usuário. Para escrever uma história do usuário, é importante pensar em três questões, segundo Sutherland (2014):

Quem? Esta pergunta vai indicar o autor da ação;

O quê? Indica a funcionalidade desejada;

Por quê? Indica o valor agregado pela funcionalidade;

Veremos o exemplo dado por Sutherland (2014); vamos pensar em um sistema para um e-commerce de livros. No sistema, irão existir dois tipos de usuários: gerente e cliente. Algumas das histórias:

“Como cliente, eu quero colocar um livro em um carrinho de compras para que eu possa comprá-lo”.

“Como gerente de administração de produtos, eu quero poder rastrear as compras de um cliente para que eu possa oferecer livros específicos para ele com base nas compras anteriores”.

Podem ainda ser acrescentados critérios de aceite (condições, restrições, regras de negócio) como, por exemplo, restrições de acesso, mensagens de sucesso, erros. Assim, para o exemplo da história do gerente de administração, poderíamos ter como critérios de aceite:

O rastreamento deve considerar as últimas 10 compras.

Agora que já entendemos como especificar requisitos utilizando a abordagem Orientação a Objeto ou Modelos Ágeis, vamos entender como verificar ou validar esses requisitos. Para assegurar que o software adequado e correto seja desenvolvido, é muito importante analisar o que foi especificado. Está correto? Está consistente? No próximo tópico, estudaremos esse processo. Vamos lá?

2.4 - Processo de verificação e validação de requisitos

O objetivo da verificação e validação é identificar se o software possui defeitos e assegurar que o software funcione de acordo com o que foi especificado. Mas qual a diferença entre verificação e validação?

Para Rios e Moreira (2003):

Verificação: engloba as inspeções ou revisões sobre os produtos gerados pelas diversas etapas do processo de desenvolvimento.

Validação: avalia se o sistema atende aos requisitos do projeto (usuário). Os testes unitários, de integração, de sistema e de aceitação podem ser classificados como testes de validação.

E no contexto de requisitos, como fazemos a verificação e a validação dos requisitos representados por casos de usos ou histórias de usuários? Nesse momento, não temos ainda a possibilidade de realizar testes. Não é mesmo?

Na verificação de requisitos, tentamos determinar se estamos construindo o produto da maneira correta. Já na validação de requisitos, verificamos se a especificação de uma fase ou do sistema completo é apropriada e consistente com os requisitos dos stakeholders, ou seja, se é o produto correto.

E como fazemos isso? Por meio de inspeções. Mas, qual o objetivo de uma inspeção?

Fique Ligado!

O principal objetivo da técnica de inspeção é a descoberta antecipada de falhas para garantir que estas não se propaguem para o passo seguinte do processo de construção ou manutenção software.

A inspeção visa encontrar erros, analisando e entendendo o que o documento descreve e checando as propriedades de qualidade. Pode ser realizada de várias formas, a seguir, citamos a inspeção ad hoc e a técnica de inspeção baseada em checklist (CASTRO et al, 2014):

- A inspeção ad hoc não propõe nenhuma técnica formal de leitura; cada leitor (que não é o autor do documento) lê o documento do seu modo, identificando falhas. Por este motivo, essa técnica torna-se dependente da experiência do leitor.

- A técnica de leitura baseada em checklist propõe o uso de uma lista de verificação (checklist) que auxilia a encontrar os defeitos. O uso desse checklist permite ao inspetor seguir uma lista de defeitos com características a serem revisadas.

A seguir, listamos alguns dos defeitos mais citados com relação à verificação:

Defeito de ambiguidade: um requisito tem várias interpretações devido aos diferentes termos utilizados para uma mesma característica.

Clareza - os requisitos foram escritos em linguagem não-técnica permitindo um bom grau de entendimento? Foi elaborado um glossário de termos para auxiliar no entendimento?

Completeza - Todos os termos necessários foram definidos? Algum requisito deveria ter sido especificado em mais detalhes? Algum requisito deveria ter sido especificado em menos detalhes?

Consistência - Existe algum requisito que descreve duas ou mais ações que conflitam logicamente? Existe algum requisito impossível de ser implementado? Cada um dos requisitos é relevante para a solução do problema?

E com relação à validação? Vamos ver alguns defeitos?

Defeito de omissão: informações necessárias ao sistema são omitidas, faltam funções requeridas pelo cliente.

Defeito de fato incorreto: há informações nos artefatos do sistema que são contraditórias com o domínio da aplicação.

Fique Ligado!

No modelo ágil, a verificação e a validação, normalmente, são realizadas nas atividades de avaliação da documentação e da criação da documentação de testes.

Na atividade de avaliação da documentação, os profissionais de teste definem os cenários, fluxos e escrevem os casos de testes. Essa equipe deve ler, entender e analisar as histórias de usuário, validar esses documentos, participar do refinamento e eliminar possíveis brechas. Nessa função, deve-se ter sempre em mente a visão do usuário e quais são os possíveis casos reais que ele poderá executar.

Na criação da documentação de testes com os artefatos revisados, melhorados e atualizados, deve-se dar início à escrita de casos de testes. A documentação de testes deve ser simples, direta e clara na medida do possível e deve descrever o passo a passo para validação dos critérios de aceite. Fazem parte dessa atividade a criação, a manutenção e a atualização dos documentos, além da criação de scripts.

Saiba mais!

Vamos assistir ao vídeo abaixo? Ele descreve muito bem o processo de requisitos.

https://www.youtube.com/watch?v=C8heVblhmKo

Encerramento

Caro estudante, nesta unidade você estudou a importância da modelagem e suas ferramentas. Entendeu os principais conceitos relacionados à UML, que é uma linguagem utilizada para modelar sistemas. Também aprendeu as técnicas de levantamento de requisitos e sua importância. Levantar requisitos não é uma tarefa fácil e as técnicas podem ajudar muito nosso trabalho. Na construção de um produto de software, estamos transformando uma abstração, que são as ideias do cliente, em um software. Esse processo envolve muita comunicação, ruídos, expectativas, necessidades etc. Qualquer ferramenta que ajude a elicitarmos com mais eficiência só pode agregar valor ao nosso trabalho.

Estudamos também técnicas de especificação de requisitos para a abordagem orientada a objeto e para os Métodos ágeis. Conhecemos o documento de visão, as principais notações do diagrama de caso de uso e o que deve conter a descrição do caso de uso. Com relação aos Modelos ágeis, entendemos como elaborar Histórias de usuário. Ao final, vimos o processo de Verificação e Validação de requisitos, sua importância e como pode ser implementado em uma organização para melhorar a qualidade das especificações. Espero que tenha gostado e assimilado estes conhecimentos de forma a aplicá-los na sua vida profissional.

Nas próximas unidades, detalharemos melhor esses conceitos e aprenderemos outros.

Vamos lá?

Questões de autoaprendizagem

1. Você faz parte de uma equipe de desenvolvimento de um sistema que possui vários stakeholders. Cada um desses stakeholders é responsável por uma parte do sistema e é necessário reuni-los para discutir as fronteiras e obter consenso sobre os requisitos a serem atendidos e sua ordem de prioridade. Qual a técnica de levantamento de requisitos seria mais adequada?

a.( ) entrevista
b.( ) análise de documentos
c.( ) workshop
d.( ) prototipação
e.( ) Nenhuma das alternativas anteriores

Solução – C. O workshop possibilita o envolvimento de todos os interessados, agiliza o entendimento do projeto do Sistema, facilita a obtenção de consenso, pois agrega as partes interessadas.

2. Um diagrama de caso de uso é um diagrama comportamental da UML que evidencia a visão externa do sistema a ser desenvolvido. Ele mostra quais são as funcionalidades desse sistema e quem interage com elas. São notações do diagrama de caso de uso: ator, caso de uso, associação, Generalização/especialização, Extensão, Inclusão. Acerca dessas asserções, assinale a opção correta.

a.( ) As duas asserções são proposições verdadeiras; a segunda é uma complementação da primeira.

b.( ) As duas asserções são proposições verdadeiras, mas a segunda não é uma complementação da primeira.

c.( ) A primeira asserção é uma proposição verdadeira, e a segunda, uma proposição falsa.

d.( ) A primeira asserção é uma proposição falsa, e a segunda, uma proposição verdadeira.

e. ( ) Tanto a primeira quanto a segunda asserções são proposições falsas.

Solução – A. As duas proposições estão corretas, e a segunda é complementação da primeira. Um diagrama de caso de uso é um diagrama comportamental da UML que evidencia a visão externa do sistema a ser desenvolvido. Ele mostra quais são as funcionalidades desse sistema e quem interage com elas. São notações do diagrama de caso de uso: ator, caso de uso, associação, Generalização/especialização, Extensão, Inclusão.

Unidade 3

Caracterização e aplicação de metodologias e ferramentas de modelagem de sistemas orientados a objetos

3.1 Conceitos de modelagem de sistemas, modelos e suas perspectivas

Neste tópico, estudaremos os conceitos de modelagem de sistemas, modelos e suas perspectivas com relação a modelagem de sistemas orientados a objetos.

Já vimos anteriormente o termo Modelagem. Lembra? No tópico II, falamos de Modelagem de requisitos. Agora vamos entender um escopo maior que é a modelagem de sistemas.

O que é modelagem de sistemas?

Modelagem de sistemas é a atividade de construção de modelos que expliquem /ilustrem a forma de funcionamento de um software.

Modelar é desenhar ou projetar o software antes da codificação. Segundo Bezerra (2006), existem muitas razões para se modelar sistemas, tais como: facilitar a comunicação entre as pessoas envolvidas, prever o comportamento futuro do sistema, gerenciar a complexidade, reduzir os custos do desenvolvimento.

E modelagem de sistemas orientados a objetos?

O termo orientação a objetos relaciona-se a organizar o mundo real como um conjunto de objetos que incorporam:

Estrutura de dados (propriedades ou atributos) e;

Um conjunto de operações ou métodos (que manipulam esses dados).

Para entender melhor, veja o exemplo abaixo e a figura 1. Todos os objetos têm um nome ou identificador, suas propriedades e operações.

Objeto: Pessoa

Propriedades ou atributos: nome, data de nascimento, peso, altura

Operações: andar, falar, dormir

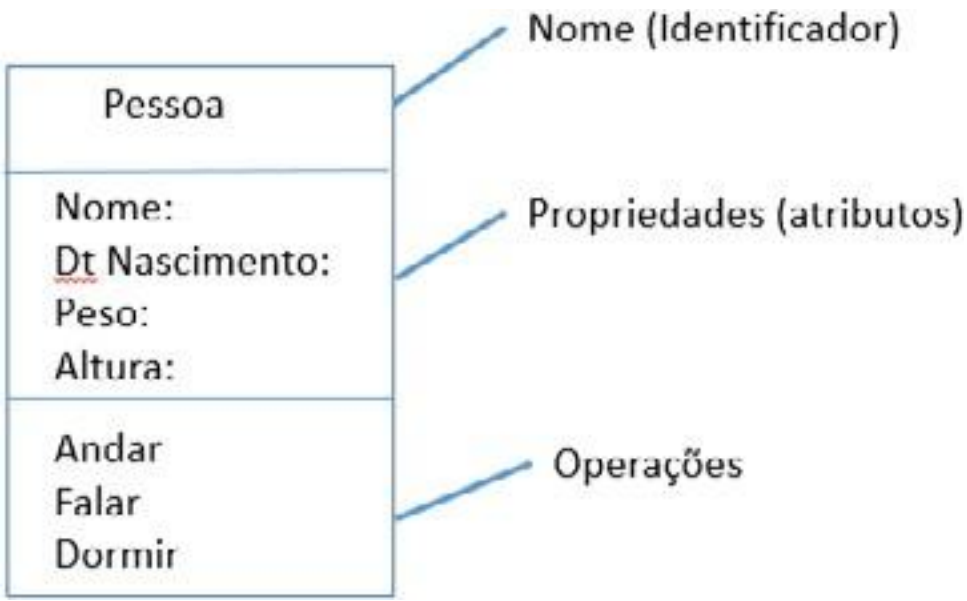


Figura 1 – Representação de Pessoa na orientação a objetos
Fonte: elaborado pela autora.

Questões de autoaprendizagem

E como seria um objeto veículo? Quais as propriedades ou atributos? Quais as operações ou métodos?

Dica:

Os nomes dos objetos geralmente são substantivos no singular, tais como: cliente, conta-corrente, pessoa, etc.

As propriedades ou atributos são substantivos, exemplo: cor, tamanho, peso, idade, número, etc.

Já as operações ou métodos, usualmente, são verbos, como: acelerar, validar, verificar, calcular, etc.

Então, a modelagem de sistemas com a abordagem orientada a objetos consiste na construção de objetos que colaboram para construir sistemas mais complexos.
São alguns dos princípios da Orientação a Objeto, segundo Bezerra (2006):

Qualquer coisa é um objeto.

Objetos realizam tarefas por meio da requisição de serviços a outros objetos;
Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares.

Mas o que é uma classe?

Uma classe descreve um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos (BEZERRA, 2006).

Vamos esclarecer: as classes são as partes mais importantes do modelo orientado a objetos. São utilizadas para capturar e apresentar o vocabulário do sistema que está em desenvolvimento.

Retomando o exercício proposto antes sobre o objeto veículo. Ao fazermos esse exercício, provavelmente, construiríamos algo similar a parte da Figura 2, a seguir:

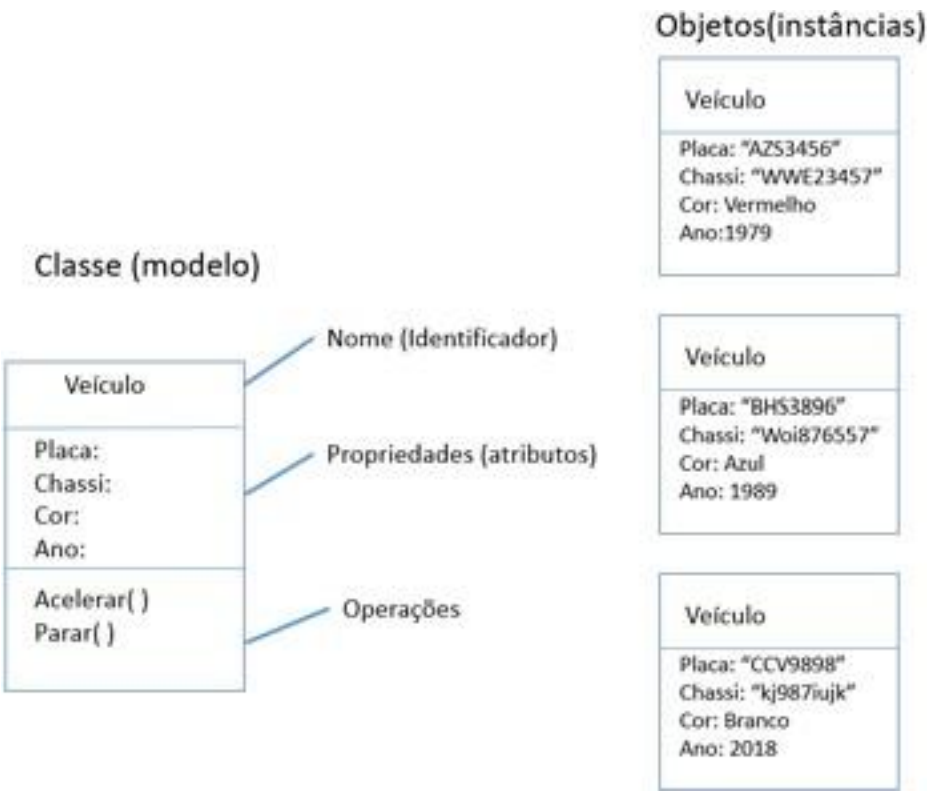


Figura 2 – Exemplo de Classe e objetos.
Fonte: elaborado pela autora.

O que definimos descreve um conjunto de objetos que compartilham os mesmos atributos, operações/métodos, etc. Todos os veículos têm placa, chassi, cor, ano. Todos os veículos têm operações de acelerar, parar. Então, definimos uma classe chamada veículo. Essa classe pode ter 1 ou mais objetos também chamados de instâncias. Elas utilizam os mesmos atributos e métodos, ou seja, todas as instâncias são veículos que possuem os mesmos atributos: placa, chassi, cor, ano. Está claro?

Vamos fazer uma analogia com a Engenharia Civil. Imagine um prédio de apartamentos. Todos os apartamentos são, rigorosamente, iguais e baseados em uma planta estabelecida (número de dormitórios, portas, fiação etc.). A planta dos apartamentos é a classe. Os apartamentos são objetos baseados naquela classe (MEDEIROS, 2004).

Ao criar uma classe, você está definindo que todos os objetos dessa classe têm os mesmos atributos e as mesmas operações/métodos.

Como definir os atributos e métodos?

Somente atributos que são de interesse do sistema devem ser descritos na classe. Isso quer dizer que se para o seu sistema a cor do veículo não é importante, a classe veículo não terá esse atributo. Claro?

E com relação aos métodos? Como criar os métodos?

Método é a implementação de uma operação. Os métodos são utilizados para realizar qualquer operação, por exemplo: cálculo salário, acender um aparelho, mostrar um saldo, manter um livro (incluir, alterar, excluir).

A Figura 3 apresenta um exemplo da classe livro.

Figura 3 – Exemplo de atributos e métodos para a classe livro
Fonte: elaborado pela autora.

Vamos entender melhor o conceito de Orientação a objetos. Assista ao vídeo, a seguir, tomando nota de todos os pontos relevantes ao que estamos discutindo aqui.

Saiba mais!

https://www.youtube.com/watch?v=7nmJ2oGRrbc

Saiba mais!

Você se lembra de que no módulo II, para modelar casos de uso, sugerimos a utilização do software Astah? Para as próximas figuras, estaremos utilizando esse software. Ok?

Você pode baixá-lo no link: http://astah.net/editions/uml-new

Para Bezerra (2006), a abordagem de orientação a objeto engloba, além de classes e objetos, os conceitos de herança, polimorfismo, encapsulamento e composição. Estudaremos esses conceitos a seguir.

Herança/Generalização

Mecanismo baseado em objetos que permite que as classes compartilhem atributos e operações baseados em um relacionamento, geralmente, generalização (RUMBAUGH et al, 1991).

Em uma Herança, uma classe “derivada” herda a estrutura de atributos e métodos de sua classe “base”, veja a figura 4. A classe base é funcionário e a classe derivada é motorista. A Figura 4, a seguir, mostra a classe funcionário com 3 atributos. Ela proporciona os atributos e métodos para a classe derivada motorista.

Mas a classe derivada pode adicionar novos métodos, aumentar a estrutura de dados ou redefinir a implementação de métodos já existentes. No exemplo, a classe motorista adicionou mais um atributo. Qual?

Quais são as vantagens da herança? Você consegue identificar? Se tivéssemos mais uma classe, secretária, com atributos “Fluência língua inglesa” e “Fluência língua espanhola”, ela poderia ser outra classe derivada? A resposta é sim. Ela herdaria também a estrutura e os métodos da classe funcionário. Veja a figura 4. Ficou claro?

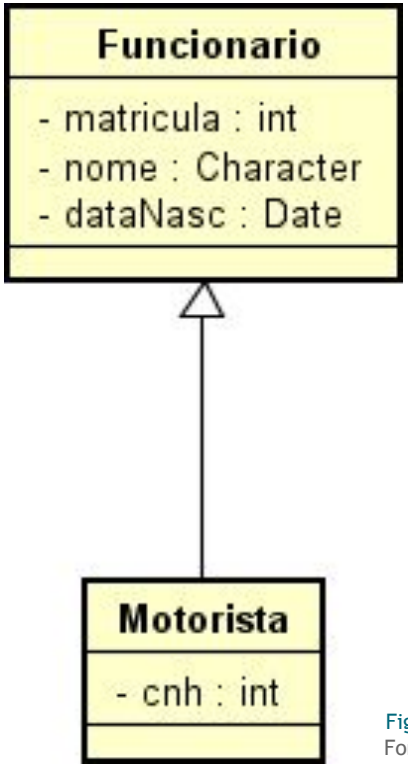


Figura 4 – Exemplo de herança
Fonte: elaborado pela autora.

Saiba mais!

Veja outros vídeos que abordam sobre herança.

https://www.youtube.com/watch?v=_PZldwo0vVo
https://www.youtube.com/watch?v=He887D2WGVw

Polimorfismo

Polimorfismo é uma operação que pode assumir múltiplas formas; a propriedade segundo o qual uma operação ou método pode comportar-se diferentemente em classes diferentes (RUMBAUGH et al, 1991).

O polimorfismo possibilita o reuso. Vamos ver o exemplo anterior, mas agora na figura 6. O método CalSalario é herdado da superclasse (Funcionário), entretanto, para motorista e para a secretária, ele tem uma implementação diferente. Por exemplo:

Para apurar o salário do motorista = (salário + adicional direção (%20 sobre horas dirigidas))

Para apurar o salário da secretária = (salário + adicional fluência (+%10 sobre salário para cada língua com fluência))

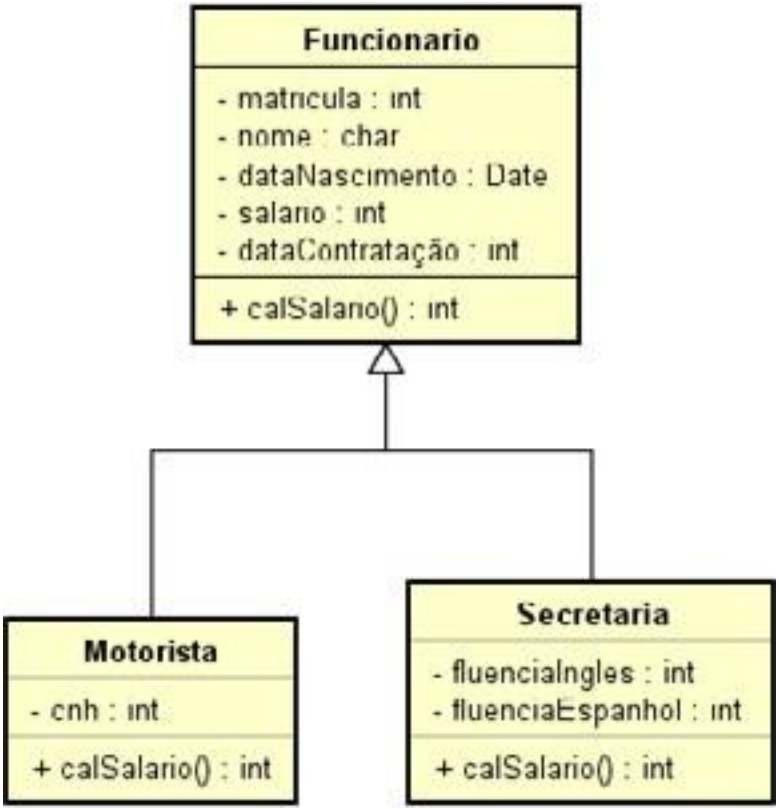


Figura 5 – Exemplo de polimorfismo.
Fonte: elaborado pela autora.

Saiba mais!

Vamos para um vídeo sobre polimorfismo?

https://www.youtube.com/watch?v=9-3-RMEMcq4

Encapsulamento

"O encapsulamento refere-se à visibilidade de atributos, métodos e classes e ao empacotamento das classes" (MEDEIROS, 2004, pág. 9). O Encapsulamento serve para controlar o acesso aos atributos e métodos de uma classe. Protege os dados manipulados dentro da classe e determina onde esta classe pode ser manipulada.

Por exemplo: você entra no carro, dirige e pisa no freio. O que acontece com o mecanismo do freio e com os pneus, você não tem a menor ideia. Só espera que o carro pare, não é? Para o motorista não é necessário entender o mecanismo interno do veículo, contudo, quem projetou e construiu o veículo se preocupou com isso e implementou este mecanismo de forma correta. Isso é o que chamamos de encapsulamento e ocultamento de informações.

Na Orientação a objeto, para possibilitar o encapsulamento, define-se o nível de acesso. O encapsulamento é dividido em dois níveis:

Nível de classe: quando se determina o acesso relacionado a uma classe inteira: public ou Package-Private (padrão);

Nível de membro: quando se determina o acesso de atributos ou métodos de uma classe: public, private, protected ou Package-Private (padrão).

Saiba mais!

Imagine uma quadra de esportes... Tem aquelas que ficam localizadas em parques e podem ser usadas por todos (*public*) – todos podem utilizar seus atributos e usar de acordo com seu desejo. Tem aquelas quadras que ficam em clubes e só podem ser usadas por seus membros (*protected*) – somente as classes do mesmo pacote têm acesso. Por fim, tem aquelas que ficam em casas e somente o dono da casa pode utilizar (*private*) – somente a classe em que está o atributo tem acesso.

No encapsulamento, um objeto que precise da colaboração de outro objeto para realizar uma operação, simplesmente, envia uma mensagem a este último (BEZERRA, 2006). Entretanto, o remetente da mensagem precisa saber quais operações o receptor pode realizar ou fornecer. "Na terminologia da orientação a objetos, diz-se que o objeto possui uma interface. Em termos simples, a interface do objeto corresponde ao que ele conhece e ao que sabe fazer, sem descrever como conhece ou faz" (Bezerra, 2006, p. 9). Ou seja, a interface é o contrato entre a classe e o mundo exterior.

Vamos fazer uma analogia?

Quando o usuário faz uma escolha na tela do computador, ele está utilizando uma interface gráfica. Correto? O que possibilita ao mundo externo colaborar com uma classe é sua interface.

Saiba mais!

Vamos complementar esse assunto com um vídeo?

https://www.youtube.com/watch?v=1wYRGFXpVlg

Composição

Quando analisamos as coisas do mundo real, podemos visualizar o conceito de composição. Por exemplo: uma televisão é composta por um painel de controle e uma tela. Esse material que você está lendo é composto por páginas. Na orientação a objeto,os objetos também podem ser compostos por outros objetos. Ou seja, podemos criar objetos a partir da reunião de outros objetos (BEZERRA, 2006).

Uma nota fiscal, por exemplo, normalmente é composta de um cabeçalho com os dados de data, CNPJ ou CPF de quem está comprando, número da nota fiscal e, também, tem uma relação dos produtos, seus preços unitários, quantidade por produto e valor total. Vamos pensar em uma classe nota fiscal com os atributos número, data e CNPJ. Também podemos ter uma classe Item NotaFiscal com os atributos número, quantidade do produto etc. É claro que deve existir, pelo menos, um item em uma nota fiscal para que ela exista. Concorda? Não existe nota fiscal sem pelo menos um produto vendido.

Esse é um exemplo de composição:NotaFiscal é composta de ItemNotaFiscal.

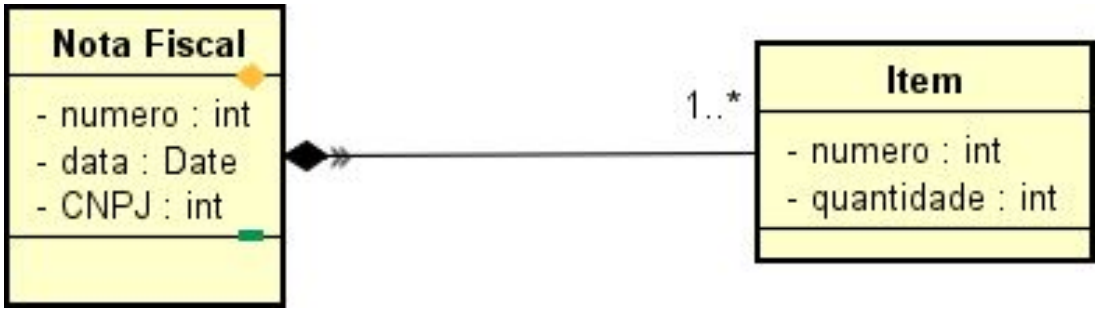


Figura 6: Exemplo de composição.
Fonte: elaborado pela autora.

Agora estudaremos alguns modelos de processo de desenvolvimento de software que utilizam os conceitos da orientação a objetos. Vamos lá?

3.2 Etapas do processo de desenvolvimento de software e metodologias

Neste tópico, detalharemos mais os processos de desenvolvimento de software. Você deve se lembrar, pois citamos o processo de desenvolvimento de software na Unidade I.

Fique Ligado!

A literatura apresenta uma série de modelos de processo de desenvolvimento de sistema, tais como: Cascata, Prototipação, Processo Unificado (Orientação a objetos), Métodos Ágeis (eXtremeProgramming - XP, SCRUM) e outros.

Fique Ligado!

Uma observação importante: existem alguns modelos prontos os quais já foram escolhidos os melhores métodos, procedimentos e ferramentas. O Processo unificado (Orientação a objetos) e os Métodos Ágeis (SCRUM, XP e outros) são exemplos desse tipo de modelos (CASTRO et al, 2014).

Não existe o modelo ideal, cada um tem vantagens e desvantagens. O conhecimento dos modelos é importante, pois no mercado algumas organizações trabalham com Métodos Ágeis, outras com Processo Unificado, outras com um modelo próprio, adaptado de algum outro. Assim, conhecer os principais modelos nos dará uma visão abrangente das variadas formas que podemos utilizar para construir um produto de software. Nesse tópico, estudaremos o modelo unificado e os modelos ágeis. Por quê?

Porque o objetivo desta unidade é estudar modelos com foco em orientação a objetos. Porque modelos como o Cascata e a Prototipação, além de não terem foco na orientação a objeto, são modelos mais antigos, apesar de algumas empresas ainda utilizarem esses modelos para construir software.

Processo Unificado

O Processo Unificado (PU) surgiu como um processo para o desenvolvimento de software visando à construção de sistemas orientados a objetos. Os precursores desse modelo foram os trabalhos de Rumbaughet al (1991), Jacobson et al (1992) e Booch (1994).

Saiba mais!

O modelo Processo Unificado tem como principais características: ser baseado em componentes que realizam interfaces, utilizar a UML, ser dirigido a casos de uso, ter o desenvolvimento iterativo e incremental e ser focado em arquitetura.

Já sabemos o que é UML e casos de uso (vimos os conceitos na Unidade II). Mas o que é ter o desenvolvimento iterativo e incremental? O que é "ser focado em arquitetura?"

Desenvolvimento iterativo é uma estratégia de planejamento em que se definem as entregas das partes do sistema.

Desenvolvimento Incremental é uma estratégia de planejamento em estágios em que várias partes do sistema são desenvolvidas em paralelo e integradas quando completas. Nesse modelo, o sistema é dividido em subsistemas por funcionalidades. Essas versões são definidas começando com um pequeno subsistema funcional, adicionando-se mais funcionalidades a cada versão (CASTRO et al, 2014).

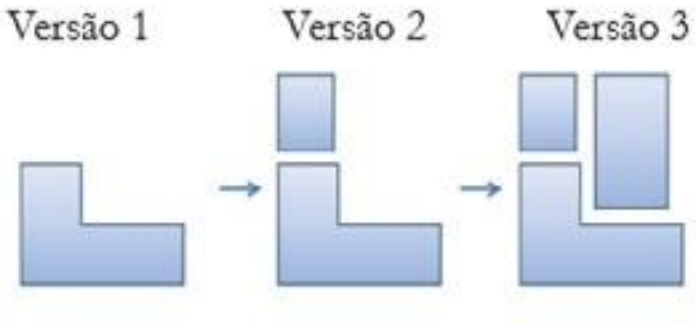


Figura 7 – Exemplo de desenvolvimento incremental.
Fonte: PFLEEGER, 2004

Ser focado em arquitetura engloba ter a visão do software como um todo. Vamos para um exemplo. Lembra do sistema de biblioteca da Unidade I?
Quando escrevemos sobre as questões da biblioteca, as tecnologias e ambientes computacionais que serão utilizados pelo novo software, estamos falando de sua arquitetura. Mais adiante retomaremos esse assunto.

Fique Ligado!

Vamos procurar na internet, no Google. Procure pelo termo “processo unificado de desenvolvimento de software”. Você vai encontrar várias referências ao RUP, não é? Mas o que é RUP?

O RUP – RationalUnifiedProcess é um refinamento do Processo Unificado. Quando falamos do RUP, é necessário identificar três dimensões (CASTRO et al, 2014):

RUP: como um processo de desenvolvimento de sistemas;

RUP: como o produto comercializado pela IBM/Rational, que contém uma Base de Conhecimento do RUP (disponível no site da IBM), chamado de IBM RationalMethod Composer;

RUP: conjunto de ferramentas comercializadas pela IBM/Rational, empregadas para apoiar o uso produtivo do RUP.

O RUP trabalha com os conceitos de disciplinas e fases, conforme a Figura 8. Na fase de Iniciação, o foco está no entendimento do negócio, no escopo (análise do problema e criação de uma “visão” da solução). Também são identificadas estimativas preliminares de prazo, custo e riscos do projeto. Na fase de Elaboração, refinam-se os requisitos, define-se a arquitetura e constrói-se um protótipo (se necessário).

A Construção tem foco na implementação do sistema, onde a maior parte do código é desenvolvido. Nesta fase, o projeto é totalmente desenvolvido e a transição corresponde às atividades de teste, treinamento e implantação do sistema em ambiente de produção.

Em cada fase, o projeto percorre várias iterações. Cada iteração é uma sequência de atividades planejadas e avaliadas no final. Cada iteração gera um executável e se baseia na iteração anterior. Ou seja, o projeto no RUP é desenvolvido de modo "iterativo e incremental".

A figura 8 mostra o modelo do RUP; cada cor representa o tempo gasto ou o esforço para aquela disciplina naquela fase.

Vamos ver se você entendeu? Identifique quais disciplinas têm maior esforço na fase de iniciação. Conseguiu? São a Modelagem de Negócios e Requisitos. Lembre-se de que, nesta fase, estamos identificando o escopo, os problemas, os objetivos, os requisitos do sistema. Tem sentido estarmos gastando mais esforço nessas disciplinas.

Quais disciplinas não têm nenhum esforço gasto nessa etapa? É a implantação não é? É óbvio que na iniciação não tem esforço para implantação.

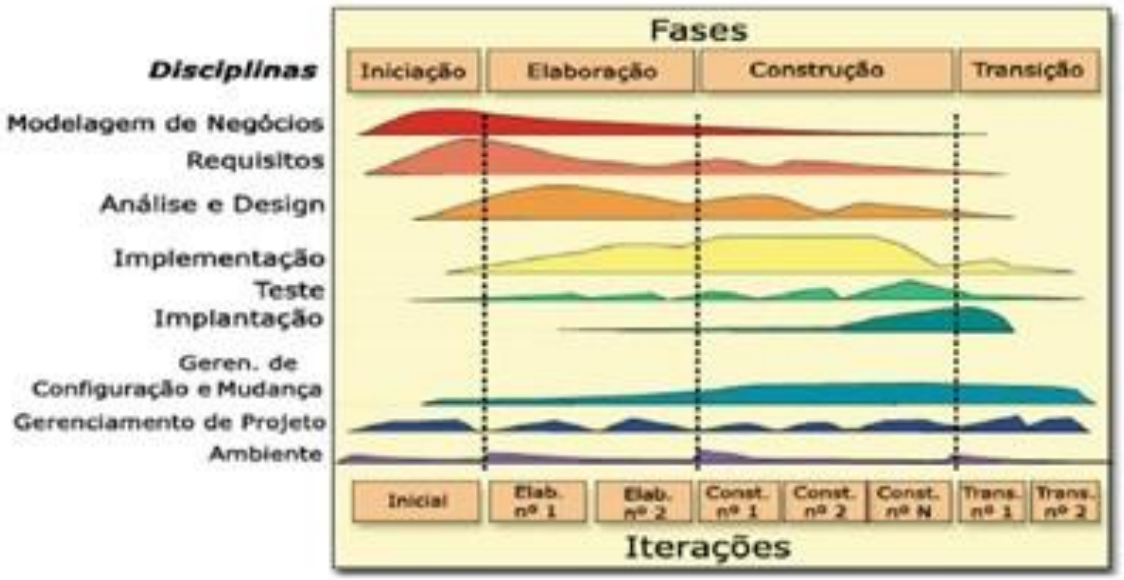


Figura 8 – Modelo RUP.
Fonte: Rational, 2002.

Além das fases, o RUP trabalha com o conceito de disciplinas. Uma disciplina consiste em conjunto de atividades sequenciadas para produzir um artefato. Em cada iteração, a equipe gasta quanto tempo for apropriado para cada disciplina, conforme figura 8.

Métodos Ágeis

A expressão "Metodologias Ágeis" ou Métodos Ágeis tornou-se conhecida em 2001, quando especialistas em processos de desenvolvimento (XP, Scrum e outros) estabeleceram princípios e características comuns destes métodos. Assim, foi criada a "Aliança Ágil" e definiu-se os 4 valores do manifesto ágil:
Indivíduos e interações mais que processos e ferramentas.

Trabalhar no software mais que documentação abrangente.

Colaboração do cliente mais que negociação contratual.

Responder às mudanças mais que seguir um plano

Assim, o grupo das Metodologias Ágeis engloba vários métodos, modelos ou frameworks, tais como: ExtremingProgramming – XP, SCRUM, LeanDevelopment e outros. A seguir, apresentaremos os principais conceitos do SCRUM.

Scrum

O SCRUM é um framework iterativo que possibilita aos desenvolvedores tratar e resolver problemas complexos e adaptativos e entregar produtos com o mais alto valor possível. É um framework dentro do qual você pode empregar vários processos ou técnicas.

Vamos ver um vídeo sobre o SCRUM?

https://www.youtube.com/watch?v=7lhnYbmovb4

O Scrum é baseado em papéis (SCHWABER ET SUTHERLAND, 2017):

- ProductOwner (PO) que é dono do produto, responsável por maximizar o valor do produto;

-Scrum Master (SM) que é o responsável por promover e suportar o Scrum como definido no Guia Scrum;

-Time de Desenvolvimento consiste de profissionais que realizam o trabalho de entregar um incremento potencialmente liberável do produto "Pronto" ao final de cada Sprint.

O Scrum possui os seguintes eventos (SCHWABER ET SUTHERLAND, 2017):

-Sprint - é um time-boxed (evento) de um mês ou menos, durante o qual um incremento do produto é criado e liberado.

-Planejamento da Sprint - um time-boxed (evento) com no máximo oito horas para uma Sprint de um mês de duração. O planejamento define a meta da Sprint e responde às seguintes questões:

O que pode ser entregue como resultado do incremento da próxima Sprint?

Como o trabalho necessário para entregar o incremento será realizado?

-Reunião Diária - é um time-boxed (evento) de 15 minutos para o Time de Desenvolvimento. A reunião diária é realizada todos os dias da Sprint. Nela, o Time de Desenvolvimento planeja o trabalho para as próximas 24 horas.

-Revisão da Sprint - é realizada no final da Sprint para inspecionar o incremento do produto e adaptar o Backlog do produto, se necessário.

-Retrospectiva da Sprint - é uma oportunidade para o Time Scrum inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima Sprint.

São artefatos do Scrum (SCHWABER ET SUTHERLAND, 2017):

-Backlog do Produto - é uma lista ordenada de tudo que é conhecido ser necessário no produto. É a única origem dos requisitos para qualquer mudança a ser feita no produto.

-Backlog da Sprint - é um subconjunto de itens do Backlog do Produto selecionados para aquela Sprint, juntamente com o plano para entregar o incremento do produto e atingir o objetivo da Sprint.

-Incremento - O incremento é a soma de todos os itens do Backlog do Produto completados durante a Sprint e o valor dos incrementos de todas as Sprints anteriores.

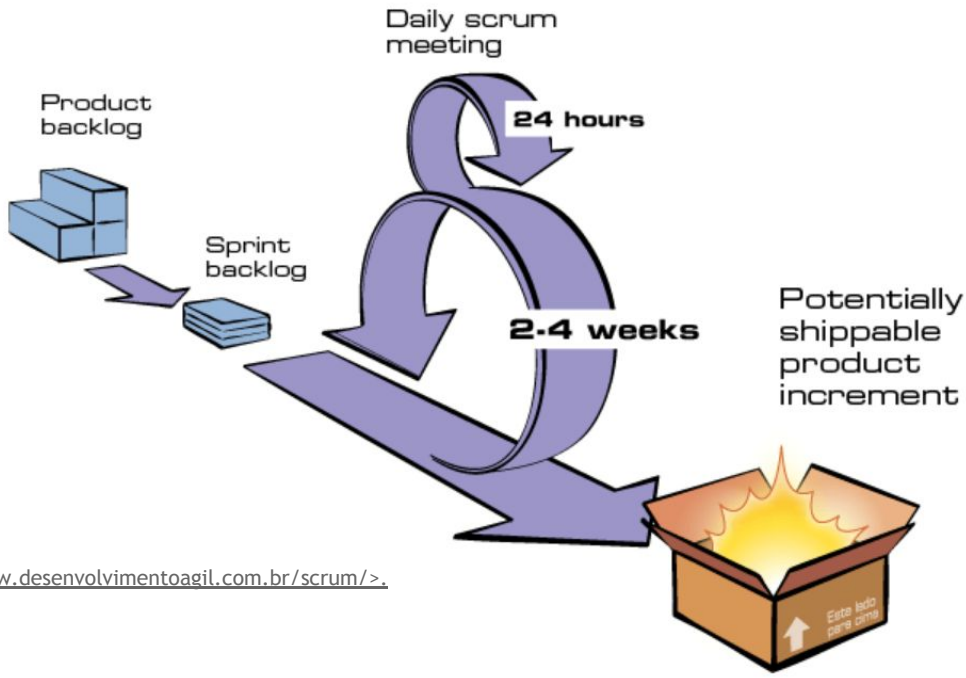


Figura 9 - Processo SCRUM
Imagem disponível em: <https://www.desenvolvimentoagil.com.br/scrum/>.

A figura 9 mostra o processo do SCRUM. Normalmente, o SCRUM trabalha com histórias de usuário que devem ser curtas e responder aquelas três questões citadas na Unidade II, lembra?

Fique Ligado!

Vamos ver o vídeo sobre histórias e modelos ágeis?

https://www.youtube.com/watch?v=3Smbhnmue7Y

Saiba mais!

Você quer saber mais sobre o SCRUM? Acesse o link: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Portuguese-Brazilian.pdf

Encerramento

Caro estudante, nesta unidade você estudou as metodologias e ferramentas de modelagem de sistemas orientados a objetos. Entendeu os principais conceitos relacionados à Orientação a objetos. Entendemos o que é um objeto, o que são classes, propriedades e operações. Entendemos, também, os principais conceitos relacionados a orientação a objetos que são: herança, polimorfismo, encapsulamento e composição. Estes conceitos serão revisados no módulo 4 quando estaremos detalhando melhor a linguagem UML.

Estudamos também as etapas do processo de desenvolvimento de software e metodologias. Conhecemos o processo unificado, discutimos o porquê de ser chamado algumas vezes de RUP. Ele é um modelo interativo e incremental.

Vimos, também, os modelos ágeis e detalhamos o SCRUM. Todos esses processos trabalham com a abordagem orientada a objetos. Espero que tenha gostado e assimilado estes conhecimentos de forma a aplicá-los na sua vida profissional.

Nas próximas aulas, detalharemos melhor esses conceitos e aprenderemos outros. Vamos lá?

Figura 10. Tirinha
Fonte:
Imagem disponível em:
https://vidadeprogramador.com.br/2011/11/16/orientado-a-objetos/

Questões de autoaprendizagem

1. Qualquer coisa é um objeto e cada objeto pertence a determinada classe. Uma classe descreve um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos.

Acerca dessas asserções, assinale a opção correta.

a. As duas asserções são proposições verdadeiras, e a segunda é uma complementação da primeira.

b. As duas asserções são proposições verdadeiras, mas a segunda não é uma complementação da primeira.

c. A primeira asserção é uma proposição verdadeira, e a segunda, uma proposição falsa.

d. A primeira asserção é uma proposição falsa, e a segunda, uma proposição verdadeira.

e. Tanto a primeira quanto a segunda asserções são proposições falsas.

Solução – A. As duas proposições estão corretas, e a segunda é complementação da primeira. Qualquer coisa é um objeto e cada objeto pertence a uma determinada classe. Uma classe descreve um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos.

2. Observe a figura a seguir e identifique a questão correta.

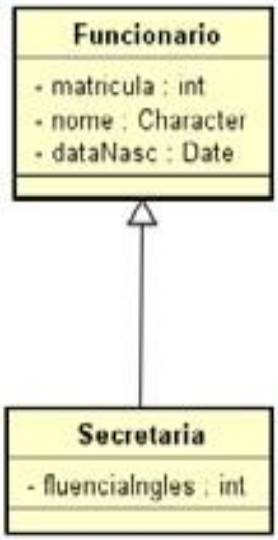


Figura 11
Fonte: elaborado pela autora.

a. É um exemplo de herança na qual a classe base é secretária e a classe derivada é funcionário. A classe secretária proporciona os atributos e métodos para a classe derivada funcionário.

b. É um exemplo de herança na qual classe base é funcionário e a classe derivada é secretária. A classe funcionário proporciona os atributos e métodos para a classe derivada secretária.

c. É um exemplo de polimorfismo, pois possibilita que o método se comporte diferente em diferentes classes.

d. É um exemplo de composição, pois mostra que os objetos podem ser compostos por outros objetos.

Solução - B. É um exemplo de herança na qual a classe base é funcionário e a classe derivada é secretária. A classe funcionário proporciona os atributos e métodos para a classe derivada secretária.

UML e seus diagramas e Introdução a design patterns

4.1 Linguagem de Modelagem Unificada (UML)

Nesta Unidade, estudaremos a UML e seus diagramas. Também veremos uma introdução a design patterns ou padrões de projeto. Já vimos na Unidade II, a descrição da notação UML, lembram?

UML - Linguagem de Modelagem Unificada (*Unified Modeling Language*) é a linguagem utilizada para modelar os sistemas. A UML auxilia os desenvolvedores a definir as características do software, seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos.

A UML já está na versão 2.0. Ela define 13 tipos de diagramas, divididos em 3 categorias: diagramas estruturais, diagramas comportamentais e diagramas de interação. Cada categoria contém os seus diagramas relacionados conforme apresenta a Figura 1 (OMG, SD).

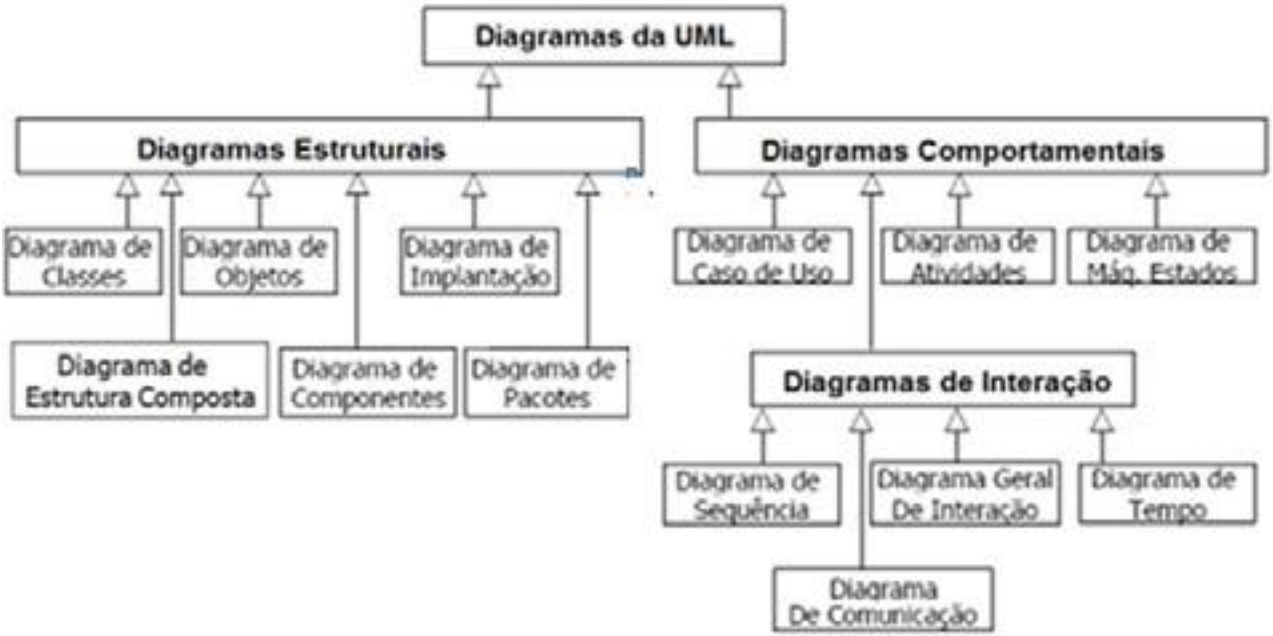


Figura 1 – Diagramas UML (Fonte: OMG, SD)

Fonte: elaborado pela autora.

Alguns autores separam os diagramas da UML, conforme o momento em que são criados dentro do processo de desenvolvimento de software. Por exemplo: Deboni (2003) classifica os diagramas em três grupos: contexto (casos de uso), conceitual (classes conceitual) e detalhado (classes completo, sequência, colaboração, atividades e outros). Estudaremos, detalhadamente, alguns diagramas nessa unidade.

Vamos entender melhor a UML?

Saiba mais!

https://www.youtube.com/watch?v=C3xYBT3o_5k

4.2 Diagrama de Caso de uso e Descrição de caso de uso

Já vimos, na Unidade II, o diagrama de caso de uso e as principais notações do caso de uso. Você se lembra? Vamos recordar?

Um diagrama de caso de uso é um diagrama comportamental da UML que evidencia a visão externa do sistema a ser desenvolvido. Ele mostra quais são as funcionalidades desse sistema e quem interage com elas (BOOCH et al, 2004).

Vamos esclarecer.

Normalmente, cria-se um diagrama de caso de uso chamado de nível 1 com a visão geral e outros casos de uso com visões mais detalhadas. Independente da forma utilizada para a construção dos diagramas de caso de uso, é importante entender que a construção de um diagrama (qualquer que seja ele) não é definitiva. Podem ocorrer e ocorrerão mudanças durante o processo de levantamento de requisitos e mesmo após a implantação do software.

Saiba mais!

Medeiros (2004) diz que um software pequeno pode ter aproximadamente dez diagramas de caso de uso e cada diagrama, em média, teria 8 a 10 casos de uso para serem trabalhados. Isso dá cerca de 80 casos de uso.

Muito trabalho para os desenvolvedores, não?

Uma sugestão de estrutura para um caso de uso segundo Medeiros (2004)

Nome Referência: UC03298 - Nome do Caso de Uso

Verbo no infinitivo (informar, comprar, pagar...)

Breve Descritivo

Descrição que informa do que trata este Caso de Uso.

Pré-Condições

Descrição que informa o que é necessário acontecer para que este Caso de Uso se inicie.

Atores Envolvidos Cenário Principal

A descrição de uma tarefa que represente o mundo perfeito, sem exceções. Verbos no presente do indicativo ou substantivos, iniciando a frase com (Registra, Compra, Seleciona, Informa...)

Cenário Alternativo

Qualquer situação que represente uma exceção de um cenário principal. Veja o significado de exceção mais adiante neste capítulo.

Requisitos Especiais

Qualquer situação não contemplada anteriormente (adjetivos), veja exemplos mais adiante.

Dados

Tipos de dados que foram encontrados durante a descrição do Caso de Uso. Aqui informamos: texto, número, data etc., ou mesmo o tipo de dado e seu tamanho, conforme a linguagem a ser utilizada, caso você os conheça.

Sugestão de estrutura de Caso de Uso.

Observação
Analista de Negócio: _____

Entrevistado: _____

Área: _____

Data: ____/____/_____

Versão: XPTO

Saiba mais!

Vamos ver o vídeo explicando como preencher cada um desses campos?

https://pt.slideshare.net/natanaelsimoes/descrio-formal-de-casos-de-uso

A seguir, estudaremos o diagrama de atividades.

4.3 Diagrama de Atividades

O diagrama de atividades tem como objetivo principal a especificação do comportamento do software, do ponto de vista funcional, ou seja, das suas funcionalidades (OMG, SD).

Esse diagrama tem vários objetivos:

Documentar o aspecto funcional do software. Ele representa o fluxo da informação que o software trabalhará e as condições ou decisões que precisam estar detalhadas ou descritas.

Mostrar aspectos específicos de alguma rotina que será automatizada pelo software. Deve-se evitar especificar toda uma funcionalidade, pois isto pode gerar diagramas de atividades complexos e difíceis de entender.

Mostrar como serão implementados os Requisitos Funcionais.

Documentar de forma macro como o sistema irá funcionar; mostrar como os módulos do sistema interagem entre si e as principais informações utilizadas referentes a entradas e saídas.
Lembra-se do caso de uso do sistema da biblioteca apresentado na Unidade II?

Para recordar:

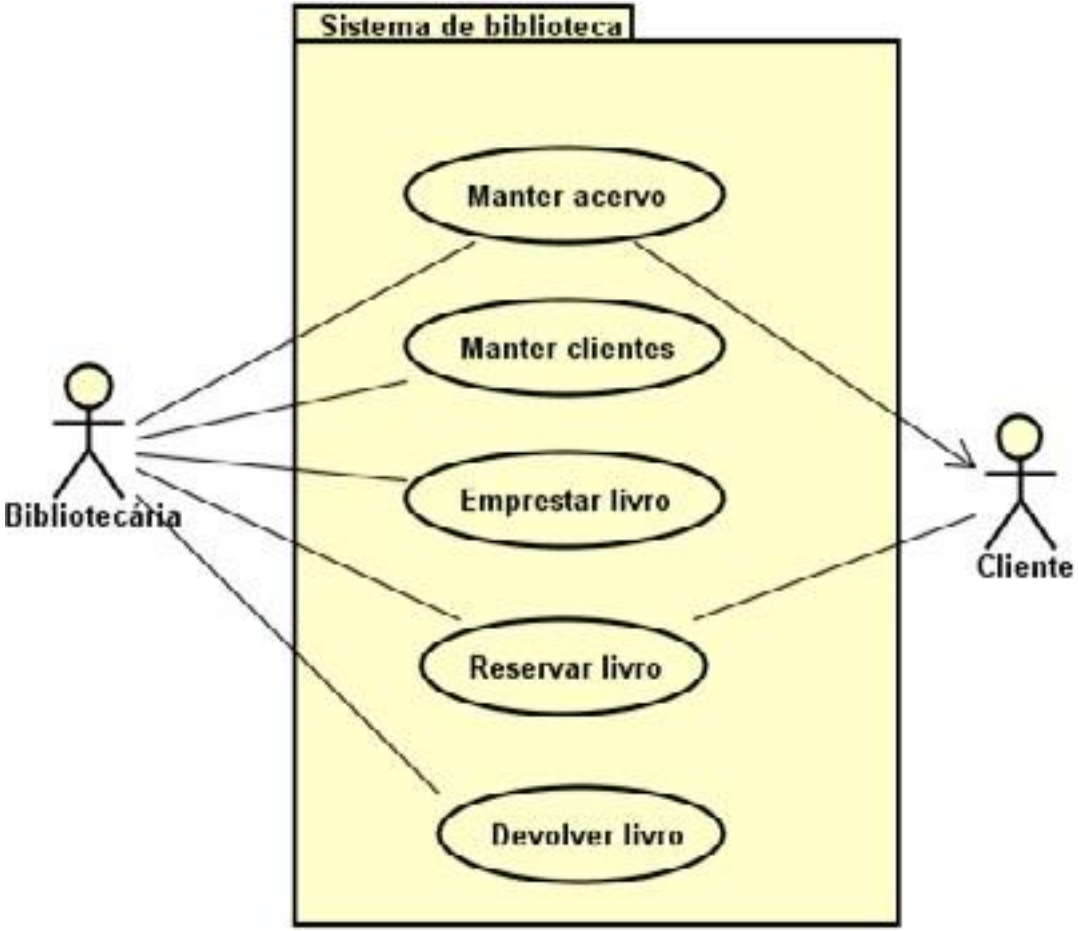


Figura 2 – Sistema de biblioteca.
Fonte: CASTRO et al, 2014.

Anteriormente, afirmamos que os diagramas podem ser modificados. Assim, em uma conversa posterior com o cliente do sistema de biblioteca identificamos que:

A biblioteca é vinculada a uma universidade e só podem ser clientes alunos desta universidade.

Dentro do caso de uso manter clientes, tem-se o envio de e-mail para confirmação de cadastro do cliente e o envio de informações (esclarecimentos) sobre como acessar a biblioteca digital.

Assim, teríamos o sistema de biblioteca, agora apresentado pela Figura 3.

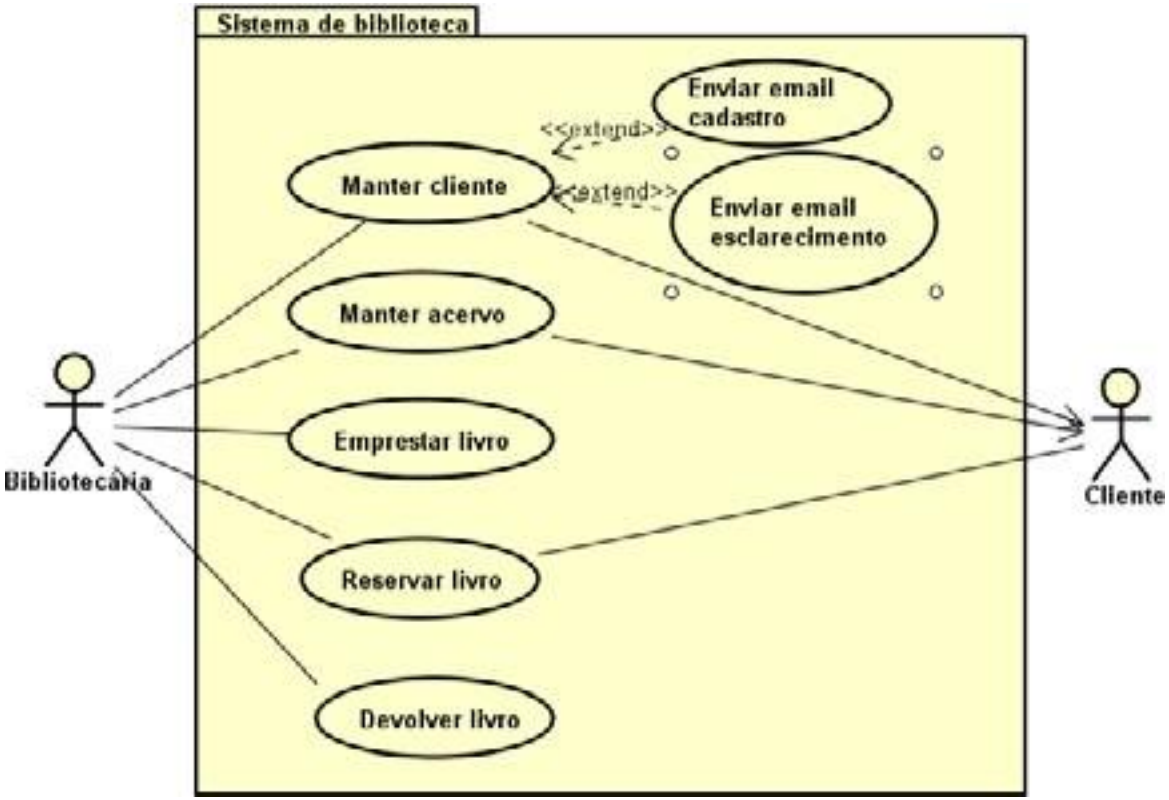


Figura 3 – Sistema de biblioteca representando as novas descobertas.
Fonte: adaptado de CASTRO et al, 2014.

A Figura 4 apresenta um exemplo bem simples para facilitar a compreensão do diagrama de atividade. Vamos entender as notações utilizadas nessa figura.

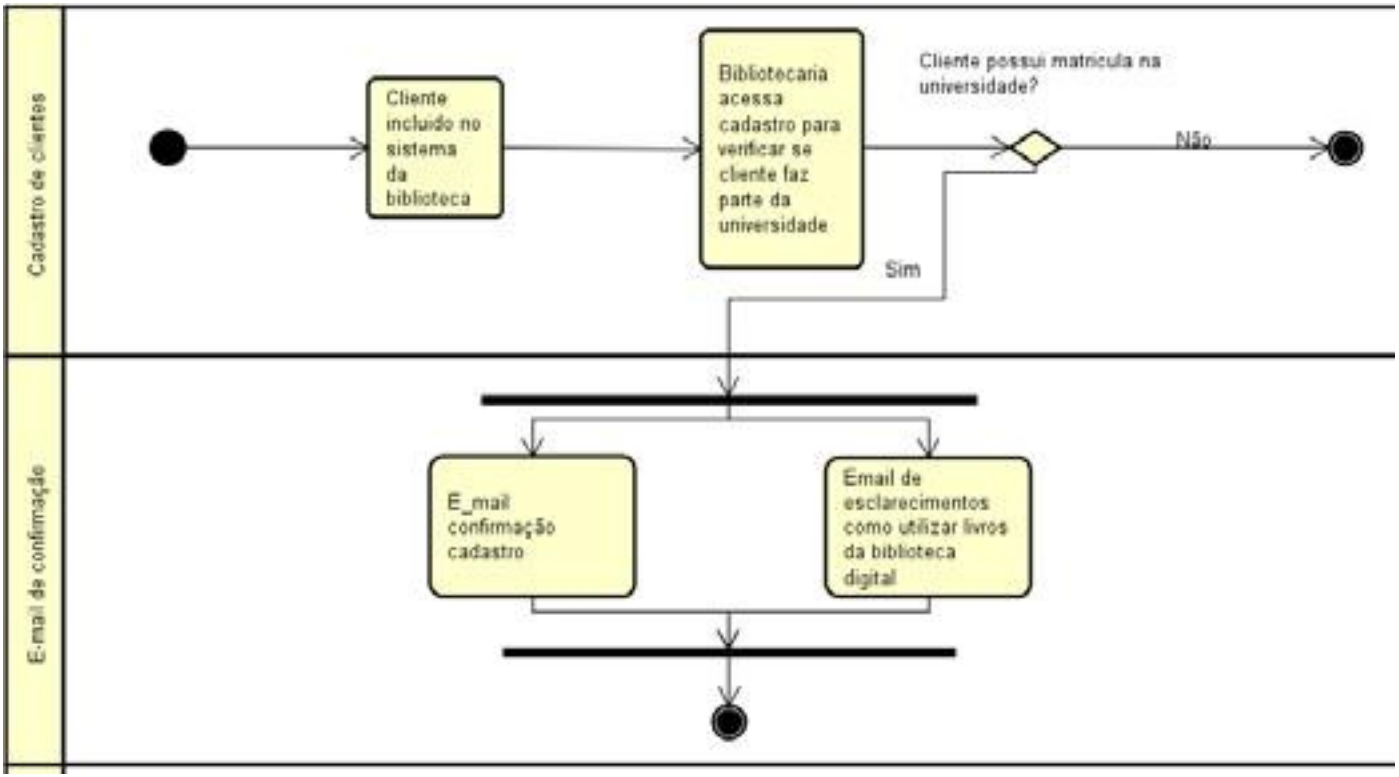


Figura 4 – Exemplo de um diagrama de atividades.
Fonte: elaborado pela autora.

Notações utilizadas em um diagrama de atividades:

- Partição ou raia: ilustra as fronteiras entre módulos, funcionalidades, sistemas ou subsistemas. Pode ser vertical ou horizontal. Uma curiosidade: o nome raia faz uma analogia com as raias das piscinas. A notação para raia é:

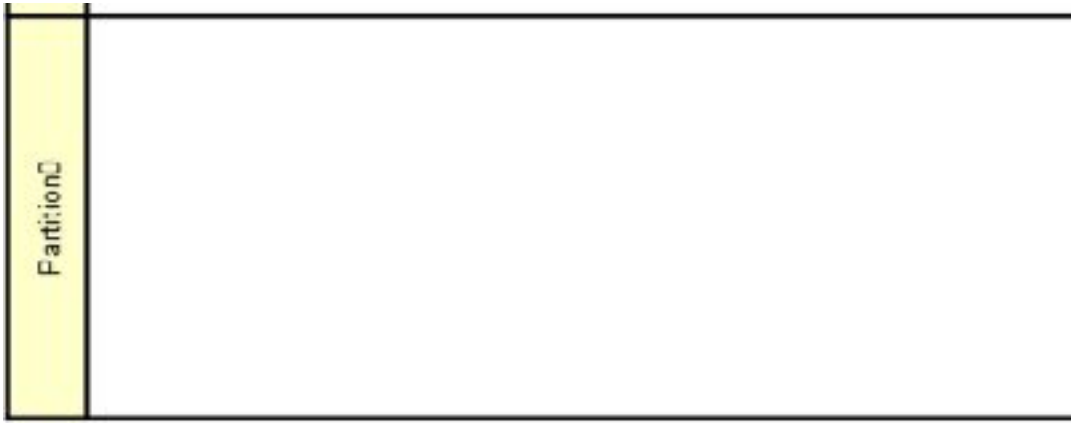


Figura 5 – Exemplo de um partição ou raias
Fonte: elaborado pela autora.

- A Iniciação ou Entrada: define o início do fluxo. Um diagrama de atividades pode ter mais de um elemento de iniciação, pois o seu início pode acontecer em mais de um "local". A notação para a iniciação é 

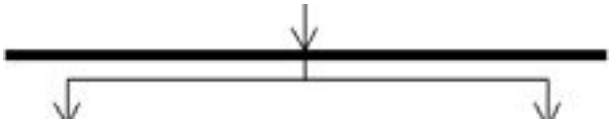
- A Atividade: representa a ação realizada. No exemplo da figura 4 temos várias atividades. Você consegue identificar quais? A notação para atividade é:

No exemplo acima temos quantas raias? Quais os nomes delas?

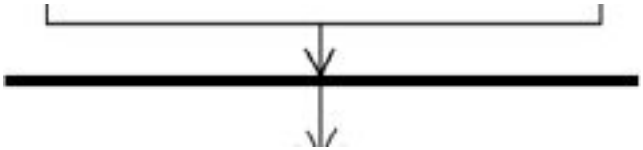
- A decisão representa uma condição que pode desviar o fluxo ilustrado no diagrama. Ela é utilizada quando lidamos com decisões. Por exemplo: "O cliente possui matrícula na universidade? Se sim, desvia para a atividade de enviar e-mail, se não, vai para o fim da atividade. A notação para decisão é:

Você identifica quantas decisões na Figura 4?

- Fork/Join - o Fork tem como objetivo dividir o fluxo em mais de uma direção. Sua notação é:



O Join tem a finalidade inversa, faz a união de várias direções do fluxo em uma única direção. A notação é:



Na Figura 4 temos Fork e Join? Quais os fluxos que eles estão dividindo ou juntando?

A Finalização ou Saída define o fim do fluxo. Um diagrama de atividades pode ter mais de uma finalização, pois o diagrama pode ter diversos pontos de saída. Sua notação é:

Na Figura 4 temos quantas saídas? Em que momento elas ocorrem?

Para melhor ilustrar o assunto, vamos ver um vídeo sobre diagrama de atividades.

https://www.youtube.com/watch?v=vReuK7_tYWc

Estudaremos, a seguir, o diagrama de classes e suas principais características.

4.4 Diagrama de classes

O Diagrama de classes tem como objetivo descrever os vários tipos de objetos no sistema e o relacionamento entre eles. Já vimos na Unidade III o conceito de classes e objetos. Você se lembra?

Nós construímos diagramas de classes baseados em casos de uso. Isso facilita a identificar quais os objetos e classes são necessários para o sistema que estamos construindo.

Um diagrama de classes pode ser apresentado em três perspectivas (CASTRO et al, 2014), são elas:

1. Conceitual - apenas classes são utilizadas. Neste tipo de perspectiva, uma classe é interpretada como um conceito. Apenas atributos são utilizados e alguns tipos de relacionamentos.

2. Especificação - tanto classes como interfaces são utilizadas neste tipo de perspectiva. O foco consiste em mostrar as principais interfaces e classes juntamente com seus métodos. Não é necessário mostrar todos os métodos, pois o objetivo deste diagrama, nesta perspectiva, é prover um maior entendimento da arquitetura do software a nível de interfaces.

3. Implementação - nesta perspectiva, vários detalhes de implementação podem ser abordados, tais como: visibilidade de atributos e métodos, parâmetros de cada método, tipos dos atributos e dos valores de retorno de cada método etc.

Nessa unidade, vamos focar, inicialmente, na perspectiva conceitual, pois estudaremos o diagrama de classes como representação das entidades, atributos e relacionamentos. Um diagrama de classes contém entidades e relacionamentos. As entidades podem ser representadas pelas classes. As classes são representadas por um retângulo dividido em três compartimentos (OMG, SD), conforme figura 6:

Nome - que conterá apenas o nome da classe modelada;

Atributos - que possuirá a relação de atributos que a classe possui em sua estrutura interna;

Operações - que serão os métodos de manipulação de dados e de comunicação de uma classe com outras do sistema.

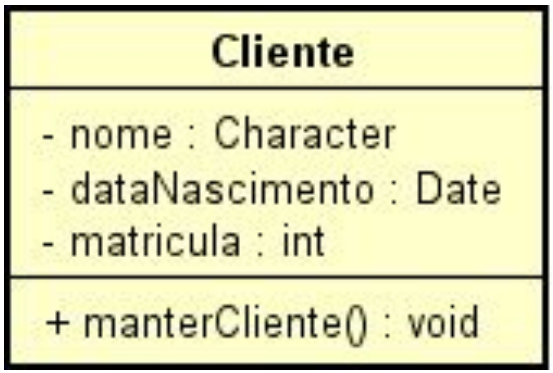


Figura 6 – Exemplo de diagrama de classes.
Fonte: elaborado pela autora.

Os relacionamentos ligam as classes/objetos entre si, criando relações lógicas entre estas entidades. O relacionamento tem:

Papel - descreve o relacionamento. A figura 7 apresenta o papel “possui” que descreve o relacionamento ou a conexão entre as classes cliente e empréstimo.

Figura 7 – Exemplo de papel.
Fonte: elaborado pela autora.

O que a figura 7 mostra? Que um objeto cliente pode se relacionar com muitos objetos empréstimo, ou seja, de uma forma bem simples: que um cliente pode ter 0 ou muitos empréstimos de livros.

Multiplicidade: para expressar a multiplicidade entre os relacionamentos, um intervalo indica quantos objetos estão relacionados no link. O intervalo pode ser de zero para um (0..1), zero para vários (0..* ou apenas *), um para vários (1..*). É também possível expressar uma série de números como (1, 4, 3..5). Se não for descrito nenhuma multiplicidade, então é considerado o padrão de um para um (1..1 ou apenas 1). A tabela 1 apresenta de forma sucinta as possibilidades, e a figura 8, um exemplo.

0..1	No máximo um. Indica que os objetos da classe associada não precisam obrigatoriamente estar relacionados.
1..1	Um e somente um. Indica que apenas um objeto da classe se relaciona com os objetos da outra classe.
0..*	Muitos. Indica que podem haver muitos objetos da classe envolvidos no relacionamento
1..*	Um ou muitos. Indica que há pelo menos um objeto envolvido no relacionamento.
3..5	Valores específicos.

Tabela 1 – Tipos de multiplicidade.
Fonte: elaborado pela autora.

Saiba mais!

Saiba mais sobre o diagrama de classes lendo o artigo: Orientações básicas na elaboração de um diagrama de classes. O artigo é bem detalhado e dá um passo a passo para a elaboração de um diagrama de classes.

https://www.youtube.com/watch?v=rDidOn6KN9k

Os relacionamentos, considerando a perspectiva conceitual, podem ser dos seguintes tipos:

Associação: é uma conexão entre classes e também significa que é uma conexão entre objetos daquelas classes. É representada por uma linha sólida entre duas classes. Pode-se também colocar uma seta no final da associação indicando que esta só pode ser usada para o lado em que a seta aponta (OMG, SD). A figura 8 apresenta esses dois tipos de representação. Esse diagrama de classes representa um modelo simples em que um cliente tem faturas e pode efetuar o pagamento. A classe pagamento é um exemplo de classe abstrata.

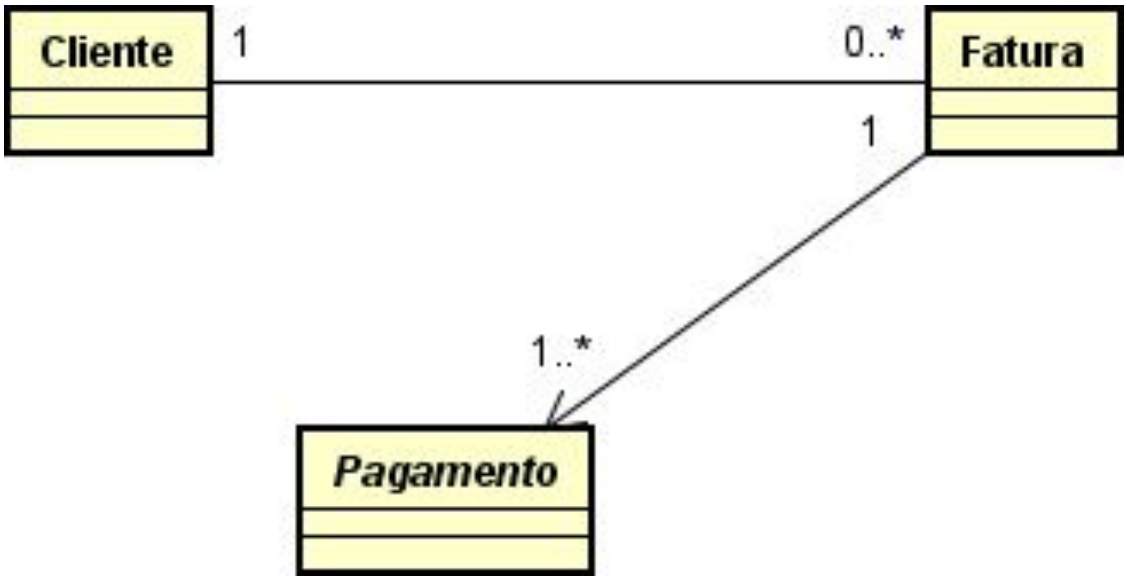


Figura 8 – Exemplo de associação.
Fonte: elaborado pela autora.

Para Refletir...

Pesquise a diferença entre classe concreta e abstrata. Analise e tente responder à pergunta:

- Por que pagamento é uma classe abstrata?

Herança/Generalização: já vimos o conceito de Herança/Generalização na Unidade III. Mecanismo baseado em objetos que permite que as classes compartilhem atributos e operações baseados em um relacionamento, geralmente, generalização.

A Representação Gráfica da Herança ou Generalização para a UML é 

A Figura 9 apresenta um exemplo de Herança/Generalização, considerando uma modificação do diagrama de classes da Figura 6. Agora foram adicionadas 3 formas de pagamento. Quais seriam elas?

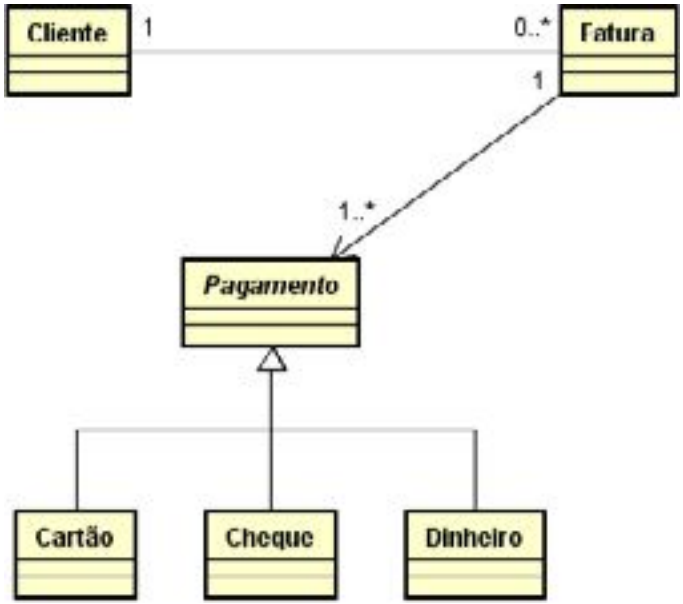


Figura 9 – Exemplo de Herança/Generalização.
Fonte: elaborado pela autora).

Saiba mais!

Veja o vídeo “UML - Diagrama de Classe - Parte 2 – Associação” para complementar o seu conhecimento.

https://www.youtube.com/watch?v=FL7zsHDqLXY

4.5 Outros diagramas

A UML possui uma série de outros diagramas conforme você pode constatar na Figura 1. A seguir, descreveremos o objetivo de alguns deles. É interessante que você veja os vídeos e leia os textos indicados para compreender melhor cada um deles. Ok?

Diagrama de Sequência - tem o objetivo de mostrar como as mensagens entre os objetos são trocadas no decorrer do tempo para a realização de uma operação. Geralmente, um diagrama de sequência é criado a partir de um diagrama de casos de uso, com a finalidade de descrever como serão as interações entre cada objeto/elemento do diagrama.

Vamos assistir ao vídeo abaixo? Ele mostra, passo a passo, a construção de um diagrama de sequência.

https://www.youtube.com/watch?v=ypP6HQdDxYM

Diagrama de objetos – o diagrama de objetos é uma instância do diagrama de classes (MEDEIROS, 2004). Na Unidade III, vimos a diferença entre classes e objetos. Você se lembra?

Os diagramas de objetos são opcionais e não são tão importantes como os diagramas de classes. Porém, eles podem ser utilizados de forma complementar para esclarecer o entendimento de diagramas complexos ajudando na compreensão do sistema tanto para os desenvolvedores como para os clientes.

Saiba mais!

Leia um exemplo do diagrama de objetos no artigo a seguir.

http://micreiros.com/diagrama-de-objetos/

Ainda existem outros diagramas que não vimos nesta unidade. Você pode conhecê-los acessando os links a seguir ou procurando outros links na internet.

Diagrama de Implantação

https://estudonahttps://estudonaweb.com.br/artigo/entendendo-o-diagrama-de-implantacaoweb.com.br/artigo/entendendo-o-diagrama-de-implantacao

Diagrama de componentes

https://estudonaweb.com.br/artigo/entendendo-o-diagrama-de-componentes

4.6 Introdução a padrões do projeto

O padrão de projeto é um modelo de uma experiência amplamente testada e aprovada e que pode ser usado como um guia para a resolução de problemas específicos de arquiteturas orientadas a objetos (GAMMA et al, 2006).

Padrões e Linguagens de padrões são formas de descrever as melhores práticas em bons projetos e capturar a experiência de uma forma que torne possível a outros reusar essa experiência.

O conhecimento de padrões de projeto permite a criação de sistemas melhores, com manutenção menos custosa e reduzindo o retrabalho. Assim, o domínio sobre padrões de projeto é muito importante ao desenvolvedor de software.

Os padrões registram as experiências bem-sucedidas, tais como:

Livro “Design Patterns” - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four- GoF);

Padrões J2EE da Sun. Disponível em: http://java.sun.com/blueprints /patterns/.

GoF (Gang of four) é a referência clássica; muitos dos padrões conhecidos são baseados no catálogo GoF (GAMMA et al, 2006). Mas devemos lembrar que novos problemas incitam a criação de novos padrões de projeto, e padrões novos sempre estão surgindo.

Autores como Pressman e Maxim (2016), em uma de suas obras sobre engenharia de software, apresentam alguns pontos importantes que nos ajudam a compreender sobre padrões, tais como: elementos essenciais e passos de como utilizar um padrão. Veja a seguir.

Para esses autores, são elementos essenciais aos padrões:

- **Nome;**
- **Descrição do problema e contexto para os quais o padrão se aplica;**
- **Descrição da solução genérica proposta;**
- **Consequências da aplicação do padrão (custos e benefícios).**

Vamos ver um exemplo?

Vamos pensar que você quer desenvolver um software de e-comerce. Segundo Pressman e Maxim (2016, p.364):

> **E-commerce: específico para sites, estes padrões implementam elementos recorrentes e aplicações para comércio eletrônico.**
>
> **Padrão: shoppingcart**
>
> **Descrição: fornece uma lista de itens selecionados para compra.**
>
> **Detalhes:lista informações de itens, quantidade, código de produto, disponibilidade (disponível, esgotado), preço, informações para entrega, custos de remessa e outras informações de compra relevantes. Também oferece a habilidade de editar (por exemplo: remover, modificar quantidade).**
>
> **Elementos de navegação: Contém a capacidade de prosseguir com a compra ou sair para encerrar as compras.**

Segundo esses autores, os padrões são semiprontos. Isso quer dizer que sempre temos que revê-los, analisá-los e adaptá-los ao nosso ambiente.

Em sua obra, eles ainda nos ajudam com alguns passos para utilizar um padrão:

1. Leia o padrão todo uma vez;
2. Estude o código fonte de exemplo;

3. Escolha nomes para os participantes do padrão dentro do seu contexto;
4. Defina as novas classes e modifique classes existentes que são afetadas;
5. Defina nomes para as operações do padrão dentro do seu contexto;
6. Implemente as operações.

Alguns padrões do GoF: Abstract Factory, Prototype, Builder, Adapter, Composite, Chain of Responsibility, Observer, etc.

Acesse o Link e veja a definição do factory

https://brizeno.wordpress.com/2011/09/17/mao-na-massa-factory-method/

Vamos complementar essas informações com o vídeo a seguir. Tente relacionar o conteúdo exibido com o que discutimos até agora.

https://www.youtube.com/watch?v=5CgwgRHhCRM

Encerramento

Caro estudante, nesta unidade, você estudou a UML que é uma linguagem utilizada para modelar sistemas e os seus diagramas. A UML tem 13 diagramas e foram revistos alguns conceitos já estudados em unidades anteriores e detalhados outros conceitos com relação a essa linguagem. Entendemos que os diagramas de Casos de Uso necessitam ser descritos para facilitar o entendimento. Estudamos o objetivo e como construir um diagrama de atividades que especifica o comportamento do software. Vimos também o diagrama de classes, sua importância e objetivo. O diagrama de classes descreve os objetos, relacionamentos e tipos de multiplicidade.

Vimos também algumas características do diagrama de sequência e do diagrama de objeto. Foram disponibilizados links para você aprender os principais aspectos do diagrama de implantação e diagrama de componentes. Ao final, vimos uma introdução a padrões de projeto. Sua importância e como podem ser implementados em um sistema para melhorar a qualidade e promover o reuso das especificações. Espero que tenha gostado e assimilado estes conhecimentos de forma a aplicá-los em sua vida profissional.

Figura 10 - Tirinha
Fonte: Imagem disponível em: <https://vidadeprogramador.com.br/2011/11/23/sistema-inovador>

Questões de autoaprendizagem

Assinale a alternativa correta.

1. É motivo para se descrever um caso de uso:

   a. ( ) Os diagramas de caso de uso não esclarecem totalmente as necessidades.
   b. ( ) Os diagramas de caso de uso podem propiciar uma interpretação equivocada.
   c. ( ) A descrição de caso de uso possibilita uma definição melhor do escopo.
   d. ( ) A descrição de caso de uso facilita a programação.
   e. ( ) Todas as alternativas anteriores

Solução – E. São vários os motivos para descrever um caso de uso: os diagramas não esclarecem totalmente as necessidades e podem propiciar uma interpretação equivocada. Além disso, descrever define melhor o escopo e facilita a programação.

2. O diagrama de atividades tem como objetivo principal a especificação do comportamento do software, do ponto de vista funcional, ou seja, das suas funcionalidades.

   O Diagrama de classes tem como objetivo descrever os vários tipos de objetos no sistema e o relacionamento entre eles.

   Acerca dessas asserções, assinale a opção correta.

   a. ( ) As duas asserções são proposições verdadeiras, e a segunda é uma complementação da primeira.
   b. ( ) As duas asserções são proposições verdadeiras, mas a segunda não é uma complementação da primeira.
   c. ( ) A primeira asserção é uma proposição verdadeira, e a segunda, uma proposição falsa.
   d. ( ) A primeira asserção é uma proposição falsa, e a segunda, uma proposição verdadeira.
   e. ( ) Tanto a primeira quanto a segunda asserções são proposições falsas.

Solução. As duas proposições estão corretas, e a segunda não é complementação da primeira. O diagrama de atividades tem como objetivo principal a especificação do comportamento do software, do ponto de vista funcional, ou seja, das suas funcionalidades. O Diagrama de classes tem como objetivo descrever os vários tipos de objetos no sistema e o relacionamento entre eles.

Referências

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. Rio de Janeiro: Campus, 2006.

BOOCH,G.;RUMBAUGHJ.;JACOBSONI.**UML**:guiadousuário .2.ed. Rio de Janeiro: Campus, 2004.

BOOCH,G. **Object- oriented analysis and design with applications**. Califórnia: Benjamin/Cummin- gs Pub,1994.

CASTRO, E.R.C, CALAZANS, A.T.S, PALDES, R.A., GUIMARAES, F.A. **Engenharia de requisitos**: um enfoque prático na construção de software orientado a negócio. Florianopolis: Bookess, 2014.

INSTITUTE OF ELECTRICALAND ELECTRONICS ENGINEERS/COMPUTERSCIENCE. **Guide to the Software Engineering Body of Knowledge** v.3(SWEBOK). 2014. Disponível em:<https://www.computer.org/web/swebok/v3 >. Acesso em 13 de dez.2018.

JACOBSON, I; ERICSSON, M; JACOBSON,A**.The Object Advantage:** Business Process Reengineering With Object Technology. Addison-Wesley Professional, 1994.

MEDEIROS, E**. Desenvolvendo Software** Com Uml 2.0 Definitivo. SÃO PAULO: Pearson Makron Books, 2004.

Referências

MOREIRA T.R.F., RIOS, E. **Projeto e Engenharia de Software**: Teste de Software.. Editora: Alta Books, 2003.

PFLEEGER, S.L. **Engenharia de software**: teoria e prática. São Paulo: Pearson, 2004.

PMI. **Um guia de conhecimento de gerenciamento de projetos** – Guia PMbok.6ª. ed. PMI -Project. Management Institute: 2018 .

RUMBAUGH, J.; BLAHA, M.; EDDY, F.; LORENSEN, W. **Object Modeling Technique** - OMT, 1991.

SBROCCO, J.H.T.C. **Metodologias ágeis**: engenharia de software sob medida. São Paulo: Erica, 2012.

SCHWABER,K. SUTHERLAND, J. **Guia do Scrum Um guia definitivo para o Scrum:** as regras do Jogo .2017. Disponível em: <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Portuguese-Brazilian.pdf.>. Acesso em: 24 de nov.2018.

SOMMERVILLE, I. **Engenharia de software**.9ª. edição. São Paulo: Pearson Prentice Hall, 2011.

SUTHERLAND, J. **Scrum: a arte de fazer o dobro do trabalho na metade do tempo** /; tradução de Natalie Gerhardt. -São Paulo: LeYa, 2014.

Currículo da autora

Possui doutorado em Ciência da Informação pela Universidade de Brasília (2008) e mestrado em Gestão do Conhecimento e TI pela Universidade Católica de Brasília (2003). Atuou, por 28 anos, como especialista em TI da Caixa Econômica Federal. Tem experiência na área de Ciência da Computação e Ciência da Informação. Professora universitária há mais de 10 anos, atuando na graduação e pós graduação de cursos de TI. Pesquisadora, atuando principalmente nos seguintes temas: análise por pontos de função, métricas, data mart, data warehouse e processo de desenvolvimento de software, qualidade de software, qualidade da informação, metodologia de pesquisa, contratação de serviços de TI, modelos ágeis, engenharia de requisitos, metodologias ativas de aprendizagem, tecnologia da informação e comunicação.

Angélica Toffano Seidel Calazans

INSTITUTO FEDERAL
Brasília

Secretaria de
Educação Profissional
e Tecnológica

Ministério da
Educação