# ANÁLISE E PROJETO DE SISTEMAS

Gláucia Silva Bierwagen

E-book 1

FAM ONLINE

# Neste E-Book:

# INTRODUÇÃO

Nesta unidade você aprenderá sobre a modelagem de projetos ou sistemas de softwares, por meio da *Unified Modeling Language (UML),* também chamada de Linguagem de Modelagem Unificada. Por se tratar de uma linguagem padrão para a elaboração e estruturação de sistemas de software, é muito importante que você conheça seus conceitos gerais.

Estudaremos os conceitos do paradigma de objetos: abstração, encapsulamento, polimorfismo e herança. Você conhecerá as principais fases do processo de desenvolvimento de sistemas, conforme seus problemas e necessidades. Conheceremos o que são Ciclos de Vida: em cascata, incremental e iterativo, prototipagem, espiral, métodos ágeis e extreme programming. Além disso, você aprenderá quais são os principais diagramas de UML.

E, por fim, estudaremos as principais notações gráficas relacionadas aos Diagramas de Casos de Uso, observando modelos e exemplos.

E, então, preparado? Aproveite e bons estudos!

# CONCEITOS GERAIS DE *UNIFIED MODELING LANGUAGE* (UML)

Na construção civil, para realizar a construção de casas, prédios ou qualquer outra estrutura, é necessário, antes, elaborar modelos de projetos que possibilitem visualizar, dimensionar e calcular a obra que será construída. Para isso, os profissionais dessa área – que são vários e com diferentes habilidades – elaboram uma planta baixa ou croqui para que se possa planejar o tamanho dos cômodos, das paredes, localização de janelas, portas, pias, etc. e, possivelmente, para a visualização de detalhes de cores, iluminação, estilos, etc., construam um projeto 3D. Em alguns casos, talvez seja necessária a elaboração de modelos adicionais para expressar as gradações do projeto. Os modelos são elaborados a partir de paradigmas (padrões) e representações características – imagens, símbolos, ideias, etc. Na área de sistemas de informação, de modo similar, é necessária a elaboração de projetos para que os softwares e os sistemas sejam construídos. Mas, como isso é feito?

# Paradigma de programação e o conceito de modelagem

Primeiramente, vamos entender o que é o paradigma de programação e o conceito de modelagem? Paradigma de programação "é um modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns", enquanto que a modelagem é a construção de diagramas, ou seja, uma representação gráfica da realidade (GONÇALVES, 2015, p. 12). Os paradigmas mais populares são os estruturados – que usam a modelagem de diagrama de fluxo ou fluxograma de dado, conforme figura 1 – e os paradigmas orientados ao objeto – que usam a modelagem *Unified Modeling Language* (UML).

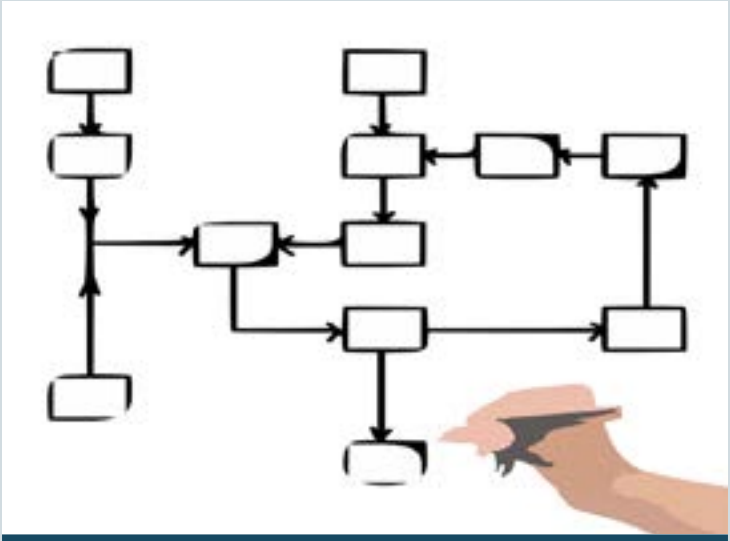

***Figura 1:*** *Fluxograma.* ***Fonte:*** *Pixabay.*

Os modelos são utilizados para elaboração de sistemas para aperfeiçoar: (1) o gerenciamento da complexidade, descrevendo perspectivas desses sistemas; (2) comunicação das pessoas envolvidas; (3) redução dos custos de desenvolvimento; e (4) a previsão futura do sistema (BEZERRA, 2007).

A partir do que estudamos, para a compreensão do paradigma orientado ao objeto, você precisa conhecer alguns conceitos. Dentre eles:

● um *objeto*, que pode ser qualquer coisa abstrata ou concreta, pode ser construído e complexo; realizar uma tarefa por meio da requisição de serviços (*emissão de mensagens)* de outros objetos e pertencer a uma classe. Por exemplo, o objeto pode ser um carro, uma reserva de passagem aérea, uma instituição, etc.

● uma *classe* é um conjunto de atributos e serviços comuns a um grupo de objetos. Dessa forma, pode-se entender que o objeto é uma instância (ou está sob o campo) de uma classe.

● a *abstração* é um processo mental em que nos concentramos no que o objeto é e faz para, a partir daí, formularmos os conceitos gerais.

● o *encapsulamento* consiste somente na visualização de aspectos externos de um objeto – sua interface –, ou seja, seus detalhes internos ficam ocultos aos demais objetos.

● o *polimorfismo* representa o estado de um objeto ser capaz de assumir várias formas, ou seja, o objeto,

ao receber uma mensagem, é quem determinará qual será sua ação ou operação.

● a herança é um mecanismo que permite que as características comuns a diversas classes possam ser generalizadas ou especializadas em uma mesma classe. Por exemplo, carro (subclasse), motos (subclasse), ônibus (subclasse) podem ser generalizados na classe veículos automotores (superclasse). Nesse caso, dizemos que houve a generalização (universalização). Em outro caso, podemos ter a especialização (tipificação) de uma classe; por exemplo, em estudante (superclasse), podemos ter estudantes de pós-graduação (subclasse), estudantes de ensino médio (subclasse), de ensino fundamental (subclasse), etc.

● a *composição* possibilita que elaboremos outros objetos a partir da reunião de objetos; por exemplo, um carro pode ser a reunião de objetos mais simples como: motor, rodas, volante, etc. (GONÇALVES, 2015; BEZERRA, 2007; FOWLER, 2005).

**Acesse o Podcast 1 em Módulos**

Visualmente, você pode identificar que a abstração permeia, ou seja, transpassa, percorre os outros conceitos do paradigma orientado a objetos.

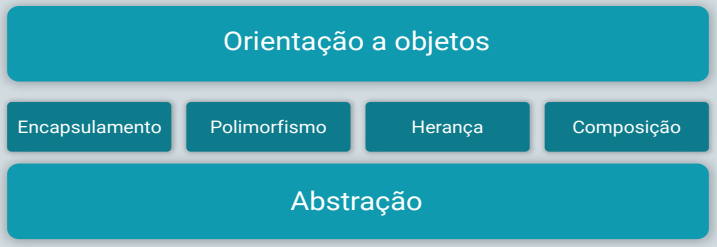


***Figura 2:*** *Princípios da orientação a objetos podem ser* ***w to s*** *como aplicações de um princípio mais básico, o da abstração.* ***Fonte:*** *Adaptado Bezerra (2007).*

Mas, afinal, o que é a modelagem UML? Vamos encontrar resposta para essa pergunta? Continue lendo e estudando.

## UML – Unified Modeling Language

Até o momento, você aprendeu sobre o paradigma de programação, sendo que um deles é o orientado a objetos. Mas, o que exatamente é *Unified Modeling Language*? UML é

> *uma linguagem visual para modelar sistemas orientados a objetos. Isso quer dizer que a UML é uma linguagem que define elementos gráficos (visuais) que podem ser utilizados na modelagem de sistemas. Esses elementos permitem representar os conceitos do paradigma da orientação a objetos. Através dos elementos gráficos definidos nesta linguagem pode-se construir diagramas que representam diversas perspectivas de um sistema (BEZERRA, 2007, p. 15).*

## SAIBA MAIS

Para saber mais sobre UML, leia o primeiro capítulo do livro **"UML Essencial: um breve guia para a linguagem-padrão de modelagem de objetos"**, de Martin Fowler, que traz o conceito da UML, como se chegou a ela, e quais foram os pesquisadores e desenvolvedores que iniciaram o trabalho com a UML para a elaboraração dos sistemas. O livro está disponível em sua biblioteca virtual (Minha Biblioteca).

Então, com ela podemos especificar, visualizar, documentar e construir elementos de um sistema por meio de notações gráficas e visuais. E por que vamos aprender sobre a UML? Aprenderemos que a UML é bastante vantajosa porque:

**1.** Serve como linguagem para expressar decisões de projeto que não são óbvias ou que não podem ser deduzidas do código.

**2.** Provê uma forma concreta o suficiente para a compreensão das pessoas e para ser manipulada pelas máquinas.

**3.** É independente da linguagem de programação, ou seja, pode ser utilizada para desenvolvimento de um sistema através das principais linguagens de programação orientadas a objetos, como Java, C++ ou PHP 5.

**4.** É independente dos métodos de desenvolvimento, ou seja, pode ser utilizada para modelar sistemas

em organizações que utilizam métodos ágeis, RUP (*Rational Unified Process*), ou outro método como base.

**5.** Não é uma metodologia de desenvolvimento de sistemas, mas define uma representação gráfica que pode ser usada para modelar sistemas orientados a objeto.

**6.** Não depende de ferramentas de modelagem, embora estas demonstrem alta produtividade; mas pode ser feita em papel ou algum editor gráfico (GONÇALVES, 2015, p. 14).

Como você pôde perceber, existem inúmeras vantagens na aplicação da UML na modelagem de sistemas. Mas, como se dá o processo de elaboração de um software?

# CONCEITOS DE ENGENHARIA DE SOFTWARE PARA ELABORAÇÃO DE SISTEMAS

Para a elaboração de um sistema de software, vamos utilizar os conceitos de engenharia de software, o que implica dizer que existem fases que devem ser seguidas para que o software seja elaborado. Cada uma das fases é composta por tarefas com entradas (necessidade para que a tarefa seja realizada), saídas (a produção da tarefa), e a atribuição (quem realizará a tarefa) (GONÇALVES, 2015).

Cada empresa cria seu próprio processo de desenvolvimento, conforme os problemas e necessidades, mas, de modo geral, seguem as seguintes fases:

1. **Levantamento de requisitos** ou **elicitação de requisitos** tem o objetivo de estabelecer quais funcionalidades o sistema a ser desenvolvido deverá ter, ou seja, é a fase que visa a compreender os problemas e as necessidades – requisitos – dos usuários do sistema, documentando isso no documento de requisitos (posteriormente, compreenderemos os tipos de requisitos e como levantá-los perante aos usuários).

2. **Análise de requisitos** ou **engenharia de requisitos** tem o objetivo de fazer um estudo detalhado dos requisitos levantados na fase anterior, gerando diagramas por meio de uma linguagem de modelagem específica (diagramas de casos de uso, de classes, etc.), a fim de realizar uma primeira aproximação

da descrição do documento de requisitos com a implementação, ou seja, com o que o usuário quer.

3. **Projeto** (desenho) que determinará "como" o sistema funcionará para atender os requisitos, de acordo com os recursos tecnológicos existentes (a fase de projeto considera os aspectos físicos e dependentes de implementação). São considerados na fase de projeto: arquitetura do sistema, padrão de interface gráfica, linguagem de programação, gerenciador de banco de dados, etc.

4. **Desenvolvimento** ou **implementação** tem o objetivo de implementar o sistema em uma determinada linguagem de programação, tendo como base os modelos criados na fase de análise e evoluídos na fase de projeto.

5. **Teste** tem o objetivo de submeter o software desenvolvido a algumas situações e analisar os resultados produzidos, com a finalidade de verificar se o software está de acordo com o requisito utilizado para implementá-lo. Além disso, o teste aborda algumas situações a que o software é vulnerável, e verifica se falhas acontecem. Outros testes ainda podem ser feitos, relacionados às diversas finalidades, por exemplo, verificar o desempenho do sistema. Geralmente, é elaborado um relatório de testes (documento) com os erros detectados no software.

6. **Validação** é onde o cliente analisa se o produto gerado atende suas necessidades. Portanto, nesta fase, os analistas precisam assegurar que a especi-

ficação com que construíram do software é correta, consistente, completa, realista e sem ambiguidades.

7. **Implantação** tem o objetivo de disponibilizar o uso do sistema no ambiente de sua operação, treinando usuários, configurando o ambiente de produção e, muitas vezes, convertendo bases de dados (LARMAN, 2004; BEZERRA, 2007; GONÇALVES, 2015).

Como você pôde notar, o desenvolvimento de um sistema envolve diversas fases. Ao encadeamento específico dessas fases para a construção do sistema dá-se o nome de **Modelo de Ciclo de Vida**. Observe o **Modelo em Cascata** (figura 3), no qual as atividades são sequenciais e uma fase deve ser terminada para a outra começar.

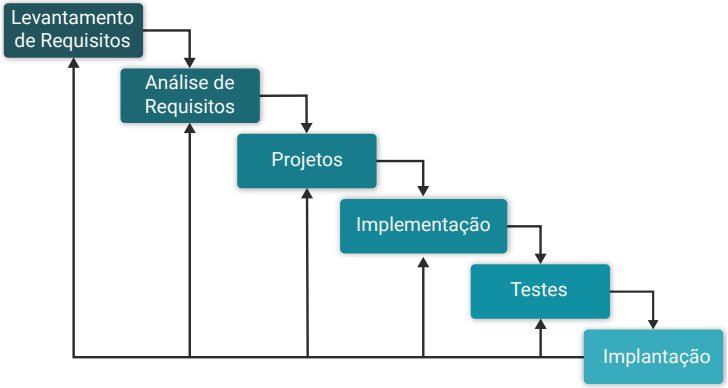


***Figura 3:*** *A abordagem de desenvolvimento de software em cascata.*
***Fonte:*** *Adaptado Sommerville (2019).*

Há ainda o **Modelo do Desenvolvimento Incremental e Iterativo** (figura 4), em que as atividades são intercaladas, sendo desenvolvidas em vários passos

similares (iterativos) e, em cada passo, o sistema é estendido com mais funcionalidades (incremental). Objetiva-se dar feedback rápido ao cliente. O principal representante da abordagem de desenvolvimento incremental e iterativa é o denominado Processo Unificado Racional (*Rational Unified Process – RUP*) (SOMMERVILLE, 2019).

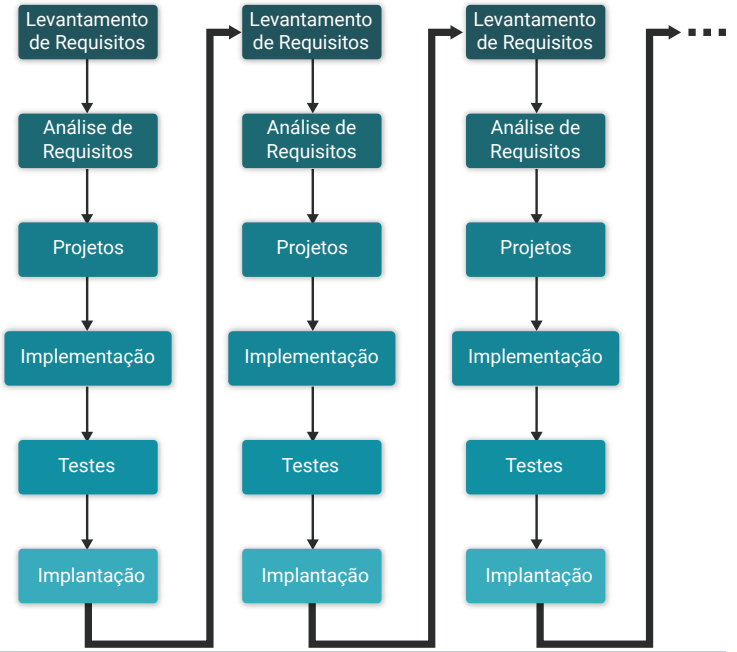


***Figura 4:*** *A abordagem de desenvolvimento de software iterativa e incremental.* ***Fonte:*** *Adaptado Sommerville (2019).*

Outro modelo é o da **Prototipagem**, que serve de complemento à análise de requisitos construindo-se protótipos. É um esboço de alguma parte do sistema. Protótipos podem ser construídos para telas de

entrada, telas de saída, subsistemas, ou mesmo para o sistema como um todo (BEZERRA, 2007).

Há, também, o **Modelo Espiral**, que combina os modelos incremental e iterativo com prototipagem, sendo o protótipo feito nas primeiras versões e o modelo incremental nas últimas versões do software.

Temos os **Modelos de Métodos Ágeis**, em que "o software não é desenvolvido como uma única unidade, mas como uma série de incrementos – cada incremento inclui uma nova funcionalidade do sistema" (SOMMERVILLE, 2019, p. 40). E, por fim, temos o **Modelo Extreme Programming** (XP) (figura 5), em que "os requisitos são expressos como cenários (chamados de estórias do usuário), que são implementados diretamente como uma série de tarefas" (SOMMERVILLE, 2019, p. 44).

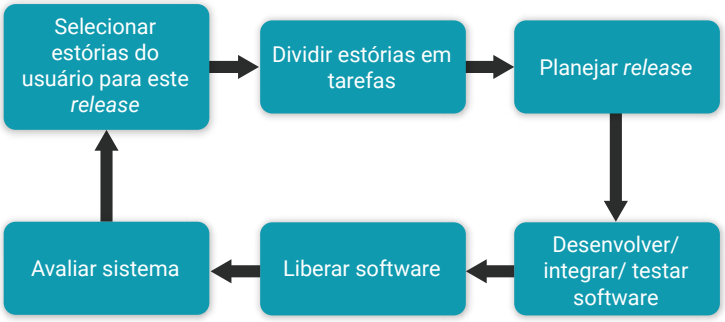


***Figura 5:*** *A abordagem de desenvolvimento extreme programming (XP).* ***Fonte:*** *Adaptado Sommerville (2019).*

Destacamos, ainda, que existem sistemas de software que são utilizados para dar suporte ao ciclo de vida de desenvolvimento de um sistema. Dois tipos

de softwares que têm esse objetivo são as ferramentas CASE (Engenharia de Software Auxiliada por Computador) e os ambientes de desenvolvimento. Tais ferramentas facilitam a criação de diagramas, interação com o código-fonte, e o rastreamento de requisitos (BEZERRA, 2007).

Como você deve ter percebido, existem fases ou etapas para o desenvolvimento de sistemas, além de exemplos de modelos de elaboração. Agora, você conhecerá quais são os principais tipos de diagrama de UML e as várias vantagens da aplicação da UML na modelagem de sistemas. Para começar, como se dá o processo de elaboração de um software?

**Acesse o Podcast 2 em Módulos**

# COMPOSIÇÃO DOS DIAGRAMAS

Por que em UML há tantos diagramas? Existem diversos tipos de diagramas e os estudaremos melhor mais adiante, porque o principal objetivo é proporcionar múltiplas visões do sistema a ser modelado, analisando-o e modelando-o sob diversos aspectos e, com isso, permitir que cada diagrama complemente os outros.

## FIQUE ATENTO

É importante que você saiba que cada diagrama da UML analisa o sistema, ou parte dele, sob uma determinada ótica. Alguns apresentam o sistema de forma mais geral e externamente, como é o caso dos **Diagramas de Casos de Uso**. Usando-se diversos diagramas é possível descobrir falhas e evitar erros futuros (GUEDES, 2011).

Cada elemento gráfico da UML possui uma sintaxe e uma semântica, sendo que a sintaxe identifica como o elemento deve ser desenhado. Por exemplo, uma classe deve ser representada por meio de um retângulo com três compartimentos. Enquanto a semântica define o que significa o elemento, e com que objetivo ele deve ser utilizado. Esses elementos podem ser classificados em:

1. **Entidades**: Representam blocos de construção da orientação a objetos como: classes, interfaces, pacotes, anotações;

2. **Relacionamentos**: São utilizados para conectar as entidades que possuem alguma ligação entre si, como: herança, agregação e chamada de método;

3. **Diagramas**: Disponibilizam diferentes visões do sistema. Cada diagrama possui um conjunto de entidades e de relacionamentos que podem ser representados (GONÇALVES, 2015, p. 14).

Os diagramas de UML são classificados em três categorias:

> • **Diagramas Estruturais** *(ou Estáticos) que são aqueles que tratam o aspecto estrutural tanto do ponto de vista do sistema quanto das classes, a partir da representação de seu esqueleto e estruturas "relativamente estáveis". Os aspectos estáticos de um sistema de software abrangem a existência e a colocação de itens como classes, interfaces, colaborações, componentes. Os diagramas estruturais de UML são os diagramas de: Classes, Objetos, Componentes, Estrutura Composta, Pacotes e Implantação.*

> • **Diagramas Comportamentais** *que são aqueles que descrevem o sistema computacional modelado quando em execução, isto é, a modelagem dinâmica do sistema. São usados para representar as partes que "sofrem alterações", como o fluxo de atividades, a interação entre usuários e o sistema e a transição dos estados. Os diagramas comportamentais de UML são os diagramas de: Casos de Uso, Atividade e Máquina de Estados.*
>
> • **Diagramas de Interação** *que são aqueles que também descrevem o sistema computacional modelado quando em execução, isto é, a modelagem dinâmica do sistema. São utilizados para representar a comunicação entre entidades e a sequência de operações a ser invocada. Os diagramas comportamentais de UML são os diagramas de: Sequência, Comunicação (ou Colaboração), Tempo e Interatividade (Visão geral de interações) (GONÇALVES, 2015, p. 15).*

Visualize esquematicamente as categorias dos diagramas de UML:

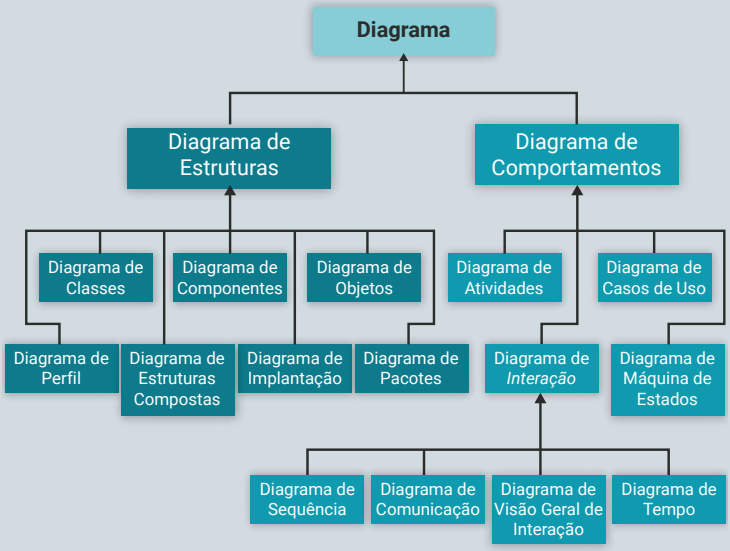


***Figura 6:*** *Categorais de diagramas de UML.* ***Fonte:*** *Adaptado WIkipedia.*

Vamos verificar, a seguir, quais são as funções dos diagramas de caso de uso.

# FUNÇÃO DOS DIAGRAMAS DE CASO DE USO

Os diagramas de caso de uso são comportamentais e usados nas fases iniciais de um projeto, portanto, seu principal objetivo é evidenciar, de forma macro (sem detalhes), os requisitos funcionais de um sistema, partindo do ponto de vista do usuário. Sua utilização é muito importante para delimitar a finalidade do sistema e determinar as funcionalidades oferecidas ao usuário. Assim sendo, o diagrama aplica-se em compreender o problema a ser resolvido, e não a detalhes de implementação do sistema.

## Diagramas e representações gráficas

Cada caso de uso de um sistema pode apresentar uma descrição narrativa – ou seja, um texto – das interações que ocorrem entre o(s) elemento(s) externo(s) e o sistema. Por exemplo, na situação "Realizar Saque", por meio do formato de texto contínuo, a narrativa pode se iniciar da seguinte forma: ***Este caso de uso inicia-se quando o cliente chega ao caixa eletrônico e insere seu cartão. O sistema requisita a senha do cliente...*** Já no formato de texto numerado, teremos a seguinte situação: ***1. Cliente insere seu cartão no caixa eletrônico. 2. O sistema requisita a senha do cliente. 3. Cliente digita a senha...***

(BEZERRA, 2007). Mas, quais são esses elementos que interagem entre si e o sistema? É importante que você saiba que, antes de apresentar os conceitos dos elementos, apresentaremos também junto a eles os diagramas de caso de uso (DCU), que são notações gráficas e textuais que representam os atores, casos de uso e relacionamentos entre esses elementos. São os seguintes:

*Os Atores* (figura 7) representam papéis (perfis) desempenhados por usuários ou qualquer entidade externa ao sistema. Segundo Bezerra (2007), os atores podem ser agrupados em categorias como funções (empregado, cliente, vendedor, etc.); organizações ou subdivisões (fornecedores, adminstradora de cartões, estoque, etc.); sistemas de softwares (sistema de cobranças, de produtos, etc.) e equipamentos com os quais o sistema pode se comunicar (sensores, leitores de QR Code, etc.).

## FIQUE ATENTO

Nos ***casos de uso*** existe a representação de alguma forma de interação – ou seja, de troca de informações – entre o sistema e o ator. Uma mesma pessoa pode desempenhar mais de um papel como ator. Por exemplo, no caso de um banco, pode ser funcionário e cliente.

Temos os **atores primários**, que iniciam a sequência de interações nos casos de uso e são agentes externos. Os **secundários** auxiliam na utilização do

sistema pelos atores primários. Para ilustrar quem pode ser, pense no seguinte: para um usuário (ator primário) acessar uma página de internet ele precisa de um navegador (ator secundário) (BEZERRA, 2007).

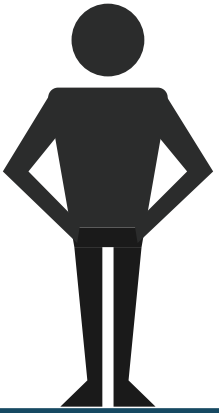

***Figura 7:*** *Representação gráfica de Nome do ator.* ***Fonte:*** *Elaboração própria.*

Os **casos de uso** (figura 8) expressam uma funcionalidade que o sistema deve conter.

***Figura 8:*** *Representação gráfica de Nome de caso de uso.* ***Fonte:*** *Elaboração própria.*

Como os atores se relacionam com os casos de uso, temos vários tipos de relacionamentos, por exemplo:

● **Comunicação**, também chamada *de associação*, informa a que caso e uso o ator está associado, significando que o ator troca informações com o sistema por meio daquele caso de uso. Pode ter também comunicação de ator para ator e de caso de uso para caso de uso. Temos como exemplo (figura 9) o **ator usuário** que pode se comunicar com o caso de uso *reservar livro*.

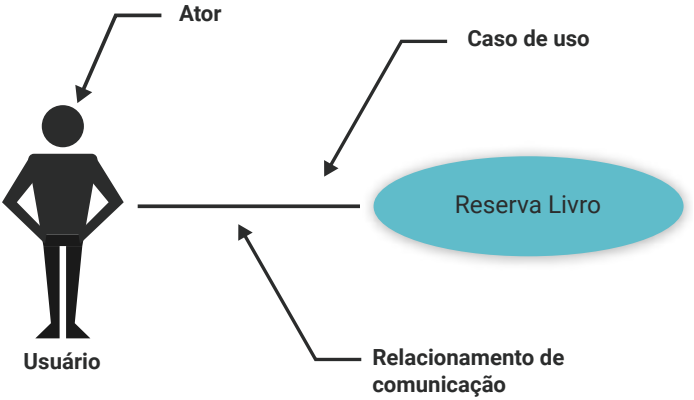


***Figura 9:*** *Representação gráfica para ator, caso de uso e relacionamento de comunicação.* ***Fonte:*** *Adaptado Bezerra (2007).*

● **Inclusão** é usado quando um caso de uso base incorpora ou inclui um ou mais casos de uso e o faz de modo obrigatório. Geralmente, é usado quando o mesmo comportamento se repete em mais de um caso de uso. Por exemplo, um **ator cliente** pode em três casos de uso – *obter extrato, realizar saque, realizar transferência* – e incluir o caso de uso *fornecer identificação* (figura 10).

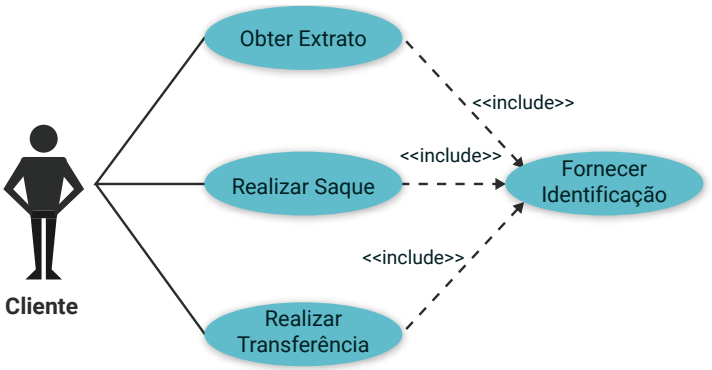


***Figura 10:*** *Representação gráfica para relacionamento de inclusão.*
***Fonte:*** *Adaptado Bezerra (2007).*

• **Extensão** é utilizado quando um caso de uso base se estende em um ou mais casos de uso e o utiliza de maneira opcional. Normalmente, é usado em um comportamento eventual de um caso de uso. Exemplificando, temos o **ator escritor** que, no caso de uso, pode *editar documentos*, mas essa edição se estende para os casos de uso *substituir texto* e *corrigir ortografia* (figura 11).

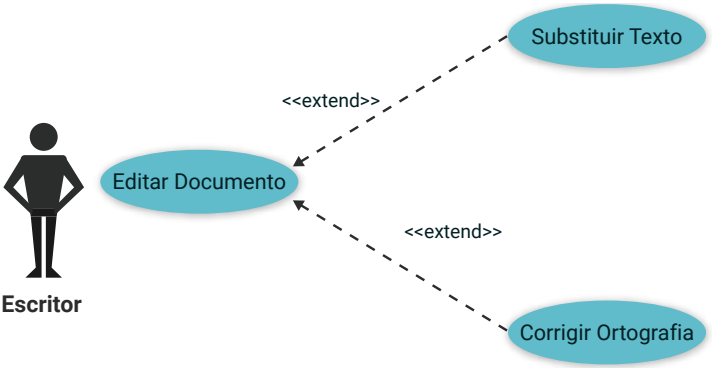


***Figura 11:*** *Representação gráfica para relacionamento de extensão.*
***Fonte:*** *Adaptado Bezerra (2007).*

● **Generalização** ou **Especialização** (especificação) ocorre quando as entidades devem ser do mesmo tipo, ou seja, dois casos de uso ou dois atores. A generalização entre casos de uso é utilizada quando se identifica dois ou mais casos de uso com comportamentos similares. A generalização entre atores tem de ser usada quando se precisa definir um ator que desempenhe um papel que já é desempenhado por outro ator em relação ao sistema, mas que também possui um comportamento particular adicional. Por exemplo, o **ator usuário** pode ter dois casos de uso – *reservar livros, pesquisar catálogos* –, mas adicionalmente como **ator professor** pode, no caso de uso, *solicitar a compra de um título de livro* (figura 12).

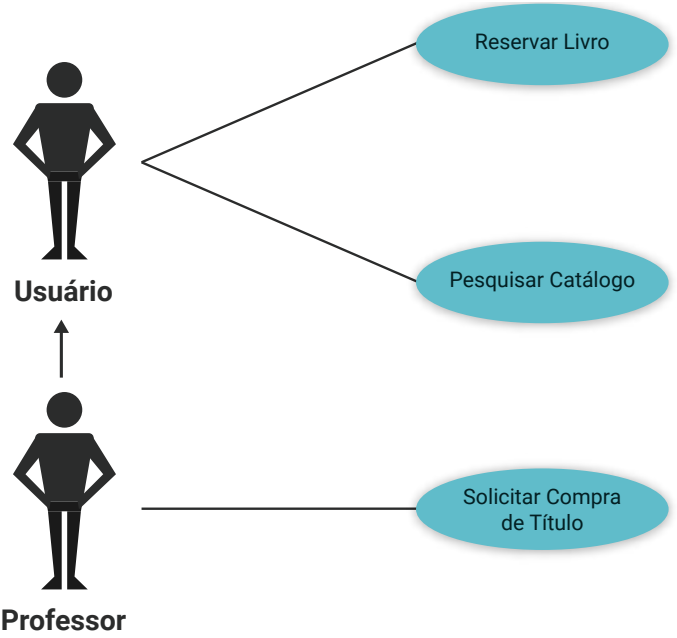


***Figura 12:*** *Representação gráfica para relacionamento de generalização ou especialização.* ***Fonte:*** *Adaptado Bezerra (2007).*

Resumidamente, temos os principais diagramas de UML.

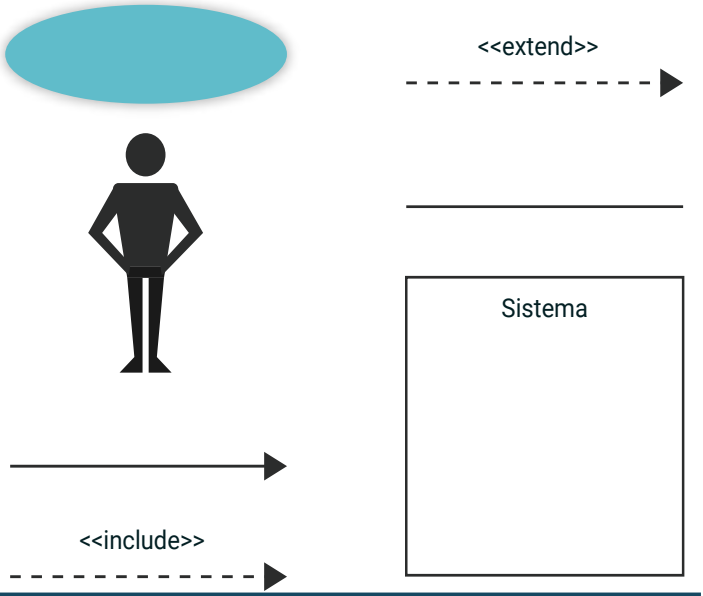


***Figura 13:*** *Elementos gráficos para a elaboração dos diagramas de casos (DCU).* ***Fonte:*** *Adaptado Bezerra (2007).*

Observe esquematicamente as possibilidades de relacionamentos entre elementos do modelo (BEZERRA, 2007):

| Elementos | Comunicação | Extensão | Inclusão | Generalização |
|---|---|---|---|---|
| Casos de Uso e Caso de Uso | | X | X | X |
| Ator e ator | | | | X |
| Casos de Uso e Ator | X | | | |

***Tabela 1:*** *Possibilidade de relacionamentos entre elementos do modelo.* ***Fonte:*** *Bezerra (2007) adaptado.*

## Documentação de casos de uso

Apresentaremos uma proposta para a descrição de um caso de uso, mas é importante lembrar que se trata apenas de uma sugestão, pois cada equipe de desenvolvimento pode retirar ou acrescentar itens que sejam adequados ao seu projeto. Portanto, os itens podem ser:

● O *nome* do caso de uso, que deve ser o mesmo usado no diagrama e deve ter um nome único.

● O identificador, que é um código único para cada caso de uso que permite fazer referência cruzada entre diversos documentos relacionados com o modelo de caso de uso. Como convenção de nomenclatura, recomenda-se que se use um prefixo CSU seguido de um número sequencial. Por exemplo: CSU01, CSU02.

● Sumário, que é uma pequena descrição dos objeitvos do ator ao usar o caso de uso.

● Ator primário é quem inicia o caso de uso ou que seja alvo do resultado produzido pelo caso de uso. Um caso de uso possui apenas um ator primário.

● Ator(es) secundário(s) são os nomes dos demais elementos externos participantes do caso de uso, os atores secundários. Um caso de uso possui zero ou mais atores secundários.

● Precondições são aqueles casos de uso cuja realização não faz sentido em qualquer momento, mas, ao contrário, somente quando o sistema está em um determinado estado com certas propriedades.

● Fluxo principal de um caso de uso, por vezes chamado de fluxo básico, corresponde à sua descrição da sequência de passos usual de quando o caso de uso é usado. O texto descritivo desse fluxo (assim como dos fluxos alternativos e de exceção, descritos a seguir) deve ser claro e conciso. Os casos de uso devem ser escritos do ponto de vista do usuário e usando a terminologia deste.

○ Fluxo(s) alternativo(s) podem ser utilizados para descrever o que acontece quando o ator opta por utilizar o caso de uso de uma forma alternativa, diferente da descrita no fluxo principal, para alcançar o seu objetivo. Fluxos alternativos também podem ser utilizados para descrever situações em que há diversas alternativas e somente uma deve ser realizada.

○ Pós-condições em alguns casos, em vez de gerar um resultado observável, o estado do sistema pode mudar após um caso de uso ser realizado. Essa situação é especificada como uma pós-condição e é um estado que o sistema alcança após certo caso de uso ter sido executado. A linguagem é descrita no pretérito.

○ Regras do negócio são políticas, condições ou restrições que devem ser consideradas na execução dos processos existentes em uma organização. Por exemplo, o número máximo de alunos por turma é igual a 30.

## Vamos observar um exemplo?

Pense em um estudo de caso sobre uma universidade que precisa de um sistema que necessite controlar alguns processos acadêmicos, como inscrições em disciplinas, lançamento de notas. Após o levantamento de requisitos iniciais desse sistema, a que chamaremos de Sistema de Controle Acadêmico (SCA), os analistas chegaram à seguinte lista de requisitos funcionais:

**R1.** O sistema deve permitir que alunos visualizem as notas obtidas por semestre letivo.

**R2.** O sistema deve permitir o lançamento das notas das disciplinas lecionadas em um semestre letivo e controlar os prazos e atrasos neste lançamento.

**R3.** O sistema deve manter informações cadastrais sobre disciplinas no currículo escolar. E outros

Identificaram como regras iniciais do negócio:

**RN01**. Quantidade de alunos possíveis: uma oferta de disciplina em uma turma não pode ter mais que 40 alunos inscritos.

Definiram que o sistema se comporia dos seguintes atores:

> *Aluno: indivíduo que está matriculado na faculdade, que tem interesse em se inscrever em disciplinas do curso.*

> *Professor: indivíduo que leciona disciplinas na faculdade. Coordenador: pessoa interessada em agendar as alocações de turmas e professores, e visualizar o andamento de inscrições dos alunos.*
>
> *Departamento de Registro Escolar (DRE): departamento da faculdade interessado em manter informações sobre os alunos matriculados e sobre seu histórico escolar.*
>
> *Sistema de Recursos Humanos: este sistema legado é responsável por fornecer informações cadastrais sobre os professores.*
>
> *Sistema de Faturamento: este sistema legado tem interesse em obter informações sobre inscrições dos alunos para realizar o controle de pagamento de mensalidades (BEZERRA, 2007, p. 93).*

Os analistas identificaram os seguintes casos de uso que foram agrupados nos seguintes grupos:

**1.** *Gerenciamento de Inscrições*

● Realizar Inscrição.

● Cancelar Inscrição – Aluno cancela inscrições em uma ou mais ofertas de disciplinas em que havia solicitado inscrição.

● Visualizar Grade Curricular – Aluno visualiza a grade curricular atual.

● Visualizar Andamento de Inscrições.

● Abrir Turma – Coordenador abre uma turma.

● Fechar Turma – Coordenador fecha uma turma.

● Atender Listas de Espera (BEZERRA, 2007, p. 94).

**2.** *Gerenciamento de Recursos Acadêmicos*

● Manter Grade Curricular – Coordenador define informações sobre uma grade curricular.

● Manter Disciplina.

● Manter Aluno – DRE mantém informações sobre aluno.

● Fornecer Grade de Disponibilidade.

● Fornecer Habilitações – Professor informa as disciplinas da grade curricular que está apto a lecionar.

● Atualizar Informações sobre Professor (BEZERRA, 2007, p. 94).

**3.** *Acompanhamento de Semestre Letivo*

● Lançar Avaliações e Frequências.

● Obter Diário de Classe – Professor obtém o diário de classe para determinado mês do semestre letivo.

● Visualizar Avaliações e Frequências.

● Solicitar Histórico Escolar – Aluno solicita a produção de seu histórico escolar (BEZERRA, 2007, p. 94).

Observe como ficariam os diagramas de caso de uso para o Sistema de Controle Acadêmico.

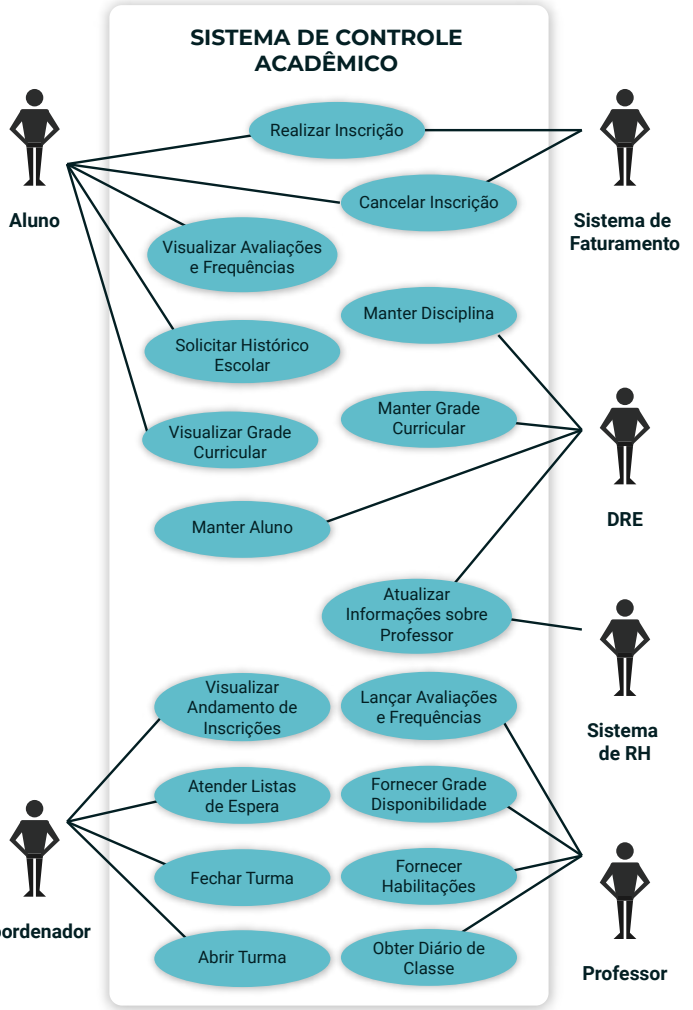


***Figura 14:*** *Diagramas de caso de uso do Sistema de Controle Acadêmico.* ***Fonte:*** *Adaptado Bezerra (2007).*

## REFLITA

Você já imaginou como descreveria os casos de uso da Visualização de Avaliações e Frequência? Como seria o Sumário? Poderíamos escrever o seguinte: aluno visualiza avaliação que recebeu (notas e frequência) nas turmas de um semestre letivo. E quem seria o ator primários e as precondições? Pense sobre isso.

Observe, como exemplo, a descrição de um caso de uso.

| |
|---|
| **Sumário:** Aluno usa o sistema para realizar inscrição em disciplinas.<br>**Ator Primário:** Aluno<br>**Atores Secundários:** Sistema de Faturamento<br>**Precondições:** O Aluno está identificado pelo sistema. |
| **Fluxo Principal** |
| 1. O Aluno solicita a realização de inscrição. |
| 2. O sistema apresenta as disciplinas para as quais o aluno tem pré-requisitos (conforme a RN01), excetuando-se as que este já tenha cursado. |
| 3. Para cada disciplina selecionada, o sistema designa o aluno para uma turma que apresente uma oferta para tal disciplina. |
| 4. O Aluno define a lista a lista de disciplinas que deseja cursar no próximo semestre letivo e as relaciona para inscrição. |
| 5. O sistema informa as turmas para as quais o Aluno foi designado. Para cada turma, o sistema informa o professor, os hotários e os respectivos locais das auls de cada oferta de disciplina. |
| 6. O Aluno confere as informações fornecidas. Aqui é possível que o caso de uso retorne ao passo 3, conforme o Aluno queira revisar (inserir ou remover) a lista de disciplinas a cursar. |
| 7. O sistema registra a inscrição do Aluno, envia os dados sobre a mesma para o Sistema de Faturamento e o caso de uso termina. |

| Fluxo Alternativo (4): Inclusão em lista de espera |
| --- |
| a. Se não há oferta disponível para alguma disciplina selecionada pelo aluno (conforme a RN01), o sistema reporta o fato e e fornece a possibilidade de inserir o Aluno em uma lista de espera. |
| b. Se o Aluno aceitar, o sistema o insere na lista de espera e apresenta a posição na qual o aluno foi inserido na lista. O caso de uso retorna ao passo 4. |
| c. Se o Aluno não aceitar, o caso de uso prossegue a partir do passo 4. Fluxo de Exceção (4): Violação de RN01 |
| **Pós-condições:** O aluno foi inscrito em uma das turmas de cada uma das disciplinas desejadas, ou foi adicionado a uma ou mais listas de espera. |
| **Regras de Negócio:** RN01 |

***Tabela 2:*** *Descrição do caso de uso* ***Realizar Inscrição****, CSU01.* ***Fonte:*** *Adaptado Bezerra (2007, p. 96).*

# CONSIDERAÇÕES FINAIS

Neste módulo, você aprendeu sobre a modelagem de projetos ou sistemas de softwares por meio da *Unified Modeling Language (UML)* que, em língua portuguesa, é chamada de Linguagem de Modelagem Unificada. Estudou os conceitos gerais dessa linguagem padrão para a elaboração e estruturação de sistemas de softwares.

Você aprendeu sobre os principais conceitos do paradigma de objetos: abstração, encapsulamento, polimorfismo e herança; conheceu as principais fases para o processo de desenvolvimento de sistemas, conforme seus problemas e necessidades. Você pôde observar, ainda, o que são os Ciclos de Vida: em cascata, incremental e iterativo, prototipagem, espiral, métodos ágeis e extreme programming. Ademais, você aprendeu quais são os principais diagramas de UML; além de ter conhecido as principais notações gráficas relacionadas aos Diagramas de Casos de Uso, observando modelos e exemplos.

# SÍNTESE

## ANÁLISE E PROJETO DE SISTEMAS

### MODELAGEM DE SISTEMAS DE SOFTWARE COM UML

**1) Paradigma de programação:** um modelo de programação suportado por linguagens que agrupam certas características comuns.

**2) Principais conceitos do paradigma orientado a objetos.**

| Conceito | Descrição |
|---|---|
| Objeto | Qualquer coisa abstrata ou concreta. |
| Classe | Conjunto de atributos e serviços comuns a um grupo de objetos. |
| Abstração | Processo mental em que nos concentramos noo que o objeto é e faz. |
| Encapsulamento | Visualização de aspectos externos de um objeto – sua interface. |
| Polimorfismo | Representa o estado de um objeto ser capaz de assumir várias formas. |
| Herança | Mecanismo que permite que as características comuns a uma mesma classe possam ser generalizadas ou especializaas em uma mesma classe. |
| Composição | Elaboração de outros objetos a partir da reunião de outros objetos. |

**3) UML (*Unifield Modeling Language*):** linguagem visual para modelar sistemas orientados a objetos.

**4) Fases de desenvolvimento de sofwares e sistemas.**

| Fase | Descrição |
|---|---|
| Levantamento de Requisitos ou elicitação de requisitos | Estabelecer quais funcionalidades o sistema a ser desenvolvido deverá ter. |
| Análise de requisitos ou engenharia de requisitos | Conjunto de atributos e serviços comuns a um grupo de objetos. |
| Projeto (Desenho) | Determinará "como" o sistema funcionará para atender aos requisitos. |
| Desenvolvimento ou implementação | Implementar o sistema em uma determinada linguagem de programação. |
| Teste | Verificar se o mesmo está de acordo com requisito que foi utilizado para implementá-lo. |
| Validação | O cliente analise se o produto gerado atende suas necessidades. |
| Implatanção | Disponibilizar o uso do sistema no ambiente de sua operação. |

**5) Modelos de Ciclo de Vida** é o encadeamento das fases de elaboração de um sistema.

**6) Diagramas de casos de uso e suas representações**

- **Atores** representam papéis desempenhados por usuários ou qualquer entidade externa ao sistema; iniciam as sequências nos casos de uso.
- **Casos de uso** – expressam uma funcionalidade que o sistema deve conter.
- **Inclusão** – quando um caso de uso base incorpora ou inclui um ou mais casos de uso e o faz de modo obrigatório.
- **Extensão** – quando um caso de uso base se estende em um ou mais casos de uso e o utiliza de maneira opcional.
- **Generalização ou Especialização (especificação)** – ocorre quando as entidades devem ser do mesmo tipo, ou seja, dois casos de uso ou dois atores.

# Referências Bibliográficas & Consultadas


BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Elsevier, 2007.

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

GONÇALVES, E. J. T. **Análise e projeto de sistemas**. 3. ed. Fortaleza: EdUECE, 2015.

GUEDES, G. T. A. **UML 2**: uma abordagem prática. 2. ed. São Paulo: Novatec, 2011.

LARMAN, C. **Utilizando UML e padrões**. 3. ed. Porto Alegre: Bookman, 2004. [Minha Biblioteca]

MARINHO, A. L. **Análise e modelagem de sistemas**. São Paulo: Pearson Education do Brasil, 2016. [Biblioteca Virtual]

MEDEIROS, E. **Desenvolvendo software com UML 2.0**: definitivo. São Paulo: Pearson Makron Books, 2004. [Biblioteca Virtual]

PAGE-JONES, M. **Fundamentos do desenho orientado a objeto com UML**. 1. ed. São Paulo: Makron Books, 2001. [Biblioteca Virtual]

PAULA FILHO, W. P. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. Rio de Janeiro: LTC, 2009. Disponível em: [Minha Biblioteca]

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

SOMMERVILLE, I. **Engenharia de software**. 10. ed. São Paulo: Pearson Brasil, 2019. [Biblioteca Virtual]

FAM
ONLINE