Seção 3

**Métodos orientados a objetos**

Caro aluno, nesta seção estudaremos a linguagem que unificou os métodos para modelagem de software orientado a objetos, a UML (Linguagem Unificada de Modelagem ou *Unified Modeling Language*); Essa linguagem permitiu aos desenvolvedores e engenheiros de software **um padrão único** para elaborar a modelagem de programas e projetos de desenvolvimento.

Iniciamos esta unidade com o estudo do paradigma orientado a objeto, **que revolucionou** a forma de pensar o desenvolvimento do código. Porém, a análise estruturada com seus DFDs (diagramas de fluxo de dados) e outras ferramentas não atendiam às novas necessidades da A/POO (Análise e Projeto Orientados a Objetos). O processo unificado **(PU) e o RUP** (Processo Unificado Racional) vieram ao encontro dessa necessidade, apresentando um método bastante robusto para acompanhar **todo o ciclo** de desenvol­vimento e da vida de um software.

Além de conhecer os principais diagramas de UML, você também terá uma breve introdução do que vêm a ser **métodos ágeis** e quais os mais utilizados atualmente.

Para que você possa aplicar na prática os conhecimentos com os quais terá contato nesta seção, você poderá unir a metodologia de processo unificado (PU) e a UML (*Unified Modeling Language*) na continuidade da elaboração do projeto de software para a fábrica de bolos do projeto “Empreendedores de Sucesso”. Nessa etapa, você deverá **escolher os diagramas** necessários para modelar o sistema, a partir do esboço do caso de uso inicial apresen­tado na Figura 4.8, **e elaborar** os refinamentos necessários, desenvolvendo os diagramas UML que forem essenciais para conclusão e apresentação do projeto ao cliente.

Assim, manter uma documentação precisa e atualizada desse projeto de software será muitíssimo importante para garantir sua qualidade e manutenção.

**A Empresa Rational Software decidiu reunir os métodos de Booch, Jacobson e Rumbaung e criaram a UML**

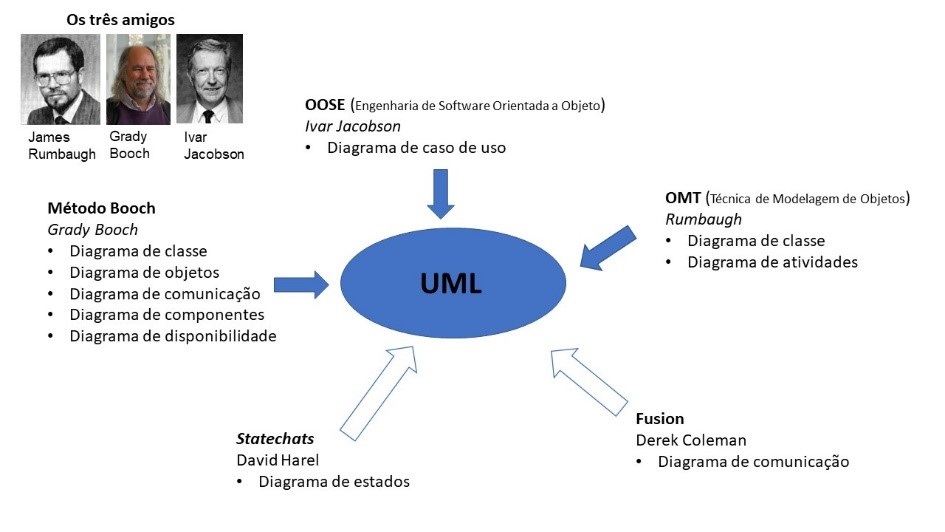
Surgia, assim, um padrão para modelagem de sistemas, que passou a ser utilizada pela maioria das empresas; Surgia, assim, um padrão para modelagem orientada a objetos;

**A UML faz uso de uma linguagem gráfica,** o que nos permite visualizar com mais facilidade os objetos e suas interações (relacionamentos), bem como construir, especificar e documentar os **artefatos** gerados por um software.

Basicamente a UML é composta de diagramas. UML (Unified Modeling Language) é uma família de notações gráficas, apoiada por um metamodelo único, particularmente daqueles construídos utili­zando o estilo orientado a objetos

As especificações da UML são controladas pela OMG (*Object Management Group*),

Figura 4.9 | Contribuição dos métodos para a UML

****

Fonte: elaborada pela autora.

Segundo Cris Kobryn, 20% da UML ajuda você a fazer 80% do seu trabalho (2005 *apud* FOWLER, 2005, p. 7). Portanto, é importante dedicar uma atenção a esses 20% fundamentais para o domínio da UML e, especialmente, para os diagramas marcados na Figura 4.11.

A versão 2.5 apresenta 14 diagramas subdivididos em duas categorias:

**diagramas de estrutura:**

os diagramas de estrutura representam as estruturas estáticas do sistema por meio de objetos, relações e atributos. Seu objetivo é mostrar como os objetos se relacionam

**diagramas de comportamento**:

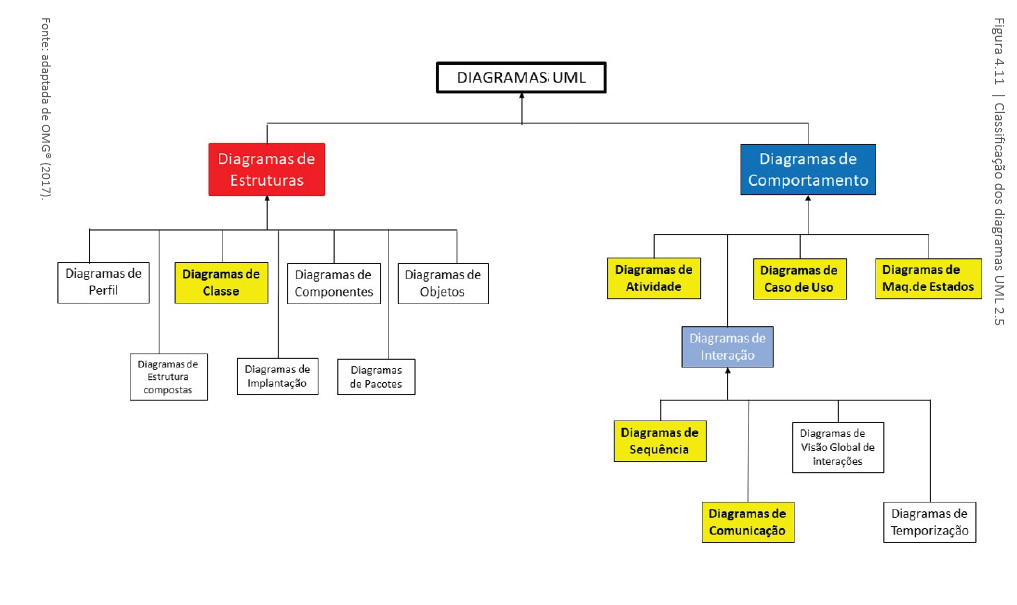
representam os aspectos dinâmicos do sistema,

são as mudanças que ocorrem no sistema com o passar do tempo

mudanças de estados internos e/ou externos no sistema

**subdivisão, denominada diagrama de interação:**

quatro diagramas

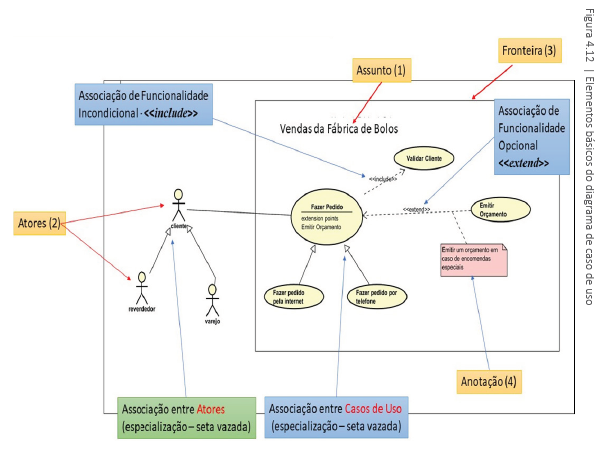


A existência de tantos diagramas tem como objetivo permitir visões múltiplas do sistema a ser modelado, como afirma Guedes (2011).

1. **Diagrama de caso de uso**

Esse diagrama é bastante utilizado. O diagrama de caso de uso fornece uma **visão geral dos objetivos** que os usuários (**os atores)** **desejam alcançar** utilizando o sistema.

Os elementos mais importantes são os atores, os relacionamentos e o fluxo de eventos;

****

1. **Diagrama de classe**

**Define a estru­tura das classes utilizadas pelo sistema,** determinando os atributos e métodos que cada classe tem, além de estabelecer como as classes se relacionam e trocam informações entre si”.

Para entender o diagrama de classe é importante você **relembrar os conceitos** fundamentais de orientação a objeto.

Uma **classe** é uma representação abstrata de um objeto.

Um **atributo** é uma característica de uma classe.

Um **método** é um comportamento de uma classe.

**Relembrando:** a classe cliente tem atributos como nome, endereço, telefone, etc., e comportamentos como fazer pedido, efetuar pagamento, aprovar orçamento.

E o “Pedro” é um objeto do tipo cliente

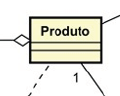
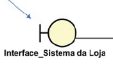
A representação da classe é um retângulo dividido em três partes: o nome da classe; os atributos e; os métodos,

Um mecanismo de extensibilidade da UML bastante utilizado nos diagramas de classe são os estereótipos.

A finalidade de um estereótipo é permitir **classificar elementos** do diagrama que tenham algo em comum entre si.

Os estereótipos podem ser definidos pelo desenvolvedor ou predefi­nidos.

os estereótipos predefinidos mais comuns são três:

1. <<***entity***>> **Estereótipo identidade**, que identifica classes de persis­tências. Essas classes **armazenam** dados **recebidos** pelo sistema.
2. Além da notação textual, <<*entity*>>, também pode ser representada pelo símbolo indicado na Figura 4.14 pela classe Produto. 
3. <<***boundary***>> **Estereótipo fronteira**, o qual **identifica** uma **classe de fronteira**. Essas classes servem de **comunicação** entre atores **externos** e o **sistema**.
4. Sua representação gráfica está indicada na Figura 4.14 pela *interface\_Sistema\_da Loja*. 
5. <<***control***>> **Estereótipo de controle.** Esse estereótipo geralmente é formado por classes que indicam regras de negócio. Sua representação gráfica é apresentada na Figura 4.14 pelo *controlador\_Sistema\_da\_loja* .

<<include>>

“Include” (seta com linha tracejada): esse relacionamento mostra que o tipo de relacionamento entre dois Casos de Uso implica na obrigatoriedade da execução do Caso de Uso que está sendo incluído.

<<extend>>

Muitos casos de uso podem compartilhar pedaços de pequenas funcionalidades. A “Extend” (seta com linha tracejada): esse relacionamento é usado para mostrar um comportamento opcional.

**Relacionamento das Classes especificados na UML (OMG®, 2017) são:**

1. **Dependência:** É o tipo de relacionamento **mais fraco** entre duas classes, **chamado de relação semântica** entre duas classes, na qual **uma alteração na classe independente pode afetar a classe dependente**.

Veja o exemplo na Figura 4.14, em que há um relacionamento do tipo dependência entre as classes Produto e a classe Receita;

Ambas existem independentemente, porém uma alteração na classe Receita poderá afetar a classe Produto.

1. **Associação:** Este é o tipo de relacionamento **mais comum**, e indica que a **classe A** tem uma relação com a **classe B**.

É um relaciona­mento **genérico**.

1. **Agregação:** É uma **associação específica**, em que a classe filha pode existir **independentemente** da classe pai.

É o caso do relacionamento de agregação mostrado na Figura 4.14 entre a classe Produto (pai) e a classe Cobertura (filha), em que, se um objeto do tipo Produto deixa de existir, o objeto do tipo Cobertura (filha) continua a existir.

1. **Composição:** é uma **associação específica** em que, se o objeto da classe pai **é destruído** o outro objeto associado, **não fará sentido** a existência da classe filha.

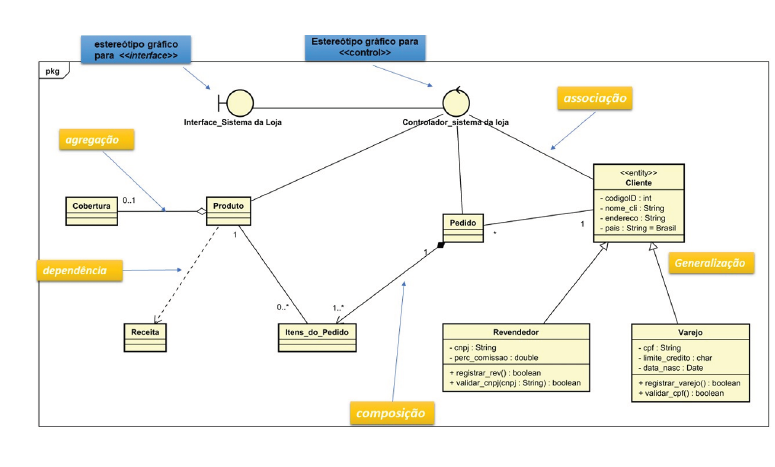
Exemplo: entre as classes Pedido e Item\_de\_Pedido temos um relacionamento de composição, pois os Itens\_de\_Pedido são partes do Pedido.

Se excluímos o Pedido, o Item\_de\_Pedido também deve ser excluído.

**5- Generalização/especialização:** indica que a **classe filha herda as características da classe pai**, também conhecida como especialização da classe.

As classes Revendedor e Varejo são classes filhas da classe Cliente;

**6 -Multiplicidade:** indica **quantas instâncias** dos objetos **estão envol­vidos na associação**,



1. **Diagrama de sequência**

Entre os diagramas de **interação**, esse é o **mais utilizado**.

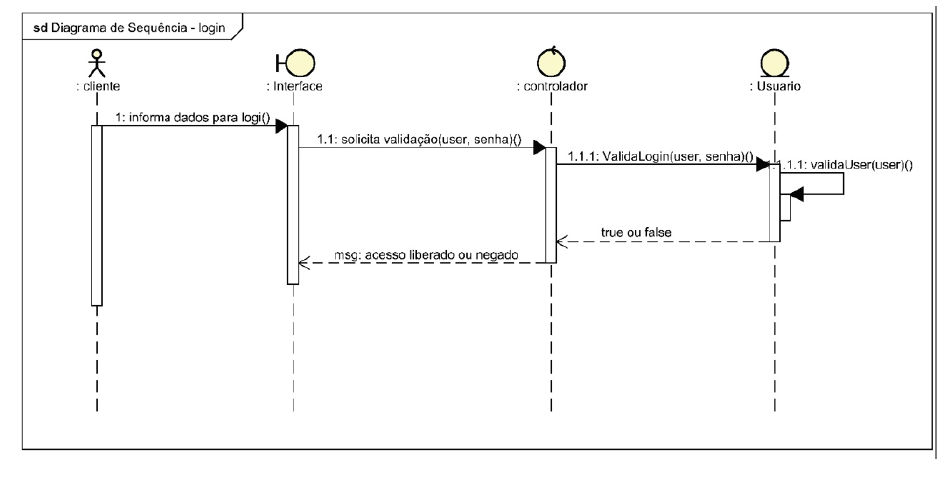
O diagrama de sequência **mostra a interação** entre **os participantes do cenário** ao longo da vida, a qual é mostrada verticalmente e na ordem de cima para baixo. Esse diagrama é muito intuitivo e quase não requer muitas explicações

No diagrama de sequência podemos inserir elementos para indicar repetição e operação opcional.

Vamos supor que o cliente possa efetuar o login apenas três vezes.

Nesse caso, basta envolvermos os elementos da Figura 4.15 por um retângulo com a indicação do termo **loop** e uma mensagem indicativa desta condição, como pode ser visto na Figura 4.16.

O diagrama de sequência apresenta, como o próprio nome diz, a sequência dos passos realizados pelos objetos

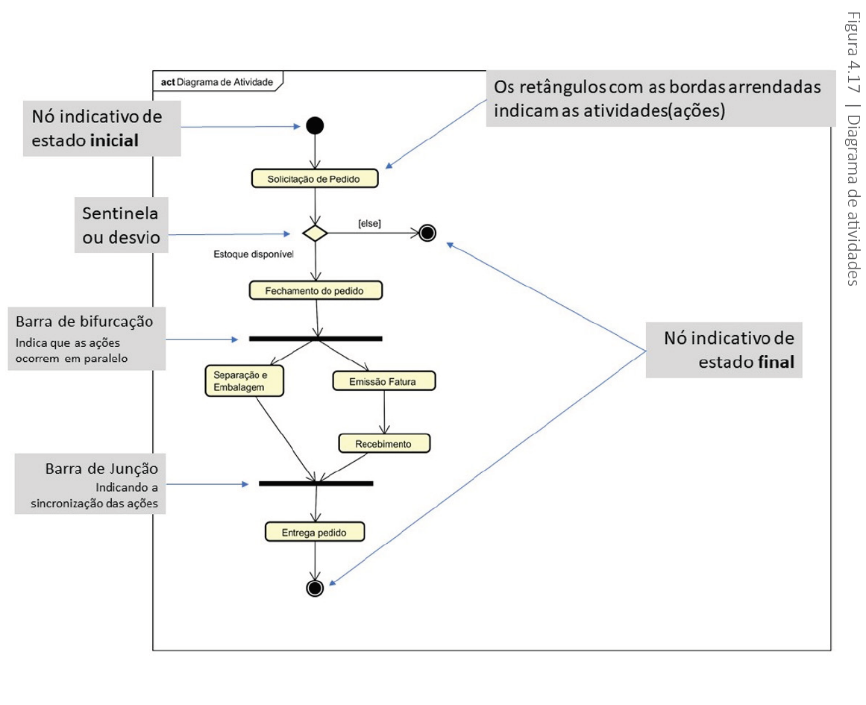


1. **Diagrama de atividade**

O diagrama de atividade tem por objetivo **descrever os passos que devem ser seguidos** para a execução de uma determinada **atividade**.

Esse diagrama assemelha-se muito com as funções de um fluxograma, exceto pelo fato que o diagrama de atividades pode representar atividades em paralelo.

Os elementos básicos do diagrama de atividade são: ações (atividades), sentinela (desvios), estados inicial e final, barra de bifurcação e barra de junção. Esses elementos podem ser vistos na Figura 4.17.



1. **Diagrama de máquina de estados**

O diagrama de máquina de estados é mais um diagrama de comporta­mento da UML. Esse diagrama visa mostrar a transição de um estado a outro dos objetos do sistema.

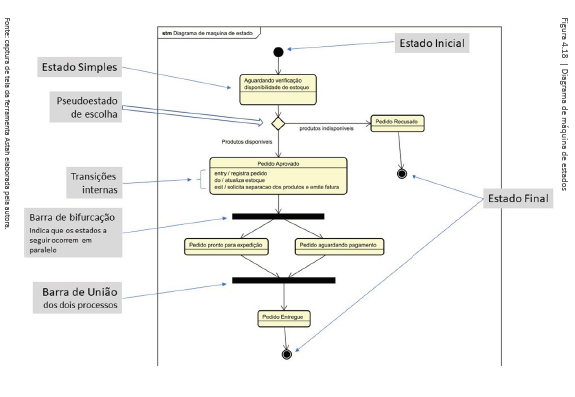
Na versão 1.x da UML, o diagrama de atividades era um caso específico desse diagrama. Assim, é possível observar muita semelhança entre eles na Figura 4.18, na qual se vê que os símbolos de estado inicial, estado final, barra de bifurcação e pseudoestado (desvio) são iguais aos utilizados no diagrama de atividade. **Uma diferença está no símbolo de estado que apresenta transições internas**, as quais existem apenas no diagrama de máquina de estados.

As transições internas de estado são três, a saber:

***Entry:*** indica ações internas do estado executada pelo objeto ao assumir um estado.

***• Do:*** são ações internas do estado executadas durante o período que o objeto se encontra em um estado.

***• Exit*:** neste caso, são ações executadas pelo objeto quando ele sai de um estado.



**Reflita**

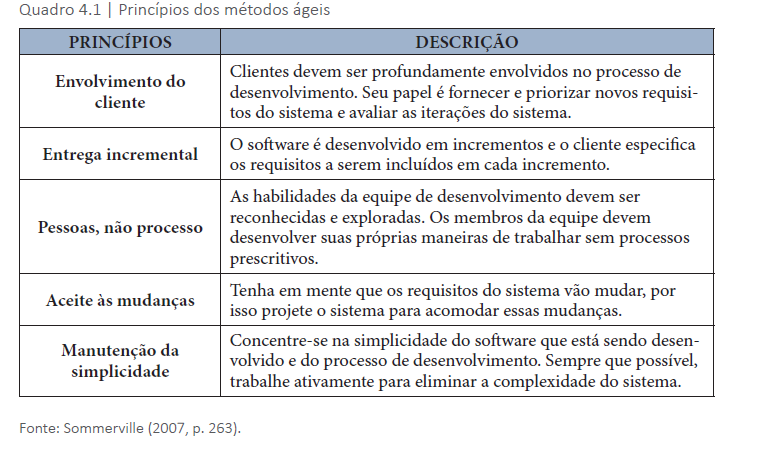
Acompanhe esse raciocínio: um fundamento essencial do PU é a **iteração**. Assim, sempre que necessário, retornamos a uma atividade já realizada e a refinamos à medida que avançamos no desenvolvimento da A/POO (Análise e Projeto Orientado a Objeto). Isso é o que a carac­teriza o processo iterativo.

Continuando nosso raciocínio, vale lembrar que a UML é uma linguagem que foi criada com um dos objetivos de facilitar esse processo, e com 20% dos diagramas UML é possível elaborar quase 80% dos softwares, o que representa três diagramas. Quais seriam os três diagramas mais utilizados no processo de modelagem? Todos são estáticos?

Para concluir esta seção, vamos conhecer um pouco mais sobre o desen­volvimento ágil.

Alguns métodos ágeis são focados em responder rapidamente à entrega do sistema; outros são voltados ao reuso e principalmente utilização de componentes, e há métodos ágeis direcionados especificamente para o desenvolvimento de sistemas críticos.

Atualmente o eXtreme Programming e o Scrum estão entre os métodos ágeis mais utilizados no mercado. Ambos seguem o mesmo conjunto de princípios apresentados no Quadro 4.1 por Sommerville (2007).



O **método Extreme Programming** (XP) exige uma abordagem “extrema” para o processo **iterativo**. **O envolvimento do cliente é parte do processo, é adaptável a mudanças, busca a simplicidade do software e seu foco é no valor do negócio**. Uma característica interessante deste método, por ser colabo­rativo (equipe), é que o desenvolvimento do código é feito aos pares. Isso permite que um programador mais experiente atue junto com outro não tão experiente e vá instruindo-o, ou, em casos mais críticos, quando dois pensam juntos, a solução vem mais rápido e fácil. Assim, o resultado esperado de agilizar as entregas é alcançado.

O outro método ágil bastante usado é o Scrum, e como já falamos, segue os mesmos princípios expostos no Quadro 4.1. O método Scrum dá alguns **nomes específicos para algumas disciplinas**, por exemplo: *sprint*, que funciona como os marcos das iterações do PU, só que o tempo de cada *sprint* é relativamente curto e diariamente a equipe avalia os resultados. Ao final de um *sprint* deve ser entregue um produto ao cliente (minissistema).

Nesta seção você pôde conhecer alguns diagramas UML e a importância de ter um bom conhecimento em pelo menos três diagramas. Também percebeu que os métodos ágeis foram uma evolução na engenharia de software, aprimorando e adaptando as necessidades do PU

Seguindo as disciplinas do processo unificado (PU), concluiremos a primeira iteração antes de começar a fase de implementação do código.

Com os conhecimentos de UML adquiridos nesta seção, será possível modelar o sistema utilizando alguns diagramas para responder: “o que o sistema deve fazer”, “como deverá fazer” e “quem irá fazer”.