



Linguagem de Modelagem Unificada (UML)



fundaçāo bradesco | escola virtual

Sumário

Apresentação	4
Módulo 1	6
Linguagem de Modelagem Unificada	6
Linguagem de Modelagem Unificada	7
Introdução à UML	7
Processo de desenvolvimento de software com UML e suas fases	13
Levantamento de requisitos	15
Análise de requisitos	16
Projeto (desenho)	18
Implementação (programação) e Testes	19
Implantação	20
Módulo 2	24
Desenvolvimento de software orientado a objetos	24
Desenvolvimento de software orientado a objetos	25
Classes, abstração e instanciação	25
Propriedades ou atributos	28
Métodos, operações ou comportamentos	29
Herança ou generalização	31
Polimorfismo	32
Encapsulamento	34

Sumário

Módulo 3	36
Notações e desenvolvimento de uma UML	36
Notações e desenvolvimento de uma UML	37
Modelo de visões	37
Blocos de construção	42
Itens da UML	43
Relacionamentos	48
Diagramas	50
Regras UML	62
Mecanismos básicos	64
Fechamento	68
Referências	69

Apresentação

Bem-vindo(a) ao curso **Linguagem de Modelagem Unificada (UML)**!

O objetivo do curso é introduzir os principais conceitos da UML. Esta notação virou referência e padrão internacional a ser seguido na indústria de engenharia de software, pois é independente, ou seja, não está relacionada a nenhuma linguagem de programação ou a um processo de desenvolvimento de software específico, o que facilita a sua adaptação à necessidade do projeto. Além disso, é muito empregada na programação orientada a objetos (POO) e possibilita que equipes de desenvolvimento e projeto possam vislumbrar o sistema a ser concebido em diagramas padronizados, melhorando o entendimento do sistema como um todo.

Desejamos a você um bom aprendizado!



Vídeo

Confira o [vídeo](#) de apresentação do curso.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

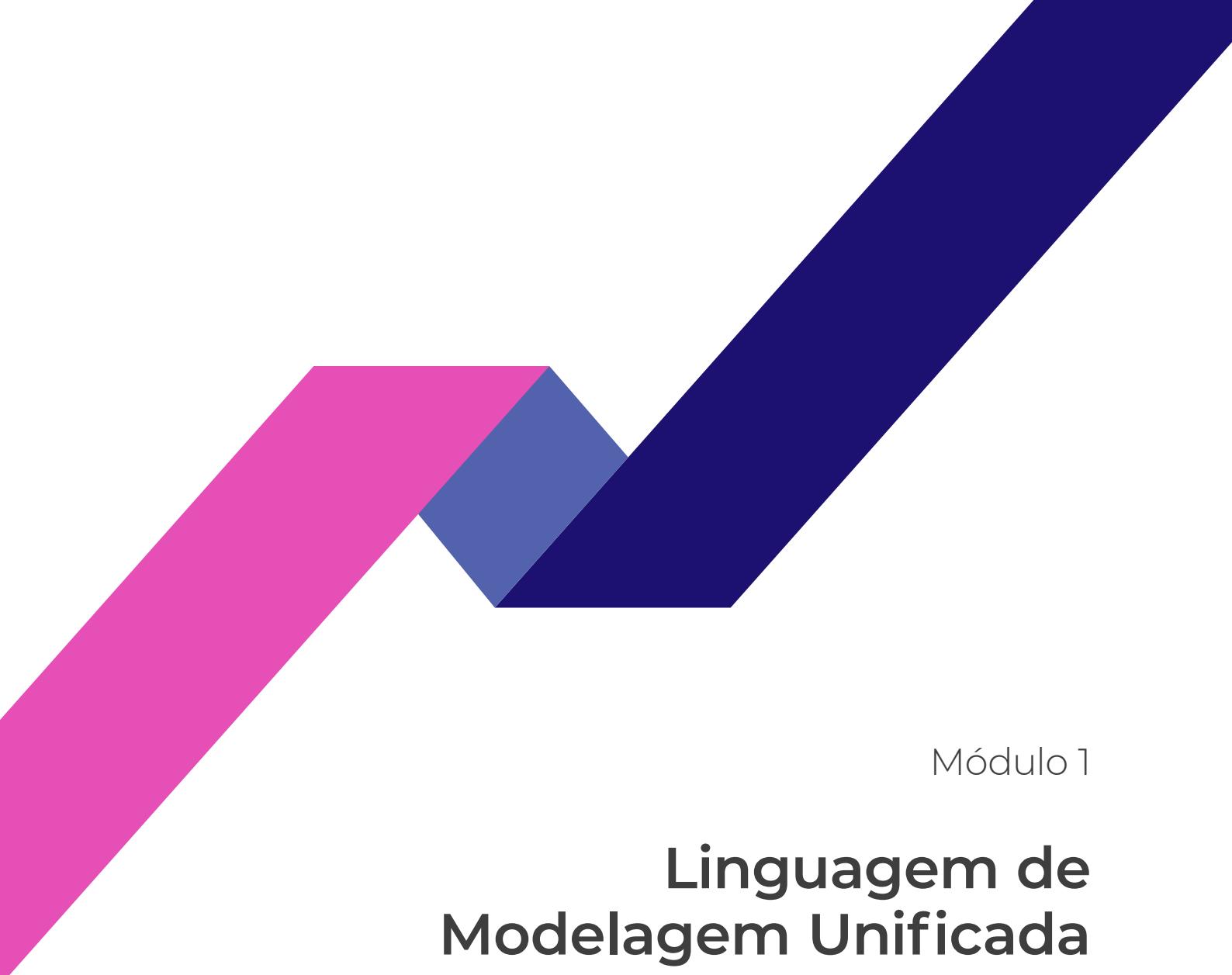
Olá! Seja bem-vindo ao curso Linguagem de Modelagem Unificada (UML).

No decorrer desta trajetória você entenderá do que se trata esse padrão de linguagem internacional e para que ele serve no dia a dia do profissional da área.

Para isso, será apresentada uma introdução do assunto para compreender, depois, como utilizar a UML no desenvolvimento de softwares. Você descobrirá, que esse processo inclui algumas fases, como o levantamento de requisitos, a análise de cada um deles, o projeto, a implementação, os testes e a implantação. Ainda, estudará sobre o desenvolvimento de software orientado a objetos, incluindo conceitos como polimorfismo e encapsulamento.

Além disso, verá sobre notações e desenvolvimento de UML, abordando modelo de visões, blocos de construção, regras da linguagem e mecanismos básicos.

Vamos começar nossa jornada?



Módulo 1

Linguagem de Modelagem Unificada

Linguagem de Modelagem Unificada

Neste primeiro módulo, você verá que a UML é um padrão de linguagem internacional para a criação da estrutura dos projetos de software. Ela pode ser utilizada para visualizar, especificar, construir e documentar os artefatos que fazem parte de sistemas com qualquer tipo de complexidade. Trata-se, de uma linguagem extremamente expressiva, compreendendo todas as visões fundamentais, desde o desenvolvimento até a implantação de um projeto de sistemas de software.

Mesmo sendo muito empregada em processos centrados em casos de usos com foco na arquitetura, incremental e iterativa, a UML é totalmente independente de processo. Vamos seus iniciar seus estudos com um breve histórico!

Introdução à UML

Para melhor compreender o que é uma UML, precisamos resgatar sua história e como ela ganhou forma, não é mesmo? Pensando nisso, ouça ao *podcast* que preparamos a seguir!



Podcast

Confira o [podcast](#) sobre a origem da Linguagem de Modelagem Unificada (UML).

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

Olá! Tudo bem? Agora que você já sabe o conceito de UML, vamos iniciar nossa caminhada por sua história e descobrir como ela ganhou forma?

A *Unified Modeling Language* (UML) ou Linguagem de Modelagem Unificada, apesar de ser confundida como uma metodologia, trata-se de uma linguagem visual universal que serve para detalhar elementos em geral. Sua finalidade é auxiliar na visualização, especificação, criação e documentação da interação entre objetos, sendo possível observar o desenho de todo esse processo.

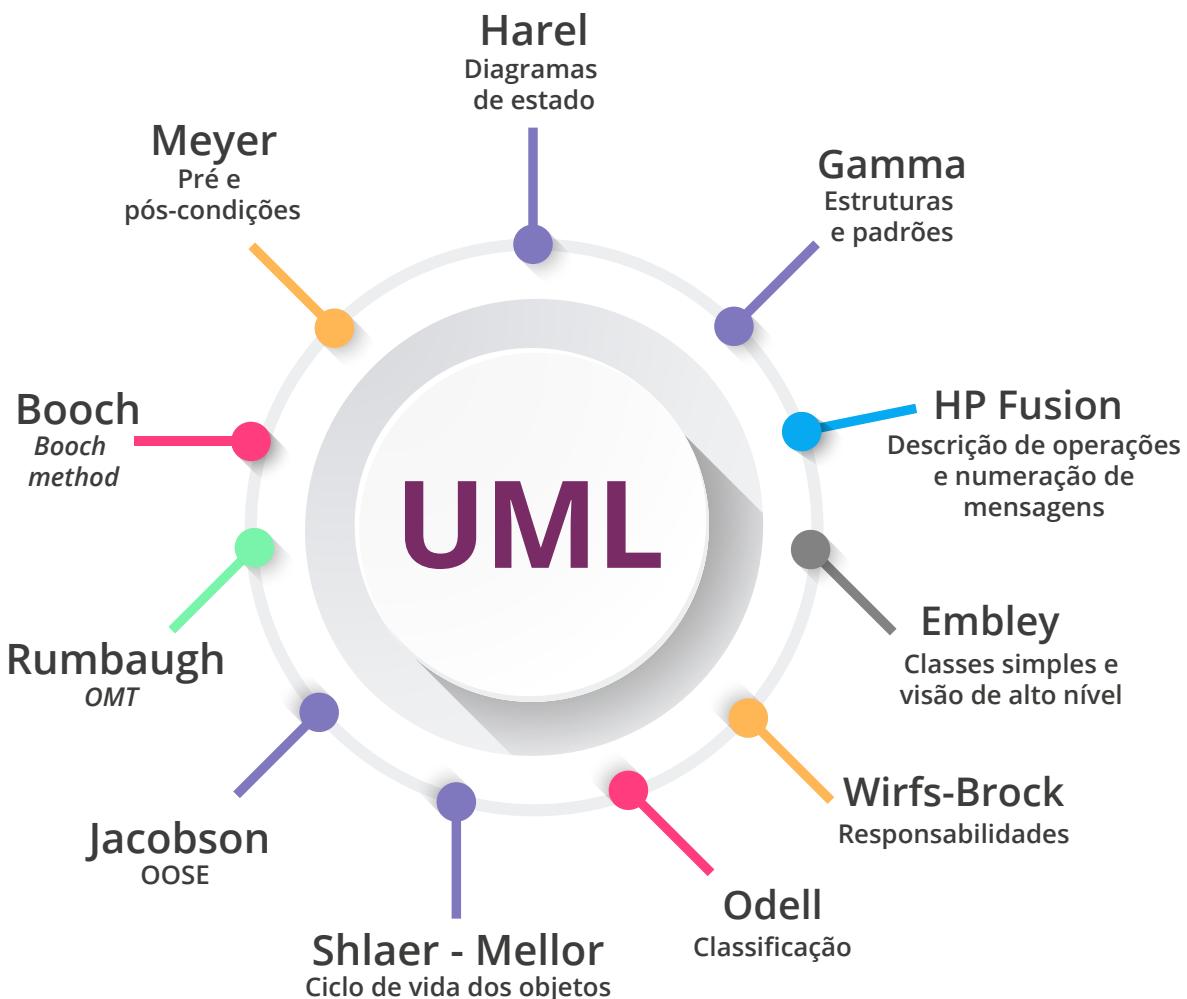
Com isso, a UML permite que desenvolvedores visualizem o artefato criado em diagramas padronizados. Sendo possível, assim, moldar sistemas por meio de conceitos de programação orientada a objetos e criar uma linguagem de modelagem que atenda a diferentes públicos, viabilizando determinar a união de metodologias conceituais para que sejam exercitáveis e práticas.

Então, com o apoio e o incentivo financeiro da *Rational Software Corporation*, a UML foi desenvolvida por três pesquisadores: Ivar Jacobson, Grady Booch e James Rumbaugh. E sua criação aconteceu a partir de três metodologias: Metodologia de Booch, Metodologia OMT (*Object Modeling Technique*) de Jacobson e Metodologia OOSE (*Object-Oriented Software Engineering*) de Rumbaugh.

Veja em mais detalhes essa história na sequência.

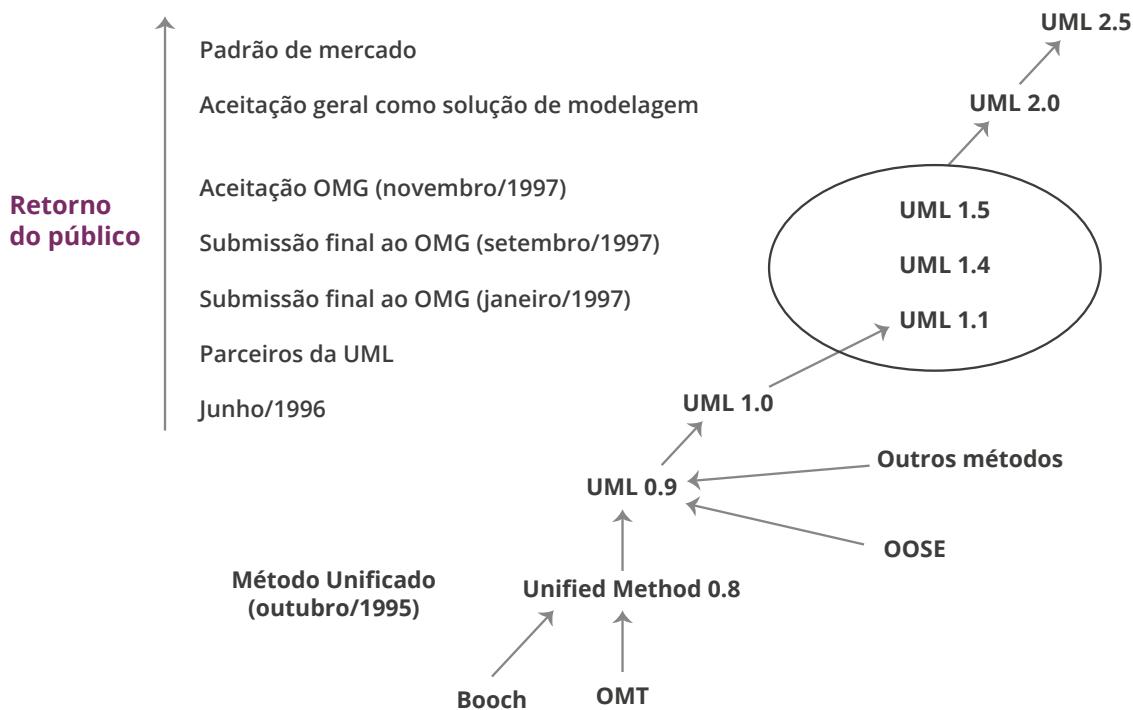
A primeira versão da UML foi lançada em 1996. Posteriormente, ela obteve várias sugestões de melhorias e expansão da linguagem por parte de empresas atuantes em modelagem e desenvolvimento de software.

Na imagem abaixo, você pode observar as principais contribuições da UML.



#PraCegoVer: na imagem, temos um esquema indicando as principais contribuições UML, incluindo Harel (diagramas de estado), Gamma (estruturas e padrões), HP Fusion (descrição de operações e numeração de mensagens), Embley (classes simples e visão de alto nível), Wirfs-Brock (responsabilidades), Odell (classificação), Shlaer – Mellor (ciclo de vida dos objetos), Jacobson (OOSE), Rumbaugh (OMT), Booch (Booch Method) e Meyer (pré e pós-condições).

Em 1997, a *Object Management Group* (OMG), organização internacional que determina e aprova padrões para orientação a objetos, reconheceu a UML como uma linguagem padrão para modelagem. Desde então, foram criadas diversas versões, conforme você pode conferir na imagem.



#PraCegoVer: na imagem, temos um esquema demonstrando o histórico UML. Do lado esquerdo, encontramos uma linha do tempo. De cima para baixo, há padrão de mercado, aceitação geral como solução de modelagem, aceitação OMG (novembro/1997), submissão final ao OMG (janeiro/1997), parceiros da UML e junho/1996. Do lado da linha, há "retorno do público". Já do lado direito, temos uma linha do tempo esquemática. Ela inicia com Booch e OMT, passa para Unified Method 0.8, UML 0.9, OOSE, outros métodos, UML 1.0, UML 1.1, UML 1.4, UML 1.5, UML 2.0, finalizando com UML 2.5.

Geralmente, a UML é utilizada para sistemas de maior complexidade, em domínios como sistemas de informações corporativos, distribuídos e baseados na web, financeiros e bancários, científicos, de telecomunicações, de eletrônica médica, de transportes, de vendas de varejo, de defesa ou de espaço aéreo. Portanto, a UML pode ser empregada para detalhar qualquer tipo de sistema, conforme você verá a seguir!

Sistemas de informação

Servem para guardar, analisar, editar e apresentar informações para os usuários, tendo um volume alto de informações complexas que são armazenadas em bancos de dados orientados a objetos ou relacionais.

Sistemas técnicos

Realizam a manutenção e administração de maquinário técnico, como processos industriais, equipamento militar e telecom. Devem conter interface especial do equipamento e pouca programação de software em relação aos sistemas de informação. Em sua maioria, são *real-time*.

Sistemas *real-time* integrados

São implementados em componentes de hardware integrados a carros, telefones celulares, alarmes, entre outros. Têm uma programação de nível baixo e suporte *real-time*.

Sistemas distribuídos

Como o próprio nome indica, são distribuídos em máquinas para que as informações sejam direcionadas entre elas com facilidade. Necessitam de meios de comunicação sincronizados, a fim de assegurar a integridade das informações contidas e compartilhadas. Em sua maioria, são implementados em mecanismos como Java Beans/RMI e CORBA.

Sistemas de software

São utilizados para especificar uma infraestrutura que outros sistemas utilizam, como bancos de dados, sistemas operacionais e ações de usuários que realizam procedimentos de nível baixo no hardware. Concomitantemente, oferecem interfaces de uso genérico de outros sistemas.

Sistemas de negócio

Empregados para reportar as especificações, as normas e o trabalho criado com base nos processos de negócio.

Contudo, as técnicas da UML não são exclusivas da Tecnologia da Informação (TI), pois elas podem ser empregadas em diferentes contextos, para descrever qualquer ação e processo que necessite de análise. Nesse sentido, seguidamente a UML é revista e tem sua versão atualizada.

Saiba mais



A UML faz parte da documentação oficial de projetos, bem como de outras metodologias, a exemplo do *Project Management Body of Knowledge* (PMBOK). Pensando nisso, caso queira obter mais informações sobre a UML, incluindo suas versões e a documentação oficial, sugerimos que acesse o [site](#) da OMG e o [site](#) da própria linguagem, que trazem informações bem interessantes!

Outro ponto interessante é que existem três tipos de conjuntos de diagramas executados pela UML. Você sabe quais são eles? Vamos conhecê-los!

Diagramas comportamentais

Destacam o que irá ocorrer no processo de negócio ou sistema. São utilizados para detalhar as funções de um sistema. Como exemplos, temos os diagramas de atividades, máquinas e estado.

Diagramas de interação

Detalham os fluxos para controlar os distintos elementos de um sistema, sendo um subconjunto dos diagramas comportamentais. Como exemplos, podemos mencionar os diagramas de tempo, comunicação, visão geral de interação e sequência.

Diagramas estruturais

Definem o que deverá ser desenvolvido em relação aos componentes do sistema. São benéficos para definir a arquitetura de um sistema que não depende do tempo. Como exemplos, temos os diagramas de perfil, objetos, pacotes, classes, implantação, componentes e estrutura composta.

É importante ressaltar, ainda, que não são utilizados, necessariamente, todos os diagramas existentes, uma vez que devemos aplicar aqueles que fazem mais sentido e que são úteis para o projeto que está sendo criado.

Depois de você ter estudado sobre a história da UML, as suas particularidades, a relação com a Programação Orientada a Objetos (POO) e os três tipos de conjuntos de diagramas executados, podemos seguir adiante e conhecer o processo de desenvolvimento de um software com UML. Acompanhe!

Processo de desenvolvimento de software com UML e suas fases

A modelagem de um sistema resulta na concepção de modelos de softwares em que um modelo captura a ideia de um sistema físico de modo abstrato, com algum objetivo. Podemos citar como exemplo quando queremos detalhar questões pertinentes à estrutura ou ao comportamento do sistema. Assim, a modelagem auxilia a definir o que pode ser incluído e o que é desnecessário dentro do modelo. Detalhando os dados do sistema físico que são importantes para o projeto.

Saiba mais



Existem várias metodologias de desenvolvimento de software as quais são divididas em dois grupos principais: tradicionais (como as metodologias incrementais, espiral e cascata) e ágeis (que englobam as metodologias *scrum*, *XP*, *cleanroom* etc.). Para conhecer um pouco mais de cada uma, assim como o ciclo de vida e suas características principais, sugerimos a leitura do artigo [Evolução da Metodologia do Desenvolvimento de Sistemas](#), escrito por Juliana Prado Uchôa. Também vale assistir ao vídeo [APS Modelos Cascata, Iterativo Incremental, Prototipação, Espiral e Métodos Ágeis](#), que traz complementos interessantes!

Ter uma metodologia de desenvolvimento nos permite categorizar as atividades em tarefas executadas durante a concepção de um sistema. Atualmente, inclusive, há muitas metodologias de desenvolvimento, mas, segundo os profissionais da área de desenvolvimento, não há uma abordagem melhor que a outra, uma vez

que cada uma possui sua metodologia própria e terá especificidades para obter e coordenar os procedimentos no desenvolvimento de software.

De qualquer modo, é possível diferenciar esses procedimentos, os quais, com algumas alterações, tornam-se comuns à maior parte dos processos existentes. As abas abaixo exibem o ciclo de vida padrão para o desenvolvimento de sistemas.

Análise

Engenharia de Requisitos: especificação e análise de requisitos.

Desenho

Arquitetura e Modelos.

Implementação

Codificação e Desenvolvimento do Sistema.

Teste

Testes, Validação e Qualidade.

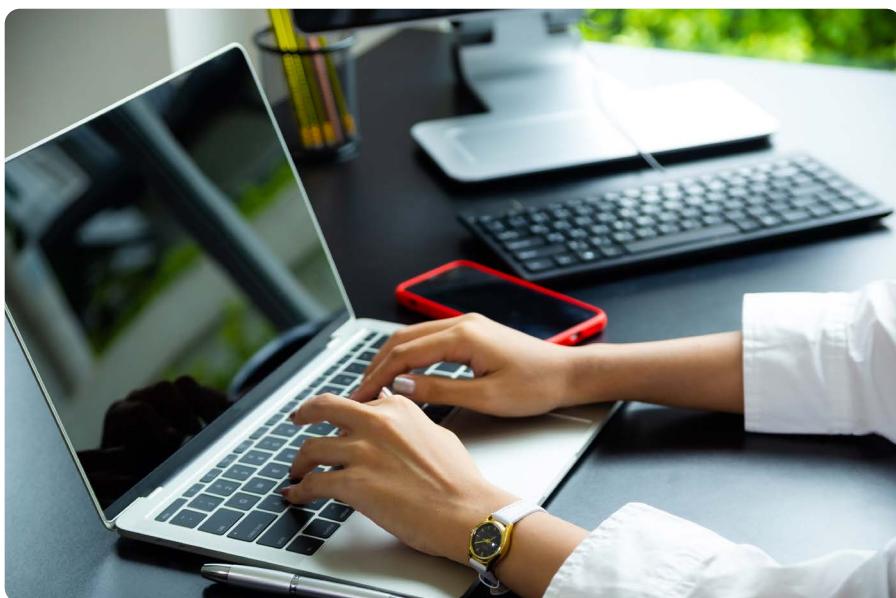
Implantação

Instalação, Suporte e Manutenção.

Aqui você pôde conhecer de modo objetivo, o ciclo de vida padrão para o desenvolvimento de sistemas. Inclusive, com esse contexto em mente, no próximo tópico, estudará cada etapa do desenvolvimento de software. Vamos lá!

Levantamento de requisitos

O levantamento de requisitos é uma das etapas iniciais no desenvolvimento de software, enquanto as demais (análise de requisitos, projetos que estão na principal etapa da modelagem, programação, teste e implantação) sugeridas por autores, de acordo com a metodologia escolhida, recebem nomes diferentes. Algumas são unidas em uma única fase ou divididas em mais fases.



#PraCegoVer: na imagem, temos a fotografia de uma pessoa digitando em um notebook. Aparecem apenas os braços e as mãos do indivíduo (à direita). Em cima da mesa preta, encontramos o notebook, um teclado e um smartphone com capa vermelha.

Entenda o Conceito



Esse levantamento tem por objetivo entender o problema a ser resolvido e definir “o que” é esperado do software, identificando se há possibilidade de desenvolvê-lo. Isso é feito por meio de entrevistas com o analista de sistemas, engenheiro de software, desenvolvedores etc. (representando a TI), que procuram compreender a necessidade do cliente (usuário) e quais funcionalidades e qual desempenho são esperados do software.

O resultado é a documentação do software, a qual conterá o detalhamento de todos os requisitos do sistema, classificados conforme você pode observar abaixo.

Requisitos funcionais

Determinam as funcionalidades do sistema.

Requisitos não funcionais

Definem características de qualidade que o sistema deverá contemplar.

Requisitos normativos

É a definição das restrições existentes.

Dessa forma, você compreendeu a primeira fase do ciclo de vida padrão para o desenvolvimento de sistemas: o levantamento de requisitos. Essa etapa envolve o objetivo da criação do software, o que ele irá resolver, sendo que os requisitos que deverão ser identificados são os funcionais, os não funcionais e os normativos.

Depois de realizarmos esse levantamento, precisamos, logicamente, analisar os requisitos encontrados. Descubra como fazer isso adiante.

Análise de requisitos

A análise de requisitos diz respeito às etapas de levantamento e análise de requisitos. Com o levantamento concluído, a fase da análise, também conhecida como análise ou especificação de requisitos, permitirá que os profissionais envolvidos — como o analista de sistemas — realizem o estudo com base na documentação do escopo, a fim de criar modelos com UML para representar o sistema. Dessa maneira, pode-se validar se todos estão em conformidade com o que está na documentação e se o sistema em questão realmente é viável.

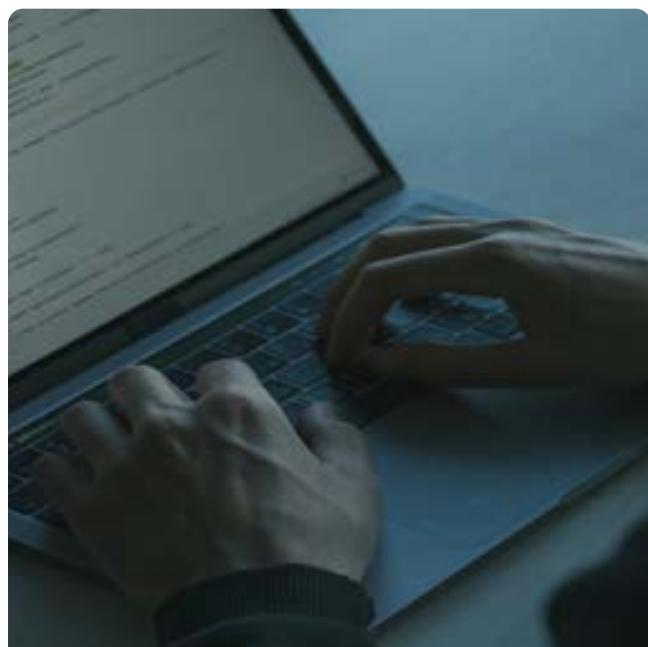


Tal como no levantamento, a análise não se baseia no ambiente de tecnologia em que será implementado o sistema, mas tem como foco criar alguma estratégia, sem evidenciar como essa solução será provida.

#PraCegoVer

Na imagem vista de cima, há um homem sentado em uma mesa de escritório e digitando em um teclado de frente a uma tela de computador. Ao lado dele, há um notebook aberto e fotos impressas.

O intuito é ter a melhor maneira de solucionar o problema do cliente, mas não focando nas peculiaridades tecnológicas.



#PraCegoVer

Na imagem que está com um enquadramento fechado, um homem com roupa escura está de frente a um notebook aberto e está digitando linhas de programação, sendo que ele faz isso em cima de uma mesa de escritório.

Outro ponto importante nessa fase de análise é a validação, ou seja, a verificação se os requisitos do cliente estão sendo cumpridos.

Após você ter visto a segunda etapa do ciclo de vida padrão para o desenvolvimento de sistemas, ou seja, a análise de requisitos, é possível definir o escopo do projeto e quais modelos com UML serão criados a partir dele, bem como validar a verificação junto a quem solicitou os tais requisitos.

Se você compreendeu todos os conceitos até aqui, podemos seguir para o próximo passo, que é a fase de projeto, ou desenho. Vamos lá?

Projeto (desenho)

Ao passo que a fase de análise foca no domínio do problema (aspectos independentes e lógicos da implementação), a fase de projeto ou desenho é voltada para a solução, buscando determinar o meio como o software realizará o que foi especificado na etapa de análise.

A maior parte da arquitetura do sistema é concebida nessa fase, quando é levado em conta se os recursos de tecnologia estão disponíveis para sanar o problema reportado pelo cliente, qual linguagem de programação será empregada, como será a interface, qual será o sistema de gerenciamento de banco de dados, como o sistema será compartilhado, entre outros detalhes pertinentes à arquitetura.

Observe no recurso abaixo, quais são os dois procedimentos primordiais que o projeto pode contar:

Projeto detalhado (projeto de baixo nível)

Realiza-se a modelagem das associações entre objetos de cada módulo, visando implementar as funcionalidades. Os diagramas da UML usados nessa etapa são os de classes, casos de uso, interação, estados e atividades.

Projeto da arquitetura (projeto de alto nível)

As classes de objetos do sistema são distribuídas em subsistemas e componentes. O diagrama da UML usado é o de implementação.

A etapa de projeto, ou ainda conhecida como desenho, é o momento de maior atenção por parte do programador que usa os modelos de criação de software com UML, visto que é nela em que será definida a maneira pela qual as necessidades da etapa da análise serão solucionadas. Para isso, existem dois procedimentos primordiais: o projeto detalhado e o projeto de arquitetura.

Assim, com o projeto realizado, passa-se para a etapa de implementação, também conhecida como programação, quando temos a codificação do sistema.

Implementação (programação) e Testes

Após a etapa do projeto, em que foram estabelecidas as soluções voltadas às necessidades mapeadas na etapa de análise, chegamos à implementação, isto é, almeja-se a realização do que foi projetado.

Por conta disso, tenha sempre em mente o seguinte:

Atenção



Na etapa de implementação, a equipe de desenvolvimento realiza a **codificação (programação) do sistema**, em que o detalhamento provido pela etapa de projeto é traduzido para uma ou mais linguagens de programação.

Em virtude disso, na programação orientada a objetos — com linguagens como Java, C#, entre outras — ocorre a concepção do código relacionado às classes de objetos do sistema.

Além disso, é nessa fase que é possível realizar o reuso de componentes para acelerar as atividades da implementação.

Porém, ainda que exista a etapa de implementação, ela sozinha não garante o suficiente para que o software criado com os modelos de UML esteja funcionando adequadamente. É preciso que ele seja testado, com o intuito de que eventuais erros e imprevistos possam

ser averiguados e corrigidos antes de sua disponibilização ao cliente.

Desta forma, após a implementação, temos a fase de testes, isto é, quando vários procedimentos são executados para nos certificarmos de que o software desenvolvido está rodando sem erros, de acordo com o detalhamento realizado na etapa do projeto.



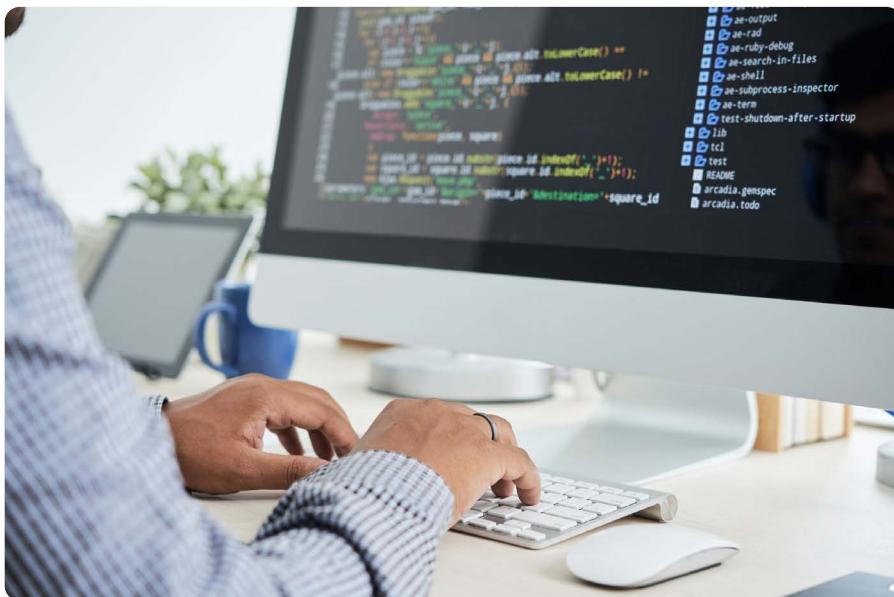
#PraCegoVer: na imagem vista de cima, um programador está debruçado em sua mesa de escritório, além de escrever observações em um caderno. Ao lado dele, está um notebook com a tela aberta e, à frente, está um papel em que há figuras impressas que remetem a etapas de um projeto, sendo que existem anotações escritas pelo profissional nelas.

O relatório de todos os testes, assim como os erros encontrados, são o resultado primordial dessa etapa, uma vez que são necessários para que o time de programadores faça as correções precisas. Dessa forma, todos os módulos do sistema são incorporados para que este seja finalizado para a implantação.

Implantação

Depois de finalizada a etapa de teste, em que todos os possíveis erros e imprevistos foram mapeados, temos a implantação.

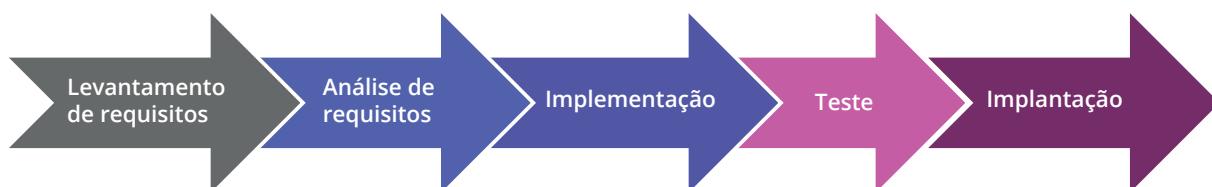
É nela que o sistema é compartilhado e implementado no ambiente especificado pelo cliente. Os manuais do software também são desenvolvidos nessa etapa, bem como os arquivos são carregados, as informações cruciais são importadas e os usuários recebem um treinamento para usar o sistema de maneira adequada.



#PraCegoVer: na imagem, temos a fotografia de uma pessoa digitando em um teclado frente a um monitor. O indivíduo está à esquerda, aparecendo apenas parte de seus braços e suas mãos. Ele veste uma camisa social quadriculada branca e cinza. Em cima da mesa, encontramos um teclado, um mouse e um monitor.

Lembre-se que, nessa fase, pode haver a migração de sistemas de software para casos em que os dados já existiam.

Com a implantação, finalizamos o processo de desenvolvimento de software com a UML. Contudo, antes de encerrarmos, convidamos você, a analisar a imagem e as informações a seguir, com o intuito de consolidar o que aprendeu em relação às etapas do ciclo de vida padrão para o desenvolvimento de sistemas. Confira!



#PraCegoVer: A imagem contém cinco setas apontando para a direita, uma em seguida da outra e coloridas. Em cada uma delas contém um título. São eles, respectivamente: Levantamento de requisitos, Análise de requisitos, Implementação, Teste e Implantação.

Levantamento de requisitos

Entender o problema a ser resolvido pelo do software.

Análise de requisitos

Estudo da documentação do escopo, a fim de criar modelos com UML para representar o sistema.

Projeto, ou desenho

Definir como o software realizará o que foi especificado na etapa de análise.

Implementação

Criação da codificação do sistema a partir de linguagens de programação (Java, C# etc.).

Teste

Aplicação de procedimentos que identificam possíveis erros.

Implantação

Compartilhamento e inserção do software no sistema ou no ambiente especificado pelo cliente, além da criação do manual de funcionamento do produto.

Parabéns! Você chegou ao final do Módulo 1!

Aqui você pôde entender a história da UML, as suas particularidades, a relação com a Programação Orientada a Objetos (POO) e os três tipos de conjuntos de diagramas executados.

Além disso, você viu o ciclo de vida padrão para o desenvolvimento de sistemas, sobretudo em relação às suas etapas, as quais são o levantamento de requisitos, a análise de requisitos, o projeto e a implantação.

Agora que já passamos por essas etapas, no próximo módulo vamos nos aprofundar sobre um assunto relevante para o seu futuro profissional: o desenvolvimento de um software orientado a objetos. Vamos lá?



Módulo 2

Desenvolvimento de software orientado a objetos

Desenvolvimento de software orientado a objetos

O desenvolvimento de um software orientado a objetos nada mais é que a aproximação do manejo das coisas de um mundo real com o manejo das estruturas de um sistema de software. Em virtude disso, temos o conceito de objeto como sendo algo genérico, que poderá retratar qualquer coisa tangível do mundo real, como animais e pessoas, os quais carregam consigo determinadas características (**atributos**) e certos comportamentos (**ações**).

Desse modo, é essencial entendermos o conceito de orientação a objetos, visto que a UML é comumente empregada nesse paradigma. Assim, você verá a seguir todos os principais conceitos dessa abordagem.

Classes, abstração e instanciação

Inicialmente, você entenderá os conceitos de classes, abstração e instanciação de modo mais aprofundado, acompanhe o vídeo a seguir!



Vídeo

Confira o [vídeo](#) de classes, abstração e instanciação.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Aqui neste vídeo vamos falar sobre classes, abstração e instanciação, o que cada um deles representa e como se relacionam!

Você sabia que na infância, de modo inconsciente, aprendemos a pensar de modo orientado? Pois todo conhecimento é demonstrado a partir de classificações e abstrações, sendo que tudo são coisas, ou seja, são objetos. E tais elementos, como casa ou carro, determinam as classes, que dizem respeito a um agrupamento de determinados objetos que trazem características e comportamentos iguais. Assim, podemos entender que uma classe é como uma “forma” a partir da qual os objetos são criados. Logo, um objeto é a instância de uma classe.

Vamos usar como exemplo o carro. Sendo classificado de carro, qualquer objeto de metal e com quatro rodas que vá de um lugar para outro, carregando pessoas. Então, quando se fala em abstração, é porque entendemos que um carro pode ter diferentes formatos, estilos, cores e marcas. Porém, cada objeto “carro” tem características parecidas, como duas portas, quatro rodas, volante, freio, vidros nas laterais, assim como traseiros e frontais, podendo, ainda, transportar pessoas.

Ao compreendermos que um objeto com características já definidas será disposto no grupo carro, ele também estará na instância da classe carro. Assim, o modelo Fusca e o modelo Ferrari, são instâncias da classe carro. Observe que mesmo contendo os mesmos atributos, os objetos de uma classe não são iguais, visto que cada um contém informações diferentes em seus atributos.

Desse modo, ao instanciar um objeto de determinada classe, será criado um novo objeto do conjunto que é retratado por aquela classe, contendo o comportamento e as características padrões. Portanto, o exemplo de criar uma classe, um tipo ou um grupo estabelece o conceito da instanciação ou instância de uma classe.

Por exemplo, Maria, Aline e Alex, são instâncias da classe “pessoa”, em que terão atributos (dados ou propriedades) comuns, como nome, data de nascimento e documento de identidade.

Simples não é mesmo? Agora que você compreendeu os conceitos de classe, abstração e instanciação, continue ligado para se aprofundar mais acerca de cada um deles.

Diante do que viu, de modo resumido, podemos dizer que a classe é o nome dado a um grupo de objetos com características em comum, já o conceito de abstração representa as diferentes manifestações de um objeto pertencente a uma classe. Por sua vez, o termo instanciação se refere a uma manifestação específica de um objeto referente a uma classe.

Na prática



A partir disso, a fim de que esses conceitos se tornem mais concretos no seu entendimento, imagine o seguinte: existe o objeto no mundo denominado “cão”. Por sua vez, a classe dele é a canina, sendo que a sua abstração é a de um poodle, de um vira lata, de um basset e, finalmente, uma instanciação dele é o cão lassie.

Neste tópico você pôde entender os conceitos de classes, abstração e instanciação. Inclusive, não só eles são importantes para compreensão da Linguagem de Modelagem Unificada, bem como auxiliam no entendimento de outro conceito: o de propriedades (ou ainda denominado de atributos).

Diante disso, conforme você verá no tópico adiante, as propriedades, ou atributos, possuem um papel muito relevante que é estabelecer as características e os atributos dos elementos de uma classe.

Propriedades ou atributos

As propriedades, também conhecidas como atributos, trata-se de recursos de uma classe ou qualquer elemento de dados. Em outras palavras, são as características da estrutura de uma classe. Em determinadas linguagens de programação, os atributos são chamados de “membros de dados”, ou “variáveis de instância de classe”.

Os atributos são apresentados na segunda divisão da classe e, geralmente, possuem dois dados, que são: tipo de informação que o atributo armazena (*character, integer* etc.) e nome que o identifica.

Os atributos possuem características referentes à classe pertencente. Todas as instâncias de uma mesma classe terão propriedades iguais, mas podem conter valores diferentes. Na classe carro, por exemplo, atributos como cor, modelo e marca do automóvel podem ter valores distintos.

Outra característica importante é a visibilidade de um atributo, que poderá ser privada, pública, de pacote e protegida. Veja mais detalhes na tabela!

Privada	Só a própria classe a qual o atributo pertence pode acessá-lo (enxergá-lo). É simbolizada com um sinal de subtração (-).
Pública	O atributo pode ser acessado (enxergado) por outras classes. É simbolizada pelo sinal de adição (+).
De pacote	O atributo só poderá ser acessado pelas classes do pacote que o contém. É simbolizada pelo til (~) e disponibilizada em algumas linguagens, como C# e Java.
Protegida	O atributo só poderá ser acessado pela própria classe (superclasse) e suas subclasses. É simbolizada pelo sustenido (#).

Vale ressaltar que o valor inicial e o tipo de dado do atributo dependerão da linguagem de programação utilizada.

Outra característica importante das classes são os **comportamentos** (ações) que elas podem ter. Ainda utilizando o exemplo da classe “pessoa”, esta pode ter o comportamento de andar, deitar-se, correr, nadar, dirigir, trabalhar, comer etc.

Portanto, um objeto associado à criação de um software, a partir do qual a linguagem UML será empregada, possui um conjunto de atributos, ou seja, as características da estrutura de uma classe juntamente com os seus comportamentos.

Contudo, ainda é válido abordar outro conceito atrelado aos anteriores e que são muito relevantes. Estamos falando das operações. Acompanhe na sequência!

Métodos, operações ou comportamentos

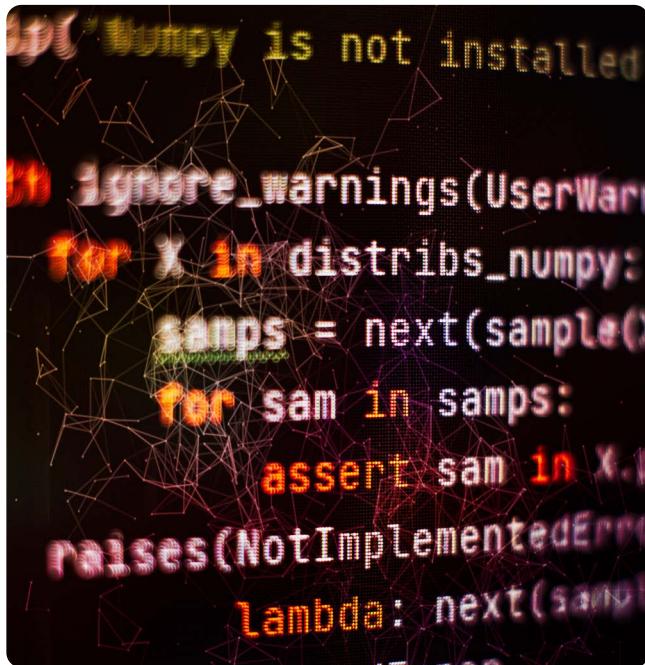
As classes carregam operações — termo utilizado pela UML — também conhecidas como métodos ou comportamentos (ações).

Um método é definido por uma ação que um objeto de determinada classe pode realizar, ou pelos serviços suportados por essa classe. Ele é guardado na terceira parte da classe.

Na prática



Para que você entenda melhor, veja este caso: na criação de um software, um dos objetos representados é um fusca, logo, esse objeto está inserido na classe “carro”, sendo que um dos métodos desempenhados por essa classe é a de “transportar pessoas”.



#PraCegoVer: Na imagem, existem várias linhas de programação com cores claras sobre um fundo preto.

Pode-se dizer, então, que o método é a abstração de um comportamento parecido entre objetos distintos, em que uma operação tem as seguintes características: tipo de retorno, nome, lista de argumentos e visibilidade. Considerando o exemplo anterior, é como se, no momento da programação, o software entendesse que, dentro da classe dos carros, tanto um fusca, quanto uma kombi podem transportar pessoas.

Na criação de um software, quando se aplica às linhas de programação voltadas para os objetos, esses realizam a comunicação e interação por meio de mensagens, as quais identificam os métodos que serão executados no objeto receptor. Para chamar um método, é enviada uma mensagem para ele com a especificação do nome do método, da lista de argumentos necessários e do nome do objeto.

Ou seja, é como se a mensagem presente na linguagem de programação tivesse o seguinte comando: “o objeto fusca dentro da classe dos carros deve transportar pessoas”.



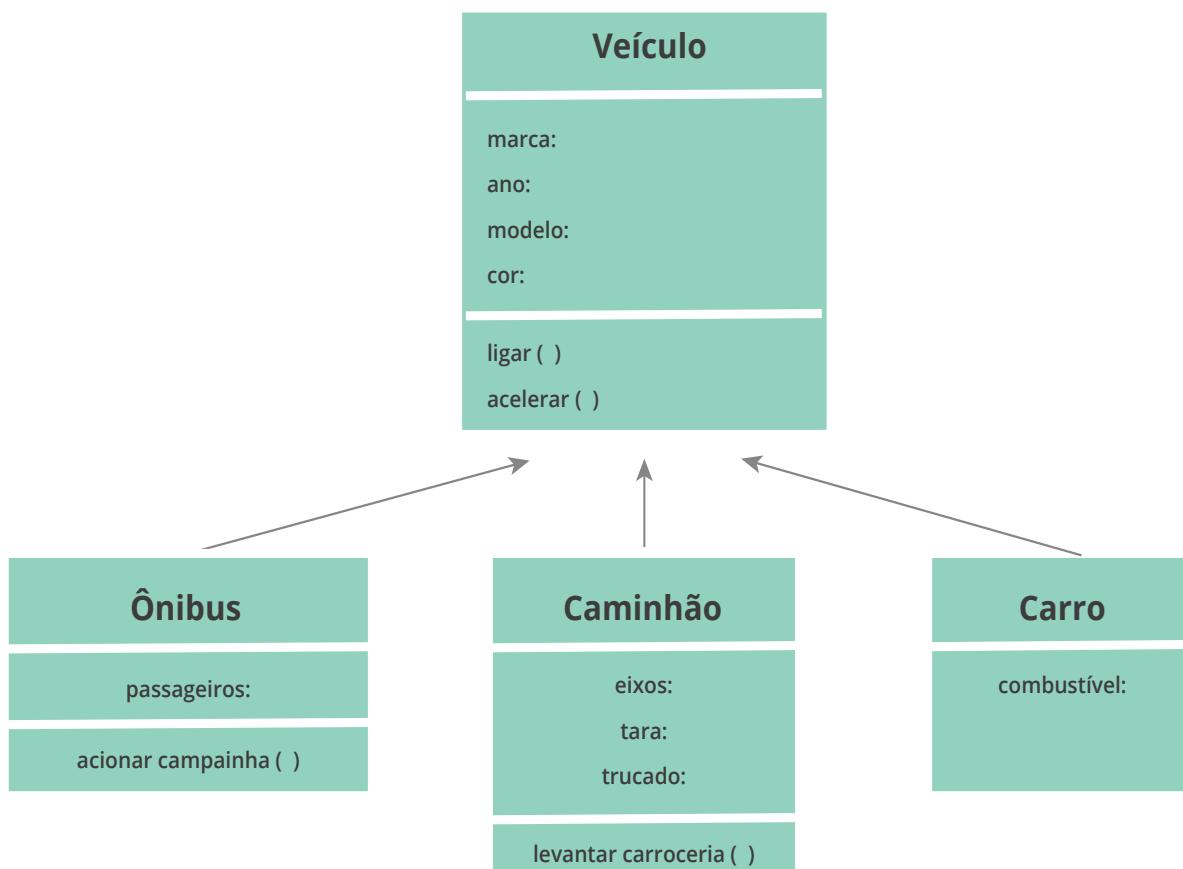
#PraCegoVer: Na imagem, existem várias linhas de programação com cores claras sobre um fundo preto. Em destaque e sobre tudo isso, há uma lupa e a sombra de quem a carrega.

A partir do exemplo anterior, você viu o conceito de métodos e de comportamentos, os quais estão associados à classe, à abstração e à instanciação.

Entretanto, há outro conceito essencial neste contexto: a herança. Isto é, o fato de que uma classe filha (subclasse) pode herdar o comportamento e/ou a estrutura de uma classe mãe (super classe). Vamos conferir como ela funciona? Siga em frente!

Herança ou generalização

A herança, também conhecida por generalização, possibilita o reaproveitamento de métodos e atributos. Com isso, otimiza tempo no desenvolvimento do software. Observe a imagem com atenção!



#PraCegoVer: na imagem, temos um diagrama de classes representando a herança. Há quatro caixas de códigos, sendo uma principal e três relacionadas abaixo. A principal diz respeito ao veículo (na qual estão escritas as seguintes palavras: "marca: , ano: , modelo: , cor: , ligar() e acelerar()"), enquanto as demais são ônibus (na qual estão escritas as seguintes palavras: "passageiros: , "acionar campainha()", caminhão (na qual estão escritas as seguintes palavras: "eixos: , tara: , trucado: , levantar carroceria()") e carro (na qual está escrita a seguinte palavra: "combustível").

Note que a generalização envolve os conceitos de classe mãe (superclasse) e classe filha (subclasse). A superclasse é uma classe que possui várias outras criadas a partir dela, que são as subclasses. Logo, estas, criadas a partir da superclasse, herdam todos os seus métodos e atributos.

No exemplo utilizado na imagem, a classe “veículo” é a classe mãe (superclasse), enquanto as classes “carro”, “ônibus” e “caminhão” são as classes filhas (subclasses ou derivadas) que herdaram os atributos de marca, ano, cor e modelo, assim como os comportamentos de acelerar e ligar da classe mãe.

Além disso, as classes filhas terão seus atributos e métodos exclusivos que os distinguiram, como o “carro”, que tem o atributo exclusivo “combustível”; o “ônibus”, que terá o atributo exclusivo “passageiro” e o comportamento (operação) exclusivo “acionar campainha”; e “caminhão”, que terá os atributos exclusivos “eixos”, “tara” e “trucado”, com o comportamento de “levantar carroceria”.

Até aqui você compreendeu o conceito de herança e como ele se aplica na relação entre classe mãe e filha. Vale ressaltar que a herança é um excelente recurso na criação de software, sobretudo, porque ele otimiza esse processo.

Siga adiante para conhecer o conceito de polimorfismo e como ele pode ajudá-lo no desenvolvimento de seu software.

Polimorfismo

Você saberia explicar do que se trata? Entenda melhor seu conceito, com o *podcast* que preparamos para você!



Podcast

Confira o [podcast](#) sobre polimorfismo.

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

Olá! Você já ouviu a palavra Polimorfismo? Se sim, sabe do que se trata? Neste podcast vamos falar um pouco sobre ele, começando pela sua definição!

O polimorfismo são classes derivadas de uma mesma superclasse, ou seja, elas podem chamar as operações com a mesma assinatura, desde que possuam comportamentos distintos em cada subclasse, cujos produtos são de resultados diferentes. Trata-se da habilidade de objetos diferentes terem operações com nomes e argumentos iguais, mas que realizam tarefas de maneiras distintas.

Imagine um veículo, como exemplo de superclasse. Uma moto, um caminhão, um carro e até uma bicicleta podem ser caracterizados como um veículo, porém são de subclasses diferentes. Pensando nisso, um objeto da subclasse caminhão iria herdar a operação frear, do mesmo modo que a subclasse bicicleta. Contudo, o comportamento e a implementação de ambos são distintos. Isso porque, o caminhão utiliza o motor, enquanto a bicicleta é impulsionada pela ação humana de pedalar.

Viu só como assim fica mais fácil de entender esse conceito? Ainda falando sobre as informações em uma classe, confira na sequência que elas podem estar visíveis ou não, com a definição de encapsulamento.

Simples não é mesmo? Você percebeu que, no processo de criação das classes, ainda existem subclassificações, necessitando de um recurso que auxilie na derivação entre uma superclasse e as suas subclasses, sendo que isso é feito a partir do conceito de polimorfismo.

Todavia, é interessante observar que as informações contidas em uma classe mãe ou filha podem, ainda, estar visíveis ou não para outros objetos, que é o conceito de encapsulamento, que você estudará agora. Vamos lá?

Encapsulamento

A ocultação de informações, mais conhecida por encapsulamento, refere-se ao procedimento de separar as questões internas das externas de um objeto. Dessa maneira, alguns detalhes ficam ocultos dos outros objetos, dizendo respeito somente ao próprio objeto a que pertencem.

Observe com atenção na imagem, como o encapsulamento pode ser público (+), *protected* (#), *private* (-) ou *package* (~).

público (+) que é visível para qualquer elementos que possa ver a classe.

protected (#) que é visível a outros elementos dentro da classe e de subclasses.

private (-) que é visível a outros elementos dentro da classe.

package (~) que é visível a elementos do mesmo pacote.

Carro
-Tipo: Texto
-Cor: Texto
-Placa: Texto
-Nº Portas: Inteiro
...
+Acelerar (): void
+Frear(): void
+TrocarMarcha(x): void
+Buzinar(): void
...

#PraCegoVer: na imagem, temos um exemplo de encapsulamento. Do lado esquerdo, encontramos as possibilidades (público, que é visível para qualquer elementos que possa ver a classe; protected, que é visível a outros elementos dentro da classe e de subclasses; private, que é visível a outros elementos dentro da classe; e package, que é visível a elementos do mesmo pacote). Do lado direito, há uma caixa com a classe "carro", sendo que, dentro dela, estão escritas as seguintes palavras "-Tipo: Texto, -Cor: Texto, - Placa: Texto, Nº Portas: Inteiro ..., +Acelerar (): void, +Frear(): void, TrocarMarcha(x): void, +Buzinar(): void".

Dessa forma, o principal benefício do encapsulamento é a possibilidade de isolar (proteger) os dados. Quando dizemos que os atributos estão encapsulados, queremos explicar que estão protegidos por um código, de modo que só serão acessíveis (visíveis) na operação em que foram concebidos. Ele se aplica para as operações, quando estas estão visíveis apenas ao objeto ao qual pertencem.

É importante salientar que o ato de encapsular cria uma **dependência direta** com a herança, visto que os objetos da classe filha herdaram todos os atributos e todas as ações (operações) da classe mãe.

Saiba mais



Para saber mais detalhes sobre a programação orientada a objetos, sugerimos que leia o artigo [POO: Tudo sobre Programação Orientada a Objetos!](#), escrito por Cairo Noleto. Também indicamos que assista ao vídeo [O que é Programação Orientada a Objetos - Conceitos Básicos de POO](#) para que os conceitos fiquem mais claros!

Parabéns! Você chegou ao final do Módulo 2!

Aqui abordamos os conceitos essenciais para a sua futura atuação profissional no contexto da POO (Programação Orientada a Objetos), que é uma importante forma de se desenvolver softwares a partir do uso da linguagem UML.

Para isso, você entendeu os conceitos de classes, abstração e instanciação, atributos e comportamentos, herança e polimorfismo.

Agora você está por dentro de o que é a programação orientada a objetos e os conceitos relacionados! Siga para o próximo módulo para aprender sobre as notações e o desenvolvimento de uma UML. Confira!



Módulo 3

Notações e desenvolvimento de uma UML

Notações e desenvolvimento de uma UML

Neste módulo, você conhecerá a forma como a UML é descrita e desenvolvida. Além disso, aprenderá sobre os conceitos, as definições, a notação, a simbologia e os artefatos da UML.

Assim, será necessário criar um modelo conceitual. Para isso, você aprenderá sobre os três componentes essenciais para o desenvolvimento de UML:



Confira!

Modelo de visões

Na UML, temos que os sistemas são compostos por modelos que traduzem pontos de vista distintos, em que cada um detalha alguma particularidade da mesma solução. Tais pontos são chamados de **visões**, ou seja, eles focam no mesmo sistema, porém sob óticas diferentes.

Cada visão faz uso de uma notação e um grupo de componentes para seu entendimento. Em outras palavras, um usuário tem uma visão diferente do desenvolvedor, do analista de sistemas, da equipe de projetos etc.

Convidamos você a assistir o vídeo a seguir, para entender melhor sobre esse assunto.



Vídeo

Confira o [vídeo](#) de modelo de visões.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Os modelos de visões, como vimos, são os pontos de vistas distintos dos sistemas da UML.

Para exemplificar, vamos pensar nos mapas, em que determinada área geográfica pode ter vários deles para descrevê-la sob perspectivas diferentes, como mapa de relevo, mapa político, mapa de estradas, entre outras possibilidades. A partir desse entendimento, podemos afirmar que a UML traz cinco visões: visão de caso de uso, visão de processo, visão lógica, visão de implementação e visão de implantação. Acompanhe.

A visão de caso de uso é utilizada como um acordo entre cliente e programador para definir o conceito e as funções que o sistema deverá possuir para atender aos requisitos. Trata-se das funcionalidades sistêmicas que deverão estar presentes, de modo a satisfazer aos requisitos do cliente.

O olhar do usuário sobre o sistema é a principal das visões, pois será a base do desenvolvimento das outras, sendo imprescindível para as fases de análise, desenho, teste do sistema e planejamento anterior.

Agora vamos para a visão de processo, de como é possível compreender a estrutura dos processos do sistema. Ela é criada com base nas visões de caso de uso e lógica, possibilitando capturar as características

estáticas e dinâmicas em diagramas de estado, interação e atividade. Ela, também, foca em peculiaridades, como tempo de resposta, escalabilidade, concorrência, ritmo de transferência (*throughput*), tolerância a falhas e paralisação do sistema (*deadlock*).

Já a visão lógica ou (estática) viabiliza organizar e estruturar o desenho do sistema de modo lógico. Por isso, é considerada uma visão arquitetônica. Ela permite a especificação de classes, subclasses e pacotes lógicos do sistema. Ainda que tenha o apoio do usuário final — ocasionalmente pela utilização de diagramas de objetos — a visão lógica transita pelo olhar das partes interessadas, como programadores, analistas, entre outros.

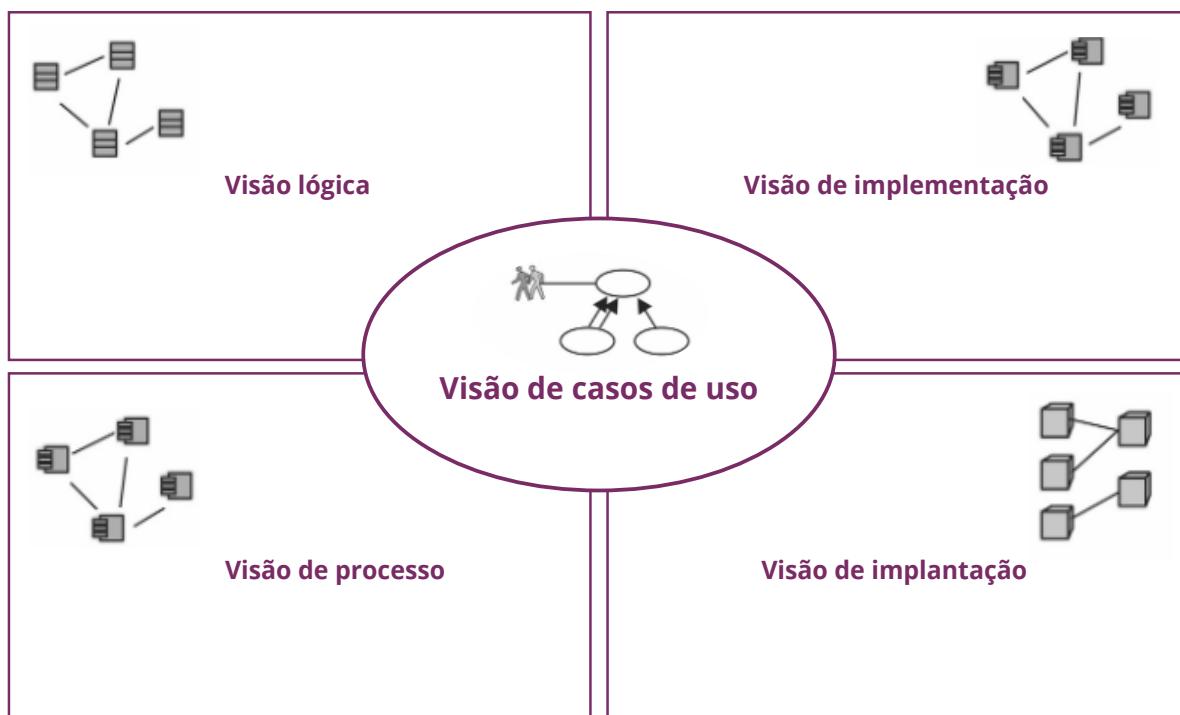
Indo para a visão de implementação, ela permite pegar as deliberações em relação à arquitetura para implementar no sistema, detalhando os subsistemas com suas dependências e seus elementos estruturados em categorias e hierarquias. Portanto, dentro de um planejamento sistêmico, ela possibilita determinar e compartilhar as tarefas relativas à implementação, estimar a quantidade de códigos que será desenvolvido ou reutilizado e detalhar as datas de entrega dos módulos do sistema, se necessário. Em virtude do propósito dessa visão, ela costuma ser de exclusividade da equipe do projeto, dos programadores e dos analistas.

E, por último, temos a visão de implantação, que trata da distribuição física do software, viabilizada por meio de um grupo de nós do ambiente em que será instalado. Esses nós se referem à infraestrutura do ambiente para a execução do sistema (*mainframes*, estações clientes, servidores de aplicação etc.), abrangendo o compartilhamento físico de *threads* e processos.

Dessa forma, como pudemos ver aqui neste vídeo, as visões são um recurso muito importante para a criação do produto, pois com elas é possível visualizar o micro e o macro do software.

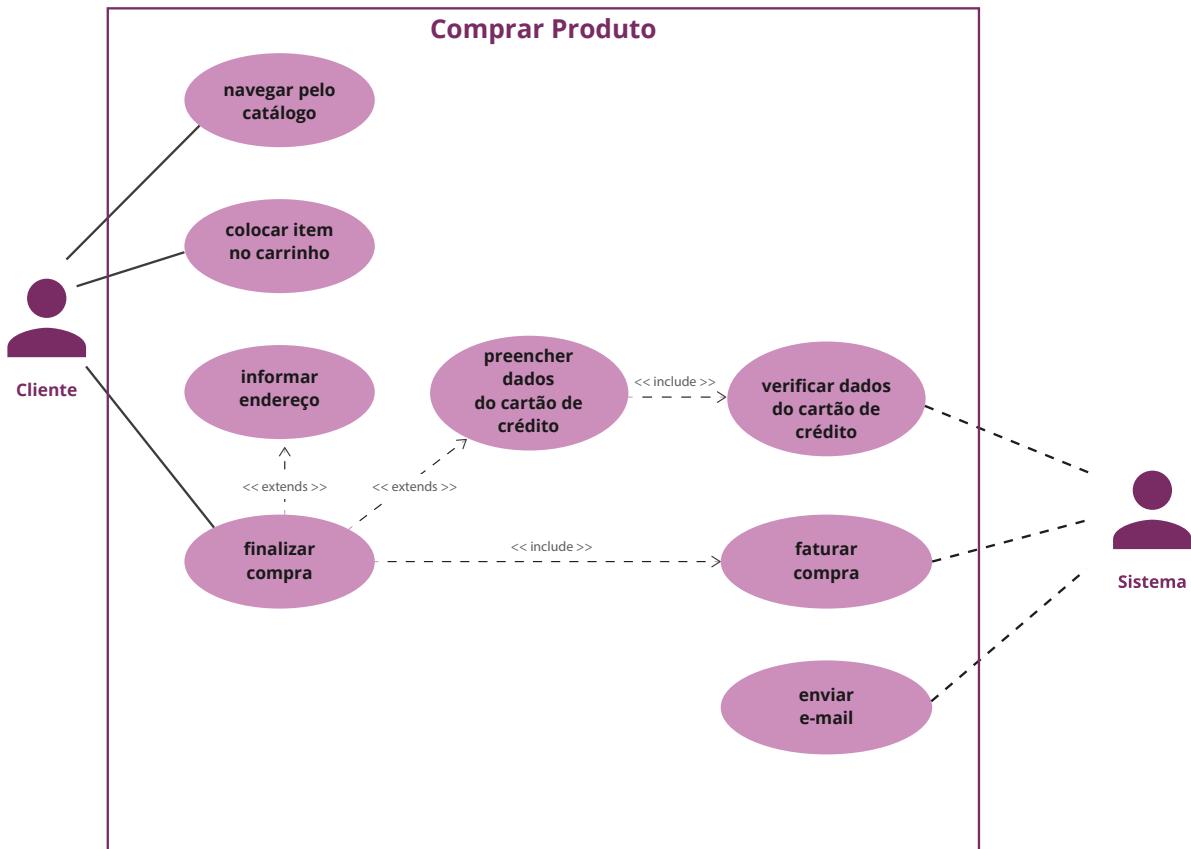
Como você pôde conferir no vídeo, as visões auxiliam o desenvolvedor a enxergar o micro e ao mesmo tempo o macro de um software, configurando-se como um importante recurso no momento da criação do produto.

Agora que você já comprehendeu as cinco visões da UML, acompanhe abaixo as ilustrações as quais vão lhe ajudar a consolidar o que aprendeu.



#PraCegoVer: na imagem, temos um esquema exemplificando as visões da UML em cinco quadrantes. Do lado esquerdo, temos as visões lógica (acima) e de processo (abaixo). No centro, encontramos a visão de casos de uso. Já do lado direito, temos as visões de implementação (acima) e implantação (abaixo).

Note que primeiramente, detalham-se as funcionalidades e os requisitos a partir da visão de caso de uso, sem se atentar quanto à tecnologia, ao banco de dados, ao servidor ou à linguagem a ser utilizada em um primeiro momento, pois o primordial é detalhar o máximo possível a solução que o cliente quer. Uma vez definida essa parte, passa-se para as demais visões, finalizando com a implantação.



#PraCegoVer: na imagem, temos um esquema exemplificando a visão de caso de uso para compra de um produto. Do lado esquerdo, encontramos o cliente com suas ações (navegar pelo catálogo, colocar item no carrinho, informar endereço e finalizar compra). Desta ação, temos o preenchimento de dados do cartão de crédito. Do lado direito, encontramos o sistema, que verifica os dados do cartão de crédito do cliente, fatura compra e envia e-mail.

Na imagem acima, é possível perceber a visão de uso, isto é, o ponto de vista do usuário e do sistema que será programado pelo desenvolvedor o qual empregará a linguagem UML. Por estarem em posições diferentes (visão), logo, existirão diferentes ações (uso).

Sendo assim, as visões da UML são abstrações dos modelos, sendo que cada uma dá atenção a um aspecto específico do sistema, enquanto os demais ficam de lado. Isso facilita o desenho e a modelagem sistêmica, de modo a melhorar a gestão e manutenção dos modelos, garantindo um produto final com maior qualidade.

Saiba mais



A **visão de caso de uso** é um dos diagramas mais utilizados nos projetos de UML. Pensando nisso, para que possa entender melhor a respeito do que é um caso de uso e o que são atores e relacionamentos dentro desse contexto, sugerimos que assista ao vídeo [Tutorial de Caso de Uso UML](#). Vale a pena conferir!

Desta maneira, você observou como é o conceito de visões, os seus cinco tipos, bem como a aplicação delas no contexto da UML, principalmente, quando se está criando um software. Inclusive, você notou a relevância de tais visões por propiciarem outras perspectivas as quais revelam situações específicas que podem afetar o desempenho do software a ser criado.

Com esses conceitos em mente, agora vamos descobrir o que são os blocos de construção? Continue seus estudos com atenção!

Blocos de construção

A linguagem da UML comprehende três tipos de blocos essenciais de construção: itens, relacionamentos e diagramas. Juntos, são conhecidos por blocos de construção da UML, em que cada um tem um papel fundamental, conforme você pode ver a seguir:

Itens

Referem-se às abstrações distinguidas como cidadãos de primeira classe em um modelo.

Relacionamentos

São as conexões que unem os itens.

Diagramas

Servem para agrupar coleções interessantes de itens.

Os blocos de construção, que constituem a linguagem da UML, vão lhe auxiliar no momento da construção das linhas de programação.

Na sequência, você conhecerá cada um desses tipos de blocos de construção mais detalhadamente!

Itens da UML

Os itens são blocos de construção básicos orientados a objetos da UML, os quais são utilizados para detalhar modelos bem formados. No total, podemos contabilizar quatro tipos de itens da UML, vamos conhecer cada um deles:

Entenda o Conceito



Os **itens estruturais** estão relacionados aos substantivos aplicados em modelagem UML. No modelo, são consideradas as partes mais estáticas, podendo retratar componentes físicos ou conceituais. Em conjunto, são chamados, também, de classificadores.

Observe abaixo os componentes que fazem parte dos itens estruturais:

Classes

Como sabemos, são detalhamentos de grupos de objetos que dividem operações, atributos, semântica e relacionamentos iguais, podendo desenvolver uma ou mais interfaces. Os retângulos dizem respeito à sua especificação gráfica.

Interface

Trata-se de um conjunto de operações que detalham os serviços de um componente ou uma classe, descrevendo comportamentos externos visíveis e podendo representar todos os comportamentos de um componente ou uma classe.

Geralmente, não aparecem sozinhas. Graficamente, podem aparecer como círculos se forem providas por uma classe ao mundo externo, ligados por uma linha. Caso sejam providas por outro tipo de classe, aparecem como semicírculos anexados à caixa de classes, ligados por uma linha.

Colaborações

Determinam as interações. São sociedades de papéis e outros componentes que trabalham em conjunto para possibilitar um comportamento cooperativo superior à soma de todos os componentes. Possuem dimensões comportamentais e estruturais, podendo uma classe ter várias colaborações. Estas são retratadas graficamente como elipses com linhas tracejadas que, de modo geral, trazem somente seus nomes.

Caso de uso

Detalha a sequência de atividades realizadas pelo sistema, providenciando resultados visíveis de valor para determinado ator. O caso de uso é feito por uma colaboração. Ele viabiliza estruturar o comportamento de itens para um modelo. São retratados, graficamente, por uma elipse com linhas contínuas que também trazem somente seus nomes.

Classes ativas

São classes em que os objetos possuem um ou mais processos “threads”, podendo iniciar a atividade de controle. São parecidas com uma classe, diferenciando-se por seus objetos retratarem elementos dos quais o comportamento é concorrente com outros elementos. São representadas por retângulos, porém com linhas duplas.

Componentes

São partes modulares sistêmicas que escondem o seu desenvolvimento atrás de um grupo de interfaces externas. Aqueles que dividem as mesmas interfaces podem ser trocados ao mesmo tempo em que mantêm o comportamento lógico. São retratados, graficamente, como uma classe com um ícone especial no canto superior direito.

Nó

É um componente físico que existe no tempo de execução e reproduz um recurso computacional com alguma memória e capacidade de processamento. Um conjunto de componentes poderá estar integrado em um nó, assim como mover de um nó para outro. Ele é retratado por um cubo.

No caso, os únicos elementos físicos entre todos os itens estruturais mencionados são os nós e os componentes.

Entenda o Conceito



Os **itens comportamentais**, por outro lado, referem-se aos verbos de um modelo, os quais representam os comportamentos no espaço e tempo. Inclusive, eles são considerados a parte dinâmica da UML e possuem um total de três tipos.

Confira quais são os tipos dos itens comportamentais a seguir!

Interação

Comportamento que engloba um grupo de mensagens trocadas entre um grupo de objetos, em alguma circunstância, para realizar propósitos específicos. Envolve elementos como mensagens, ações e conexões entre outros objetos. De modo gráfico, as mensagens são retratadas por uma linha cheia com seta, normalmente contendo o nome de suas operações.

Máquina de estado

Comportamento que detalha as sequências de estados dos quais interações ou objetos percorrem durante sua existência em resposta a eventos. É retratada, graficamente, por um retângulo com ângulos arredondados, geralmente contendo o nome e o respectivo subestado, se tiver.

Atividade

Comportamento que detalha a sequência de etapas que um processo computacional executa. É retratada por um retângulo com ângulos arredondados, juntamente com o nome que indica a sua finalidade.

Entenda o Conceito



Temos ainda os **itens de agrupamento**, que se referem a blocos em que os modelos podem ser decompostos. São considerados as partes organizacionais do modelo UML.

Em relação a isso, existem três itens de agrupamento, os quais você pode verificar no recurso abaixo:

Pacotes

Eles são procedimentos de propósito geral para a entidade do próprio projeto, diferentemente das classes, que organizam os construtos de implementação.

Itens comportamentais, estruturais e outros

Os itens comportamentais, estruturais e outros poderão ser adicionados em pacotes, sendo que um pacote diferente de outros elementos presentes em tempo de execução só existe durante esse tempo de desenvolvimento, o que significa que é um pacote conceitual.

Os diretórios dos pacotes

Além disso, os pacotes são retratados por diretórios com guias, geralmente contendo apenas o nome e, às vezes, o conteúdo.

Conceito



Os **itens anotacionais** são os comentários inclusos para detalhar, esclarecer e realizar observações sobre qualquer elemento do modelo. São considerados as partes explicativas dos modelos de UML.

Também existe apenas um tipo de item anotacional:

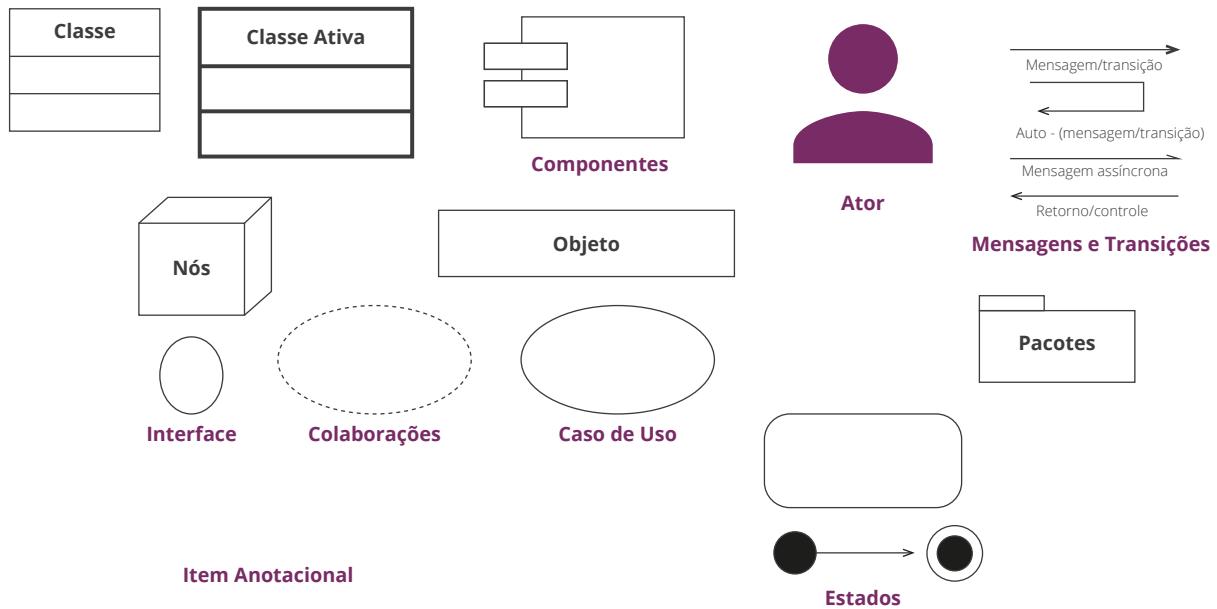


Item Anotacional

#PraCegoVer: um retângulo com um dos cantos contendo uma dobra de página, abaixo dele está escrito "Item Anotacional".

A nota é um símbolo que serve para retratar os comentários ou restrições anexadas a um elemento ou a uma coleção. A representação gráfica é um retângulo com um dos cantos contendo uma dobra de página, seguido de um comentário gráfico ou texto.

Antes de encerrarmos o tópico, veja a imagem a seguir, ela representa uma síntese dos principais itens estudados na linguagem UML.



#PraCegoVer: na imagem, temos exemplos de principais itens UML. Da esquerda para a direita, encontramos a classe (caixa de texto), a classe ativa (caixa de texto em negrito), os nós (cubo), a interface (oval na vertical pequeno), o item anotacional (papel com a ponta dobrada), as colaborações (oval na horizontal tracejado e ampliado), os componentes (quadrado com retângulos anexados), o objeto (retângulo ampliado), o caso de uso (oval na horizontal ampliado em negrito), o autor (representado por um ícone de perfil de uma pessoa), os estados (retângulo com as pontas arredondadas e dois círculos ligados por uma seta, sendo que o da direita é um círculo preenchido dentro de um vazio), as mensagens e transações (representadas por tipos de flechas) e os pacotes (pasta).

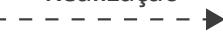
Até aqui, você conheceu os itens fundamentais da linguagem UML, isto é, a classe e a classe ativa, os nós, a interface, o item anotacional, as colaborações, os componentes, o objeto, o caso de uso, o autor, os estados, as mensagens, as transações e os pacotes.

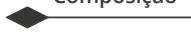
Todavia, existe um aspecto importante nesse contexto que promove a articulação desses itens estudados, são os relacionamentos. Acompanhe!

Relacionamentos

Após você ter estudado os principais itens da UML, chegou a hora de compreender os relacionamentos. Eles são os blocos relacionais básicos de construção. No geral, há sete relacionamentos principais na UML, conforme você pode observar em mais detalhes a seguir.

Os sete relacionamentos na UML

<p>Dependência Refere-se a um relacionamento semântico entre dois elementos, em que a modificação de um elemento independente poderá alterar a semântica do elemento que é dependente deste. Essa dependência é retratada por linhas tracejadas, em sua maioria com setas e, às vezes, com um rótulo.</p>	<p>Dependência</p>  <p>#PraCegover: linha horizontal traçada com uma seta na ponta direita.</p>
<p>Associação É um relacionamento estrutural entre classes que detalham um grupo de conexões entre objetos que são instâncias das classes. A representação gráfica da associação diz respeito a linhas sólidas, com a possibilidade de serem direcionadas e terem rótulos, mas, geralmente, trazem outros adornos, como papéis, nomes, multiplicidade etc.</p>	<p>Associação</p>  <p>#PraCegover: na imagem, há uma linha horizontal.</p>
<p>Agregação Refere-se a um relacionamento de especialização, em que os itens dos objetos especializados (filhos) são substituíveis por objetos do item generalizado (pais). Desse modo, os filhos dividem o comportamento e a estrutura dos pais. A representação gráfica da generalização é uma linha sólida com uma seta indicando o pai.</p>	<p>Agregação</p>  <p>#PraCegover: na imagem, há uma linha horizontal com um losango na ponta esquerda.</p>
<p>Realização É um relacionamento semântico entre classificadores, em que um classificador estabelece um acordo que o outro classificador deve realizar. Esse tipo de relacionamento é identificado em duas localidades (entre casos de uso e colaborações que os executam ou entre classes e interfaces ou componentes que as executaram). Sua representação gráfica é uma linha tracejada com seta branca entre generalização e relacionamento de dependência.</p>	<p>Realização</p>  <p>#PraCegover: na imagem, há uma linha horizontal pontilhada com uma seta na ponta direita.</p>
<p>Herança O relacionamento do tipo herança proporciona o compartilhamento entre as classes de métodos e atributos específicos por meio da hierarquia delas.</p>	<p>Herança</p>  <p>#PraCegover: na imagem, há uma linha horizontal com uma seta maior na ponta direita.</p>

Composição O relacionamento do tipo composição ocorre, quando há ligação entre mesmos objetos de classes distintas, ou seja, é possível reaprovar objetos, sem recorrer à duplicação das classes.	 <i>#PraCegover: na imagem, há uma linha horizontal com um losango preenchido na ponta esquerda.</i>
Navegabilidade Por fim, existe o relacionamento navegabilidade. Basicamente, ele representa a trajetória da navegação entre objetos associados.	 <i>#PraCegover: na imagem, há uma linha horizontal com uma seta na ponta direita.</i>

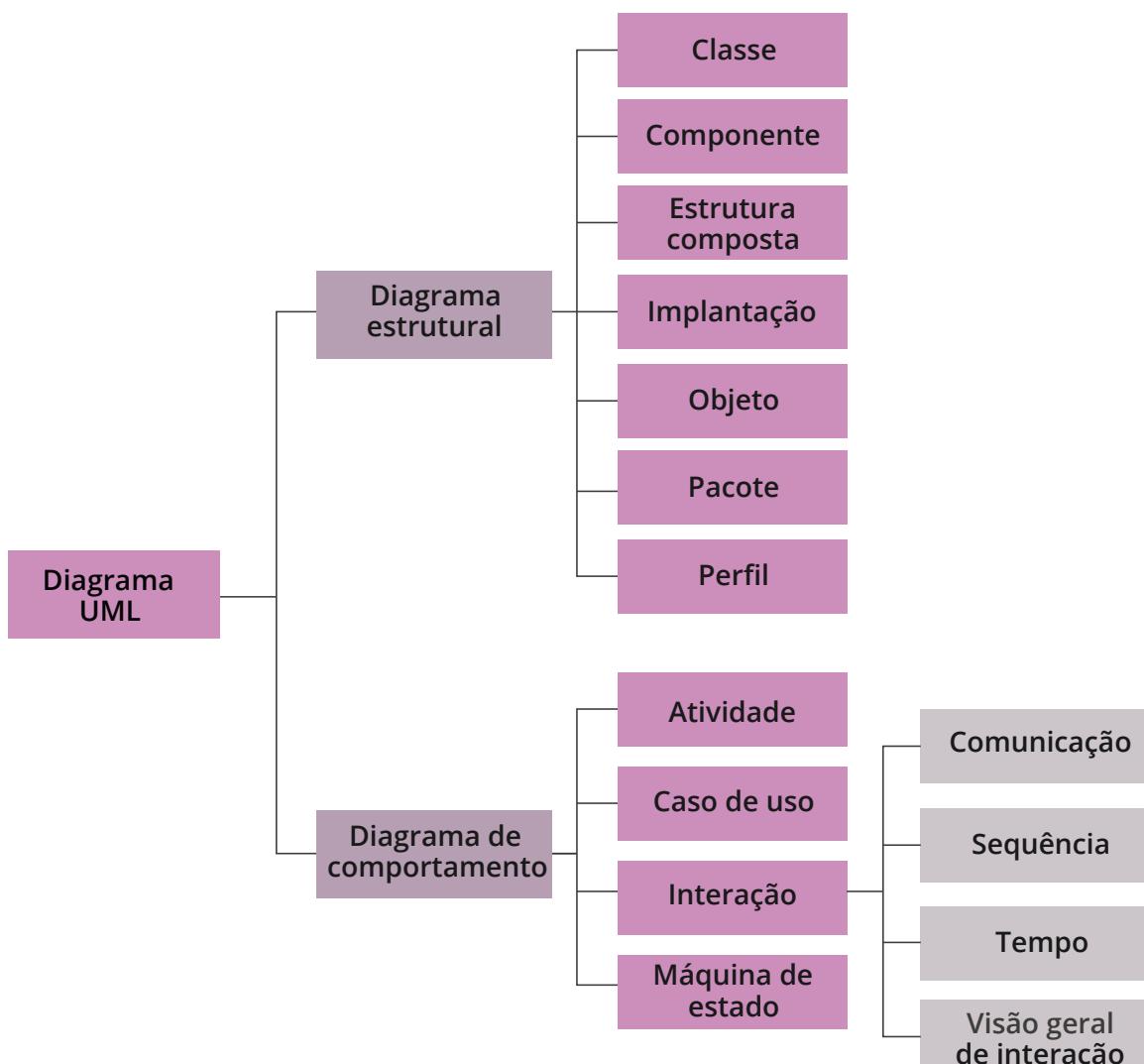
Após conhecer os sete relacionamentos existentes na UML, observando as suas principais características e funções, chegou a hora de entender o conceito de diagramas, importante elemento que auxilia na estrutura do sistema complexo dos softwares.

Diagramas

Na UML, um diagrama é a representação gráfica de um grupo de elementos, em sua maioria caracterizados como gráficos de relacionamentos (arcos) e itens (vértices).

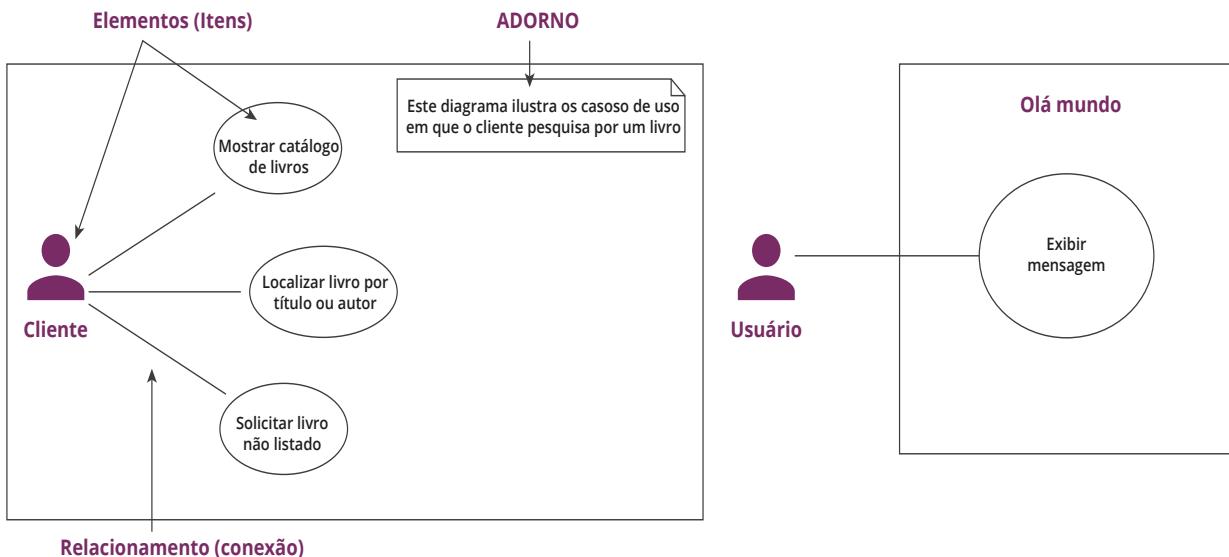
Desta forma, ela proporciona ilustrações que possibilitam visualizar um sistema sob diferentes óticas, ou seja, em todos os tipos de sistemas, um diagrama retrata uma visão parcial dos itens que irão construir-los.

No que diz respeito a isso, acompanhe a imagem abaixo. Ela é um exemplo de um diagrama de UML 2.5. Nele, observe a maneira pela qual os componentes, classes, objetos etc. são estruturados, a fim de que o sistema complexo do software funcione.



#PraCegoVer: na imagem, temos um diagrama de UML 2.5. Da esquerda para a direita, encontramos o diagrama UML dividido em diagrama estrutural (acima) e diagrama de comportamento (abaixo). O diagrama estrutural é ramificado em classe, componente, estrutura composta, implantação, objeto, pacote e perfil. Já o diagrama de comportamento é ramificado em atividade, caso de uso, interação e máquina de estado. A interação, por sua vez, é dividida em comunicação, sequência, tempo e visão geral de interação.

É importante mencionar que um diagrama poderá ter qualquer combinação de elementos e relacionamentos, como você pode observar na imagem a seguir, em que temos os atores “cliente” e “usuário”, o adorno para ilustrar os casos de uso em que o ator “cliente” pesquisa por um livro, as linhas para indicar relacionamentos de conexões, os balões para indicar comportamento/ação etc.



#PraCegoVer: na imagem, temos dois exemplos de diagramas com elementos. Do lado esquerdo, encontramos os elementos ou itens, que seria o cliente com suas ações (montar catálogo de livros, localizar livro por título ou autor e solicitar livro não listado), mantendo-se um relacionamento (conexão). Há, também, o adorno, sendo que o diagrama ilustra os casos de uso em que o cliente pesquisa por um livro. Do lado direito, temos o usuário de fora, com “olá mundo” e “exibir mensagem” do lado de dentro do diagrama, sendo que o usuário está ligado com o círculo dentro de um retângulo que contém o termo “exibir mensagem”.

Contudo, na prática, note que o diagrama apresenta um número pequeno de combinações comuns, as quais estarão de acordo com as cinco visões mais úteis da arquitetura de um software.

Na UML, um modelo não é somente um diagrama, pois ele é uma representação visual. Além dos gráficos, há especificações escritas para os itens, como os cenários para casos de uso.

A UML especifica três relacionamentos visuais principais em diagramas. Veja a seguir quais são eles!



#PraCegoVer: na imagem, há um programador em seu escritório. Ele está de costas, sentado, com um fone de ouvido e olhando para a tela de um computador, a qual contém várias linhas de código. Ao lado da tela, há um notebook aberto. Do outro lado, há uma prateleira com pastas e arquivos. Na parede à frente, outras linhas de código são projetadas.

Conexões

Conexões são linhas que ligam símbolos e ícones, formando caminhos ligados aos símbolos em suas duas extremidades.

Recipientes

Recipientes podem ser caixas, círculos e demais imagens que possuem outros símbolos e outras linhas.

Anexos

Anexos são itens que estão visualmente agrupados e podem dar a sugestão de um relacionamento devido à sua proximidade.

Podemos ter dois tipos principais de categorização de diagramas, que são os estruturais e os comportamentais. Conheça-os melhor!

Entenda o Conceito



Os **diagramas estruturais** têm por finalidade a especificação, visualização, construção e documentação dos aspectos estáticos de um sistema. Tenha em mente os aspectos estáticos do sistema como uma caracterização do seu esqueleto e estrutura relativamente regulares.

Esses aspectos estáticos de um software englobam a existência e colocação de elementos, como interfaces, classes, componentes, colaborações e nós. Assim, na UML, os diagramas estruturais são ordenados em razão dos principais grupos de elementos existentes na modelagem de um sistema.

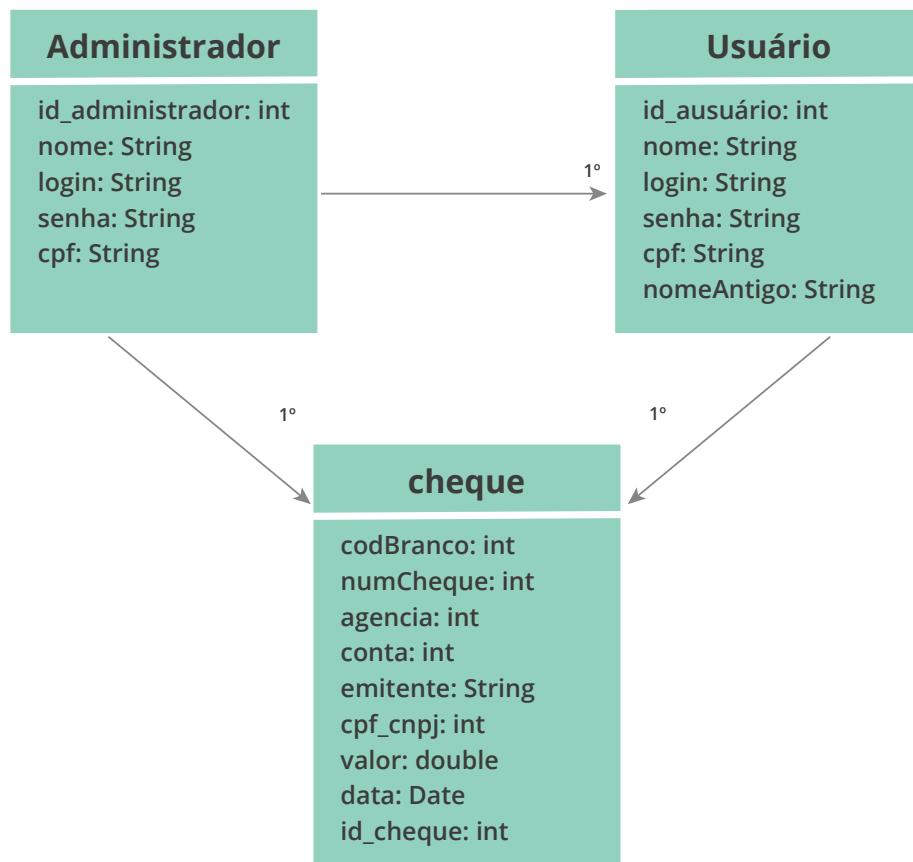
Na UML 2.5, há sete diagramas estruturais, observe-os.

Diagramas de classes

Exibem um conjunto de classes, colaborações, interfaces e seus relacionamentos. São os diagramas mais usuais em sistemas com paradigma orientado a objetos para exemplificar a visão estática em um projeto sistêmico, contendo classes ativas.

Exemplo cheque:

Negócio



Diagramas de componentes

São empregados para exibir as partes internas, as portas e os conectores que criam um componente, ou seja, provêm uma visão mais simples de um sistema complexo, uma vez que o fragmenta em componentes menores para ilustrá-lo. Ao estanciar um componente, as cópias de suas partes internas são igualmente instanciadas.

Diagramas de estrutura composta

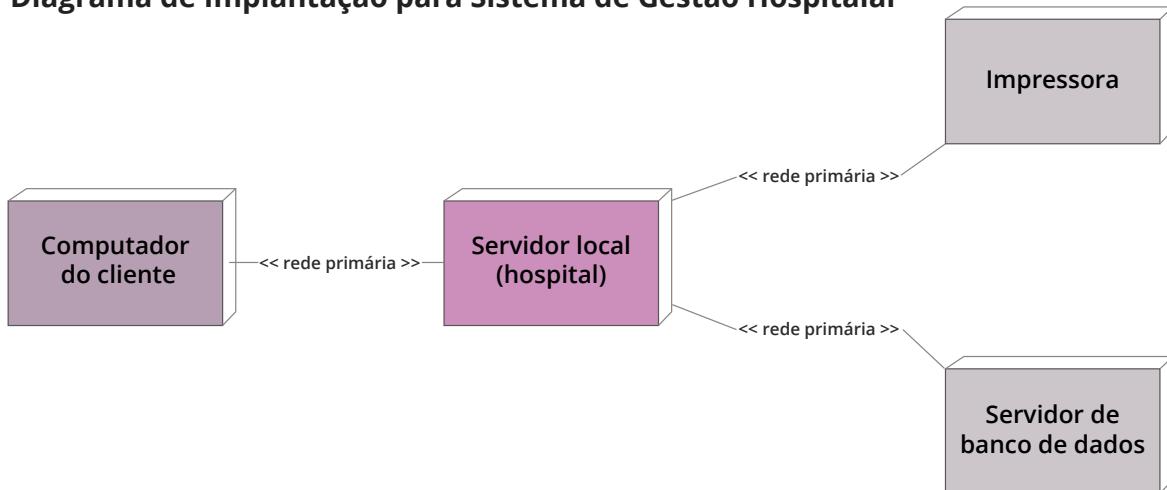
Utilizados para exibir a estrutura interna de uma colaboração ou classe.

Diagramas de implantação

Exibem um conjunto de nós e seus relacionamentos. São empregados para exemplificar a visão estática da implantação de determinada arquitetura, estando relacionados aos diagramas de componentes, pois, geralmente, um nó contém um ou mais componentes.

Exemplo Hospitalar:

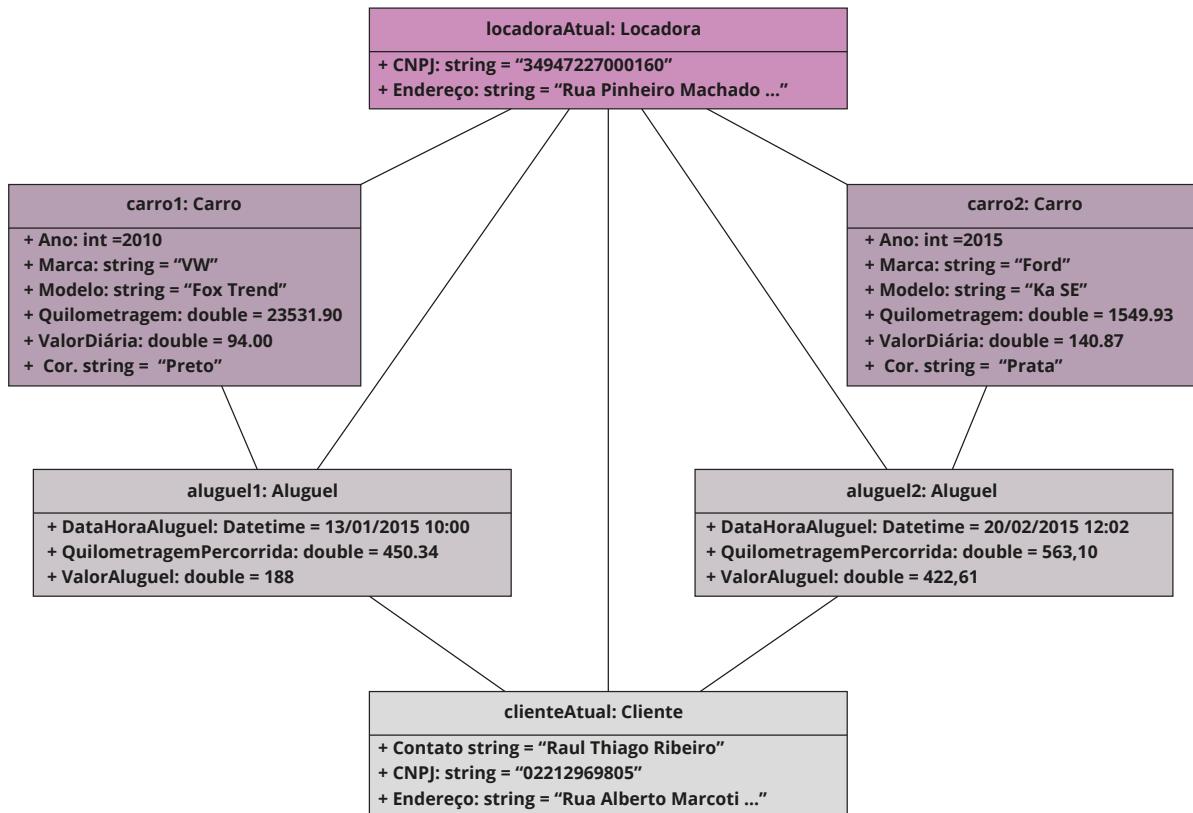
Diagrama de Implantação para Sistema de Gestão Hospitalar



Diagramas de objetos

Exibem um conjunto de objetos e seus relacionamentos. São empregados para demonstrar as estruturas de dados e os registros estáticos de instâncias dos elementos localizados nos diagramas de classes. Eles direcionam a visão estática de um projeto sistêmico ou a visão do processo de um sistema, assim como os diagramas de classes, mas consideram protótipicos ou casos reais.

Exemplo: Locadora de carros e cliente:



Diagramas de pacotes ou módulos

Têm por finalidade ilustrar os subsistemas ou submódulos pertencentes a um sistema, de modo a determinar as partes que o compõem. Podem ser utilizados de maneira isolada ou associada a outros diagramas. São empregados, também, para auxiliar na demonstração da arquitetura de uma linguagem, como a UML, ou para a definição das camadas de um software ou de um processo de desenvolvimento.

Diagramas de perfil

Foram criados a partir da versão UML 2.5, tendo por objetivo o desenvolvimento de uma visão — ou a personalização de um metamodelo já concebido com construções específicas para um domínio determinado — em um relacionamento entre classes para suportar um domínio. Isto é, auxiliam na criação de novas propriedades e semântica em diagramas UML, determinando

restrições, estereótipos customizados e valores marcados, possibilitando a customização de um metamodelo da UML para plataformas e domínios distintos.



#PraCegoVer: na imagem, temos a fotografia de três pessoas sentadas em uma mesa, e frente umas para as outras, estando duas na parte da frente e uma atrás. Elas observam seus monitores à frente. Em cima da mesa, ainda há duas lâmpadas.

Entenda o Conceito



Já os **diagramas comportamentais** têm por finalidade especificar, visualizar, documentar e construir os aspectos dinâmicos de algum sistema. Tenha em mente que esses aspectos retratam as partes do sistema que foram modificadas.

Dessa forma, na UML, esses diagramas são ordenados com base nos principais modos disponibilizados para realizar a modelagem da dinâmica de um sistema. Acompanhe as definições abaixo, para que você conheça mais sobre os diagramas comportamentais:

Diagramas de atividades

Também conhecidos como modelagens ou mapeamentos do processo de negócio, têm por finalidade exibir o fluxo de uma atividade para outro sistema. Ela exibe um grupo de atividades, o fluxo ramificado ou sequencial de uma atividade para outra e os objetos que realizam ou sofrem ações. Utilizamos esses diagramas, aliás, para exemplificar a visão dinâmica de um sistema. Eles são cruciais em virtude de realizarem o desenvolvimento das funções para um sistema e destacam o fluxo de controle na execução de um comportamento. A grosso modo, ilustram um processo passo a passo, com início e fim, sendo essenciais para o atingimento de um objetivo. Podem ser empregados em qualquer ambiente de negócios, além do desenvolvimento de softwares.

Diagramas de casos de uso

Exibem um conjunto de casos de uso, seus atores (um tipo especial de classe) e relacionamentos. Têm por finalidade desenhar a visão estática do caso de uso de um sistema e são essenciais para organizar e modelar os comportamentos de um sistema, exibindo os requisitos funcionais do sistema. Isto é, um diagrama de casos de uso detalha o que o sistema realiza e de que modo isso ocorre, por isso, traz um conjunto de eventos que surgem quando um ator utiliza o sistema para concluir um processo. Importante destacar que um ator é qualquer indivíduo ou item que exerce interação com o sistema (organização, aplicativo, pessoa etc.), estando de fora dele.

Diagramas de máquina de estado

Também conhecidos por gráficos de estados, exibem a máquina de estados, que são transições, estados, eventos e atividades. São essenciais para se realizar a modelagem de comportamento de uma classe, interface ou colaboração, sendo empregados quando o comportamento de um objeto é complexo e os detalhes são fundamentais. Assim, auxilia a detalhar o comportamento de um operador ou objeto, bem como o modo que ele altera, baseado em eventos externos e internos.

Diagramas de interação

Descrevem como conjuntos de objetos colaboram para algum comportamento.

Ainda a respeito dos diagramas de interação, é importante mencionar que há quatro diagramas englobados nessa categoria: de visão geral da interação, de tempo ou temporização, de sequência e de comunicação.

O **diagrama de visão geral da interação** passou a existir a partir da UML 2 e trata de uma variação do diagrama de atividades, visto que ambos exibem uma sequência passo a passo. Por conta disso, podem ser entendidos como um híbrido do diagrama de sequência com o de atividades. Entretanto, é um diagrama de atividades formado por diferentes diagramas de interação, utilizando as mesmas anotações do primeiro, mas que adiciona itens, como uso da interação, restrição de tempo, interação e restrição da duração.

O **diagrama de tempo ou temporização**, por outro lado, detalha a alteração na condição ou no estado de instância de uma classe ou seu papel durante um período. Geralmente é empregado para exibir a alteração no estado de um objeto no tempo, em resposta a eventos externos, dependendo das restrições de duração.

Em resumo, o diagrama de tempo demonstra como os atores e objetos se comportam ao longo de uma linha do tempo. Os principais componentes são:

Linha de vida

Elemento individual.

Linha do tempo do estado

Estados diferentes cuja linha de tempo passará.

Restrição de duração

Tempo que se precisa para que uma restrição seja executada.

Restrição de tempo

Tempo especificado para que algo seja executado.

Ocorrência de destruição

Ponto em que a linha de vida de um objeto termina, não havendo nada posteriormente.

Já o **diagrama de sequência** é comportamental e se preocupa com a ordem temporal em que as mensagens são trocadas entre os objetivos envolvidos em determinado processo. Geralmente, refere-se a um caso de uso determinado pelo diagrama de mesmo nome e se apoia no diagrama de classes para definir os objetos das classes envolvidas no processo.

Esse diagrama identifica o evento gerador do processo modelado, assim como o ator responsável pelo evento, definindo como o processo deverá proceder e ser concluído por meio de uma chamada de métodos disparados por mensagens, as quais são enviadas entre os objetos.

Atenção



O diagrama de sequência é útil para multitarefas e ideais que exibem todos os tipos de processos de negócios, uma vez que demonstram a estrutura sistêmica, com a exibição de sequência de interações e mensagens entre os objetos e atores de maneira cronológica.

Para finalizarmos essa parte, trazemos o **diagrama de comunicação**. Até a versão 1.5 da UML, era conhecido como “diagrama de colaboração”, mas, a partir da versão 2, passou a se chamar “diagrama de comunicação”. Ele destaca a comunicação entre objetos e exibe a ordenação daqueles que fazem parte de uma interação, mostrando ramificações e interações mais complexas. Está completamente relacionado ao diagrama de sequência, sendo que um completa o outro.

As informações exibidas no diagrama de comunicação são quase as mesmas do diagrama de sequência, mas com um foco diferente, uma vez que o de comunicação não se preocupa com o tempo do processo, mas sim em como os itens do diagrama estão relacionados e quais mensagens trocam entre si no processo.

Saiba mais



Existem algumas ferramentas gratuitas para a criação de diagramas UML. Para saber mais a respeito do assunto e conhecer que ferramentas são essas, sugerimos a leitura do artigo [7 Ferramentas On-line Gratuitas para Criar Diagramas UML](#), escrito por Eduardo Harada. É uma ótima maneira de complementar seus estudos!

Com o que viu até aqui, você passou a ter contato com os diferentes tipos de diagramas presentes na linguagem UML. Nesse sentido, ficou evidente a relevância deles, sobretudo, na estruturação dos sistemas associados ao desenvolvimento dos softwares.

É importante ressaltar que, para construí-los de modo adequado e eficaz, você deve ter ciência das principais premissas para a criação de artefatos UML. A seguir, você estudará mais sobre isso.

Regras UML

No contexto da UML, existem os diferentes tipos de relacionamentos entre objetos, classes etc., sendo que os diagramas ajudam na estruturação deles.

Por conta disso, é relevante citar que, na UML, não pode haver combinações

aleatórias para os blocos de construção, por isso, são estabelecidas algumas regras para determinar o que um modelo bem formado deverá conter.

Nesse sentido, há regras semânticas para nomes, escopo, visibilidade, integridade e execução. Confira abaixo, para saber mais:

Nomes

Quais nomes poderão ser escolhidos para relacionamentos, diagramas e itens.

Escopo

Contexto que define um significado específico para um nome.

Visibilidade

Como os nomes serão visualizados e empregados pelos outros.

Integridade

Como os itens se relacionarão entre si de maneira consistente e adequada.

Execução

Qual é o significado de executar ou simular um modelo dinâmico.

Os modelos bem formados são aqueles autoconsistentes semanticamente e harmônicos com todos os modelos relacionados a eles.

Além disso, os modelos construídos durante o desenvolvimento de software geralmente evoluirão a ponto de serem vistos por outros integrantes de modos diferentes e em momentos distintos. Assim, a equipe de programação não desenvolverá somente modelos bem formados, mas outros, como os listados abaixo.

Parciais

Alguns elementos ficarão ocultos para simplificar a visão do modelo.

Inconsistentes

A integridade do modelo não é garantida.

Incompletos

Alguns elementos podem ser omitidos.

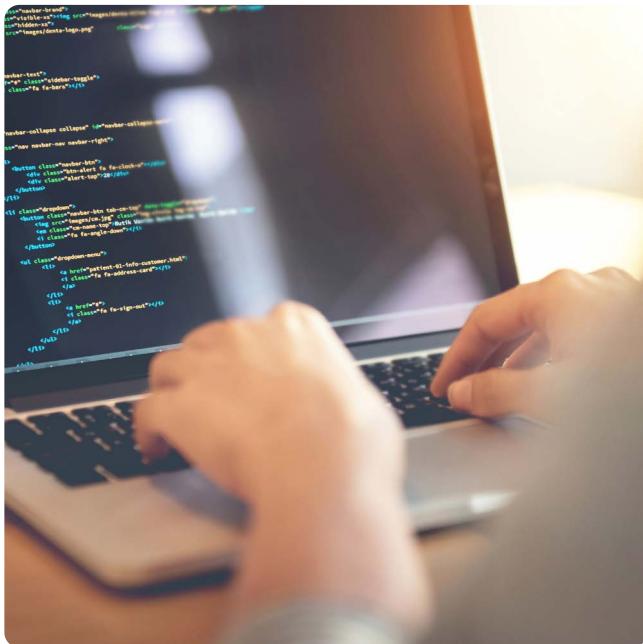
Neste tópico, você viu sobre as normas semânticas que regem as configurações dos modelos referentes à UML, os quais são essenciais para o desenvolvimento dos softwares.

Para finalizarmos, veja quais são os mecanismos básicos da UML.

Mecanismos básicos

Depois de ter visto as regras voltadas aos blocos de construção a partir da linguagem UML, neste momento, você estudará sobre os mecanismos básicos direcionados para essa linguagem.

A UML se torna mais simples e harmoniosa em razão da presença de quatro mecanismos essenciais, introduzidos de modo consistente na linguagem. São eles: as especificações, os adornos, as divisões comuns e os mecanismos de extensão.

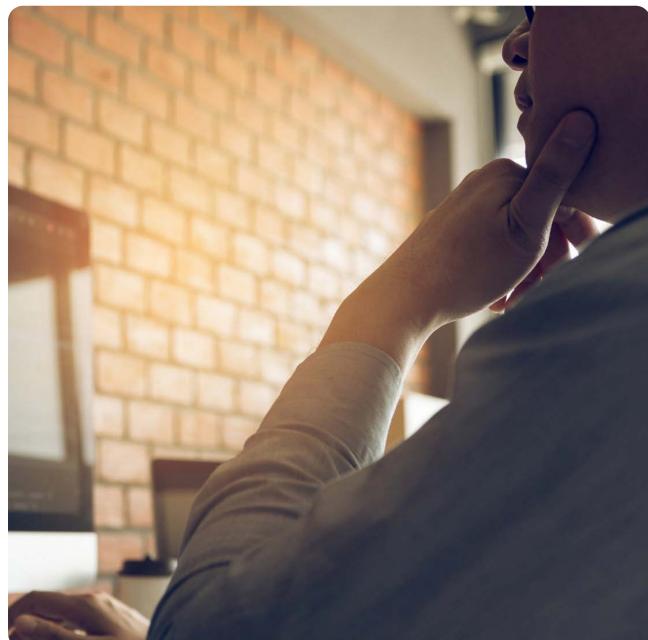


As **especificações** provêm um repertório semântico que possui todas as partes dos modelos de um sistema, sendo cada parte ligada às demais, de modo consistente. Logo, os diagramas da UML são ilustrações criadas a partir desse repertório, em que cada diagrama revela um aspecto específico do sistema.

#PraCegoVer:

Na imagem com um enquadramento fechado, existe um notebook com uma tela aberta em cima de uma mesa de madeira com uma pessoa digitando.

Normalmente, os elementos da UML têm uma notação gráfica direta e única, propiciando a reprodução visual das características mais importantes. Por exemplo, a notação de classe expõe os aspectos mais relevantes dela, como seu nome, suas operações e seus atributos. Já a especificação da classe poderá conter outros aspectos, como se a classe fosse abstrata ou como é a visibilidade de suas operações e atributos.



#PraCegoVer: Na imagem com um enquadramento fechado, existe um programador que está sentado em sua mesa de escritório em frente a um computador desktop. Ele está com a mão no queixo observando a tela, enquanto digita com a outra mão.

Em resumo, a maioria das características mencionadas pode ser representada como **adorno textual ou gráfico** para a notação básica da classe.

Note que todos os itens da notação UML serão criados com um símbolo básico inicial, do qual poderá ser adicionada uma variedade de adornos.

No caso das **divisões comuns**, no paradigma orientado a objetos, o mundo costuma ser dividido, pelo menos, em três partes, as quais podem ser vistas abaixo:

Divisão entre classes e objetos

Uma classe é uma abstração, enquanto um objeto é a manifestação concreta dessa abstração.

Separação de implementação e interface

Esta declara um contrato e a implementação representa a execução completa desse contrato, responsável pela semântica completa da interface.

Separação de papel e tipo

Este declara a classe de uma entidade (objetos, atributo ou parâmetro), enquanto o papel detalha o significado de uma entidade em seu contexto.

Para finalizarmos, temos os **mecanismos de extensão**. A UML provém uma linguagem padrão para criar a estrutura de projetos de sistemas, mas nem sempre uma única linguagem é suficiente para expressar todas as possíveis nuances dos modelos em qualquer domínio. Dessa forma, a UML é aberta, possibilitando que se amplie a linguagem de modo controlado.

Conheça, abaixo, as principais características que os mecanismos de extensão possuem:

Estereótipos

Expandem o vocabulário da UML, possibilitando a criação de novos tipos de blocos de construção, os quais são derivados dos já existentes, mas específicos a determinados problemas.

Valores atribuídos

Expandem as propriedades dos blocos de construção, possibilitando a criação de novas informações na especificação de um elemento.

Restrições

Costumam ampliar as semânticas dos blocos de construção, possibilitando adicionar novas regras ou alterar as já existentes.

Aqui, neste módulo, você estudou sobre os modelos de visões (de uso, de processo, de lógica, de implementação e de implantação), os blocos de construção, os itens presentes em tais blocos, os diferentes tipos de relacionamentos e diagramas e, por fim, as regras que os estabelecem.

É importante ressaltar que o conteúdo estudado contribuirá para que você, futuro programador, tenha uma visão micro e macro dos diferentes elementos e das fases que envolvem a criação dos softwares orientados a objetos, sobretudo, quando a linguagem UML for aplicada.

Fechamento

Parabéns! Você finalizou o curso de Linguagem de Modelagem Unificada (UML).

Como notou durante seus estudos, a UML nada mais é que um padrão de linguagem internacional expressiva utilizada na criação de projetos de softwares. Ela torna possível visualizar, especificar, construir e documentar os artefatos que fazem parte dos sistemas, mesmo sendo independente. Desse modo, as equipes de desenvolvimento conseguem visualizar e compreender o sistema de maneira mais organizada e padronizada.

Sempre que precisar, você pode retornar e rever o curso.

Boa sorte em sua jornada.

Até a próxima!

Referências

APS modelos cascata, iterativo incremental, prototipação, espiral e métodos ágeis. [S. I.], 30 ago. 2020. 1 vídeo (21 min). Publicado pelo canal Marcus Vinicius Amaral Rodrigues. Disponível em: https://www.youtube.com/watch?v=hh_SEogkF0M. Acesso em: 1 maio 2021.

ARAÚJO, E. C. de. Sobrecarga, herança, polimorfismo e exceção em C#. **Linha de Código**, [s. I.], [s. d.]. Disponível em: <http://www.linhadecodigo.com.br/artigo/2622/sobrecarga-heranca-polimorfismo-e-excecao-em-csharp.aspx>. Acesso em: 21 fev. 2021.

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

BOOCH, G.; JACOBSON, J.; RUMBAUGH, J. **UML**: guia do usuário. 2. ed. Rio de Janeiro: Elsevier, 2017. v. 2.

FREEPIK. Disponível em <https://br.freepik.com>

GUEDES, G. **UML 2**: uma abordagem prática. 3. ed. São Paulo: Novatec, 2018.

HARADA, E. 7 ferramentas on-line gratuitas para criar diagramas UML. **Profissionais TI**, [s. I.], 22 ago. 2018. Disponível em: <https://www.profissionaisti.com.br/7-ferramentas-online-gratuitas-para-criar-diagramas-uml/>. Acesso em: 2 maio 2021.

LIMA, A. da S. **UML 2.5**: do requisito à solução. São Paulo: Saraiva Educação S.A., 2018. v. 3.

NOGUEIRA, A. UML - Unified Modeling Language - Adornos e nós. **Linha de Código**, [s. I.], [s. d.]. Disponível em: <http://www.linhadecodigo.com.br/artigo/874/uml-unified-modeling-language-adornos-e-nos.aspx>. Acesso em: 2 maio 2021.

NOLETO, C. POO: tudo sobre programação orientada a objetos! **Trybe**, [s. I.], 9 ago. 2020. Disponível em: <https://blog.betrybe.com/tecnologia/poo-programacao-orientada-a-objetos/>. Acesso em: 1 maio 2021.

O QUE é programação orientada a objetos - Conceitos básicos de POO. [S. I.], 28 out. 2020. 1 vídeo (23 min). Publicado pelo canal Bóson Treinamentos. Disponível em: <https://www.youtube.com/watch?v=dG7LIYne2VA>. Acesso em: 1 maio 2021.

PAULA FILHO, W. de P. **Engenharia de software**: fundamentos, métodos e padrões. 2. ed. São Paulo: LTC, 2003.

PEXELS. Disponível em <http://www.pexels.com>

SILVA, A. M. R. da; VIDEIRA, C. A. E. **UML, metodologias e ferramentas**: case. Lisboa: Centro Atlântico, 2001. Disponível em: http://www.cesarkallas.net/arquivos/livros/informatica/UML_Metodologias_e_Ferramentas_CASE_portugues_.pdf. Acesso em: 2 maio 2021.

SILVA, A. P. C. da. Introdução a diagrama de classes e UML. **Slide Player**, [s. l.], 2014 Disponível em: <https://slideplayer.com.br/slide/363952/>. Acesso em: 21 fev. 2021.

UCHÔA, J. P. Evolução da metodologia do desenvolvimento de sistemas. **Linha de Código**, [s. l.], [s. d.]. Disponível em: <http://www.linhadecodigo.com.br/artigo/2108/evolucao-da-metodologia-do-desenvolvimento-de-sistemas.aspx>. Acesso em: 1 maio 2021.

UNIFIED Modeling Language. **OMG**: Object Management Group, Milford, 2017. Disponível em: <https://www.omg.org/spec/UML/About-UML/>. Acesso em: 31 jan. 2021.

VIEIRA, R. UML: diagrama de casos de uso. **Medium**, [s. l.], 12 dez. 2015. Disponível em: <https://medium.com/operacionalti/uml-diagrama-de-casos-de-uso-29f4358ce4d5>. Acesso em: 21 fev. 2021.

WAZLAWICK, R. S. **Análise e design orientados a objetos para sistemas de informação**: modelagem com UML, OCL e IFML. São Paulo: LTC, 2015.

