



# ANÁLISE E PROJETO DE SISTEMAS E INFORMAÇÃO I



# **ANÁLISE E PROJETO DE SISTEMAS E INFORMAÇÃO I**

**Copyright © UVA 2019**

Nenhuma parte desta publicação pode ser reproduzida por qualquer meio sem a prévia autorização desta instituição.

Texto de acordo com as normas do Novo Acordo Ortográfico da Língua Portuguesa.

**AUTORIA DO CONTEÚDO**

Claudio Ribeiro da Silva

**PROJETO GRÁFICO**

UVA

**REVISÃO**

Lydianna Lima

Janaina Senna

Theo Cavalcanti

Caroline Maganhi

**DIAGRAMAÇÃO**

UVA

S586

Silva, Claudio Ribeiro da.

Análise e projetos de sistemas de informação I [livro eletrônico] / Claudio Ribeiro da Silva. – Rio de Janeiro: UVA, 2020.

3,95 MB : PDF.

ISBN 978-65-5700-044-1

1. Sistemas de Informação Gerencial. 2. Projeto de Sistemas. 3. Casos de Uso (Engenharia de Sistemas). 4. Métodos Orientados a Objetos (Computação). 5. UML (Computação). I. Universidade Veiga de Almeida. II. Título.

CDD – 658.4038  
BN

# SUMÁRIO

|                     |          |
|---------------------|----------|
| <b>Apresentação</b> | <b>6</b> |
|---------------------|----------|

|              |          |
|--------------|----------|
| <b>Autor</b> | <b>7</b> |
|--------------|----------|

## UNIDADE 1

|  |          |
|--|----------|
| <b>Fundamentos da orientação a objetos</b> | <b>8</b> |
|--|----------|

- Conceitos da orientação a objetos
- A linguagem UML 16
- Os diagramas da linguagem UML

## UNIDADE 2

|                                 |           |
|---------------------------------|-----------|
| <b>Modelagem de caso de uso</b> | <b>29</b> |
|---------------------------------|-----------|

- Transição dos requisitos funcionais para caso de usos
- O Diagrama de Caso de Uso
- A descrição dos casos de usos

# SUMÁRIO

## UNIDADE 3

### **Modelagem de classe de domínio** **51**

- O diagrama de classes
- O relacionamento entre as classes
- O Dicionário de Classes

## UNIDADE 4

### **Diagramas da UML para a construção do modelo conceitual do sistema** **79**

- O Diagrama de Sequência
- O Diagrama de Transição de Estado
- O Diagrama de Atividade

# APRESENTAÇÃO

Desenvolvimento de software representa a capacidade que um profissional da tecnologia da informação – TI possui em transformar os desejos e necessidades do usuário ou cliente em um sistema de informação – SI que atenda a essas necessidades. Construir um SI demanda disciplina e controle, pois o tempo existente entre a identificação dos requisitos a serem implementados e a conclusão da sua implantação pode fazer com que algumas das necessidades identificadas não sejam implementadas e outras sejam desenvolvidas sem que tenham sido solicitadas. Esta disciplina tem como objetivo oferecer o material necessário para a construção de SIs mediante o uso dos conceitos da orientação a objetos utilizando a linguagem UML (unified modeling language) de forma a evitar que tais situações ocorram.

A rotatividade dos profissionais existente nas empresas faz com que eles estejam constantemente mudando de trabalho e atividades; dessa forma, o uso de uma linguagem de modelagem permite criar um padrão de documentação que facilita o entendimento da análise e projeto de um SI mesmo que outros o tenham iniciado. Um padrão gráfico de representação faz com que profissionais que tenham esse conhecimento possam dar continuidade a projetos de desenvolvimento de SI em qualquer lugar que utilize esse padrão.

Por intermédio de alguns dos diagramas da UML, apresentados na disciplina, é possível conhecer o “passo a passo” na construção de um SI de forma organizada por meio de um encadeamento entre eles, sendo apresentados cinco dos principais diagramas propostos pela linguagem. Aliado a eles serão apresentados os conceitos da orientação a objetos, que tem sido, nos dias de hoje, a principal metodologia utilizada nas principais linguagens de programação, o que faz com que haja um padrão de conceitos em todas as etapas do ciclo de desenvolvimento de software.

Dessa forma, nesta disciplina, será possível conhecer as atividades realizadas por um profissional de TI responsável pelo desenvolvimento de um SI desde o recebimento dos requisitos funcionais identificados na fase de levantamento até o início da fase de implementação do software.

## CLAUDIO RIBEIRO DA SILVA

Claudio Ribeiro da Silva é doutor em Engenharia de Sistemas e Computação pelo Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia da Universidade Federal do Rio de Janeiro – Coppe/UFRJ, mestre em Ciência da Informação pela UFRJ, pós-graduado em Análise e Desenvolvimento de Sistemas pela Pontifícia Universidade Católica do RJ – PUC-RJ e graduado em Ciências Contábeis pela Faculdade de Administração São Paulo Apóstolo – Faspa, cuja mantenedora era a Sociedade Educacional São Paulo Apóstolo – Sepa. É professor dos cursos de graduação nas modalidades presencial e a distância na área de Gestão de Tecnologia da Informação, Sistema da Informação e Análise e Desenvolvimento de Sistemas. Atuou por mais de 30 anos como tecnologista da informação em autarquia federal de médio porte realizando atividades de desenvolvimento de software e gerenciamento de projetos de desenvolvimento de software.



**C. Lattes**

# UNIDADE 1

Fundamentos da  
orientação a objetos



# INTRODUÇÃO

Para o desenvolvimento de um SI é necessária a aplicação de um conjunto de técnicas e metodologias que irão permitir que a documentação produzida ao longo desse processo possa ser compartilhada e entendida por todos que dela necessitem. Exercer esse tipo de atividade em qualquer uma das suas etapas, nos dias de hoje, requer compartilhar conhecimentos com a equipe da qual participa ou ter de assumir responsabilidades de outros que não fazem mais parte dela. Esse compartilhamento de informações com pessoas próximas ou não somente é possível por meio de um padrão de documentação e confecção de diagramas que permitirão o entendimento de forma clara e objetiva do que está sendo produzido.

Nesta unidade você irá identificar os fundamentos da orientação a objetos aplicados à linguagem UML, para que possa estar familiarizado com o padrão internacional de produção da documentação utilizada para o desenvolvimento de um SI.



## OBJETIVO

Nesta unidade você será capaz de:

- Conhecer os conceitos da orientação a objetos aplicados aos diagramas propostos pela UML para a documentação das etapas do desenvolvimento de um sistema.

# Conceitos da orientação a objetos

Segundo Bezerra:

[...] o paradigma da orientação a objetos visualiza um sistema de software como uma coleção de agentes interconectados chamados “objetos”, sendo que cada objeto é responsável por realizar tarefas específicas. Através da interação entre objetos uma tarefa computacional é realizada. (BEZERRA, 2007)

## **Diante dessa definição, surge a primeira pergunta: o que é objeto?**

Como resposta, podemos afirmar: objeto é tudo o que a aplicação manipula, ou seja, qualquer coisa é um objeto.

O principal conceito da orientação a objetos faz com que cada objeto tenha “vida”. Dessa forma, podemos destacar alguns fatores como, por exemplo, o fato de os objetos se comunicarem com outros objetos realizando tarefas por meio de requisição de serviço ou eventos.



### **Exemplo**

Como exemplo, podemos citar a necessidade de saber a que curso o aluno Carlos pertence. Dessa forma, o objeto Carlos se comunica com um dos objetos que representa o curso ao qual ele pertence.

Observe que no exemplo acima falamos de “aluno” e “curso”, que representam um conjunto de todos os alunos e cursos existentes, conhecidos como “classe”. Portanto, uma classe representa um conjunto de objetos que possuem as mesmas características e comportamentos. Na orientação a objetos, a comunicação entre os objetos, chamada de “evento”, realiza a interligação entre classes, permitindo o encadeamento, fator fundamental no desenvolvimento de um sistema.

Vamos exemplificar um pouco mais:



### Exemplo

- A classe Aluno é composta por todos os alunos cadastrados, sendo cada um deles um objeto.
- A classe Curso é composta por todos os cursos cadastrados, sendo cada um deles um objeto.
- A comunicação (evento) entre um objeto da classe Aluno (um aluno específico) e um objeto da classe Curso (um curso específico) representa a comunicação entre essas duas classes, sendo que, nesse contexto, representa o curso no qual o aluno está inscrito.

Figura 1: Comunicação entre objetos de duas classes.



Fonte: Elaborada pelo autor (2019).

Analisando o exemplo da Figura 1, observamos:

- A classe Aluno possui quatro objetos (João, Maria, Rosa e Pedro).
- A classe Curso possui dois objetos (Análise de Sistemas e Sistemas de Informação).
- Cada objeto de aluno comunica-se com um objeto de curso, sendo possível saber que os objetos João e Maria pertencem ao curso de Análise de Sistemas, e Rosa e Pedro, ao curso de Sistemas de Informação.

Cada classe é formada por características e comportamento. As características representam as informações que pertencem a ela, ou seja, os seus atributos. Algumas características que podemos atribuir à classe Aluno são os dados que pertencem a cada objeto (por exemplo, o nome do aluno, sua data de nascimento, estado civil etc.). Cada objeto tem um conjunto de características ou atributos.

As classes também possuem comportamento, conhecidos como métodos. Métodos são as ações realizadas pelas classes. No mundo real, podemos atribuir aos alunos os comportamentos de falar, andar, comer etc. No entanto, no ambiente tecnológico, métodos são representados por pequenos procedimentos a serem desenvolvidos que irão realizar ações sobre os objetos.



### Exemplo

Por exemplo, consultar um objeto ou um conjunto deles, criar um objeto e atualizar alguns dos seus atributos são alguns dos exemplos de métodos aplicados a uma classe em um SI.

## Tipos de classes

As classes podem ser classificadas de diferentes formas:

- **Classes de entidades ou classes de negócio** – Modelam a informação persistente, sendo tipicamente independente da aplicação.
  - São as classes candidatas mais fortes a serem reutilizadas.
  - Geralmente são necessárias para cumprir alguma responsabilidade do software.
  - Normalmente estão associadas às entidades de banco de dados pela necessidade em armazenar o conteúdo dos seus atributos.
  - Exemplos: Cliente, Aluno, Professor, Fornecedor etc.
- **Classes de interface** – Tratam da comunicação com o ambiente do produto.
  - Modelam as interfaces do produto com usuários e outros sistemas.
  - Durante a análise de domínio, considera-se que as classes de fronteira surgem de cada par ator-caso de uso.

- **Classes de controle** – Coordenam o fluxo de um caso de uso complexo, encapsulando lógica que não se enquadre naturalmente nas responsabilidades das entidades.
  - São tipicamente dependentes da aplicação.
  - Controlam o sequenciamento do comportamento do caso de uso.
  - Por exemplo, podem-se definir classes de controle para coordenar um subfluxo ou um fluxo alternativo mais complexo, um algoritmo, a geração de um relatório ou uma regra de negócio complexa.
  
- **Classes abstratas x classes concretas** – Na hierarquia de classes é comum especificar que certas classes são abstratas, significando que não podem apresentar instâncias diretas.
  - Classe abstrata:
    - > As classes abstratas são constituídas mediante a união de conceitos genéricos, que serão adequados ao domínio da aplicação por meio apenas de suas subclasses.
    - > Servem como modelo para outras classes que dela herdem, não podendo ser instanciadas.
    - > Para ter o seu objeto é necessário criar uma classe mais especializada herdando dela para que a nova classe seja instanciada.
    - > Exemplo: é definido que a classe Animal seja herdada pelas subclasses Gato, Cachorro, Cavalo etc., mas ela mesma nunca será instanciada.
  - Classe concreta:
    - > Por contraste, uma classe concreta é aquela que pode ser instanciada.
    - > Representa as classes de negócio, em sua maioria.

## Outros conceitos da orientação a objetos

Vimos até o momento, de forma detalhada, dois importantes conceitos: classes e objetos; entretanto, vários outros são aplicados à orientação a objetos. Muitos desses conceitos são utilizados também na programação orientada a objetos – POO, que representa o desenvolvimento do software em uma linguagem de programação que utiliza esses conceitos.



### Importante

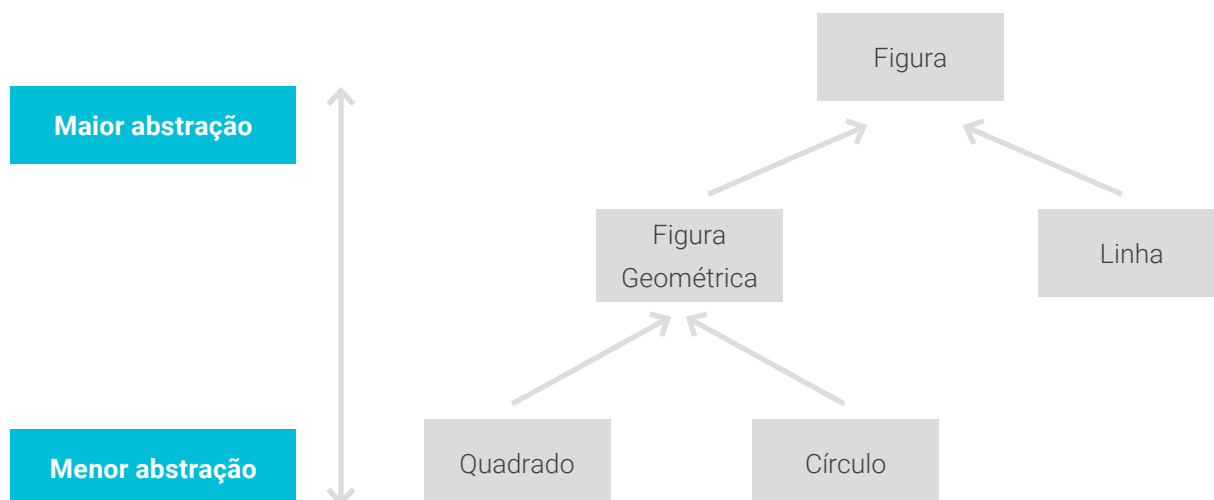
A definição dos conceitos da orientação a objetos não se altera tanto na fase de implementação do sistema utilizado na POO quanto o uso na análise orientada a objetos. Os conceitos devem ser identificados na fase de análise para que possam ser implementados posteriormente, quando o sistema estiver na fase de desenvolvimento.

Além das definições de classes, objetos, atributos e comportamentos, outros conceitos são aplicados à orientação a objetos, tais como:

**Instância** – Representado por um elemento (objeto) da classe. A partir dos atributos e métodos definidos para uma classe, é possível criar objetos com essa estrutura, sendo eles identificados como uma instância. Os conceitos objeto e instância se confundem, no entanto não existe consenso de que eles representam exatamente a mesma situação.

**Herança** – O conceito de herança é aplicado sempre que uma classe “herda” atributos ou métodos de outras classes. Esse conceito pode ser aplicado quando duas classes possuem características ou comportamentos semelhantes, ou seja, as informações comuns a um conjunto de objeto podem ser abstraídas em uma classe, evitando, dessa forma, a duplicação dessas informações. Esse conceito é bastante comum tanto nas etapas de análise quanto na implementação de um sistema.

Figura 2: Representação de herança.



Fonte: Elaborada pelo autor (2019).

**Encapsulamento** – Esse conceito vem do verbo “encapsular”, que representa detalhar o máximo possível os métodos de uma classe, permitindo que eles possam ser utilizados de forma independente.

Como exemplo, imagine que você queira apenas fazer uma consulta, obtendo os dados de um objeto. Para isso, basta criar um método que tenha apenas esse objetivo, fazendo com que ele seja simples e “leve”. Não é necessário saber de que forma ele obtém esses dados, é importante apenas saber que, se você der o número da matrícula de um aluno, você irá obter os seus dados. Por meio do encapsulamento, a única coisa que um objeto precisa saber para pedir a colaboração a outro objeto é conhecer a sua interface. No nosso dia a dia, você sabe o que ocorre internamente na televisão ou no controle remoto quando você aperta o botão de liga/desliga? Seria necessário ter esse conhecimento para ligar ou desligar uma TV? Basta apenas saber que, se apertar aquele botão, a TV irá ligar, se estiver desligada, ou desligar, caso esteja ligada.

**Polimorfismo** – Significa “várias formas”. Representa a capacidade de abstrair diferentes implantações em uma única interface. Como exemplo, um controle remoto que aciona vários equipamentos. É possível realizar o envio da mesma mensagem para objetos semelhantes, mas que implementem a interface de diferentes formas. Na orientação a objetos, é definido como sendo um código que possui “vários comportamentos” ou produz “vários comportamentos”.

# A linguagem UML

A linguagem UML (do inglês, *unified modeling language*, traduzida como “linguagem unificada de modelagem”) representa uma linguagem padrão utilizada para modelagem de sistemas que utiliza o modelo de orientação a objetos. Ela representa uma linguagem de notação, ou seja, uma representação gráfica utilizada na análise e projeto de sistemas de informação, que representa, por meio dos diagramas e elementos propostos, a relação existente entre os objetos que compõem o sistema.

O objetivo de utilizar representações gráficas é permitir o entendimento por meio de notações previamente conhecidas e aceitas internacionalmente. Dessa forma, situações como a rotatividade de profissionais, muito comum nas empresas comerciais, têm o impacto minimizado, caso substituições de profissionais sejam necessárias durante o projeto de desenvolvimento de um sistema.

Um dos problemas existentes no desenvolvimento de sistemas nas metodologias que não utilizam representação gráfica em todas as suas fases é o entendimento. Por esse motivo era comum que um profissional de TI realizasse um entendimento dos requisitos de um sistema e, ao repassá-los, o entendimento fosse equivocado por não ter sido expresso de forma clara. Com o uso de diagramas, a forma de representação se torna mais objetiva, eliminando eventuais dúvidas que surjam.



Dessa forma, a UML é considerada uma linguagem formada por elementos gráficos que permite a modelagem de sistemas utilizando o paradigma da orientação a objetos em sistemas concorrentes e distribuídos.

Figura 3: Profissional de TI expressando os requisitos na forma não gráfica.



## A história da UML

A UML surgiu da união de três metodologias existentes na época: BLOOCH (Rational), OMIT (Rumbaugh) e OOSE (Jacobson), a partir de 1994.

A evolução da linguagem pode ser descrita no quadro abaixo:

Tabela 1: Evolução da UML.

|                                   |  |
|-----------------------------------|--|
| <b>Outubro/1995</b>               | Versão 0.8 do Unified Process (Processo Unificado) por meio da unificação das metodologias da Rational e Rumbaugh. |
| <b>Junho/1996</b>                 | Versão 0.9 da UML com a incorporação da metodologia OOSE.  |
| <b>Janeiro/1997</b>               | Aprovada pelo Object Management Group – OMG como padrão de orientação a objetos, sendo criada a versão 1.0.        |
| <b>Novembro/1997 a março/2003</b> | Desenvolvimento das versões 1.1 a 1.5.   |
| <b>Julho/2005 a agosto/2011</b>   | Desenvolvimento das versões 2.0 a 2.4.1.   |
| <b>Maio/2015</b>                  | Desenvolvimento da versão 2.5.   |
| <b>Dezembro/2017</b>              | Desenvolvimento da versão 2.5.1 (atual em setembro/2019).  |

Fonte: [projetodiario.net.br](http://projetodiario.net.br) e [omg.org](http://omg.org). Adaptado.

A cada versão lançada, novas funcionalidades são acrescentadas com o objetivo de atender às necessidades de negócios das empresas nos dias de hoje, em termos de comportamento e estrutura dos sistemas de informação.



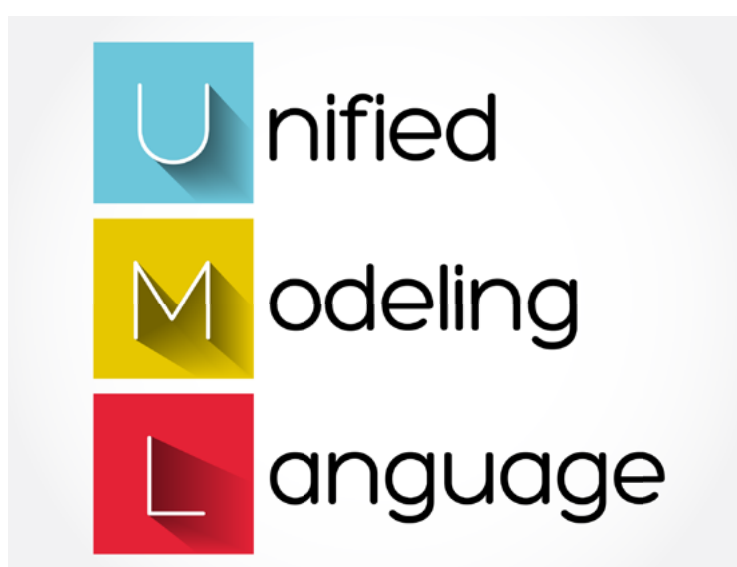
### Curiosidade

Pesquise no endereço <https://www.mindomo.com/pt/mindmap/versoes-uml-164c4e92e9c8458ca29cd8e569b1511f> e conheça as mudanças introduzidas em cada uma das versões desenvolvidas até a 2.4.1.

Segundo Ventura (2019), a UML “ajuda a deixar o escopo claro, pois centraliza em uma única visão (o diagrama) um determinado conceito, utilizando uma linguagem que todos os envolvidos no projeto podem facilmente entender”, no entanto ela deve ser utilizada apenas quando é necessária para facilitar a comunicação entre as partes envolvidas no projeto.

Dessa forma, a linguagem pode ser considerada como sendo universal para os profissionais que atuam na área de desenvolvimento de sistema, por criar um padrão simples de entendimento por todos os diferentes tipos de perfil profissional envolvidos nas diferentes etapas do desenvolvimento.

Figura 4: *Unified modeling language – UML.*



## Vantagens e desvantagens da orientação a objetos e UML

Uma das principais vantagens da orientação a objetos está associada à reutilização de código.

### O que significa isso?

Se você trabalha em uma empresa que iniciou agora o uso desse conceito, essa vantagem inicialmente não será observada, pois será necessário desenvolver tudo, no entanto, conforme os sistemas forem sendo criados, algumas classes passam a ficar disponíveis para serem utilizadas por outros sistemas.



## Exemplo

Por exemplo, vamos imaginar uma universidade que quer reescrever todos os seus sistemas passando a utilizar a orientação a objetos. Quando o primeiro sistema que utiliza a classe Aluno for criado, nele serão construídos e testados alguns métodos, como os de consulta e atualização, por exemplo. Quando um segundo sistema que utiliza essa classe estiver em desenvolvimento, a classe de consulta aos dados do aluno não será desenvolvida, pois já está pronta, bastando apenas conhecer a sua interface. Com o passar do tempo, várias classes passam a ser criadas, reduzindo a necessidade de desenvolvê-las e, como consequência, o tempo para esse desenvolvimento.

Outra vantagem é a facilidade de entendimento a partir dos diagramas da UML, sendo uma importante ferramenta de comunicação com o usuário e documentação para o desenvolvedor.

Considerando que as linguagens de programação, na sua maioria, utilizam o conceito da programação orientada a objetos, passa a haver uma integração entre todas as fases de desenvolvimento, permitindo maior controle sobre as funcionalidades a serem criadas.

Uma das desvantagens no uso da UML é a dificuldade em manter os diagramas atualizados em caso de alteração. Essa situação é preocupante, uma vez que a dinâmica de uma alteração é rápida, nem sempre dispondo do tempo necessário para realizar essa atualização. Para solucionar esse problema, a empresa deve dispor do tempo, ou equipe, necessário para atualizar, mesmo que após a sua implantação.

Outro problema existente atualmente está associado ao uso de sistemas gerenciadores de banco de dados – SGBDs no modelo relacional na maioria das empresas. Esse tipo de modelo não implementa alguns dos recursos disponíveis na orientação a objetos, sendo necessário realizar a transformação de um modelo para outro, alterando alguns dos conceitos utilizados na orientação a objetos. A solução para essa situação seria o uso de SGBDs orientados a objetos por essas empresas, no entanto é necessário elevado investimento, adequando os dados e acessos a eles.

## Comparando a orientação a objetos com a análise estruturada

A análise estruturada é uma metodologia de desenvolvimento bastante utilizada na década de 1990, sendo bastante usada em um período no qual a UML ainda não havia sido desenvolvida e, posteriormente, ainda não estava sendo empregada de forma mais universalizada.

Essa metodologia tinha como base os processos desenvolvidos pela aplicação, ou seja, o que o sistema precisa fazer, no qual um dos seus diagramas apresenta o fluxo com que as informações transitavam entre os procedimentos existentes no processo, objeto do desenvolvimento.

Tabela 2: Algumas diferenças entre a análise estruturada e a orientação a objetos.

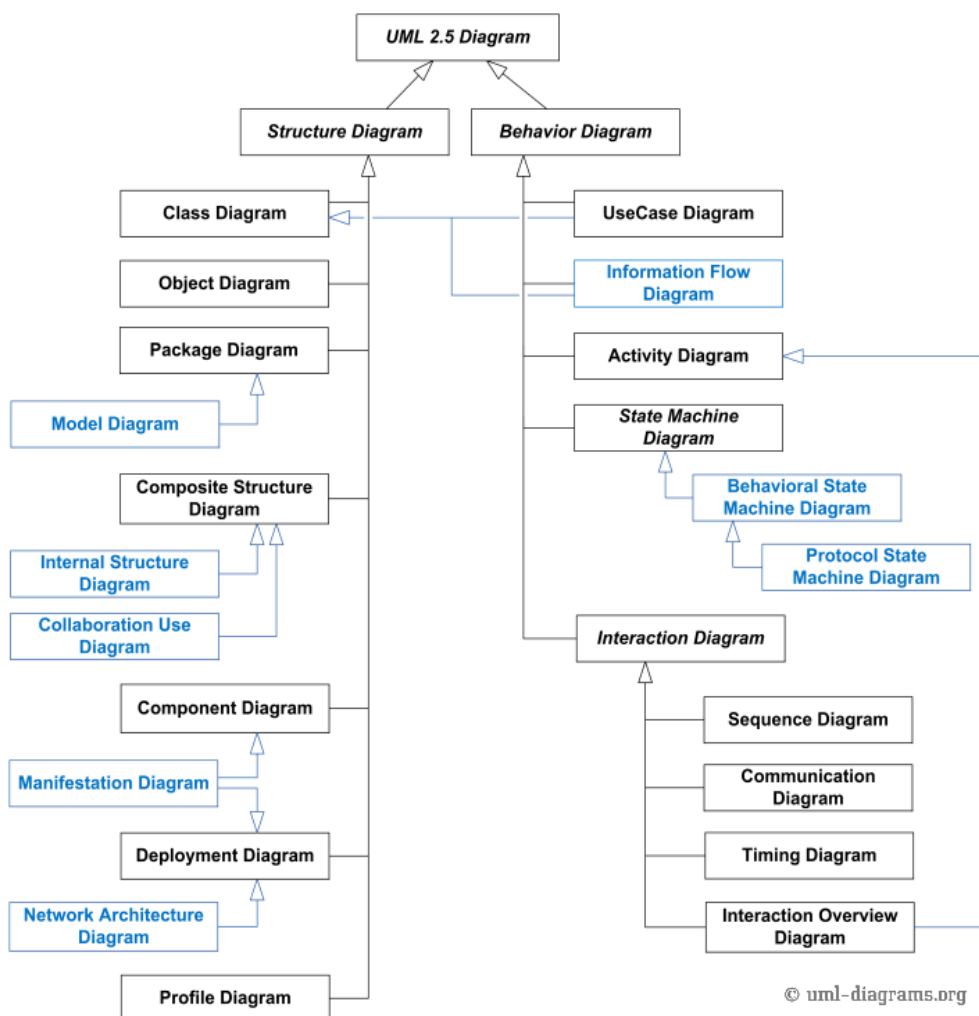
| DESCRIÇÃO   | ESTRUTURADA   | ORIENTAÇÃO A OBJETO                   |
|---|---|---------------------------------------|
| Ótica do sistema.                                 | Processos.  | Classes.                              |
| Disponibilização de rotinas e funções do sistema. | Espalhada pelos programas que a utilizam.                         | Concentrada na classe correspondente. |
| Reúso de código.                                  | As rotinas são duplicadas.  | Não existe duplicação.                |
| Exemplo: Sistema Acadêmico.                       | Funções: Cadastrar disciplina, Matricular aluno, Consultar notas. | Classes: Disciplina, Aluno, Nota.     |

Observando a Tabela 2, verificamos duas grandes diferenças: processos x classes e reúso de código. Essas diferenças permitem que a orientação a objetos tenha agilidade no desenvolvimento de um software, sendo essa uma das suas principais vantagens.

# Os diagramas da linguagem UML

A definição de nomes e padrões nos diagramas da UML vem sofrendo alterações ao longo do tempo, conforme observamos em cada versão. Na versão 2.5, esses diagramas foram divididos em dois grupos: diagramas estruturais e diagramas comportamentais.

Figura 5: Estrutura de diagramas da UML versão 2.5.



Fonte: [uml-diagrams.org](http://uml-diagrams.org).



## Observação

Os itens em azul na Figura 5 não fazem parte da taxonomia oficial da UML 2.5.

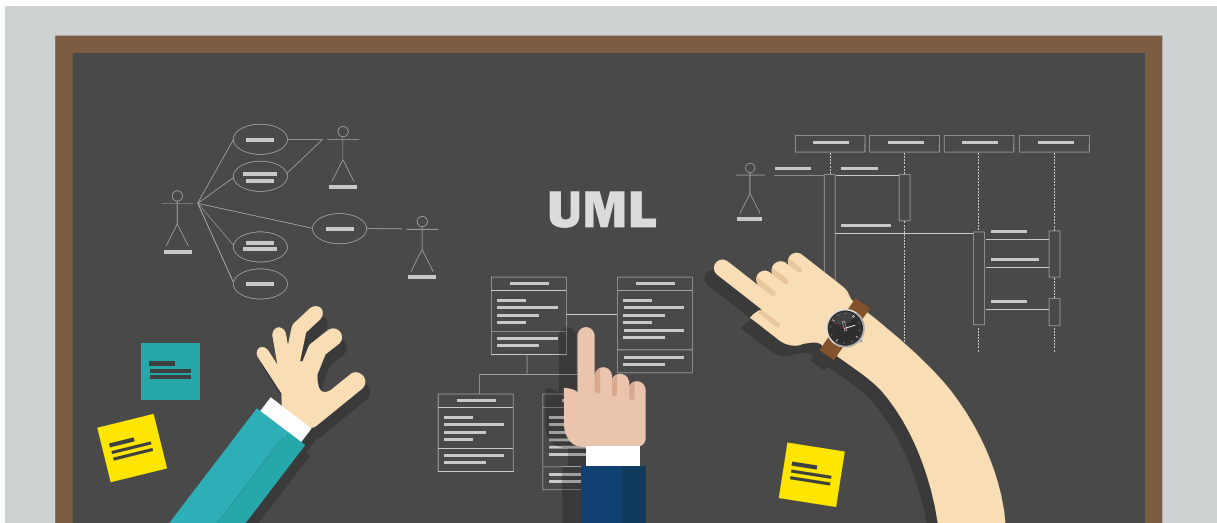
## Diagramas estruturais

Os diagramas estruturais representam a estrutura estática dos componentes do sistema nos diferentes níveis de abstração e implementação e a relação entre eles. Esses componentes representam os principais conceitos do sistema, incluindo as abstrações, os conceitos do mundo real e de implementação e a relação entre objetos. Esses diagramas não representam conceitos relacionados a tempo e comportamentos dinâmicos.

Fazem parte desse grupo os seguintes diagramas:

- **Diagrama de classe** – Representa o relacionamento entre as classes de negócio do sistema e interfaces. Nas classes, são definidos os seus atributos (propriedades) e métodos (comportamento).
- **Diagrama de objeto** – Representa o relacionamento de uma instância de uma classe, devendo ser utilizado sempre que houver a necessidade de especificar um conteúdo específico de um objeto.
- **Diagrama de pacotes** – Representa a reorganização de componentes em estruturas maiores, apresentando o relacionamento entre elas.
- **Diagrama de componentes** – Realiza a modelagem do sistema em termos de componentes e seus relacionamentos por meio de interfaces.
- **Diagrama de implantação** – Apresenta a arquitetura do sistema, representando os diversos softwares utilizados e os locais nos quais estão instalados.
- **Diagrama de estrutura composta** – Representa a colaboração entre os diversos componentes do sistema de forma integrada.
- **Diagrama de perfil** – Define tipos padronizados de estereótipos ou metamodelos que representam valores ou restrições.

Figura 6: Integração entre os diagramas da UML.



### Importante

A classificação dos diagramas não está relacionada à etapa do desenvolvimento de um sistema em que são construídos. A maioria deles está presente na etapa de projeto, tanto no aspecto estrutural quanto no comportamental.

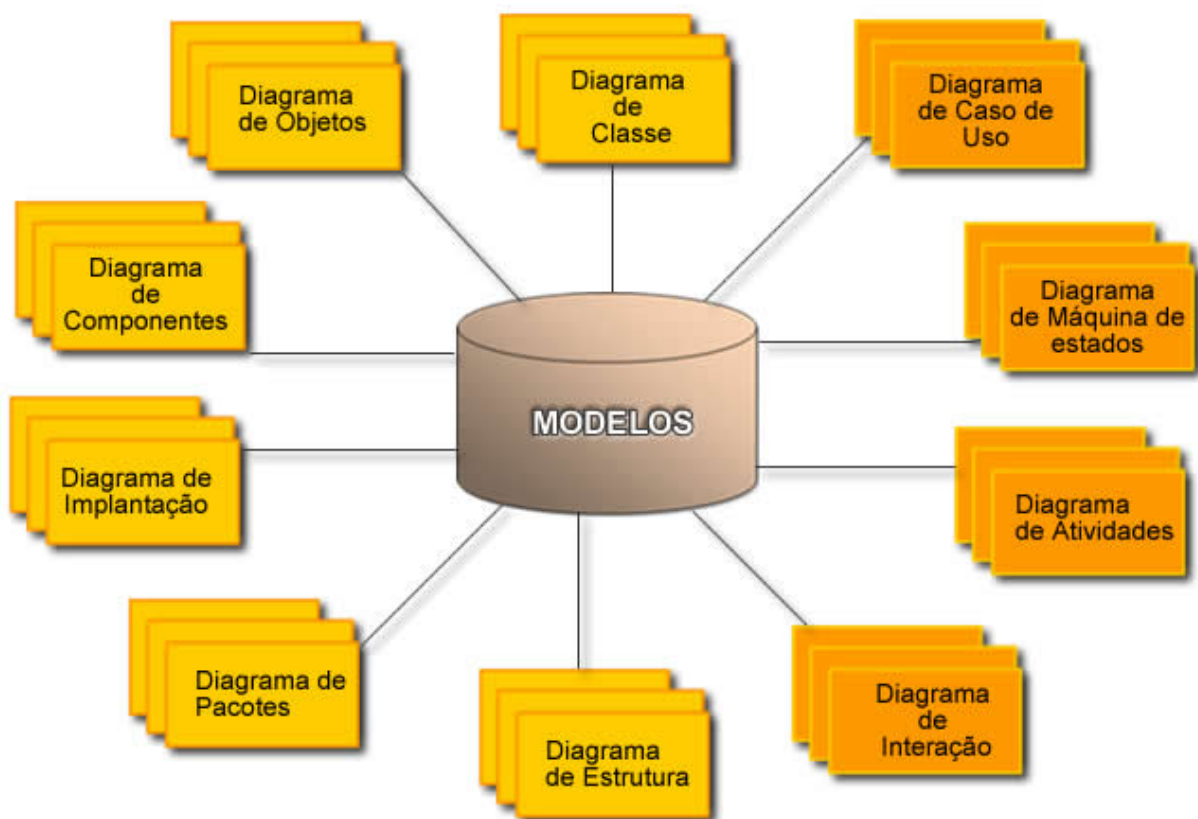
## Diagramas comportamentais

Os diagramas comportamentais ou dinâmicos representam a estrutura dinâmica dos componentes do sistema, representando as mudanças que eles sofrem ao longo do tempo. Fazem parte desse grupo os seguintes diagramas:

- **Diagrama de caso de uso** – Apresenta os requisitos funcionais do sistema sob a forma de casos de uso e a interatividade com os respectivos atores.
- **Diagrama de atividade** – Representa a forma com que uma atividade, representada por um processo de negócio ou caso de uso, é realizada por meio da sua lógica e sequência de execução.
- **Diagrama de estado** – Representa os diferentes estados que a instância de uma classe pode passar e os eventos que provocam as diversas mudanças.

- **Diagrama de interação** – É representado por um conjunto de quatro diagramas que descrevem como um grupo de objetos se relacionam (colaboram) em um determinado comportamento apresentado dentro de um caso de uso. Compõem esse conjunto os seguintes diagramas:
  - **Diagrama de sequência** – Descreve a sequência com que as mensagens são trocadas entre os objetos por meio de eventos temporais.
  - **Diagrama de comunicação** – Descreve as trocas de mensagens entre os objetos, sem a representação temporal. Complementa o diagrama de sequência.
  - **Diagrama de tempo** – Descreve as mudanças de estado da instância de uma classe ao longo do tempo.
  - **Diagrama de interação geral** – Integra os diagramas de sequência e atividade, representando a visão global de interação entre os objetos.

Figura 7: Composição dos diagramas da UML.



Fonte: [infoescola.com](http://infoescola.com).





## MIDIATECA

Para ampliar o seu conhecimento, veja o material complementar da Unidade 1 disponível na midiateca.



## NA PRÁTICA

Na realidade empresarial a rotatividade de colaboradores é constante por diversos motivos. Independentemente desses motivos, a necessidade de continuidade dos projetos de desenvolvimento de sistemas faz com que haja sempre a necessidade de substituição em meio a esses desenvolvimentos. A existência de um padrão de documentação faz com que essa substituição ocorra de forma menos prejudicial, pois o entendimento do que já foi produzido ocorre a todos aqueles que conhecem os conceitos da orientação a objetos e os diagramas apresentados pela UML.

# Resumo da Unidade 1

Nesta unidade você estudou os principais conceitos da orientação a objetos, a linguagem UML a partir da sua evolução e a sua importância no processo de desenvolvimento de sistemas. Estudou também os diagramas propostos pela linguagem para facilitar a comunicação entre a equipe de desenvolvimento e demais partes interessadas no sistema.

As empresas comerciais possuem rotatividade nos profissionais de TI nas suas diversas especialidades. A área de desenvolvimento de sistemas não é diferente. O conhecimento que um profissional adquire ao iniciar o projeto de um novo sistema pode ser perdido caso ele seja substituído e não tenha feito os registros do que foi realizado de forma clara para quem o substitua consiga entender a fim de dar continuidade a partir do ponto que estava. O objetivo da UML é permitir que essa comunicação entre profissionais, que muitas vezes nunca se viram, seja feita de forma tranquila, permitindo haver continuidade no projeto por meio de uma linguagem padronizada, para que qualquer pessoa que conheça os conceitos possa entender exatamente o que foi feito a fim de dar continuidade sem perda de tempo ou retrabalho. A facilidade de integração entre as etapas de construção de um sistema representa outro fator importante no uso dessa linguagem, pois evita inconsistências entre as suas diversas etapas de desenvolvimento.

Por esses motivos, o uso da UML é atualmente bastante difundido, assim como os conceitos da orientação a objetos, ainda mais por permitirem a integração com as principais linguagens de programação existentes no mercado. A partir da próxima unidade, você irá conhecer de forma mais detalhada alguns desses diagramas para que possa entrar no mundo da orientação a objetos.



## CONCEITO

Nesta unidade foram estudados os seguintes conceitos importantes:

- Os conceitos da orientação a objetos utilizados no desenvolvimento de um sistema.
- A evolução da UML desde o seu surgimento até a versão atual.
- As vantagens e desvantagens da orientação a objetos e UML.
- Os diagramas da versão atual da UML e os objetivos de cada um deles.
- A subdivisão da UML nos modelos comportamentais e estruturais utilizados nas diferentes etapas do ciclo de desenvolvimento de um sistema.

# Referências

BARCELAR, R. R. Modelagem de sistemas orientada a objetos com UML. *In*: \_\_\_\_\_. Engenharia de software. **Ricardo Barcelar**, [s. l.], 18 nov. 2014. Disponível em: [http://www.ricardobarcelar.com.br/aulas/eng\\_sw/mod3-uml.pdf](http://www.ricardobarcelar.com.br/aulas/eng_sw/mod3-uml.pdf). Acesso em: 25 set. 2019.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 2. ed. Rio de Janeiro: Elsevier, 2007.

FELIZARDO, J. H. L. F. História da UML. **Projeto Diário**, São Paulo, 25 jul. 2013. Disponível em: <https://www.projetodiario.net.br/historia-da-uml>. Acesso em: 25 set. 2019.

LEMOS, H. D. Encapsulamento, polimorfismo, herança em Java. **DevMedia**, Goiânia, 30 maio 2009. Disponível em: <https://www.devmedia.com.br/encapsulamento-polimorfismo-heranca-em-java/12991>. Acesso em: 5 set. 2019.

MARTINEZ, M. UML. **InfoEscola**, [s. l.], 19 mar. 2010. Disponível em: <https://www.infoescola.com/engenharia-de-software/uml/>. Acesso em: 18 set. 2019.

O QUE é um diagrama UML? **Lucidchart**, [s. l.], c2019. Disponível em: <https://www.lucidchart.com/pages/pt/o-que-e-uml>. Acesso em: 25 set. 2019.

SERMENHO, R. Conceitos básicos de orientação a objetos: parte 1. **Medium**, [s. l.], 1º maio 2016. Disponível em: <https://medium.com/gdgcampinas/conceitos-b%C3%A1sicos-de-orienta%C3%A7%C3%A3o-a-objetos-b58809b2d809>. Acesso em set. 2019.

VENTURA, P. O que é UML (Unified Modeling Language). **Até o Momento**, Belo Horizonte, 31 jan. 2019. Disponível em: <https://www.ateomomento.com.br/diagramas-uml/>. Acesso em: 18 set. 2019.

## **UNIDADE 2**

Modelagem de caso de uso

# INTRODUÇÃO

Um dos principais problemas em desenvolver um sistema é manter a integridade dos seus componentes entre as diversas etapas do desenvolvimento. Esse problema pode começar a surgir logo no início do desenvolvimento quando requisitos funcionais, regras de negócios e requisitos não funcionais identificados junto ao usuário na etapa de levantamento de requisitos podem ser “deixados de lado”, ou seja, não fazer parte do software, ou outros requisitos podem “survir” sem que haja o registro da sua necessidade ou de quem o solicitou.

Esses problemas, quando ocorrem, somente são detectados em fases adiantadas do projeto de desenvolvimento, fazendo com que os acertos prejudiquem as datas previamente acertadas das entregas, gerando prejuízos de tempo e financeiro. Saber controlar a migração dos requisitos em casos de uso para que não haja perdas ou inclusões de funcionalidades é uma tarefa simples que irá minimizar a possibilidade de problemas dessa natureza. A identificação correta dos casos de uso e a construção do diagrama correspondente associando aos atores principais e suas descrições irão gerar qualidade na documentação produzida até essa etapa.



## OBJETIVO

Nesta unidade você será capaz de:

- Realizar a transição dos requisitos funcionais identificados na fase de levantamento de requisitos em casos de usos para a construção do Diagrama de caso de uso e documentá-los por meio da descrição de cada um deles em modelo proposto.

# Transição dos requisitos funcionais para caso de usos

A fase de levantamento de requisitos de um sistema tem como resultado a identificação das principais necessidades que o usuário tem para o sistema. Requisitos funcionais, requisitos não funcionais e as regras de negócio do sistema são as principais informações coletadas por meio das várias técnicas de elicitación de requisitos.

Consideremos as definições de cada um deles, abaixo descritas:

- Os **requisitos não funcionais** estão associados a aspectos relacionados à arquitetura do sistema, como, por exemplo, aspectos relacionados à qualidade do sistema, tais como usabilidade, performance, portabilidade etc., e também outros recursos relacionados aos padrões de desenvolvimento. Esses requisitos são utilizados quando o projeto arquitetural do sistema estiver sendo desenvolvido.
- As **regras de negócio** são as políticas que as empresas utilizam nos seus processos de negócios. Essas regras são características próprias de cada empresa, mesmo se considerarmos aplicações semelhantes. Por exemplo: uma empresa de comércio eletrônico pode “pontuar” seus clientes oferecendo vantagens àqueles que possuem maior pontuação. Essa situação representa uma ação específica dessa empresa, pois outras concorrentes podem ter outros tipos de pontuação ou não ter nada equivalente. Nessa fase de desenvolvimento do sistema é importante conhecer quais são as regras que vão impactar no negócio que está em desenvolvimento para que possam ser implementadas no software.
- Os **requisitos funcionais** são aqueles com que devemos nos preocupar neste momento, pois representam as funcionalidades (necessidades) que o usuário informou que deve haver no sistema. Consultas, relatórios, processamento de algo relacionado ao negócio são alguns dos requisitos que sempre são citados na etapa de levantamento. Além destes, os procedimentos de atualização (inclusão, alteração, deleção e consulta) das classes de domínio irão compor o sistema para que processamentos e saídas possam ser realizados.



## Importante

Regra de negócio não é sinônimo de requisito funcional. Os requisitos são todas as necessidades que devem estar presentes no software e a regra de negócio são implementações de ações relacionadas ao negócio que devem ser implementadas em alguns dos casos de uso. Considerando-se o exemplo das vantagens oferecidas aos melhores clientes de uma empresa de comércio eletrônico, na funcionalidade que irá realizar a venda devem ser implementadas as políticas de pontuação e vantagens definidas pela empresa. Neste sentido, o caso de uso “Realizar venda” implementa a regra de negócio “Conceder vantagens aos melhores clientes”.

São os requisitos funcionais que devem ser analisados nesta etapa do desenvolvimento uma vez que, se eles representam as necessidades do sistema, devem estar previstos no software a ser desenvolvido, representados pelos casos de uso, e, portanto, temos que definir padrões de rastreabilidade para que haja consistência entre esses dois elementos, evitando que os requisitos funcionais “desapareçam”, ou seja, deixem de se transformar em casos de usos e também evitando que “surjam” casos de uso sem que haja requisito funcional que aponte para a sua necessidade.

Um dos principais problemas para o desenvolvimento de um software é manter a integridade entre as várias etapas e diagramas desenvolvidos. A interface existente entre as etapas de levantamento de requisitos e análise e projeto do sistema é representada pelos requisitos funcionais, regras de negócio e requisitos não funcionais, portanto para evitar que estas etapas se tornem inconsistentes é necessário que o processo de migração entre elas seja feito de forma segura.

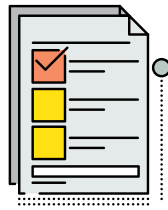
A UML ou a Engenharia de Software não propõem um modelo de migração entre essas etapas, portanto cabe ao desenvolvedor criar padrões próprios que permitam transformar com segurança os requisitos funcionais em casos de uso e identificar em qual desses casos de uso as várias regras de negócio serão implementadas. Dessa forma, será possível evitar que requisitos ou regras de negócios sejam “esquecidos” ou sejam criados casos de uso sem que o usuário tenha solicitado.

Figura 1: Situações que devem ser evitadas no desenvolvimento de um software.





Figura 2: Verificar se os requisitos funcionais foram considerados como casos de usos.



## Requisitos

Para que seja feita uma migração segura entre a etapa de levantamento de requisitos e análise e projeto de software, são necessárias apenas três atividades relativamente simples:

- **Transição dos Requisitos funcionais em casos de uso** – Neste caso, devemos identificar numericamente cada um dos requisitos funcionais (ex.: RF01, RF02, RF03...) para facilitar o manuseio e criar uma tabela associando cada um deles ao caso de uso que irá implementá-lo, também identificado numericamente (ex.: CSU01, CSU02, CSU03...), descrever o objetivo de cada um deles e um nome que será associado ao caso de uso. Além dessas informações, deve conter o(s) ator(es) responsável(veis) pela ativação do caso de uso e a(s) regra(s) de negócios que serão implementadas, caso haja alguma. É importante que essas regras também sejam identificadas de forma numérica e sequencial para facilitar o entendimento. Desta forma teremos o seguinte exemplo:

| ID    | Nome                             | Descrição  | RF   | Ator           | RN   |
|-------|----------------------------------|--|------|----------------|------|
| CSU01 | Manter beneficiários             | O sistema deverá ser capaz de manter atualizados os dados dos beneficiários dos planos de saúde.                       | RF01 | Administrativo |      |
| CSU02 | Manter clientes                  | O sistema deverá ser capaz de manter atualizados os dados dos clientes da empresa.                                     | RF02 | Administrativo | RN01 |
| CSU03 | Manter planos de saúde           | O sistema deverá ser capaz de manter atualizados os dados dos planos saúde.  | RF03 | Financeiro     |      |
| CSU04 | Manter funcionários              | O sistema deverá ser capaz de manter atualizados os dados dos funcionários da empresa.                                 | RF04 | Financeiro     |      |
| CSU05 | Consultar adimplência do cliente | O sistema deve ser capaz de permitir a confirmação da adimplência do cliente para realização/liberação para o serviço. | RF05 | Gerente        |      |

- **Tabela de atores** – Nesta tabela serão relacionados todos os atores que irão interagir com o sistema, indicando os casos de uso nos quais são responsáveis e o seu papel no sistema. Esta tabela funciona como uma “tabela reversa” àquela construída inicialmente e irá servir como base para a construção do Diagrama de Caso de Uso.

| Nome           | Descrição   | Casos de uso |
|----------------|---|--------------|
| Administrativo | Realiza as funcionalidades operacionais diárias da empresa.                     | CSU01, CSU02 |
| Financeiro     | Realiza as funcionalidades ligadas às operações financeiras diárias da empresa. | CSU03, CSU04 |
| Gerente        | Realiza atividades de gestão operacional da empresa.                            | CSU05        |

- **Tabela de regras de negócios** – Tabela que também funciona como “tabela reversa” àquela inicialmente construída tendo como objetivo relacionar todas as regras de negócio identificadas na etapa de levantamento e agora numeradas de forma sequencial (ex.: RN01, RN02, RN03...), sua descrição e o(s) caso(s) de uso que irá(ão) implementá-la(s). A criação dessa tabela visa evitar que alguma regra seja implementada, pois dessa forma ela ficaria sem caso de uso associado.

| ID   | Nome                         | Descrição   | Casos de uso |
|------|------------------------------|---|--------------|
| RN01 | Verificar maioria do cliente | Somente clientes com idade superior a 18 anos podem ser responsáveis pelo plano de saúde. | CSU02        |

Algumas regras para composição dos casos de uso:

- Deve descrever uma funcionalidade ou objetivo do sistema.
- Devem ser identificados por ações verbais que representem o seu objetivo. Ex.: Manter, Calcular, Realizar, Gerar, Consultar etc.
- Devem ser definidos por meio de nomes curtos. Ex.: Manter cliente, Calcular comissão, Consultar histórico financeiro etc.
- Caso a quantidade de casos de uso de um sistema seja grande e impeça a visualização correta pelo diagrama, é indicado unir os casos de uso similares em pacotes e representá-los em cada diagrama.

# O Diagrama de Caso de Uso

O diagrama de caso de uso é uma representação gráfica da interação entre os elementos externos que se relacionam com o sistema e as ações que cada um deles realiza. Este diagrama representa uma importante ferramenta de avaliação por permitir fácil visualização das funcionalidades que serão realizadas pelo sistema e quem fará, ou faz, cada uma delas, por parte do usuário. Por ter características gráficas não é necessário ter conhecimentos da linguagem UML para realizar esta avaliação, sendo, normalmente, a sua apresentação “conduzida” pelo profissional de TI, explicando cada uma das ações nele apresentadas.

Ele é considerado uma importante ferramenta da fase de levantamento de sistemas pelos seguintes motivos:

- Ser uma representação das funcionalidades externamente observáveis do sistema e dos elementos.
- Moldar os requisitos funcionais do sistema.

O diagrama de caso de uso faz parte da modelagem de caso de uso, juntamente com a documentação produzida na fase de levantamento do sistema, a definição dos casos de usos e o detalhamento de cada um deles.

Diante dessas explicações, surge o questionamento que dá origem a todo esse processo:

## O que é caso de uso?

Segundo Macoratti, citando Ivan Jacobson, um caso de uso é um “documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo”.

Ainda segundo Macoratti, “um caso de uso é uma técnica de modelagem usada para descrever o que um novo sistema deve fazer”. Esse diagrama deve ser construído por meio da comunicação entre o usuário e os profissionais de TI, responsáveis pelo levantamento dos requisitos do sistema, com o objetivo de descrever as ações que devem ser realizadas, identificadas pela interação usuário x sistema. Ao final dessa comunicação, o usuário deve dar o “aceite” sobre o que foi descrito, validando, assim, a forma com que o sistema irá realizar o caso de uso.

**Dessa forma, podemos concluir que o caso de uso define o uso de uma funcionalidade do sistema sem revelar a sua estrutura e o comportamento interno desse sistema.**

## Tipos de casos de usos

Embora não sejam amplamente conhecidos, existem dois tipos de casos de usos:

**Primário** – Representados por aqueles que constituem os objetivos dos atores por meio das funcionalidades que estão sendo implementadas pelo sistema. Nesta categoria se enquadram os casos de usos que agregam valor ao sistema, como por exemplo: Realizar inscrição, Gerar relatório de venda, Calcular média etc.

**Secundário** – Representados por aqueles que são necessários para que os objetivos específicos do sistema possam ser atingidos. Como exemplo, a manutenção dos cadastros, usuários etc.



### Exemplo

O objetivo de um sistema acadêmico é fornecer informações para o acompanhamento da vida acadêmica de um aluno. Para isso um aluno se inscreve em uma turma/disciplina, o professor lança os graus por ele obtido e o sistema informa se está aprovado ou não. Esses são exemplos de casos de usos primários, pois atendem ao objetivo do sistema.

Portanto, para que o aluno possa se inscrever em uma turma/disciplina, é necessário que as disciplinas sejam previamente cadastradas, assim como as turmas, associando cada uma das turmas a uma ou mais disciplinas. Sem esses cadastros não é possível o sistema operar, pois impediria que o aluno se inscrevesse. Estes são exemplos de casos de uso secundários, pois não representam objetivos específicos, mas sem eles esses objetivos não serão atingidos.

## Como construir um Diagrama de Caso de Uso

Para a construção de um diagrama de caso de uso utilizamos poucos elementos gráficos, um facilitador do seu entendimento e temos algumas regras para construção.

O primeiro passo para a construção de um diagrama é definir com um retângulo os limites do sistema. Os atores ficarão fora do retângulo e os casos de uso dentro. Esses posicionamentos têm como objetivo informar que os casos de usos estão inseridos no sistema e os atores são aqueles que interagem, ou seja, estão fora do sistema.

Após a criação do retângulo devem ser relacionados todos os atores e casos de usos identificados para o sistema. A identificação dos atores deve ser colocada abaixo do símbolo que o representa, e o nome do caso de uso, ou a identificação, deve ser colocado dentro do símbolo que representa. Lembrando sempre que os atores ficam fora do limite do sistema e os casos de usos, dentro dele.

Para finalizar, devem ser associados os casos de uso a cada um dos atores que os executam por meio de um traço, identificado como comunicação. Nessa fase o diagrama deve ser organizado para que seja evitado o cruzamento das comunicações (traços) entre diferentes atores e casos de uso.



### Importante

A construção deve ser feita tendo como base a transição dos requisitos funcionais em casos de uso, pois nela estão relacionados tudo que deve ser previsto no sistema e os atores que irão executá-los.



### Exemplo

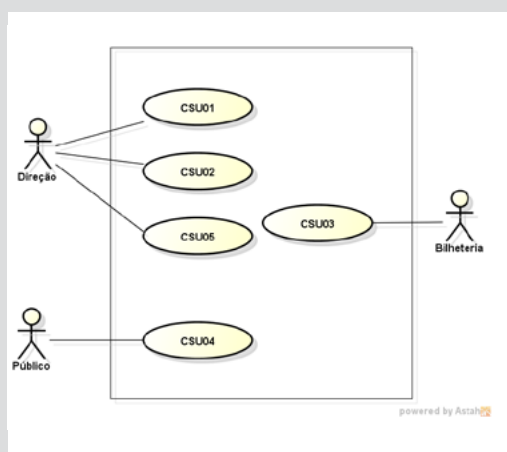
Considerando a transição abaixo, podemos construir o diagrama representado pela figura 3.

| Identificação | Requisito                                     | Descrição  | Ator       |
|---------------|---|--|------------|
| RF01          | Manter teatros                                | O sistema deverá ser capaz de manter atualizados os dados dos teatros.                 | Direção    |
| RF02          | Manter peças                                  | O sistema deverá ser capaz de manter atualizados os dados das peças.                   | Direção    |
| RF03          | Vender ingresso                               | O sistema deverá ser capaz de permitir ao público a compra de ingressos na bilheteria. | Bilheteria |
| RF04          | Vender ingresso antecipado                    | O sistema deverá ser capaz de permitir a compra antecipada de ingresso.                | Público    |
| RF05          | Emitir relatório com total de público pagante | O sistema deverá ser capaz de emitir o relatório com o total de público pagante.       | Direção    |



## Importante

Figura 3: Diagrama de caso de uso.



Fonte: Elaborada pelo autor (2019).

Para identificar o caso de uso em um diagrama devemos colocar o nome do caso de uso ou a sua identificação. Ambos têm vantagens e desvantagens.

Ao colocar o nome, os símbolos que representam os casos de usos ficarão de diferentes tamanhos, adaptando-se ao nome do caso de uso, por exemplo: Manter Peça e Emitir relatório de público pagante. Essa situação cria uma visualização (estética) ruim, no entanto, o diagrama fica documentado de forma mais clara, pois é fácil identificar que atores realizam quais casos de uso.

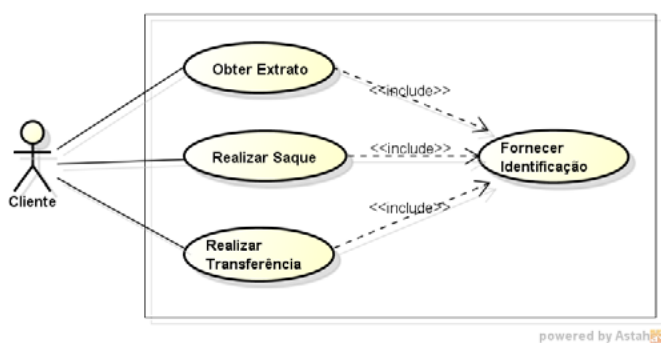
Colocando as identificações dos casos de uso, conforme diagrama apresentado na figura 3, os tamanhos ficam padronizados, pois todas as identificações têm a mesma quantidade de caracteres, no entanto, caso queira saber “quem faz o quê” será necessário consultar a documentação.

## Outros tipos de relacionamentos do Diagrama de Caso de Uso

A UML especifica três tipos de relacionamento no diagrama de caso de uso que representam situações de relação/dependência entre os casos de uso. Vamos a eles:

**Inclusão** – O mecanismo de empacotar uma sequência de instruções em uma rotina evita a repetição dessa rotina em todo lugar em que ela se faz necessária. Sempre que a sequência de instruções deve ser utilizada, a rotina correspondente é chamada. Nesse caso, podemos definir essa sequência como um caso de uso específico que é acionado sempre que outro o utilize.

Figura 4: Diagrama de caso de uso com inclusão.

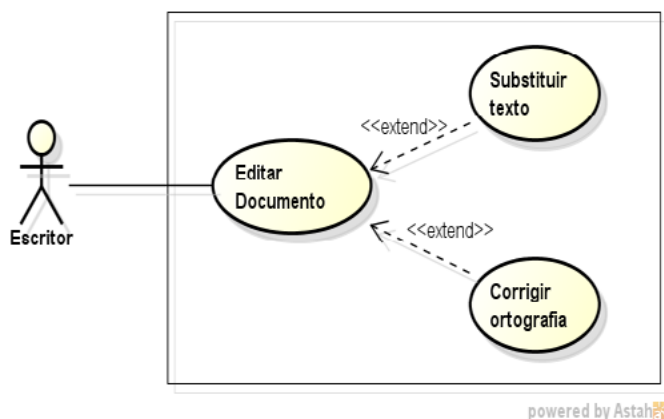


Fonte: Elaborada pelo autor (2019).

No diagrama apresentado na figura 4, o ator “Cliente” realiza três casos de usos: “Obter Extrato”, “Realizar Saque” e “Realizar Transferência”. Nas três situações o caso de uso “Fornecer Identificação” **deve** (obrigatório) ser realizado.

**Extensão** – Um relacionamento de extensão de A para B indica que um ou mais dos cenários de B **podem** (opcional) ser realizados por A.

Figura 5: Diagrama de caso de uso com extensão.

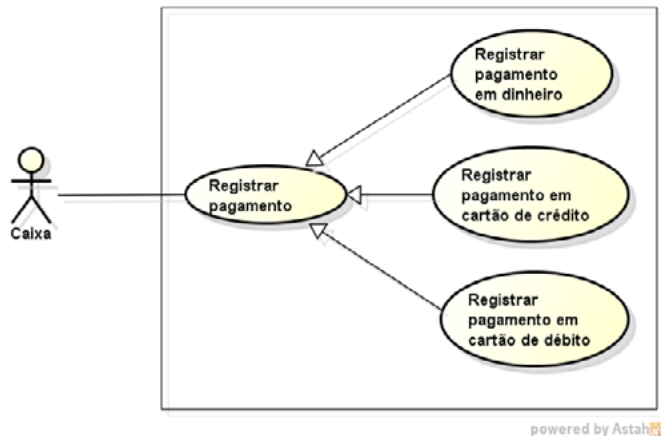


Fonte: Elaborada pelo autor (2019).

No diagrama apresentado na figura 5, o ator “Escritor” executa o caso de uso “Editar Documento”, podendo ou não também realizar os casos de uso “Substituir texto” e “Corrigir ortografia”.

**Generalização** – Ocorre sempre que um caso de uso herda sequência de comportamento de outro caso de uso. A generalização também é conhecida como herança, podendo ser de caso de uso ou atores. Esta ocorre quando um ator realiza as mesmas funções (herda) de outro ator.

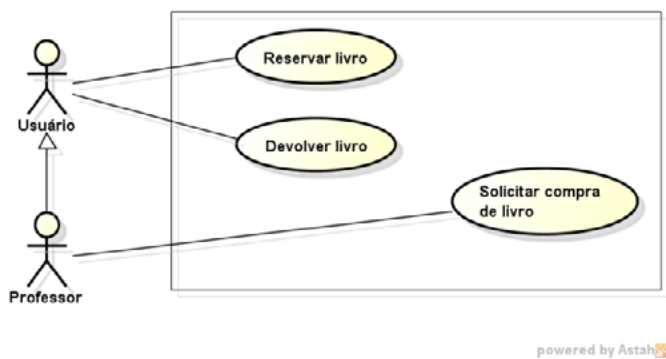
Figura 6: Diagrama de caso de uso com generalização de caso de uso



Fonte: Elaborada pelo autor (2019).

No diagrama apresentado na figura 6, o ator “Caixa” realiza o caso de uso “Registrar pagamento” e um dos três outros casos de uso. A diferença entre a generalização e a extensão é que, na primeira, pelo menos um dos casos de usos deve ser executado, enquanto na extensão pode não haver esta situação.

Figura 7: Diagrama de caso de uso com generalização de atores.



Fonte: Elaborada pelo autor (2019).

No diagrama representado pela Figura 7 existe uma generalização de atores no qual o ator “Professor” realiza a solicitação da compra de livro e as mesmas funcionalidades do ator “Usuário”, ou seja, o Professor herda os casos de uso de Usuário.



# A descrição dos casos de usos

A descrição dos casos de uso representa uma forma de documentar a interação entre o ator e o sistema. Essa interação tem como objetivo mostrar a ação que o ator faz e a resposta do sistema, portanto ela se propõe a apresentar a reação que o sistema faz a cada ação feita pelo ator na execução do caso de uso. Essas ações/reações devem ser detalhadas em um nível superficial não apresentando características tecnológicas ou especificidades de interface ou linguagem de programação. O objetivo é que a interação não seja alterada caso haja mudanças de ambientes, linguagens, arquiteturas, banco de dados ou outra nova tecnologia que surja e provoque a migração do caso de uso. Esse tipo de caso de uso é conhecido como essencial.



## Importante

Pode ser utilizada a “regra prática dos 100 anos” para identificar se um caso de uso contém algum detalhe de tecnologia. Pergunte-se, ao ler a narrativa de um caso de uso, se a narrativa seria válida tanto há 100 anos quanto daqui a 100 anos. Se a resposta para sua pergunta for “sim”, então, muito provavelmente, esse caso de uso não define aspectos tecnológicos, ou seja, é um caso de uso definido como essencial.

Os casos de uso cujas interações citam detalhes de tecnologia que serão utilizados na implementação são conhecidos como caso de uso real. O uso desse tipo de modelo sofre mudanças sempre que a tecnologia ou outro elemento da arquitetura altera, desta forma, o cuidado com a sua manutenção é maior do que com os casos de uso essenciais.

A descrição de um caso de uso é também conhecida como “Cenário”, sendo que, além desta, existem diversas formas de representar um cenário. Um cenário também é chamado de instância de um caso de uso.

## Por que instância de caso de uso?

Dentro dos conceitos da orientação a objeto, instância representa um elemento de uma classe. Neste caso, a “classe” seria um caso de uso. Assim, a descrição de um cenário, seja por meio da descrição de caso de uso ou de outra forma, deve considerar todas as possibilidades de execução desse cenário, isto é, se existem alternativas, por opções dos usuários ou do modelo, que demandem ações específicas para cada uma delas, todas devem ser documentadas para que sejam conhecidas, o que deve ser feito para cada uma.



### Exemplo

Em um processo de atualização dos dados de uma classe de negócio, o ator tem as opções de incluir uma nova instância nesta classe, alterar os dados de uma instância já existente, excluir uma instância cadastrada ou apenas consultar. Cada uma dessas ações requer ações específicas para que sejam realizadas, ações que devem ser definidas e validadas pelo usuário.

O que podemos destacar como mais importante no processo de descrição de um cenário, seja por descrição de um caso de uso ou outro tipo de representação, é a validação que deve ser feita pelo usuário, ator ou a pessoa que forneceu as informações para a sua construção. Essa validação é importante para que a sua implementação ocorra em um formato já aceito minimizando a possibilidade de retrabalho futuro.

Figura 8: Usuário deve validar as descrições realizadas pelo profissional de TI.



## Modelo para a descrição de um caso de uso

A UML não especifica um padrão para essa descrição, por este motivo, não existe um consenso quanto às informações que devem estar presentes ou à forma com que são definidas. É possível encontrar diferentes maneiras de representação com algumas diferenças de conteúdo, mas o padrão de interação entre ator/sistema deve estar presente. Para o desenvolvimento da nossa disciplina, vamos propor um modelo de descrição, lembrando sempre que outros modelos são encontrados na literatura.

Analisando o modelo proposto, observamos que muitas das informações têm suas origens na documentação produzida na etapa de levantamento e organizada na migração dos requisitos funcionais para casos de uso. Vamos analisar cada uma delas.

Figura 9: Modelo de descrição de caso de uso proposto para a disciplina.

| Nome   | Manter Fornecedor                                | Id: CSU01 |
|--|--|-----------|
| Sumário:   | Manter atualizados os dados dos fornecedores.    |           |
| Ator Primário:   | Setor de compras.                                |           |
| Ator Secundário:   |  |           |
| Pré-condição:  | Ator deve ser validado no perfil pela aplicação. |           |
| Requisito Funcional:   | RF01   |           |
| Regra de Negócio:  | RN03, RN04.                                      |           |
| Fluxo Principal  |  |           |
| 1. Descrever passo a passo os procedimentos que devem ser realizados.        |  |           |
| Fluxo Alternativo  |  |           |
| Descrever, para cada fluxo alternativo, os procedimentos a serem executados. |  |           |
| Fluxo de exceção   |  |           |
| Descrever, para cada fluxo de exceção, os procedimentos a serem executados.  |  |           |
| Pós-Condição:  | Fornecedor atualizado.                           |           |

- **Nome e identificação do caso de uso:** retirada da tabela de migração dos requisitos funcionais para caso de uso, contendo o nome e a identificação do caso de uso que será descrito.
- **Sumário:** Define o objetivo do caso de uso, também retirado da tabela de migração dos requisitos funcionais para caso de uso.
- **Ator:** Existem dois tipos de atores:
  - **Primário:** representa aquele responsável por iniciar o caso de uso, ou seja, aquele que executa o procedimento. Essa informação também é retirada da tabela de migração dos requisitos para caso de uso.
  - **Secundário:** representa um elemento externo que não executa (inicia) o caso de uso, porém, sua ação é necessária para a execução. Um ator secundário pode ser outro ator ou sistema interno ou externo à empresa. Por exemplo, se formos definir a descrição de uma venda on-line, o ator principal seria o cliente, por ter sido ele que iniciou esse processo, entretanto, caso a compra seja feita por meio de cartão de crédito, a administradora do cartão é o ator secundário, pois ela é acessada e as informações passadas interferem na execução. A empresa que está realizando a venda não possui nenhuma gestão sobre as ações realizadas pela administradora de cartão.
- **Pré-condição:** Identifica ações que devem ter ocorrido para que o caso de uso seja realizado corretamente. Existem duas características de ações:
  - **Segurança:** especifica que o ator deve ter sido validado pelo sistema, por meio de um caso de uso “Login” e está autorizado a realizar essa ação (perfil).
  - **Negócio:** define as ações de negócio que devem ter ocorrido para que o caso de uso seja realizado corretamente. Por exemplo, para que o caso de uso “Realizar compra de matéria-prima” seja feito, é necessário que a matéria-prima a ser adquirida tenha sido previamente cadastrada.
- **Requisito Funcional:** Especifica a identificação do requisito funcional a que o caso de uso está associado. Essa informação é retirada da tabela de migração dos requisitos funcionais para casos de usos.

- **Regras de negócio:** Relaciona as regras de negócios que devem ser implementadas no caso de uso. Essa informação é retirada da tabela com as regras de negócio.
- **Fluxo principal:** Descreve as ações que sempre serão realizadas pela aplicação. Cada ação deve ser numerada sequencialmente. A primeira é sempre representada pela ação do usuário ao iniciar o caso de uso e, a partir daí, a interação entre ator/sistema.

Figura 10: Exemplo de descrição do fluxo principal.

| Fluxo Principal   |
|---|
| <ol style="list-style-type: none"> <li>1. Ator escolhe opção Manter Fornecedor.</li> <li>2. Sistema exibe uma tela para realizar a manutenção dos dados do fornecedor.</li> <li>3. Atos escolhe a opção de incluir um novo fornecedor [FA01] ou Ator informa o critério de pesquisa e escolhe a opção para execução da seleção [FA04].</li> <li>4. Encerrar caso de uso.</li> </ol> |

- **Fluxos alternativos:** Descreve as ações que serão ou não realizadas conforme opção feita pelo ator. Deve ser descrito um fluxo alternativo para cada ação a ser implementada.

Figura 11: Exemplo de descrição de fluxo alternativo.

| Fluxo Alternativo: [FA01] Incluir Fornecedor   |
|--|
| <ol style="list-style-type: none"> <li>a. Sistema exibe tela para inclusão dos dados do fornecedor</li> <li>b. Ator informa os dados na tela</li> <li>c. Ator escolhe opção de Salvar [FA02] ou Cancelar [FA03]</li> </ol>   |
| Fluxo Alternativo: [FA02] Salvar   |
| <ol style="list-style-type: none"> <li>a. Sistema valida os dados [FE01] [FE02]</li> <li>b. Sistema grava os dados [FE03]</li> <li>c. Retornar para o passo 2 do Fluxo Principal</li> </ol>  |
| Fluxo Alternativo: [FA03] Cancelar   |
| <ol style="list-style-type: none"> <li>a. Retornar para o passo "2" do Fluxo Principal</li> </ol>  |
| Fluxo Alternativo: [FA04] Pesquisar  |
| <ol style="list-style-type: none"> <li>a. Sistema recupera os dados a partir do critério de seleção definido [FE03] e realiza a exibição na tela</li> <li>b. Atos seleciona um fornecedor e escolhe a opção de Alteração [FA05], Exclusão [FA06] ou Consulta [FA07]</li> </ol> |

- **Fluxos de exceção:** Descreve as ações que serão realizadas quando ocorrer uma situação não prevista (erro) na aplicação. As situações mais comuns são:
  - **Acesso à base de dados:** devem ser definidas as ações feitas pela aplicação quando ocorrer algum tipo de acesso à base de dados que produza uma situação de erro. Esses erros devem ser tratados para evitar o envio de mensagens diretamente do sistema gerenciador de base de dados.
  - **Violação dos dados:** definir as ações que devem ser realizadas quando houver algum tipo de erro na manipulação do dado informado. Ex.: não preenchimento de um campo, preenchimento com formato incorreto etc.
  - **Violação de regra de negócio:** definir as ações que devem ser realizadas quando for identificada uma situação que viole alguma regra de negócio tratada pelo caso de uso. Deve ser descrito um fluxo de

Figura 12: Exemplo de descrição de fluxo de exceção.

|  |
|--|
| <b>Fluxo de Exceção: [FE01] Validar dados obrigatórios</b>           |
| a. Enviar mensagem de erro<br>b. Retornar para o passo "a" do [FA01] |
| <b>Fluxo de Exceção: [FE02] Violação de regra de negócio</b>         |
| a. Enviar mensagem de erro<br>b. Retornar para o passo "a" do [FA01] |
| <b>Fluxo de Exceção: [FE03] Erro de acesso a banco</b>               |

- **Pós-condição:** Apresenta o resultado do cenário, caso tenha sido realizado com sucesso. Ex.: Relatório gerado, Dados atualizados etc.



### Importante

Manual para o usuário não tem o mesmo objetivo que a descrição de um caso de uso; enquanto o primeiro descreve as ações que o usuário deve realizar para atingir um objetivo, o segundo descreve a forma com que essas ações são realizadas pelo sistema para que o objetivo seja alcançado.



## MIDIATECA

Acesse a midiateca da Unidade 2 e veja o conteúdo complementar indicado pelo professor sobre um resumo dos assuntos abordados na unidade.



## NA PRÁTICA

No dia a dia do desenvolvimento de um sistema, a necessidade de apresentar produtos para o usuário sempre é superior ao tempo disponível. Aumentar a equipe para que os prazos possam ser cumpridos ajuda a melhorar essa performance, mas existem limites de orçamento e técnicos, pois muitas vezes aumentar indiscriminadamente a quantidade de pessoas não significa reduzir o tempo na mesma proporção. Criar procedimentos padronizados que sistematizem as atividades e reduzam o risco do projeto é a melhor solução para atingir a melhor relação entre custo e qualidade.

## Resumo da Unidade 2

Nesta unidade você construiu o diagrama e descrições de caso de uso a partir das informações coletadas na etapa de levantamento dos requisitos de um sistema. Para que essas construções fossem feitas de forma segura, foram apresentados alguns procedimentos que visam facilitar o processo de mapeamento dos requisitos funcionais em casos de usos.

São muitos os riscos associados a um projeto de desenvolvimento de sistema. Muitos dos problemas que surgem estão associados ao curto tempo reservado para que as necessidades que o usuário tem no sistema possam ser identificadas corretamente. Considerando que as empresas nos dias de hoje necessitam ter as suas demandas de softwares atendidas rapidamente, esse curto tempo se propaga para as demais etapas fazendo com que a documentação não tenha a devida atenção. Essa “pressa” em oferecer produtos ao usuário sem o planejamento adequado produz resultados e consequências desastrosos, gerando retrabalho e aumentando bastante os custos associados a esse projeto. Entender a demanda do mundo atual, porém sem abrir mão da qualidade do produto, é uma equação que precisa ser feita de forma bastante equilibrada. Esse resultado muitas vezes pode ser obtido com pequenas ações de sistematização de processos, utilizando pequenas ferramentas ou passos, que reduzem bastante o risco de um desenvolvimento sem impactar de forma significativa no tempo.

Por este motivo, a UML oferece o Diagrama de Caso de Uso que permite ao usuário entender o contexto das funcionalidades que irão compor o sistema e dá abertura para que outras ações possam ser realizadas para que a Modelagem dos casos de usos possa ser feita de forma consistente com os requisitos identificados. Continuando o desenvolvimento do software, na próxima unidade você irá conhecer os procedimentos propostos pela UML para a Modelagem de Classe de Domínio, em que os principais componentes de negócio do sistema e suas relações serão identificadas.





## CONCEITO

Nesta unidade, foram estudados três conceitos importantes:

- A importância da **transição entre os requisitos funcionais em casos de usos** como forma de manter a integridade nas funcionalidades identificadas.
- A construção do **Diagrama de Caso de uso**.
- A documentação dos casos de uso por meio de um modelo proposto para a **Descrição de casos de usos**.

# Referências

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 2. ed. Rio de Janeiro: Campus-Elsevier, 2007.

**CASO de uso**. In: WIKIPEDIA: the free encyclopedia. [San Francisco, CA: Wikimedia Foundation, 2010]. Disponível em: [https://pt.wikipedia.org/wiki/Caso\\_de\\_uso](https://pt.wikipedia.org/wiki/Caso_de_uso). Acesso em: 4 out. 2019.

CODIFICAR. **O que são requisitos funcionais e requisitos não funcionais**. Disponível em: <https://codificar.com.br/blog/requisitos-funcionais-nao-funcionais/>. Acesso em: 4 out. 2019.

MACORATTI, J. C. **Modelando sistemas em UML – Casos de Uso**. Disponível em: [http://www.macoratti.net/net\\_uml2.htm](http://www.macoratti.net/net_uml2.htm). Acesso em: 4 out. 2019.

SPINOLA, R. Especificação de Requisitos com Casos de uso. **DEVMEDIA**. Disponível em: <https://www.devmedia.com.br/especificacao-de-requisitos-com-casos-de-uso/10245>. Acesso em: 4 out. 2019.

VENTURA, P. Entendendo definitivamente o que é um caso de uso. **Até o momento**. 29/02/2016. Disponível em: <https://www.ateomomento.com.br/o-que-e-caso-de-uso/>. Acesso em: 4 out. 2019.

## **UNIDADE 3**

### Modelagem de classe de domínio

# INTRODUÇÃO

Se formos definir um grau de importância aos diagramas da UML, podemos colocar a modelagem de classe de domínio como um dos principais devido ao papel que representa na conexão que essa modelagem faz entre as etapas de levantamento, com a identificação de seus casos de usos e a etapa de modelagem na qual as classes e as relações existentes entre elas para produzir as funcionalidades são identificadas. Dessa forma, o diagrama de classes possui múltiplos objetivos, como a especificação de todas as classes existentes no sistema, seus atributos e métodos, possibilitando a construção de outros diagramas da UML, tais como o diagrama de sequência e estado; permite criar a estrutura inicial para a construção da modelagem de dados, especificando a estrutura das bases de dados que armazenarão os dados do sistema; e apresenta algumas das características físicas da implementação. Assim, ele representa um dos principais diagramas por causa dos vários objetivos diretos e indiretos que proporciona..



## OBJETIVO

Nesta unidade você será capaz de:

- Construir o diagrama de classe de domínio de um sistema apresentando as associações entre as classes e produzindo a documentação necessária do sistema.

# O diagrama de classes

Em uma definição simples, podemos dizer que o diagrama de classes é uma representação gráfica em que são apresentados dois importantes itens para a construção de um sistema: a estrutura das classes de negócios que compõem a aplicação, por meio de suas características (atributos) e seus comportamentos (métodos), e as relações existentes entre essas classes.

## O que é uma classe?

Para entender um pouco melhor esse diagrama, utilizaremos os seguintes conceitos:

- Um sistema de software orientado a objetos é composto de objetos em colaboração para realizar as tarefas desse sistema.
- Uma classe descreve esses objetos por meio de atributos e operações.
- Todo objeto pertence a uma classe.

Vamos exemplificar a classe “Aluno”. Um aluno representa uma classe por possuir um conjunto de objetos que correspondem aos alunos registrados no sistema (instâncias da classe). No diagrama de classes, uma classe é definida pelo símbolo apresentado na Figura 1:

Figura 1: Exemplo de uma classe.




Fonte: Elaborada pelo autor (2019).

A classe é representada por um retângulo dividido em três partes, no qual a parte superior é destinada à identificação da classe (nome).

Para identificar uma classe, devem ser realizadas as seguintes ações:

- De uma forma geral, a identificação de classes deve ser feita apontando-se as classes candidatas que são identificadas. Classes candidatas são aquelas cuja responsabilidade de atualização é do próprio sistema (os casos de uso “Manter” apresentam essas classes).
- Para identificar as classes, devem ser analisados casos de uso, relatórios, documentos, textos, outros softwares relacionados e qualquer artefato que faça parte do domínio do problema, cuja operação dos atributos se faz necessária no sistema.
- Relatórios, consultas ou outros artefatos produzidos pelo sistema **não** são classes de negócio.
- Para criar o nome para uma classe, utilize as seguintes regras:
  - Identifique substantivos para definir o nome da classe.
  - Use nome no singular para as classes conceituais.
  - Use sempre a primeira letra maiúscula.
  - Utilize nomes que identifiquem objetos únicos. Ex.: Cliente, Fornecedor, Aluno, Professor etc.

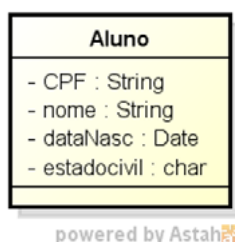
Cada classe possui seus atributos ou, segundo a notação da orientação a objeto, suas características. Um atributo representa um dado pertencente à classe, por exemplo, na classe “Aluno” cada instância possui seu próprio nome, data de nascimento, estado civil etc.



### Importante

No diagrama de classe, são definidos apenas os atributos que pertencem à classe, ou seja, codificações e matrículas não devem ser representados nesse diagrama por serem atributos inseridos apenas para utilização em banco de dados de modelo relacional.

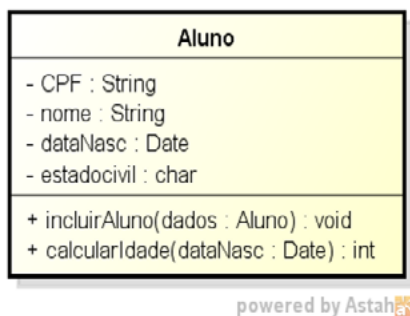
Figura 2: Classe “Aluno” com atributos definidos.



Fonte: Elaborada pelo autor (2019).

Os atributos de uma classe são definidos na segunda parte do retângulo, conforme Figura 2, na qual são relacionados todos os atributos pertencentes e suas características físicas de armazenamento (tipo).

Figura 3: Exemplo completo da classe “Aluno” com atributos e métodos.



Fonte: Elaborada pelo autor (2019).

Na terceira parte do retângulo são definidos os métodos (comportamentos) que pertencem às classes. Métodos são unidades programáveis da classe que podem ser acionadas de outras classes ou métodos e possuem um objetivo específico, podendo ou não retornar valores ou possuírem parâmetros (informações utilizadas para seu processamento).

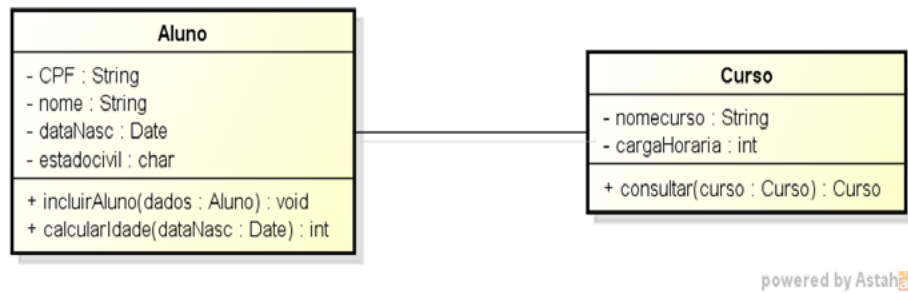
Os métodos no diagrama de classes devem ser definidos com o nome, o tipo de valor que retornam e os parâmetros, caso possuam, conforme apresentado na Figura 3.

## Associação entre classes

Existem algumas formas de representar o fato de que objetos podem se relacionar uns com os outros, sendo uma delas a associação. Uma associação representa relacionamentos (ligações) que são formados entre objetos durante a execução do sistema. Embora as associações sejam representadas entre as classes do diagrama, elas retratam as ligações existentes entre os objetos das classes envolvidas. Outros tipos de relacionamentos e maior detalhamento sobre a associação serão apresentados no próximo tópico.

Vamos representar a situação na qual um aluno está matriculado em um curso. Esse caso representa que um aluno (instância da classe “Aluno”) está inscrito (associado) em um curso (instância da classe “Curso”), conforme Figura 4.

Figura 4: Representação de uma associação entre classes.



Fonte: Elaborada pelo autor (2019).

A associação é representada por meio de um traço (segmento de reta) ligando as classes cujos objetos se relacionam.

## Multiplicidades entre as classes

Multiplicidades entre classes representam a informação dos limites inferior e superior da quantidade de associações com que um objeto pode estar associado a outro. Cada associação no diagrama de classes possui duas multiplicidades, definidas em cada extremo da linha de associação. Os diferentes tipos de multiplicidades são apresentados na Tabela 1.

Tabela 1: Os possíveis tipos de multiplicidades.

| Nome:          | Limite inferior: | Limite superior | Simbologia  |
|----------------|------------------|-----------------|---|
| Apenas um      | 1.               | 1               | 1..1 (ou 1)   |
| Zero ou muitos | 0                | *               | 0..* (ou *)   |
| Um ou muitos   | 1.               | *               | 1..*  |
| Zero ou um     | 0                | 1               | 0..1  |
| Zero ou um     | x                | y               | x..y, onde x e y são valores previamente conhecidos |

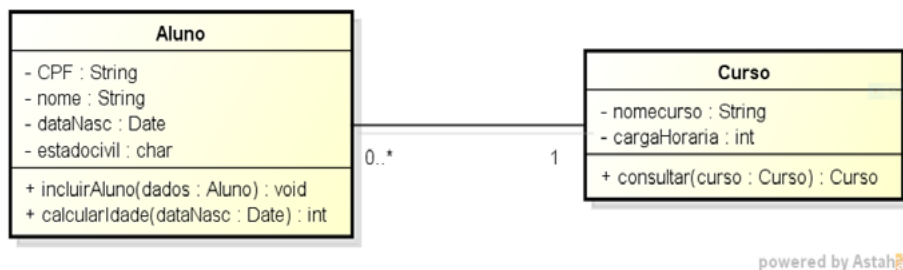


Cada tipo de multiplicidade (simbologia) deve ser representado em cada uma das extremidades da associação, simbolizando a conectividade entre elas.

Analisando a Figura 5, fazemos a seguinte interpretação:

- Um curso pode ter zero ou muitos alunos.
- Um aluno só pode estar inscrito em um curso.

Figura 5: Tipos de multiplicidade.



Fonte: Elaborada pelo autor (2019).

## Outros exemplos com conectividade

Figura 6: Conectividade um para um.



Fonte: Elaborada pelo autor (2019).

Um departamento pode ter zero ou um responsável, e uma pessoa pode ser responsável por no mínimo um e no máximo um (apenas um) departamento.

Figura 7: Conectividade um para muitos.



Fonte: Elaborada pelo autor (2019).

Um departamento pode ter zero ou muitos funcionários, e um funcionário trabalha em apenas um departamento.

Figura 8 - Conectividade muitos para muitos.



Fonte: Elaborada pelo autor (2019).

Um projeto possui zero ou muitos funcionários, e um funcionário trabalha em zero ou muitos projetos.

Figura 9: Conectividade com intervalos específicos.



Fonte: Elaborada pelo autor (2019).

Uma corrida possui no mínimo 10 e no máximo 60 velocistas, e um velocista pode ter corrido no mínimo zero ou no máximo muitas corridas.



#### Dica

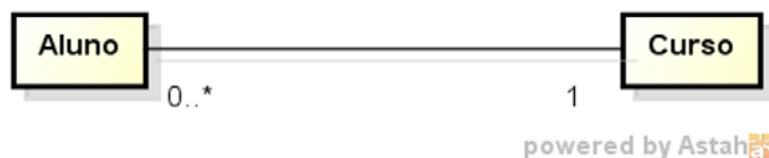
- Utilizar a conectividade “muitos” representa que o limite superior é desconhecido ou não possui um valor específico.
- Utilizar a conectividade com intervalos específicos significa que os limites inferiores e superiores são conhecidos e devem ser respeitados. Essa informação normalmente tem sua origem em uma regra de negócio.

## Modelos de classes

Existem três tipos de modelos de classes de abstração apresentados pela UML:

- Modelo de classe de domínio, ou modelo conceitual:
  - Representa apenas as classes de domínio do negócio.
  - Não considera aspectos relacionados à tecnologia.
  - É voltado para a apresentação ao cliente/usuário.
  - É construído na fase de análise do sistema.
  - Representa apenas o nome das classes e suas associações.

Figura 10: Modelo conceitual.



Fonte: Elaborada pelo autor (2019).

- Modelo de classe de especificação:
  - É obtido por meio de informações sobre solução de software acrescido ao modelo conceitual.
  - Apresenta os principais métodos.
  - Define a navegabilidade entre as classes.
  - É voltado para pessoas que não necessitam de detalhes de desenvolvimento.
  - É construído na fase de projeto do sistema.
  - Podem surgir novas classes para solução do problema.

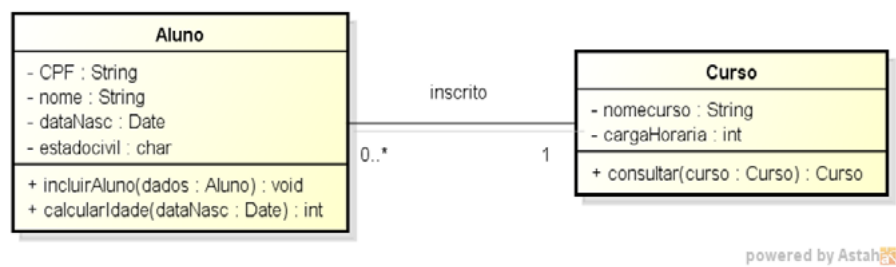
Figura 11: Modelo de especificação.



Fonte: Elaborada pelo autor (2019).

- Modelo de classe de implementação:
  - Corresponde à implementação das classes em uma linguagem de programação.
  - Aborda detalhes de implementação: atributos, tipos, métodos etc.
  - É voltado para a equipe de desenvolvimento.
  - É construído na fase de implementação.

Figura 12: Modelo de implementação.



Fonte: Elaborada pelo autor (2019).

# O relacionamento entre as classes

Relacionamento entre classes representa os vínculos existentes entre elas para que possam compartilhar informações e realizar colaborações. Existem oito tipos de relacionamentos:

## Associação

O relacionamento do tipo associação é a forma mais comum, tendo sido exemplificado no Tópico 1, em que duas classes se relacionam entre si. Também é conhecido como **associação binária**.

A representação gráfica desse relacionamento é feita por uma linha contínua ligando as duas classes, conforme a Figura 13.

Figura 13: Relacionamento do tipo associação.



Fonte: Elaborada pelo autor (2019).

O uso da associação indica que a leitura do relacionamento pode ser feita/entendida por ambos os lados, ou seja:

- Um aluno pode estar inscrito em zero ou muitas disciplinas.
- Uma disciplina pode ter zero ou muitos alunos inscritos.

## Dependência

O relacionamento do tipo dependência é utilizado quando existe dependência entre as classes. Ela é representada por uma linha tracejada com uma seta em uma das extre-

midades, que deve indicar a classe dependente. A mudança no sentido da seta altera a representação da dependência entre as classes.

Figura 14: Relacionamento do tipo dependência.



Fonte: Elaborada pelo autor (2019).



### Importante

Uma dependência representa uma ligação fraca entre os objetos das classes, sendo um tipo pouco comum de relacionamento.

A Figura 14 representa a relação entre as classes Veículo e Motor, na qual um veículo possui um motor, no entanto um motor não pode possuir um veículo. A leitura deve ser feita no sentido da seta representativa do relacionamento.

## Agregação

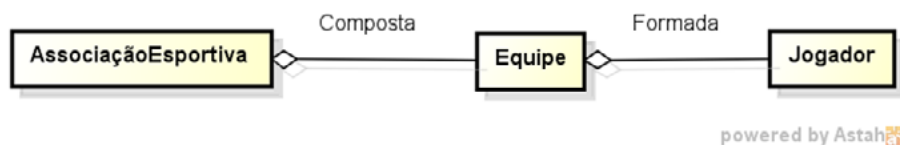
Uma agregação representa um tipo especial de associação no qual um objeto é parte do outro, no entanto a parte pode existir sem o todo. Na agregação, um objeto está contido (parte) em outro (todo). A representação gráfica é feita por meio de um losango vazado (sem preenchimento) ligado à classe dona do relacionamento (todo).

Existem algumas características particulares nessa relação:

- Agregações são assimétricas: se um objeto A é parte de um objeto B, B não pode ser parte de A.
- Agregações propagam comportamento, no sentido de que um comportamento que se aplica a um todo automaticamente se aplica às suas partes.

- Sejam duas classes associadas X e Y. Se uma das perguntas a seguir for respondida com um “sim”, provavelmente haverá uma agregação na qual X seja o todo e Y, parte.
  - X tem um ou mais Y?
  - Y é parte de X?

Figura 15: Relacionamento do tipo agregação.



Fonte: Elaborada pelo autor (2019).

A Figura 15 apresenta dois relacionamentos do tipo agregação, sobre os quais devemos fazer a seguinte interpretação:

- Uma associação esportiva (todo) é composta por equipes (parte).
- Uma equipe (todo) é composta por jogadores (parte).

No entanto:

- Um jogador pode não estar vinculado a nenhuma equipe.
- Uma equipe pode não estar vinculada a nenhuma associação esportiva.

## Composição

Uma composição é uma variação da agregação na qual o objeto parte não pode viver sem o objeto todo, ou seja, eles têm que se pertencer. A representação gráfica é feita por meio de um losango preenchido ligado à classe dona do relacionamento (todo).

Existem algumas características particulares nessa relação:

- A classe X sendo composta por elementos da classe Y significa que X e Y não podem viver um sem o outro.
- Quando o todo é criado, no mínimo uma parcela de suas partes deve ser criada, assim como quando o todo é destruído todas as suas partes também devem ser destruídas.
- Uma parte deve pertencer a apenas um todo.

Figura 16: Relacionamento do tipo composição.



Fonte: Elaborada pelo autor (2019).

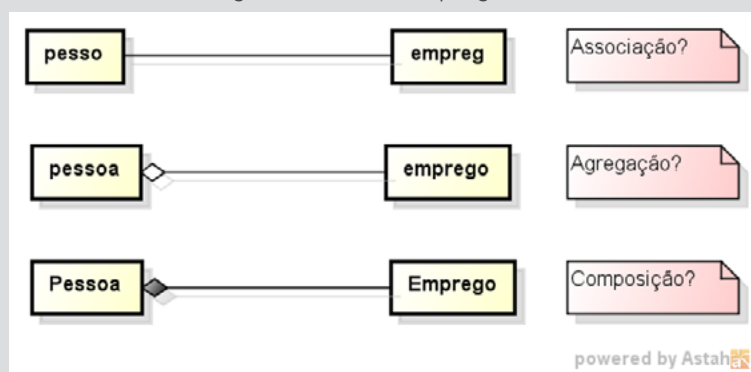
A interpretação dada à Figura 16 deve ser feita da seguinte forma: considere como item de uma nota fiscal cada linha que representa um produto comprado pelo cliente, de modo que:

- Uma nota fiscal só existirá se houver, no mínimo, um item, ou seja, um produto comprado.
- Um item de uma nota fiscal só existirá se estiver associado a uma nota fiscal.
- Se uma nota fiscal for deletada, todos os seus itens também deverão ser.



## Dica

Figura 17: Dicas de perguntas.



Fonte: Elaborada pelo autor (2019).

Como identificar se um relacionamento é associação, agregação ou composição?



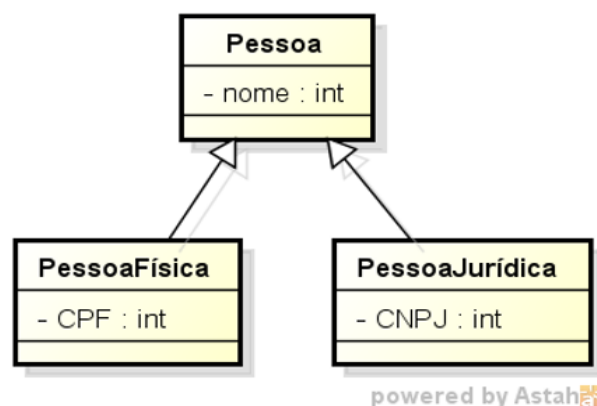
- Se deletar PESSOA, tem que deletar EMPREGO?
  - SIM – Composição.
  - NÃO – Agregação ou Associação.
- EMPREGO tem alguma utilidade sozinho?
  - SIM – Associação.
  - NÃO – Agregação.

## Generalização

Tipo de relacionamento no qual são aplicados os conceitos de herança da orientação a objeto, nos quais são apresentadas as dependências e as hierarquias entre as classes. Suas principais características são:

- Uma associação entre classes, em que uma classe herda as propriedades e os comportamentos de uma ou mais classes (superclasse).
- Uma classe herda propriedades e relacionamentos de sua superclasse imediata (classe em um nível mais alto na hierarquia).
- Toda instância de uma classe também é instância da superclasse.

Figura 18: Relacionamento do tipo generalização.



Fonte: Elaborada pelo autor (2019).

Realizando a interpretação da Figura 18, identificamos a superclasse Pessoa, que possui duas classes vinculadas (Pessoa Física e Pessoa Jurídica), representando dois diferentes tipos de pessoas que possuem atributos ou comportamentos comuns.

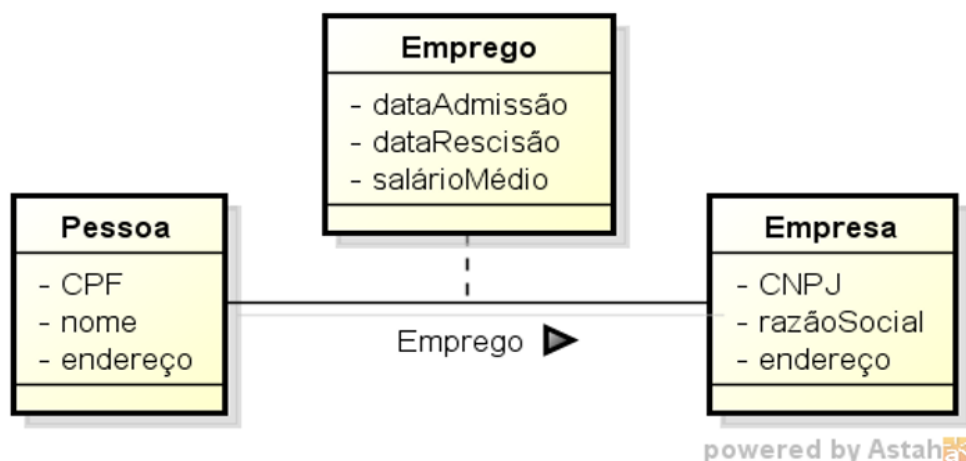
- A pessoa física possui como atributos: nome e CPF.
- A pessoa jurídica possui como atributos: nome e CNPJ.

Considerando que o atributo nome é comum às pessoas físicas e jurídicas, ele é representado na classe Pessoa e herdado por ambas as classes.

## Classe associativa

A classe associativa é definida quando existe a necessidade de ser criada classe a partir de uma relação entre duas outras classes. Essa situação ocorre quando é necessário manter informações sobre a relação, ou seja, essas informações não pertencem apenas a uma das classes que compõem a relação, mas sim a ambas. A representação gráfica é feita de forma similar a uma classe, a diferença é que essa classe está ligada à associação.

Figura 19: Relacionamento com classe associativa.



Fonte: Elaborada pelo autor (2019).

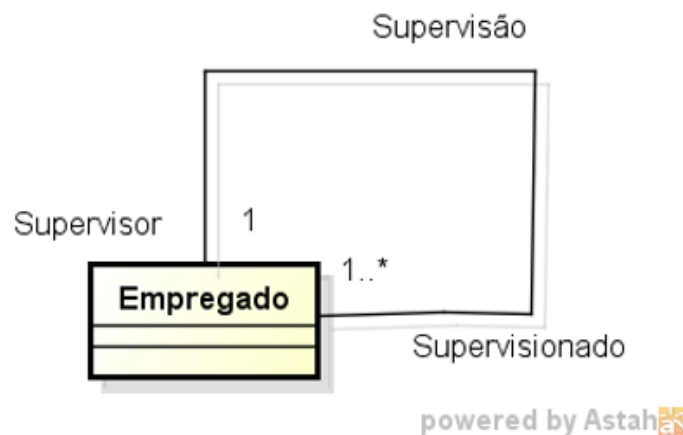
A Figura 19 representa um exemplo de classe associativa na qual identificamos as seguintes características:

- A classe Emprego surge a partir do relacionamento Pessoa x Empresa.
- Os atributos definidos na classe dependem de cada objeto Pessoa e Empresa, ou seja, uma pessoa trabalhou em várias empresas, cada uma delas com suas próprias datas de admissão e rescisão e salário médio.
- Os atributos definidos na classe Emprego não pertencem a objetos das classes Pessoa ou Empresa.

## Associação reflexiva

Este tipo de associação existe quando objetos de mesma classe se associam entre si, tendo cada um deles papéis diferentes na associação, sendo sua utilização importante para evitar ambiguidades na leitura. Uma associação reflexiva não indica que um objeto se associa a ele próprio, ao contrário, indica que objetos de uma mesma classe se associam. A representação gráfica é feita por meio da associação da classe com ela mesma.

Figura 20: Relacionamento com associação reflexiva.



Fonte: Elaborada pelo autor (2019).

Analisando a Figura 20, são identificadas as seguintes informações:

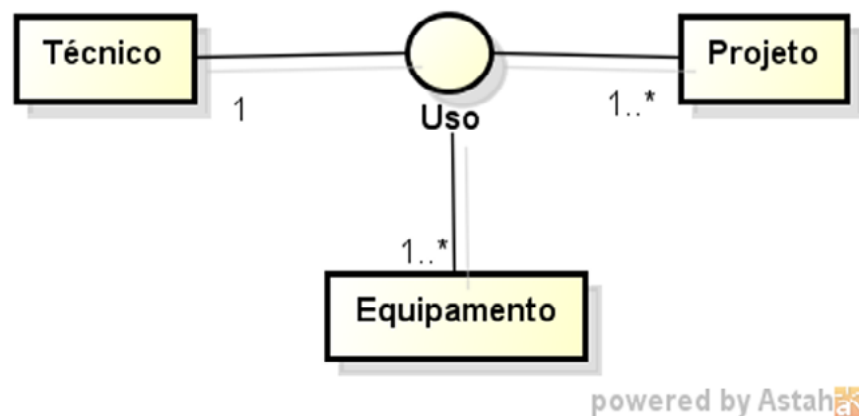
- Um objeto da classe “Empregado” se relaciona com outro objeto dessa mesma classe.

- Esse relacionamento é identificado como Supervisão, no qual um dos empregados é supervisor de um ou muitos empregados, enquanto outros empregados são supervisionados por um empregado.
- Um supervisor é um dos objetos da classe “Empregado”.

## Associação n-ária

Tipo menos comum de associação, ocorre para representar associações existentes entre objetos de n classes. O tipo mais comum, ou menos raro, ocorre nas associações ternárias (relacionamento entre três classes), mas é possível haver associações combinando quantidade maior de classes. A representação gráfica definida pela UML é feita por meio do uso de um losango interceptando as linhas de associação. No software Astah, é feita por meio de um círculo representado por uma interface.

Figura 21: Relacionamento com associação ternária.



Fonte: Elaborada pelo autor (2019).

Ao analisar a Figura 21 identificamos que um ou muitos equipamentos são utilizados por um técnico em um ou muitos projetos. Dessa forma, a relação Uso existirá somente se houver técnico trabalhando em projetos utilizando equipamentos.

# O Dicionário de Classes

Uma das principais características da UML é propor a construção de diagramas que permitirão à equipe de desenvolvimento criar padrões de documentação que possam ser entendidos por qualquer profissional conhecedor da linguagem. Essa característica é importante devido à rotatividade de profissionais que existe nas empresas, nas quais as pessoas atuam em projetos de desenvolvimento, saindo para outros locais após o término de um projeto e sendo substituídas por outras quando existe a necessidade de se realizar algum tipo de manutenção no sistema. A documentação de um sistema tem como objetivo permitir que o profissional que não participou de seu desenvolvimento possa conhecer, por meio dos diversos tipos de registros e diagramas produzidos, o que foi feito, possibilitando seu entendimento e a continuidade no trabalho sem perda de tempo para “descobrir as coisas”.

O diagrama de classes tem um importante papel nessa “transição” de profissionais, pois nele são registrados os relacionamentos entre as classes, além das características (atributos) e dos comportamentos (métodos) de cada uma delas, entretanto relacionar apenas cada um deles pode não produzir o conhecimento necessário, principalmente durante a etapa de desenvolvimento. Esse diagrama serve de estrutura para a elaboração da base de dados do sistema e como apoio ao desenvolvedor, sendo possível por meio dele permitir a criação ou a definição dos métodos que serão utilizados.

Vamos analisar a seguinte situação: o diagrama de classes é construído e repassado à equipe de desenvolvedores para que seja iniciada a construção dos códigos fontes que serão utilizados pelo sistema.

## **Essa situação é possível?**

Sim. No entanto, para isso é necessário que a equipe não conheça apenas quais são os métodos ou atributos definidos para a classe, mas também suas características físicas, como o tamanho, o tipo, os parâmetros de entrada e saída dos métodos etc. Essas informações estão contidas em um documento chamado Dicionário de Classes.

## Dicionário de Classes

A proposição da UML foi feita pela construção de padrões gráficos (diagramas), no entanto, para que outros aspectos associados à documentação possam ser apresentados, é necessário que outros documentos estejam associados a eles, permitindo o melhor entendimento ou a completude da informação.

O Dicionário de Classes não é um documento proposto pela UML, por esse motivo não existe um padrão de construção. Podemos observar que outros documentos estão nessa mesma situação, por exemplo, a Descrição dos Casos de Usos, dessa maneira, ao realizarmos buscas sobre o assunto, podemos encontrar diversas formas de representação, assim como é possível encontrar definições semelhantes com o nome de Dicionário de Dados. Outro aspecto a ser considerado na dinâmica adotada pelo mercado profissional para o desenvolvimento de sistemas em que documentar pode gerar “perda de tempo”, este documento possui um nível elevado de críticas por ser de difícil manutenção, no entanto, se for utilizada uma ferramenta automatizada para a construção do diagrama, essa dificuldade é minimizada.



### Importante

A principal diferença que podemos apontar entre Dicionário de Dados e Dicionário de Classes é que, no primeiro, apenas as informações sobre os dados são apresentadas, enquanto no segundo as informações referentes aos métodos também são descritas.

Considerando o fato de não existir um padrão de construção para o Dicionário de Classes, vamos propor um modelo que contenha as informações básicas necessárias para que o desenvolvedor possa criar os principais métodos definidos para as classes conhecendo previamente algumas das características físicas de cada um de seus elementos.

Figura 22: Documentar é preciso!



No Dicionário de Classes proposto, usaremos como referência o modelo apresentado no diagrama de classes, no qual uma classe é definida com seus atributos (características) e métodos (comportamentos), conforme Tabela 2.

As informações a serem preenchidas no Dicionário de Classes devem ter sido previamente especificadas no diagrama de classes, e ambos os documentos devem ter consistência entre eles, ou seja, deve haver um dicionário para cada classe, todos os atributos devem estar presentes com os mesmos tipos de dados e tamanhos definidos, assim como os métodos com os parâmetros e seu tipo, e o tipo de dado que será devolvido pelo método.

Tabela 2: Modelo de Dicionário de Classes proposto.

|                           |                  |             |                |
|---------------------------|------------------|-------------|----------------|
| <b>Classe:</b>            |                  |             |                |
| <b>Descrição:</b>         |                  |             |                |
| <b>Atributos</b>          |                  |             |                |
| <b>Nome</b>               | <b>Descrição</b> | <b>Tipo</b> | <b>Tamanho</b> |
|                           |                  |             |                |
| <b>Métodos</b>            |                  |             |                |
| <b>Nome</b>               |                  |             |                |
| <b>Descrição</b>          |                  |             |                |
| <b>Parâmetro Entrada:</b> |                  |             |                |
| <b>Retorno</b>            |                  |             |                |

Fonte: Elaborada pelo autor (2019).

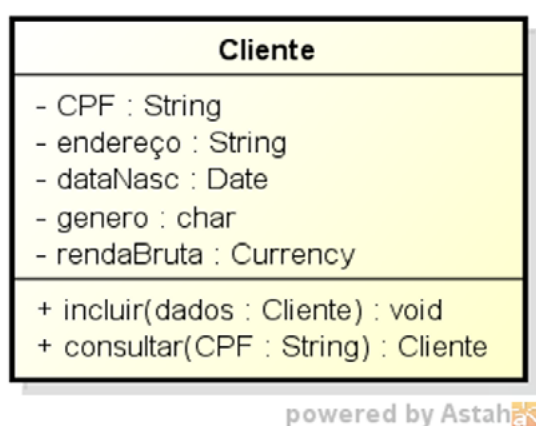
Analisando o modelo apresentado como Dicionário de Classes, faremos a divisão em três áreas:

- Área de identificação da classe: na qual são definidos o nome da classe e seu objetivo no sistema.
- Área de definição dos atributos: nessa área, será utilizada uma linha para cada atributo, nas quais serão relacionados o nome, o objetivo, o tipo e o tamanho, se necessário, para cada atributo definido para a classe.

- Área de definição dos métodos: nessa área, será utilizado um conjunto de quatro linhas para cada método, no qual em cada uma delas serão especificados o nome do método, o objetivo, a relação dos parâmetros de entrada e o tipo para cada um deles, e o tipo de retorno que será produzido pelo método.

Vamos exemplificar cada uma dessas áreas de forma separada tendo como base a classe “Cliente”, apresentada na Figura 23.

Figura 23: Definição da classe “Cliente” no diagrama de classes.



Fonte: Elaborada pelo autor (2019).

Vamos iniciar definindo a área de identificação da classe, na qual possuímos apenas duas informações: nome da classe e sua descrição (objetivo). Ao produzirmos a Dicionário de Classes de um sistema, devemos descrever todas as classes definidas no diagrama.

Tabela 3: Área de identificação da classe.

|   |
|---|
| <b>Classe:</b> Cliente.   |
| <b>Descrição:</b> Contém as informações referentes aos clientes da empresa. |

As informações de cada classe devem ser preenchidas da seguinte forma:

- Na linha referente ao nome da classe, transcrevemos aquele definida no diagrama.
- Na linha referente à descrição, faremos um breve resumo sobre essa classe. É importante destacar que, embora seja irrelevante essa informação, existem algumas situações em que características próprias da classe precisam ser registradas. Nes-



ses casos, podem utilizar esse local para fazer o registro ou criar uma linha para esse objetivo.

A área utilizada para a definição dos atributos possui a quantidade de linhas necessária para cada atributo. Dessa forma, essa área tenderá a ser extensa caso uma classe possua muitos atributos.

Tabela 4: Área para definição dos atributos da classe.

| Atributos  |   |          |         |
|------------|---|----------|---------|
| Métodos    |   |          |         |
| Nome:      | Descrição:  | Tipo     | Tamanho |
| CPF        | Número do CPF.<br>Deve ser validado na rotina Check11.                  | String   | 11      |
| endereço   | Endereço residencial.   | String   | 50      |
| dataNasc   | Data de nascimento do cliente.<br>O cliente deve ser maior que 18 anos. | Date     |         |
| gênero     | Gênero do cliente (M ou F).   | char     |         |
| rendaBruta | Renda familiar bruta.   | Currency | .       |

Para cada atributo, devem ser realizados os seguintes registros:

- Nome do atributo conforme definido no Dicionário de Classes. É indicado que seja utilizado esse nome na definição da classe durante sua implementação.
- O objetivo sucinto do atributo, acrescentando informações adicionais específicas, conforme exemplos dados.
- O tipo de atributo deve ser o mesmo utilizado na linguagem de programação e na base de dados.
- O tamanho, quando necessário, deve ser compatível com o que será definido na base de dados.

A área utilizada para a definição dos métodos possui um conjunto de quatro linhas para apresentar todas as informações sobre eles, sendo indicada a inserção de uma quinta linha em branco para separar um método de outro.

Tabela 5: Área para definição dos métodos da classe.

| Métodos              |   |
|----------------------|---|
| Nome:                | incluir.  |
| Descrição:           | Realiza a inclusão de uma instância na classe.                  |
| Paramêtros Entrada:  | Estrutura de dados da classe "Cliente".                         |
| Retorno:             | Sem retorno.  |
|                      |   |
| Nome:                | consultar.  |
| Descrição            | Recupera os dados de um cliente específico a partir de seu CPF. |
| Paârametros Entrada: | CPF do cliente (String).  |
| Retorno:             | Estrutura de dados da classe "Cliente".                         |

Para cada método, deve ser utilizada uma linha para cada uma destas informações:

- O nome do método deve ser o mesmo utilizado quando de sua implementação.
- A descrição deve representar seu objetivo, acrescido de outras informações se necessário.
- Os parâmetros de entrada devem ser definidos na sequência e de tipos compatíveis com a implementação do método.
- O tipo de retorno também deve ser compatível com a implementação.

É importante destacar que a identificação da necessidade de novos métodos surge na fase de implementação do software, dessa forma é imprescindível que tanto o diagrama como o Dicionário de Classes sejam atualizados sempre que nova informação surgir.



## MIDIAATECA

Para ampliar o seu conhecimento, veja o material complementar da Unidade 3 disponível na midiateca.



## NA PRÁTICA

O diagrama de classes de negócio de um sistema representa um dos diagramas mais importantes daqueles propostos pela UML. Por meio dele, é possível identificar a relação entre as classes de negócio, sendo este o ponto de partida para a construção da estrutura dos dados que serão armazenados na base de dados. Para permitir o melhor entendimento da relação entre as classes e a futura construção da base de dados, é produzido um Dicionário de Classes, no qual é feito o registro dos objetivos de cada característica e dos comportamentos que compõem as classes.

## Resumo da Unidade 3

Nesta unidade você construiu o diagrama de classes, a partir dos casos de usos identificados e demais informações coletadas na fase de levantamento de requisitos, e o Dicionário de Classes a partir desse diagrama. É essencial destacar nessas construções a importância da consistência entre elas e a documentação produzida até este momento.

O projeto de desenvolvimento está “caminhando”, e a preocupação com o produto (software) deve existir antes de iniciar sua implementação. A consistência entre os diagramas e os demais documentos produzidos é importante para que o resultado possa ser o esperado pelo usuário. Cada documento produzido tem como base outros já elaborados e irá servir como referência para etapas futuras do desenvolvimento, daí a necessidade de mantê-los sempre integrados, evitando distorções que poderão impactar o produto. Repetindo o que já foi exposto anteriormente, a documentação é algo que muitas vezes é considerado como “perda de tempo”, no entanto ela deve ser entendida como um “investimento de tempo”, que somente será percebido quando houver a necessidade de realizar alguma “manutenção” (ajuste) no software por profissionais que não participaram de seu desenvolvimento. Entender o que foi feito por alguém que não conhecemos sem a documentação correta seria como procurar algo sem termos a menor noção de onde poderia estar: vamos demorar um tempo muito maior sem a devida certeza sobre a qualidade do resultado.

Dessa forma, a UML propõe um mínimo de documentação por meio dos diagramas e dá a cada pessoa/empresa a oportunidade de criar seu próprio modelo de documentação complementar baseado em suas necessidades e dinamismo. Assim, nesta unidade conhecemos um dos principais diagramas propostos; na próxima, conheceremos outros diagramas que ajudarão em diferentes etapas do desenvolvimento..



### CONCEITO

Nesta unidade foram estudados três conceitos importantes:

- A construção do **diagrama de classe**.

- A **identificação dos relacionamentos entre as classes** representadas no diagrama de caso de uso.
- A documentação das características e dos comportamentos das classes por meio de um modelo proposto como **Dicionário de Classes**.

# Referências

DIAGRAMA DE CLASSES. In: WIKIPEDIA: the free encyclopedia. [San Francisco, CA: Wikimedia Foundation, 2010]. Disponível em: <http://en.wikipedia.org/wiki/Laparotomia>. Acesso em: 18 mar. 2019.

FIGUEIREDO, E. **Relacionamentos do diagrama de classes**. Disponível em: [https://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/uml-diagrama-classes-relacionamentos\\_v01.pdf](https://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/uml-diagrama-classes-relacionamentos_v01.pdf). Acesso em: 20 out. 2019.

GOMES, J. **Aula 5 - Dicionário de Dados**. 19 nov. 2014. Disponível em: <https://pt.slideshare.net/janynnegomes/aula-5-41778197>. Acesso em: 20 out. 2019.

MACORATTI, J. C. **UML – Diagrama de classes e objetos**. Disponível em: [http://www.macoratti.net/net\\_uml1.htm](http://www.macoratti.net/net_uml1.htm). Acesso em: 20 out. 2019.

SIGNIFICADOS. **Significado de diagrama de classes**. 11 jan. 2018. Disponível em: <https://www.significados.com.br/diagrama-de-classes/>. Acesso em: 20 out. 2019.

VENTURA, P. **Entendendo o diagrama de classes da UML**. 16 jul. 2018. Disponível em: <https://www.ateomomento.com.br/uml-diagrama-de-classes/>. Acesso em: 20 out. 2019.

## UNIDADE 4

Diagramas da UML para  
a construção do modelo  
conceitual do sistema

# INTRODUÇÃO

A UML propõe, com seus diagramas, não apenas um padrão de documentação, pois ela possibilita um padrão de comunicação entre os membros da equipe de desenvolvimento. No entanto, o objetivo em construí-los significa muito mais do que documentar, significa registrar padrões de comportamentos dos objetos das classes, sejam eles por meio da comunicação ou da representação de seus estados, possibilitando identificar ações antes da etapa de implementação. Três diagramas do modelo comportamental se destacam neste objetivo:

- Diagrama de Sequência, que representa a comunicação entre os objetos, permitindo identificar alguns dos métodos que serão implementados.
- Diagrama de Transição de Estado, que representa os estados de um objeto e as transições que provocam mudanças no estado, permitindo que estas transições sejam previamente identificadas e previstas no seu desenvolvimento.
- Diagrama de Atividade, que apresenta as atividades que ocorrem no modelo a ser representado.

Combinando esses diagramas, é possível identificar alguns dos métodos necessários para a construção do sistema antes de iniciar sua implementação, minimizando a possibilidade de retrabalhos nesta etapa.





## OBJETIVO

Nesta unidade serão estudados três conceitos importantes:

- A construção do Diagrama de Sequência, implementando o padrão MVC como padrão de desenvolvimento do software.
- A construção do Diagrama de Transição de Estado, apresentando os diferentes estados de uma instância de classe e os eventos que realiza cada uma das transições.
- A construção do Diagrama de Atividade para representar a sequência com que as atividades compõem o sistema ou casos de usos representados no sistema.

# O Diagrama de Sequência

O Diagrama de Sequência representa um dos diagramas propostos no modelo comportamental da UML e representa a troca de mensagens entre os objetos. Detalhando um pouco estes conceitos, entendemos como troca de mensagens a comunicação entre os objetos ou a maneira como um método aciona outro método definido em outra classe ou camada. O Diagrama é construído representando a sequência com que as ações são realizadas (sequência temporal). Dessa forma, a base para a sua construção pode ser a descrição do caso de uso que está sendo representada, onde os fluxos principal e alternativos já foram descritos.

Logo, podemos resumir que o Diagrama de Sequência representa:

- A sequência com que as ações são realizadas.
- As mensagens enviadas entre os objetos.
- Os métodos que são acionados.
- As classes que interagem no caso de uso.

Embora o Diagrama de Sequência possa ser utilizado para realizar outros tipos de representação, o seu uso é indicado para documentar cada um dos casos de uso existente no sistema, definindo um diagrama para cada um deles. O diagrama deve ser iniciado a partir do ator indicado no diagrama de caso de uso e, a partir daí, apresentar as demais interações.

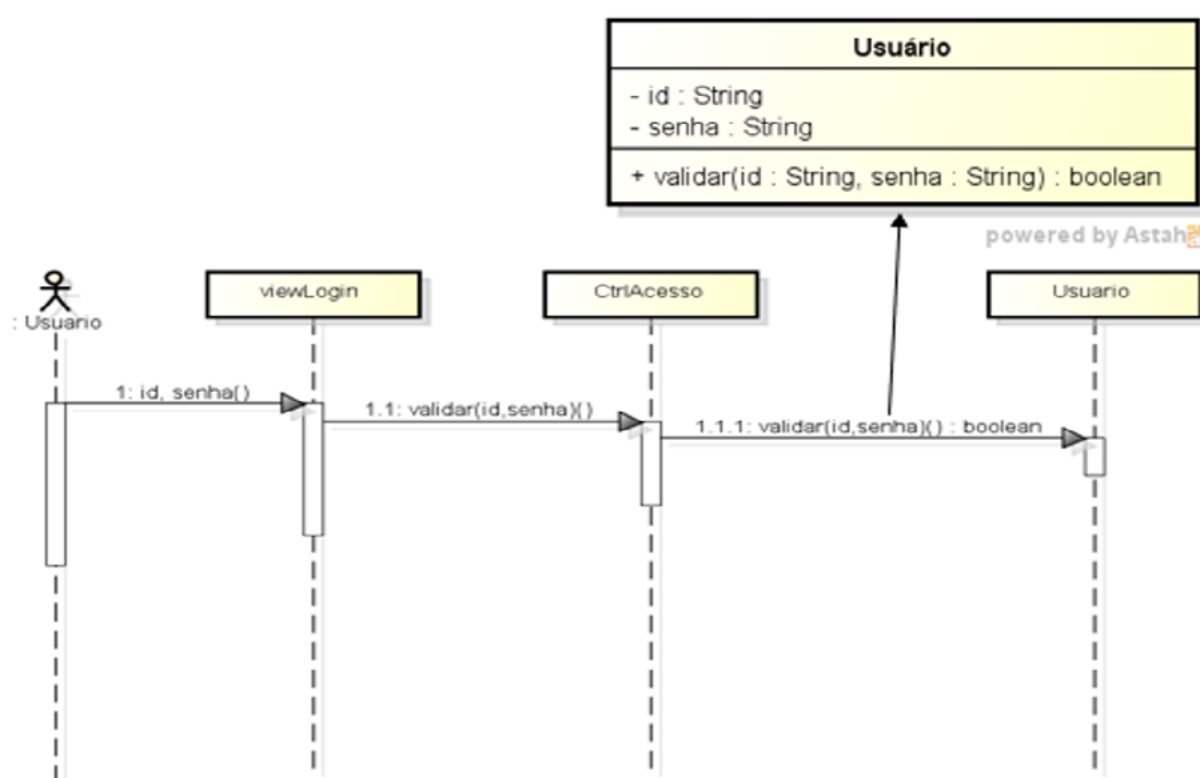


## Importante

Embora haja diferentes maneiras de representar este diagrama, nesta Unidade será adotado o padrão MVC (Model-View-Control) como forma de representação. Neste padrão, toda interação do ator será feita através da interface (View), que se comunicará exclusivamente com a classe de controle (Control). Esta classe terá o objetivo único de interagir entre a interface e a camada de negócio (Model), com o objetivo de “isolar” o negócio das interfaces, permitindo o seu reaproveitamento em outras aplicações quando necessário. As classes de negócios são as representadas no Diagrama de Classe.

A construção de um Diagrama de Sequência deve ser feita a partir da ação do ator com a interface e, a partir daí, representar as comunicações entre as camadas. A passagem do tempo é percebida através da direção vertical no sentido de cima para baixo. Quanto mais abaixo uma mensagem aparece no diagrama, mais tarde no tempo ela foi enviada. Embora nem sempre seja possível, deve-se tentar arrumar os objetos no Diagrama de Sequência de acordo com a ordem na qual as mensagens são enviadas, procurando minimizar o cruzamento de setas de mensagens com linhas de vida, embora a ordem horizontal com que os elementos são definidos no diagrama não tenha nenhum significado predefinido. Um dos aspectos que devem ser representados no diagrama é a responsabilidade da classe que possui o método representado pela mensagem, conforme exemplo apresentado na Figura 1.

Figura 1: Exemplo Diagrama de Sequência.



Fonte: Elaborada pelo autor (2019).

A Figura 1 apresenta um trecho de um caso de uso onde a interação do ator “Usuário” é feita através de uma tela de Login, que interage com a camada de controle para que seja acionado o método “Validar”, cuja responsabilidade é da classe “Usuário”, conforme representação feita na sua definição no diagrama de classe.

## Principais elementos gráficos que compõem o Diagrama de Sequência

- **Ator** – Representa o elemento externo que aciona o caso de uso, sendo utilizado o mesmo símbolo do diagrama de caso de uso.



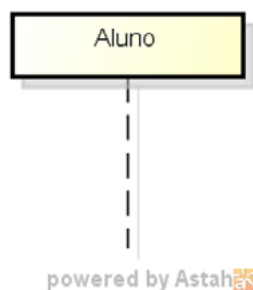
- **Objeto** – Representa a instância de uma classe utilizada no caso de uso.



- **Classes** – A representação de uma classe no diagrama é semelhante à definição de um objeto, porém o nome da classe não é sublinhado.



- **Linhas de vida** – No Diagrama de Sequência cada objeto aparece no topo de uma linha vertical tracejada. Essa linha é denominada linha de vida.



- **Mensagens** – A notação para uma mensagem em um Diagrama de Sequência é uma flecha horizontal ligando uma linha de vida a outra. O formato da ponta da seta indica o tipo de mensagem que está sendo enviada.

– **Mensagem síncrona:** o objeto chamador deve esperar até que a mensagem seja concluída.



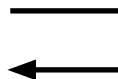
– **Mensagem assíncrona:** o objeto chamador pode continuar o processamento e não precisa esperar por uma resposta.



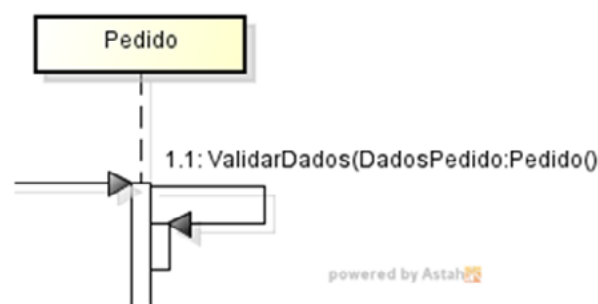
– **Mensagem de retorno:** o objeto chamado responde à mensagem com algum tipo de retorno ou resposta. Dependendo do tamanho do caso de uso, a sua representação em um diagrama de sequência pode se tornar também grande, dificultando o seu entendimento ou clareza dos componentes, por este motivo, nem sempre é indicado o uso da mensagem de retorno, por torná-lo ainda maior.



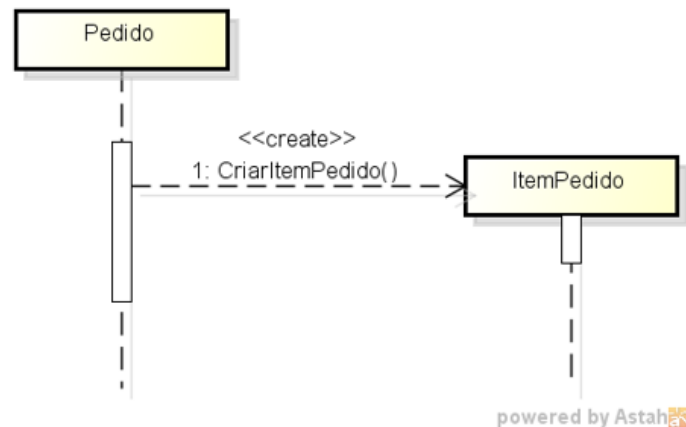
**Mensagem reflexiva ou recursiva (autochamada):** quando um objeto envia uma mensagem para si próprio.



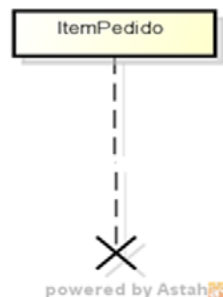
- **Focos de controle** – Correspondem a blocos retangulares posicionados sobre a linha de vida de um objeto. Um foco de controle representa o tempo em que um objeto realiza uma ação. O topo do foco de controle coincide com o recebimento de uma mensagem. A parte de baixo de um foco de controle coincide com o término de uma operação realizada pelo objeto. Chamadas recursivas provocam barras de controles empilhadas.



- **Criação de objetos** – Para criar um objeto, você desenha a seta de mensagem diretamente para a caixa do objeto. Se o objeto existe desde o início da interação, ele deve ser posicionado no topo do diagrama. Por outro lado, se o objeto é criado em um momento posterior, o seu retângulo deve ser posicionado mais abaixo no diagrama. A criação de um objeto pode ser requisitada por outro objeto através de uma mensagem. Normalmente, esta mensagem de criação é rotulada com o nome “criar”.



- **Destruição de objetos** – A destruição de um objeto é representada por um X grande. Um objeto normalmente é destruído quando ele não é mais necessário na interação.

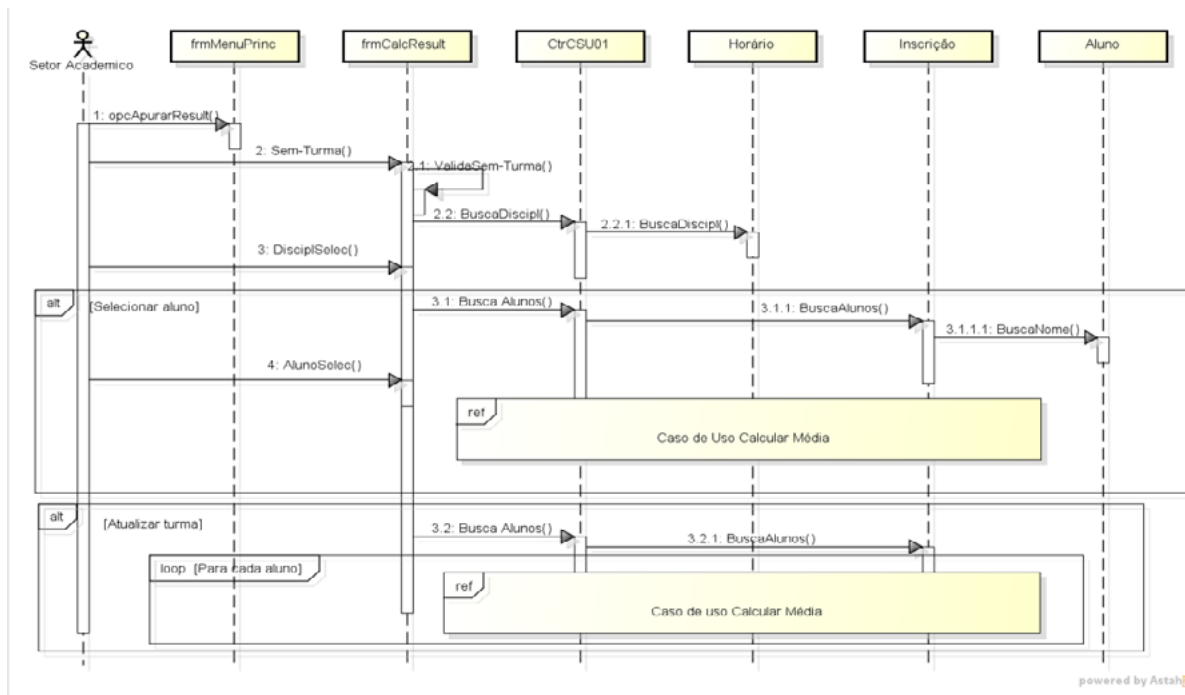


- **Laços, condicionais etc.** – A UML 1 utilizava marcadores de iteração e sentinelas. Um marcador de iteração é um asterisco (\*) adicionado ao nome da mensagem. Você pode adicionar um texto entre colchetes para indicar a base da iteração. As sentinelas são expressões condicionais colocadas entre colchetes e indicam que a mensagem é enviada somente se a sentinela é verdadeira. Embora os marcadores de iteração e as sentinelas (condições de guarda) possam ajudar, eles têm suas fraquezas. Por esse motivo, a UML 2 oferece os quadros de interação.

Tabela 1: Elementos que compõem um Diagrama de Sequência.

| Operador | Significado  |
|----------|--|
| alt      | Múltiplos fragmentos alternativos; somente aquele cuja condição for verdadeira será executado.   |
| opt      | Opcional; o fragmento é executado somente se a condição fornecida for verdadeira. Equivalente a um alt, com apenas um caminho.   |
| par      | Paralelo; cada fragmento é executado em paralelo.  |
| loop     | Laço; o fragmento pode ser executado várias vezes e a sentinela indica a base da iteração.   |
| sd       | Diagrama de Sequência; usado para circundar um Diagrama de Sequência inteiro, se você quiser.  |
| ref      | Referência; refere-se a uma interação definida em outro diagrama. O quadro é desenhado de forma a abordar as linhas de vida envolvidas na interação. Você pode definir parâmetros e um valor de retorno. |

Figura 2: Diagrama de Sequência com alguns dos elementos apresentados.



Fonte: Elaborada pelo autor (2019).

## **Etapas para a construção de um Diagrama de Sequência**

Para elaborar um Diagrama de Sequência é indicado que sejam realizados os seguintes procedimentos:

- Defina o contexto para a interação: se é um sistema, subsistema, operação ou classe ou um cenário de caso de uso ou colaboração.
- Defina o estágio para a interação, identificando quais objetos desempenham um papel na interação. Distribua-os no Diagrama de Sequências da esquerda para a direita.
- Defina a linha de vida para cada objeto. Na maioria dos casos, os objetos persistirão ao longo de toda a interação. Para aqueles objetos criados e destruídos durante a interação, defina suas linhas de vida, conforme seja apropriado, e indique explicitamente seu nascimento e morte com as mensagens adequadamente estereotipadas.
- Começando com as mensagens que iniciam a interação, distribua cada mensagem subsequente de cima para baixo entre as linhas de vida, mostrando as propriedades de cada mensagem (como seus parâmetros), conforme necessário, para explicar a semântica da operação.



# O Diagrama de Transição de Estado

Esse diagrama pode ser encontrado na literatura com diferentes nomenclaturas, tais como: Diagramas de Estado, Diagramas de Transição de Estados, Diagrama de Máquina de Estado e Diagramas de Gráficos de Estados, sendo todas estas identificações representadas pelo mesmo diagrama. No documento original da UML, ele é identificado como *State Machine Diagram*, que, traduzindo fielmente, representa “Diagrama de Máquina de Estado”, entretanto, o nome mais comum utilizado é o Diagrama de Transição de Estado (DTE), sendo o nome que iremos tratá-lo neste material.

O Diagrama de Transição de Estado representa um dos diagramas do modelo comportamental apresentados pela UML. Ele apresenta os diversos “estados” de uma instância da classe a partir do seu ciclo de vida, ou seja, esta definição é dada por meio das diversas situações (estados) que um objeto pode ter desde a sua criação na classe, representado pelo estado inicial, até ele não ser considerado como instância processável, ou seja, uma instância não mais utilizada pela aplicação, tais como: inativa, cancelada etc., representando o seu estado final.



## Exemplo

Podemos relacionar os estados de uma instância da classe funcionário:

- Ativo: quando o funcionário está trabalhando.
- Licenciado: quando ele está afastado por alguma razão não prevista.
- Férias: quando ele está em férias.
- Desligado: quando ele não faz mais parte do quadro de funcionários da empresa.

A construção de um DTE tem como objetivo identificar estes estados e as ações que devem ser previstas no sistema para produzir a mudança de um estado para outro, chamadas de transição. Estas ações são definidas através de eventos internos ou externos ao sistema, que devem ser previstas no sistema para que haja estas mudanças. Desta forma, a construção deste diagrama não tem como objetivo apenas melhorar a documentação do sistema, mas também minimizar a possibilidade de “esquecer” algum destes eventos antes de iniciarmos a fase de implementação.

Nem todas as classes possuem Estado. Dessa forma, o DTE é construído apenas para aquelas que representam estas mudanças. Algumas delas possuem apenas dois estados: Ativo e Inativo, por exemplo, representando o uso ou não desta instância pela aplicação, entretanto, outras possuem um conjunto maior de estados, como os diferentes estados de uma compra (aguardando autorização de pagamento, em separação no estoque, em processo de entrega etc.) e existem classes que podem não possuir estado, como desejar manter um registro de todos os clientes que já realizaram operações com a empresa, não havendo preocupação com o tempo da última operação. Neste exemplo, para o negócio, o cliente será sempre ativo, não havendo necessidade de representar este estado e, conseqüentemente, construir o diagrama.

Portanto, podemos fazer algumas afirmações sobre o DTE:

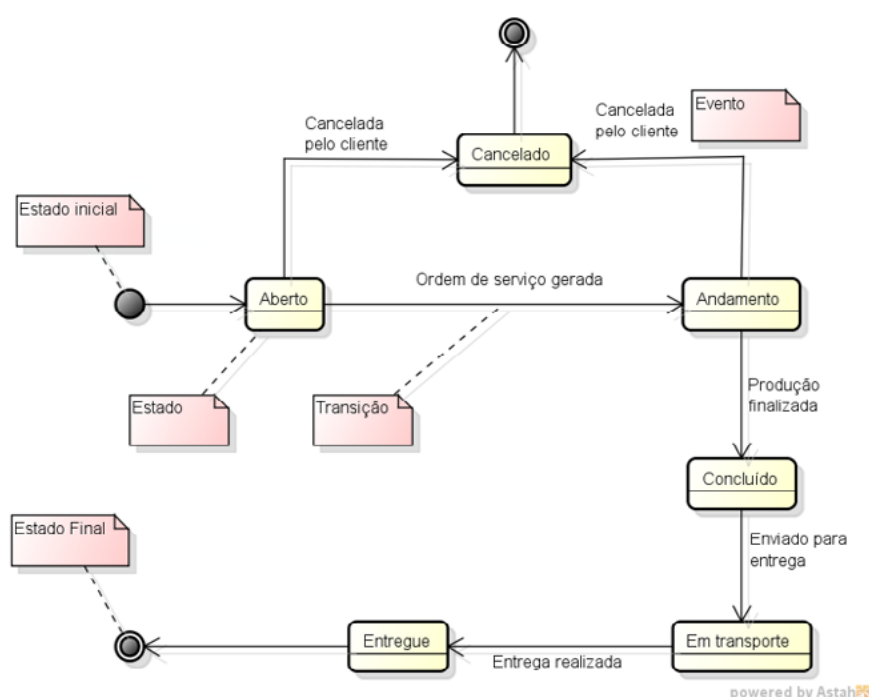
- Ele representa a mudança de estado de um objeto.
- Cada objeto se encontra em um estado particular dentro do seu ciclo de vida.
- Um objeto muda de estado quando ocorre algum evento interno ou externo ao sistema, chamado de transição de estado.
- Quando um destes eventos acontece, significa que um objeto da classe teve o seu estado alterado.
- A partir da análise das transições entre os estados de um objeto, podem-se prever as operações necessárias para que o evento seja realizado.
- Cada DTE descreve o comportamento dos objetos de uma única classe.
- O DTE é construído apenas para as classes que possuem estados conhecidos cujo comportamento é afetado e modificado pelos diferentes estados.
- O DTE descreve o ciclo de vida dos objetos de uma classe, os eventos que causam a transição de um estado para outro e as operações que provocam estas transições.

## Principais elementos gráficos do diagrama

- Estado inicial – Representado por um círculo escuro que aponta para o primeiro estado da instância. Toda classe deve ter apenas um único estado inicial.
- Estado final – Representado por um círculo com um ponto no meio, sendo este símbolo indicado pelo estado final da classe, considerando que uma classe pode ter vários estados finais.
- Representação de um estado – Representado por um retângulo com a identificação do estado descrita no seu interior.
- Transição – Uma seta que indica a transição de um estado para outro, com o nome do evento que promove esta transição.

A Figura 3 apresenta um DTE da classe “Venda” de uma empresa que produz seus próprios produtos. Analisando o diagrama, observamos um único estado inicial (Aberto), como sempre deve ser, e dois estados que indicam o término das transições (Cancelado e Entregue). Uma estância da classe Venda pode passar por seis estados diferentes (Aberto, Andamento, Cancelado, Concluído, Em transporte e Entregue), cujos eventos que realizam a transição de um estado para outro são identificados através da seta de transição.

Figura 3: DTE com elementos básicos.



Fonte: Elaborada pelo autor (2019).

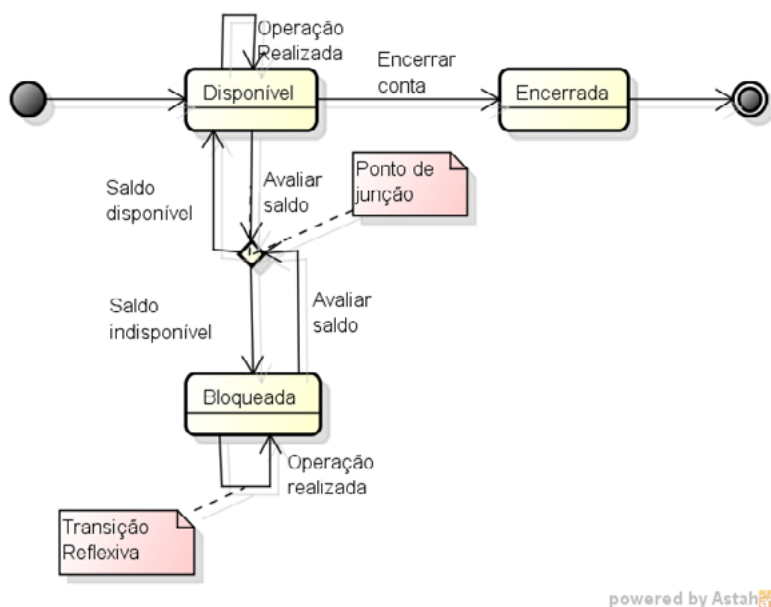
## Outros elementos gráficos

- Transição reflexiva – Uma transição cuja origem e destino são representados pelo mesmo estado.
- Ponto de junção – Representado pelo símbolo de um losango, é utilizado quando o estado a ser assumido varia de acordo com uma condição, chamada de condição de guarda.

A Figura 4 apresenta o DTE da classe “Conta Bancária” que apresenta três estados: Disponível, Bloqueada e Encerrada, sendo o estado Disponível o inicial. A cada operação

realizada na conta ela pode alterar o seu estado para Bloqueada ou Disponível, de acordo com o valor do seu saldo avaliado na condição de guarda. Ao realizar o evento “Encerrar conta” a conta se torna Encerrada, sendo este o seu estado final.

Figura 4: DTE com transição reflexiva e ponto de junção.



Fonte: Elaborada pelo autor (2019).

## Eventos

Os eventos representam ações que ocorrem no objeto da classe e provocam mudanças no seu estado. Para melhor documentação do diagrama, as transições e suas respectivas ações devem ser apresentadas em uma Tabela de Transição de Estado, conforme Tabela 2, representando a classe Conta Bancária.

Tabela 2: Tabela de Transição de Estado da classe Conta Bancária.

| Evento            | Transição  |
|-------------------|--|
| Realizar Operação | Cada operação de débito ou crédito realizada na conta. |
| Encerrar Conta    | Solicitação do cliente para encerramento da conta.     |



### Importante

A UML propõe apenas padrões em diagramas, desta forma, a tabela de transição de estado não é sempre apresentada na literatura, e quando ocorre é feita de diferentes maneiras.

## Etapas para construção de um DTE

Sistematizando o processo de construção dos Diagramas de Transição de Estado podemos seguir as seguintes ações:

- Identificar as classes que tenham estados conhecidos.
- Identificar os estados relevantes para estas classes.
- Identificar os eventos relevantes aos objetos das classes. Para cada evento, identificar as transições que ele ocasiona.
- Para cada estado, identificar as transições possíveis quando um evento relevante ocorrer.
- Para cada estado, identificar os eventos internos e ações correspondentes relevantes.
- Para cada transição, verificar se há fatores que influenciam o seu disparo, definindo a condição de guarda e as ações que devem ser executadas, se necessário.
- Para cada condição e ação, identificar os atributos e ligações envolvidos.
- Definir o estado inicial e final, se for o caso.
- Montar o diagrama, posicionando os estados de tal forma que o ciclo de vida possa ser visto de cima para baixo e da esquerda para a direita.

# O Diagrama de Atividade

O Diagrama de Atividade representa mais um dos diagramas do modelo comportamental propostos pela UML a ser apresentado neste curso. Ele define a sequência com que as ações, identificadas como atividades, são realizadas dentro de um aspecto temporal, ou seja, na ordem com que elas ocorrem. Ele possui um conjunto simples de elementos gráficos, sendo este fato um aspecto importante para que as pessoas da área de negócio consigam entender a forma como o sistema funciona ou irá funcionar.



## Importante

O Diagrama de Atividade possui um conjunto de elementos gráficos simples semelhantes ao Fluxograma, no entanto, ele é mais recente e contempla outros elementos.

O nível de detalhamento das atividades varia de acordo com o que se quer representar, desta forma, você pode representar como sendo uma atividade simples, como um comando, ou complexa, representando um módulo de um sistema ou um caso de uso. Devemos entender atividade como sendo uma ação que demanda um tempo para sua realização, podendo ser curto ou longo dependendo do que é necessário. O diagrama não detalha como a atividade é realizada, podendo ser definida em um documento complementar, mas sim, quais são elas, a sequência com que são realizadas além de aspectos relacionados a condições que podem ocorrer na sua operação.

## Quando utilizar o Diagrama de Atividade

A princípio esse diagrama pode ser utilizado em diferentes situações, sendo, entretanto, mais ou menos indicado para cada uma delas. Podemos citar algumas dessas situações, lembrando que, sempre que for preciso detalhar algum aspecto específico do software que contenha um conjunto de atividades, ele pode ser utilizado.

- Uma das principais situações em que ele pode ser utilizado é quando existe a necessidade de detalhar um processo complexo que compõe o software, seja ele um

aspecto funcional ou um processo do negócio, cuja necessidade de um melhor entendimento por parte da equipe de desenvolvimento evite que ele seja desenvolvido de forma equivocada.

- Outra situação importante é para apresentar o sistema como um todo, onde os casos de usos identificados ou módulos do sistema são colocados em sequência, mostrando eventuais dependências existentes entre eles. Lembrando que no Diagrama de Caso de Uso eles são apresentados sem nenhuma relação temporal.

Existem outras situações em que o Diagrama de Atividade pode ser utilizado, entretanto, é indicado o uso de outras ferramentas ou documentos. Nesta situação podemos apontar:

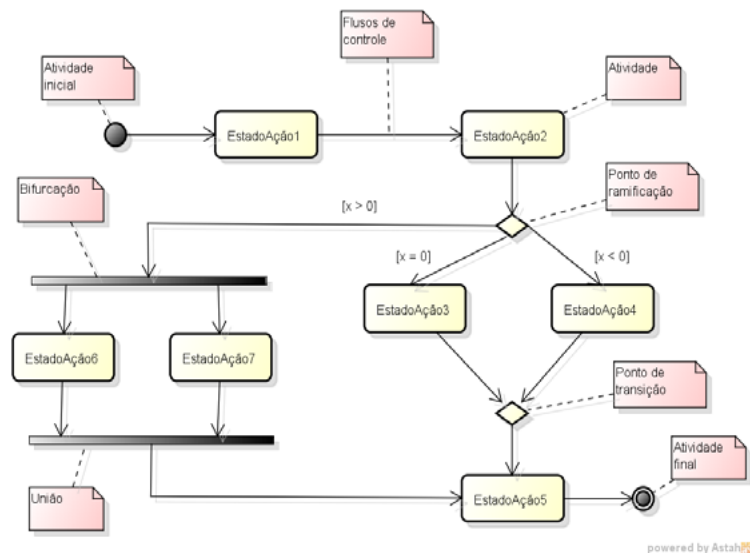
- Mapear processos – Embora o mapeamento de processos apresente atividades e elas possuam uma sequência de execução, é indicado que sejam utilizadas ferramentas específicas para este objetivo, como aquelas que utilizam os diagramas com notação BPMN (Business Process Model and Notation). Por serem próprias para este objetivo, elas realizam operações específicas, como as validações do processo, que o Diagrama de Atividade não faz, trazendo maior qualidade ao produto.
- Descrever caso de uso – Definir um caso de uso com o Diagrama de Atividade significa realizar os mesmos procedimentos da Descrição do Caso de Uso, sendo que um em formato textual e o outro de forma gráfica. O nível de detalhamento do diagrama seria muito pequeno, uma vez que as atividades seriam representadas pelos comandos do caso de uso. A construção deste diagrama para todos os casos de usos pode representar um retrabalho tanto no momento do desenvolvimento quanto na manutenção.

## Principais elementos gráficos do diagrama

- Atividade inicial – Representada por um círculo escuro que aponta para a primeira atividade que se quer representar.
- Atividade final – Representada por um círculo com um ponto no meio, sendo este símbolo indicado pela última atividade que se quer representar, considerando que pode haver várias atividades que encerram o diagrama.
- Atividade – Representada por um retângulo com a identificação da ação descrita no seu interior.
- Fluxos de controle – Uma seta que indica a mudança de uma atividade para outra, podendo ou não realizar uma identificação.
- Decisão ou ponto de ramificação – Representada pelo símbolo de um losango, é utilizado quando existe uma condição a ser avaliada que pode mudar o fluxo das atividades representadas no diagrama.

- Ponto de transição – Representado pelo símbolo de um losango, é utilizado quando duas ou mais atividades realizadas de forma exclusiva se unem para um fluxo único.
- Bifurcação e União – Representada por uma barra vertical que representa o início de atividades que podem ser realizadas em paralelo, através de fluxos paralelos (bifurcação) ou o término destas atividades (união) voltando a um fluxo único.

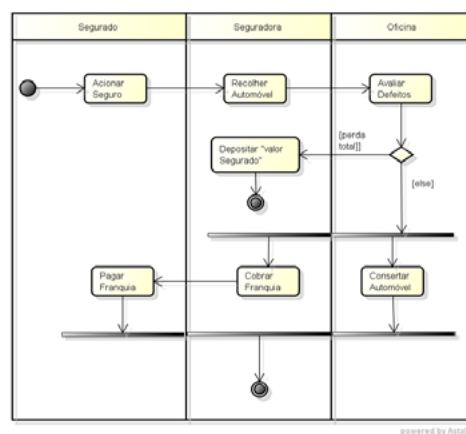
Figura 5: Exemplo de um diagrama de atividade.



Fonte: Adaptado de Bezerra (2007, p. 229).

- Raia – Originária das “raias das piscinas”, são fronteiras, verticais ou horizontais, que representam as separações entre módulos, funcionalidades ou atores que interagem com a atividade.

Figura 6: Exemplo Diagrama de Atividade com Raia



Fonte: Adaptado de Bezerra (2007, p. 231).



## Etapas para construção de um Diagrama de Atividade

Sistematizando o processo de construção dos Diagramas de Atividades, podemos seguir as seguintes ações:

- Conhecer o que será representado.
- Identificar as atividades existentes.
- Sequenciar as atividades e aquelas que podem ser realizadas em paralelo.
- Identificar as condições existentes e as situações previstas em cada uma delas.
- Identificar os atores envolvidos para representação nas raias.
- Identificar a atividade inicial.
- Identificar a(s) atividade(s) final(is).
- Construir o diagrama.



### MIDIA TECA

Para ampliar o seu conhecimento, veja o material complementar da Unidade 4 disponível na midiateca.



### NA PRÁTICA

No desenvolvimento de um sistema, a documentação, muitas vezes, não é considerada como prioritária em razão da necessidade de maior eficiência (produtividade) na entrega das funcionalidades do sistema para o usuário. Quando uma empresa identifica a necessidade de produzir documentação, alguns diagramas da UML são utilizados, como o Diagrama de Atividade na etapa de levantamento; o Diagrama de Transição de Estado para verificar se todos os

eventos necessários para a transição dos estados da instância de uma classe foram cumpridos, evitando a possibilidade de inconsistência em razão de eventuais ausências destes eventos; e o Diagrama de Sequência para representar a comunicação entre os diferentes tipos de classes e o padrão de desenvolvimento a ser utilizado na etapa de implementação.

## Resumo da Unidade 4

Nesta unidade você estudou três diagramas do modelo comportamental proposto pela UML, que proporciona compartilhamento mais adequado de informações sobre o sistema, seja na produção da documentação como no apoio ao processo de desenvolvimento de software. Para realizar este apoio, a adoção destes diagramas irá permitir melhor comunicação entre os membros da equipe no que se refere ao registro das atividades ou comportamentos identificados pelos objetos das classes ou pela própria aplicação, facilitando identificar necessidades que irão compor o sistema.

Um dos maiores problemas no desenvolvimento de um Sistema de Informação está associado ao retrabalho. É importante ressaltar que os novos padrões de desenvolvimento visam agilizar a apresentação de resultados ao cliente, principalmente no que se refere à implantação de funcionalidades do software com espaços de tempo pequeno, no entanto, agilizar entregas não significa abandonar documentação. Documentar um sistema sempre foi, e assim o será no futuro, a melhor maneira de entender o que foi feito por outra pessoa. As primeiras técnicas de desenvolvimento não priorizavam a documentação, no entanto, ao verificar a dificuldade em se manter funcionando sistemas já implantados quando alguma alteração ou nova funcionalidade precisava ser feita, verificou-se que o tempo gasto para entender o sistema era muito maior do que se fosse feita uma nova versão, entretanto, esta nem sempre é a melhor situação.

As equipes de desenvolvimento de sistemas devem saber que, embora se gaste tempo em construir documentação, esse tempo será reaproveitado no futuro, e, aos gestores, é importante terem o conhecimento de que agilidade de entrega não deve ser sinônimo de pressa, pois esta nem sempre está associada à qualidade.



### CONCEITO

Nesta unidade foram estudados três conceitos importantes:

- A construção do Diagrama de Sequência, implementando o padrão MVC como padrão de desenvolvimento do software;

- A construção do Diagrama de Transição de Estado, apresentando os diferentes estados de uma instância de classe e os eventos que realiza cada uma das transições;
- A construção do Diagrama de Atividade para representar a sequência com que as atividades compõem o sistema ou casos de usos representados no sistema.

# Referências

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 2. ed. Rio de Janeiro: Campus-Elsevier, 2007.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML – Guia do Usuário**. Rio de Janeiro: Campus, 2000.

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

GONÇALVES, G. M. D. **Diagramas de Transição de Estados**. Disponível em: <https://www.devmedia.com.br/diagramas-de-transicao-de-estados-engenharia-de-software-30/18444>. Acesso em: 30 nov. 2019.

PAULA FILHO, W. P. **Engenharia de Software**: Fundamentos, Métodos e Padrões. Rio de Janeiro: LTC, 2003.

SILVA, P. C. B. **O Diagrama de Sequência**. Disponível em: <https://www.devmedia.com.br/artigo-sql-magazine-64-utilizando-uml/12665>. Acesso em: 30 nov. 2019.

VENTURA, P. **Entendendo o Diagrama de Sequência da UML**. Disponível em: <https://www.ateomomento.com.br/diagrama-de-sequencia-uml/>. Acesso em: 30 nov. 2019.

VENTURA, P. **Entendendo o Diagrama de Atividades da UML**. Disponível em: <https://www.ateomomento.com.br/uml-diagrama-de-atividades/>. Acesso em: 30 nov. 2019.

