Visão Geral

Herysson R. Figueiredo
herysson.figueiredo@ufn.edu.br

Modelagem de sistema de software

Uma característica intrínseca de sistemas de software é a complexidade de seu desenvolvimento, que aumenta à medida que o tamanho do sistema cresce.

Modelagem de sistema de software

Para a construção de sistemas de software mais complexos, é necessário um planejamento inicial ou seja o desenvolvimento de um ***modelo.***

Razões para se utilizar modelos na construção de sistemas

Gerenciamento da complexidade: um dos principais motivos de utilizar modelos é que há limitações no ser humano em como lidar com a complexidade. Pode haver diversos modelos de um mesmo sistema, cada qual descrevendo uma perspectiva do sistema a ser construído.

Razões para se utilizar modelos na construção de sistemas

Comunicação entre as pessoas envolvidas: sem dúvida o desenvolvimento de um sistema envolve a execução de uma quantidade significativa de atividades. Essas atividades se traduzem em informações sobre o sistema em desenvolvimento. Grande parte dessas informações corresponde aos modelos criados para representar o sistema

Razões para se utilizar modelos na construção de sistemas

Redução dos custos no desenvolvimento: no desenvolvimento de sistemas, seres humanos estão invariavelmente sujeitos a cometer erros, que podem ser tanto individuais quanto de comunicação entre os membros da equipe.

Razões para se utilizar modelos na construção de sistemas

Previsão do comportamento futuro do sistema: o comportamento do sistema pode ser discutido mediante uma análise dos seus modelos. Os modelos servem como um “laboratório”, em que diferentes soluções para um problema relacionado à construção do sistema podem ser experimentadas.

O paradigma da Orientação a Objetos

Pode-se dizer, então, que o termo “paradigma da orientação a objetos” é uma forma de abordar um problema.

O paradigma da Orientação a Objetos

Há alguns anos, Alan Kay, um dos pais do paradigma da orientação a objetos, formulou a chamada “analogia biológica”. Por meio dela, ele imaginou um sistema de software que funcionasse como um ser vivo. Nesse sistema, cada “célula” interagiria com outras células através do envio de mensagens com o objetivo realizar um objetivo comum. Além disso, cada célula se comportaria como uma unidade autônoma.

O paradigma da Orientação a Objetos

De uma forma mais geral, Kay pensou em como construir um sistema de software a partir de agentes autônomos que interagem entre si. Ele estabeleceu então os seguintes princípios da orientação a objetos:

- Qualquer coisa é um objeto.
- Objetos realizam tarefas por meio da requisição de serviços a outros objetos.
- Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares.
- A classe é um repositório para comportamento associado ao objeto.
- Classes são organizadas em hierarquias.

O paradigma da Orientação a Objetos

O paradigma da orientação a objetos visualiza um sistema de software como uma coleção de agentes interconectados chamados objetos. Cada objeto é responsável por realizar tarefas específicas. Para cumprir com algumas das tarefas sob sua responsabilidade, um objeto pode ter que interagir com outros objetos. É pela interação entre objetos que uma tarefa computacional é realizada

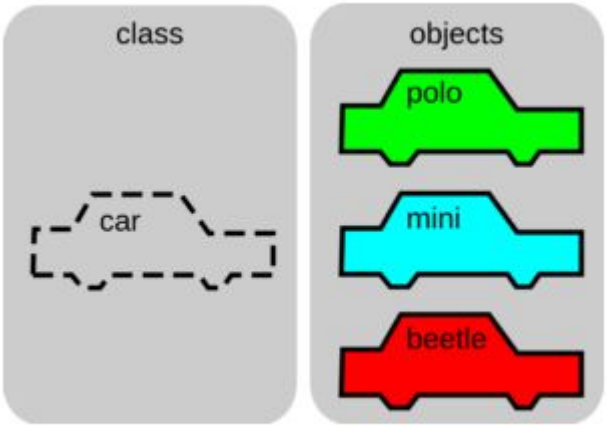
O paradigma da Orientação a Objetos

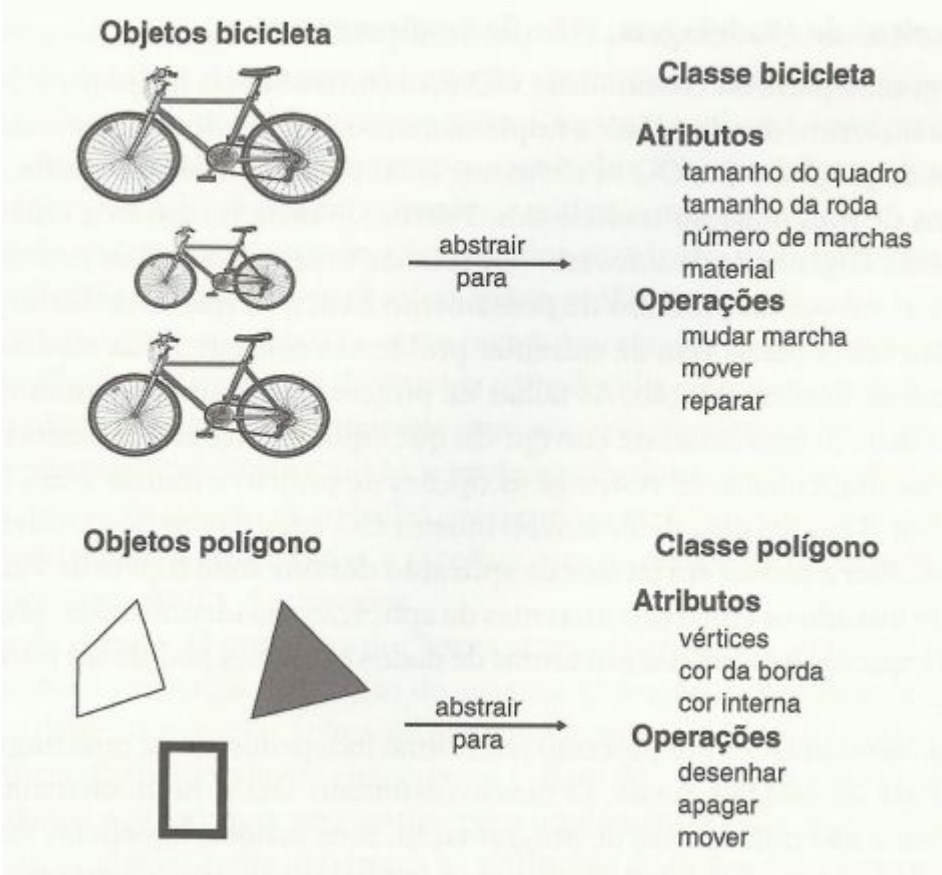
Um sistema de software orientado a objetos consiste em objetos em colaboração com o objetivo de realizar as funcionalidades desse sistema. Cada objeto é responsável por tarefas específicas. É graças à cooperação entre objetos que a computação do sistema se desenvolve.

Classe e Objeto

“Uma classe é como um molde a partir do qual objetos são construídos”

Classe e Objeto





Operação, mensagem e estado

Dá-se o nome de **operação** a alguma ação que um objeto sabe realizar quando solicitado.

Objetos não executam suas operações aleatoriamente. Para que uma operação em um objeto seja executada, deve haver um estímulo enviado a esse objeto.

Operação, mensagem e estado

Seja qual for a origem do estímulo, quando ele ocorre diz-se que o objeto em questão está recebendo uma **mensagem** requisitando que ele realize alguma operação.

Quando se diz na terminologia de orientação a objetos que **objetos de um sistema estão trocando mensagens** significa que esses objetos estão enviando mensagens uns aos outros com o objetivo de realizar alguma tarefa dentro do sistema no qual eles estão inseridos.

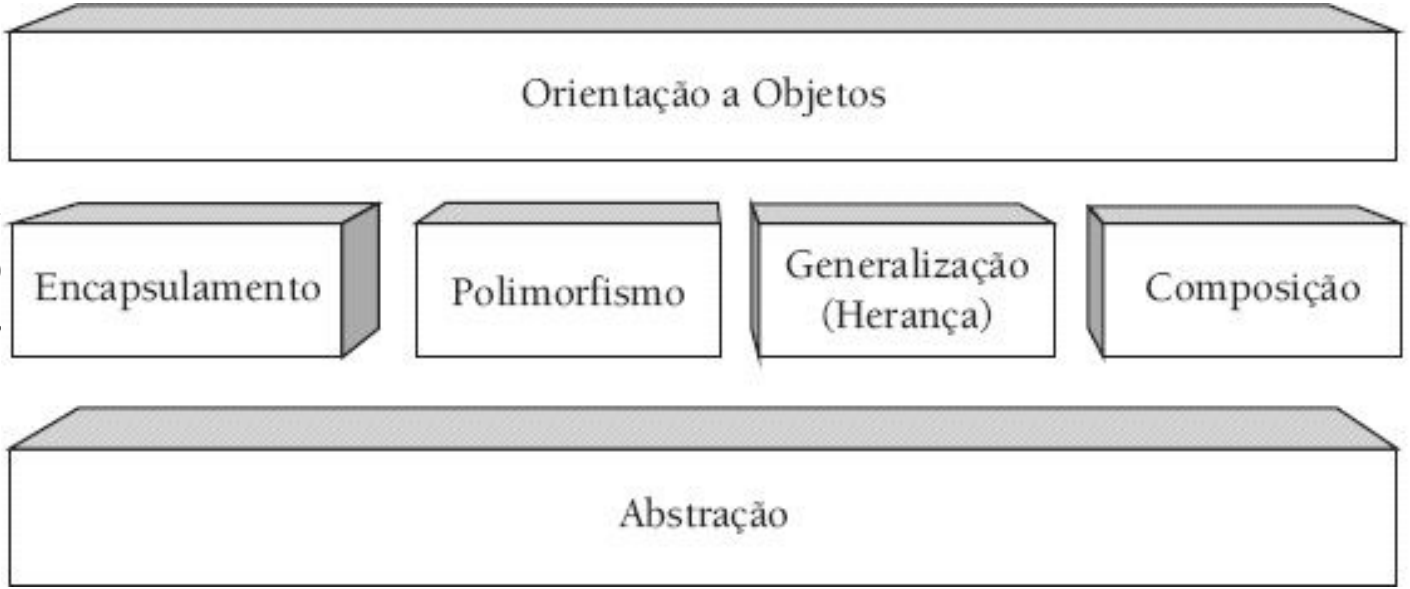
Operação, mensagem e estado

Por definição, o **estado** de um objeto corresponde ao conjunto de valores de seus atributos em um dado momento. Uma mensagem enviada a um objeto tem o potencial de mudar o estado desse objeto.

O Papel da abstração na orientação a objetos

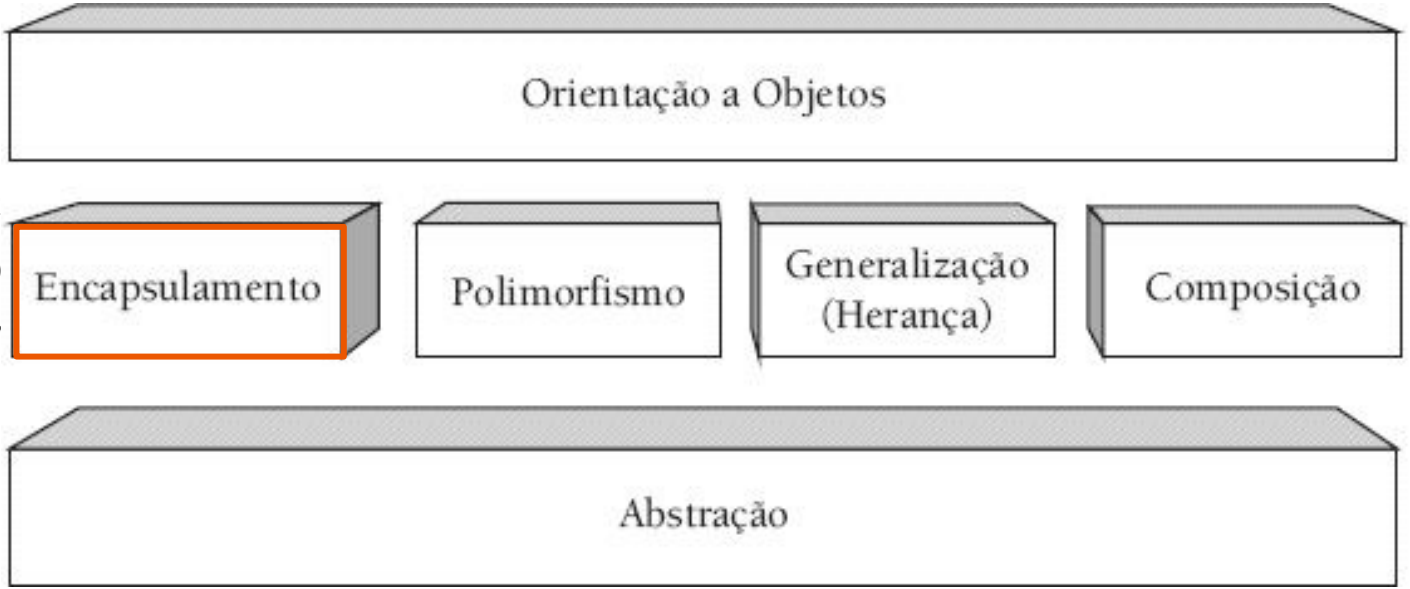
A **abstração** é um processo mental pelo qual nós, seres humanos, nos atemos aos aspectos mais importantes (relevantes) de alguma coisa, ao mesmo tempo em que ignoramos os menos importantes.

O Papel da abstração na orientação a objetos



A **ab** ...os
atem ... ao
mesm

O Papel da abstração na orientação a objetos

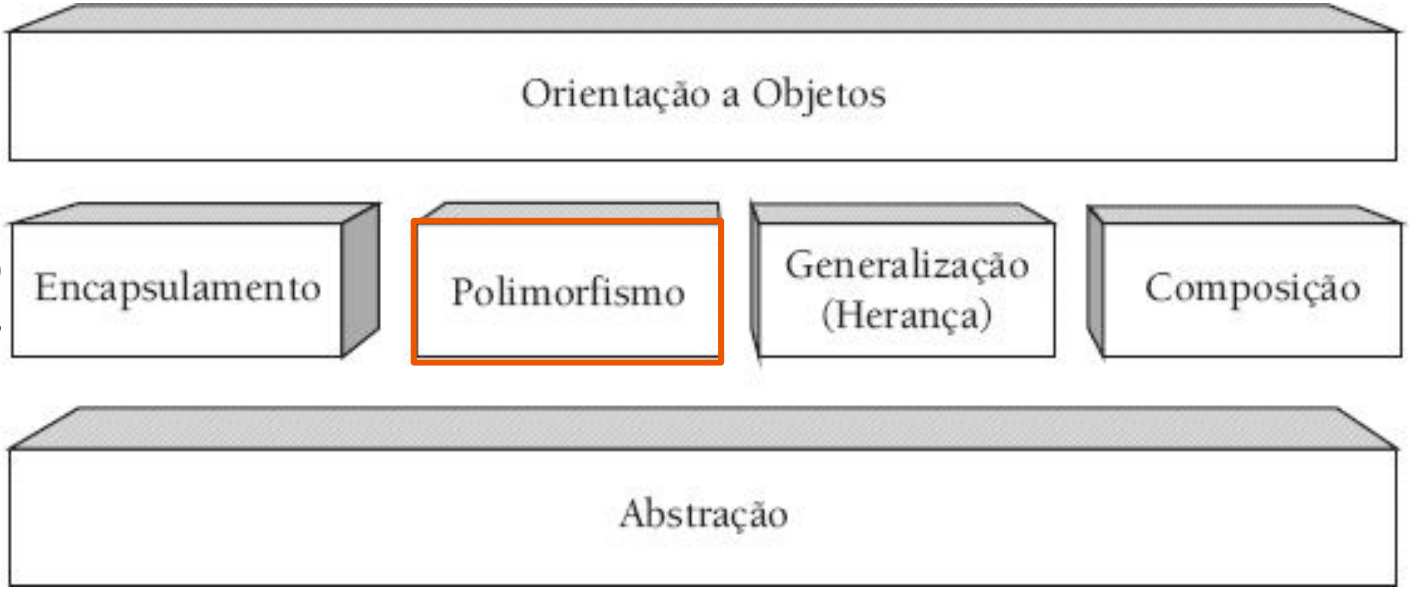


A **ab** ...os
atem ... ao
mesm

Encapsulamento

O mecanismo de **encapsulamento** é uma forma de restringir o acesso ao comportamento interno de um objeto.

O Papel da abstração na orientação a objetos

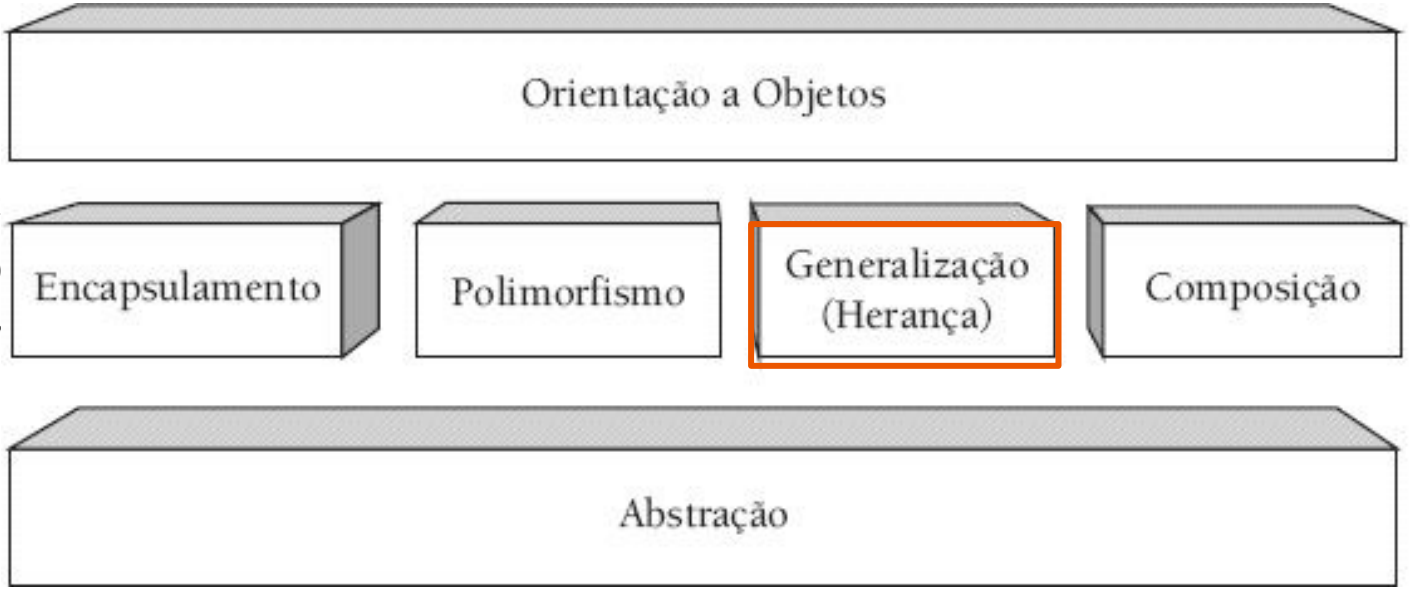


A **ab** ...os
atem ... ao
mesm

Polimorfismo

Polimorfismo é o princípio pelo qual duas ou mais classes derivadas da mesma superclasse podem invocar métodos que têm a mesma assinatura, mas comportamentos distintos.

O Papel da abstração na orientação a objetos



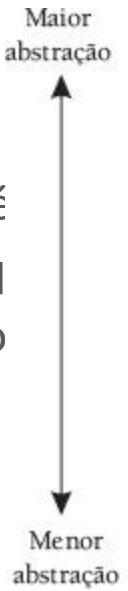
A **ab** ...os
atem ... ao
mesm

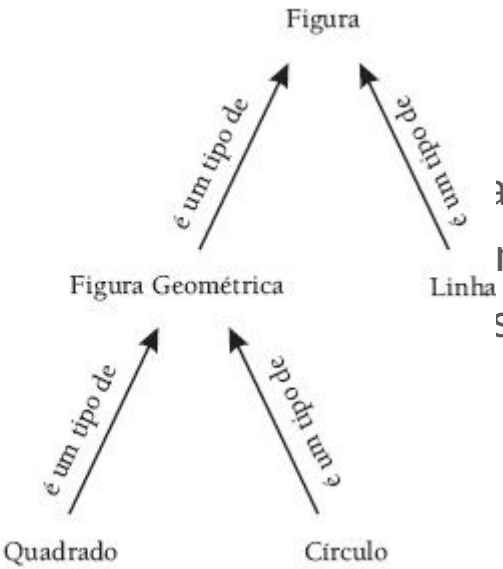
Generalização (Herança)

A generalização é outra forma de abstração utilizada na orientação a objetos. Declara que as características e o comportamento comuns a um conjunto de objetos podem ser abstraídos em uma classe.

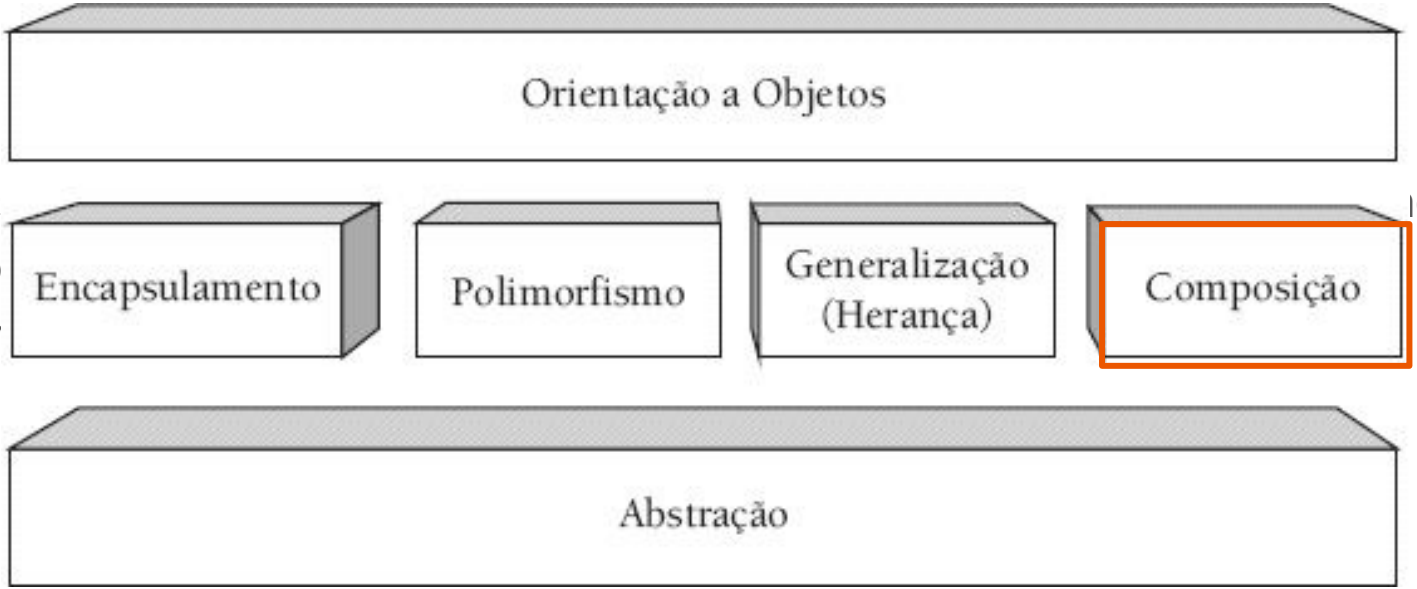
Generalização (Herança)

A generalização é ada na orientação a objetos. Declara q mento comuns a um conjunto de objeto se.





O Papel da abstração na orientação a objetos



A **ab** ...os
atem ...ao
mesm

Composição

“Objetos compostos por outros objetos”

UML e Visões de um sistema

UML

A construção da UML - *Unified Modeling Language* (Linguagem de Modelagem Unificada) teve muitos contribuintes, mas os principais atores no processo foram Grady Booch, James Rumbaugh e Ivar Jacobson. Esses três pesquisadores costumam ser chamados de “os três amigos”.

UML

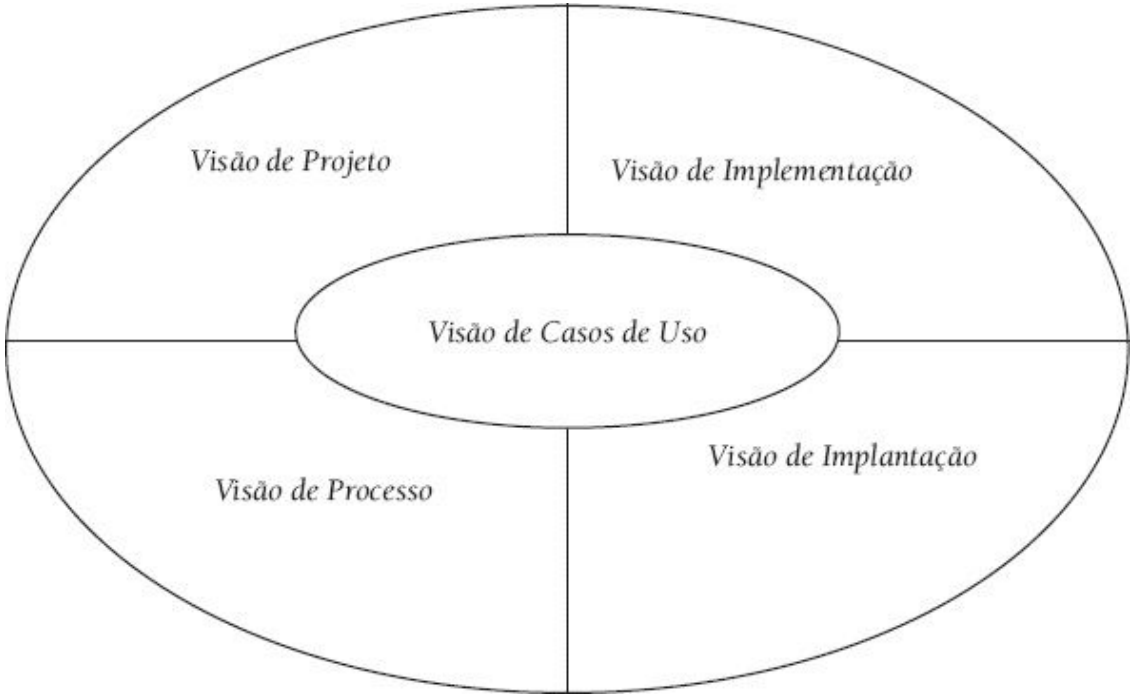
No processo de definição inicial da UML, esses pesquisadores buscaram aproveitar o melhor das características das notações preexistentes, principalmente das técnicas que haviam proposto anteriormente (essas técnicas eram conhecidas pelos nomes Booch Method, OMT e OOSE).

Visões de um sistema

O desenvolvimento de um sistema de software complexo demanda que seus desenvolvedores tenham a possibilidade de examinar e estudar esse sistema a partir de perspectivas diversas.

Os autores da UML sugerem que um sistema pode ser descrito por cinco visões interdependentes desse sistema.

Visões de um sistema



Visão de Casos de Uso

Descreve o sistema de um ponto de vista externo como um conjunto de interações entre o sistema e os agentes externos ao sistema. Esta visão é criada em um estágio inicial e direciona o desenvolvimento das outras visões do sistema.

Visão de Projeto

Enfatiza as características do sistema que dão suporte, tanto estrutural quanto comportamental, às funcionalidades externamente visíveis do sistema.

Visão de Implementação

Abrange o gerenciamento de versões do sistema, construídas pelo agrupamento de módulos (componentes) e subsistemas.

Visão de Implantação

Corresponde à distribuição física do sistema em seus subsistemas e à conexão entre essas partes.

Visão de Processo

Esta visão enfatiza as características de concorrência (paralelismo), sincronização e desempenho do sistema.

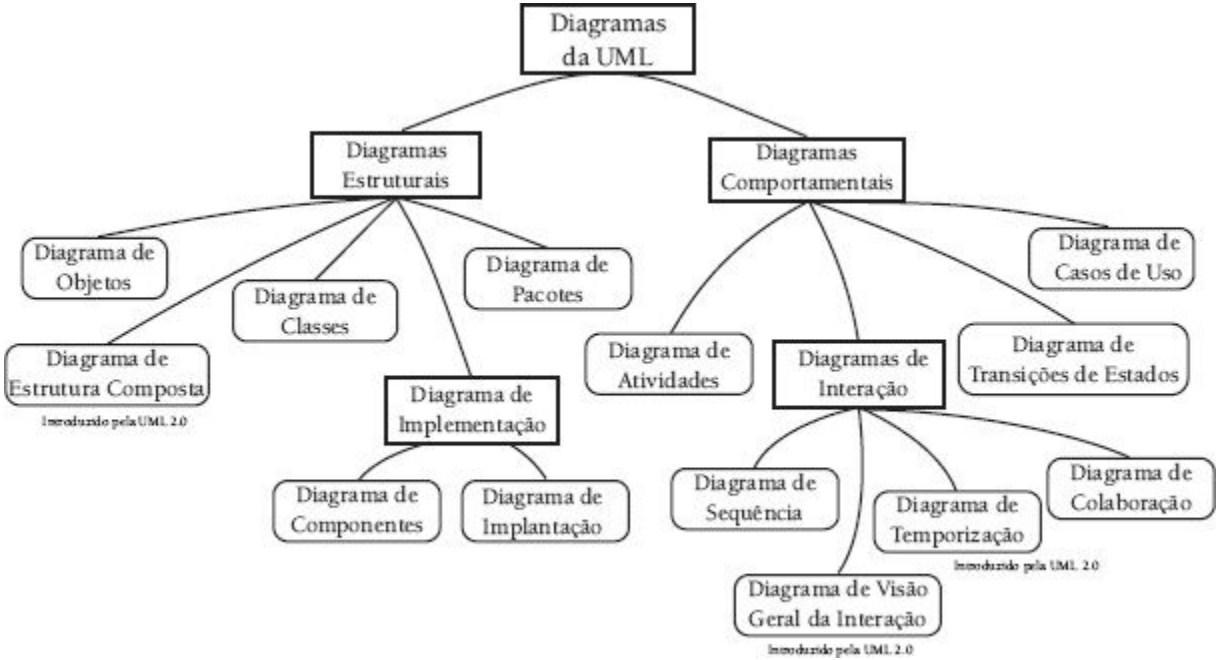
Diagramas da UML

Um processo de desenvolvimento que utilize a UML como linguagem de suporte à modelagem envolve a criação de diversos documentos. Esses documentos podem ser textuais ou gráficos. Na terminologia da UML, eles são denominados **artefatos de software**, ou simplesmente artefatos. São os artefatos que compõem as visões do sistema.

Diagramas da UML

Os artefatos gráficos produzidos durante o desenvolvimento de um sistema de software orientado a objetos (SSOO) podem ser definidos pela utilização dos diagramas da UML. Os 13 diagramas da UML 2.0

Diagramas da UML



Processo de desenvolvimento de software

Atividades típicas de um processo de desenvolvimento

Um processo de desenvolvimento classifica em atividades as tarefas realizadas durante a construção de um sistema de software. Há vários processos de desenvolvimento propostos. Por outro lado, é um consenso na comunidade de desenvolvimento de software o fato de que não existe um melhor processo, aquele que melhor se aplica a todas as situações de desenvolvimento

Atividades típicas de um processo de desenvolvimento

Cada processo tem suas particularidades em relação ao modo de arranjar e encadear as atividades de desenvolvimento. Entretanto, podem-se distinguir atividades que, com uma ou outra modificação, são comuns à maioria dos processos existentes

Levantamento de Requisitos

A atividade de levantamento de requisitos (também conhecida como elicitação de requisitos) corresponde à etapa de compreensão do problema aplicada ao desenvolvimento de software. O principal objetivo do levantamento de requisitos é que usuários e desenvolvedores tenham a mesma visão do problema a ser resolvido

Levantamento de Requisitos

Normalmente os requisitos de um sistema são identificados a partir de um **domínio**. Denomina-se **domínio** a área de conhecimento ou atividade específica caracterizada por um conjunto de conceitos e de terminologia compreendidos por especialista nessa área.

Levantamento de Requisitos

No contexto do desenvolvimento de software, um domínio corresponde à parte do mundo real que é **relevante**, no sentido de que algumas informações e processos desse domínio precisam ser incluídos no sistema em desenvolvimento. **O domínio também é chamado de domínio do problema ou domínio do negócio.**

Levantamento de Requisitos

O produto do levantamento de requisitos é o documento de requisitos, que declara os diversos tipos de requisitos do sistema. É normal esse documento ser escrito em uma notação informal (em linguagem natural). As principais seções de um documento de requisitos são: ***Requisitos funcionais, Requisitos não funcionais e Requisitos normativos.***

Requisitos funcionais

Definem as funcionalidades do sistema. Alguns exemplos de requisitos funcionais são os seguintes:

- “O sistema deve permitir que cada professor realize o lançamento de notas das turmas nas quais lecionou.”
- “O sistema deve permitir que um aluno realize a sua matrícula nas disciplinas oferecidas em um semestre letivo.”
- “Os coordenadores de escola devem poder obter o número de aprovações, reprovações e trancamentos em cada disciplina oferecida em um determinado período.”

Requisitos não funcionais

Declaram as características de qualidade que o sistema deve possuir e que estão relacionadas às suas funcionalidades. Alguns tipos de requisitos não funcionais são os seguintes:

- Confiabilidade: corresponde a medidas quantitativas da confiabilidade do sistema, tais como tempo médio entre falhas, recuperação de falhas ou quantidade de erros por milhares de linhas de código-fonte.
- Desempenho: requisitos que definem os tempos de resposta esperados para as funcionalidades do sistema.

Requisitos não funcionais

Declaram as características de qualidade que o sistema deve possuir e que estão relacionadas às suas funcionalidades. Alguns tipos de requisitos não funcionais são os seguintes:

- Portabilidade: restrições quanto às plataformas de hardware e software nas quais o sistema será implantado e quanto ao grau de facilidade para transportar o sistema para outras plataformas.
- Segurança: limitações sobre a segurança do sistema em relação a acessos não autorizados.
- Usabilidade: requisitos que se relacionam ou afetam a usabilidade do sistema. Exemplos incluem requisitos sobre a facilidade de uso e a necessidade ou não de treinamento dos usuários.

Requisitos normativos

Declaração de restrições impostas sobre o desenvolvimento do sistema. Restrições definem, por exemplo, a adequação a custos e prazos, a plataforma tecnológica, aspectos legais (licenciamento), limitações sobre a interface com o usuário, componentes de hardware e software a serem adquiridos, eventuais necessidades de comunicação do novo sistema com sistemas legados etc.

Requisitos normativos

Restrições também podem corresponder a **regras do negócio**, restrições ou políticas de funcionamento específicas do domínio do problema que influenciarão de um modo ou de outro no desenvolvimento do software

Levantamento de requisito

Uma das formas de se medir a qualidade de um sistema de software é pela sua utilidade. E um sistema será útil para seus usuários se atender aos requisitos definidos e se esses requisitos refletirem as necessidades dos usuários.

Análise

De acordo com o professor Wilson de Pádua, as fases de levantamento de requisitos e de análise de requisitos recebem o nome de engenharia de requisitos (PÁDUA, 2003).

Análise

Formalmente, o termo análise corresponde a “quebrar” um sistema em seus componentes e estudar como eles interagem entre si com o objetivo de entender como esse sistema funciona.

Análise

No contexto dos sistemas de software, esta é a etapa na qual os analistas realizam um estudo detalhado dos requisitos levantados na atividade anterior. A partir desse estudo, **são construídos modelos** para representar o sistema a ser construído.

Análise

As principais ferramentas da UML para realizar análise são o **diagrama de casos de uso** e o **diagrama de classes** (para a modelagem de casos de uso e de classes, respectivamente). Outros diagramas da UML também utilizados na análise são: diagrama de interação, diagrama de estados e diagrama de atividades

Análise

O foco principal da análise são os aspectos lógicos e independentes de implementação de um sistema.

Projeto

Na fase de projeto, determina-se “como” o sistema funcionará para atender aos requisitos, de acordo com os recursos tecnológicos existentes (a fase de projeto considera os aspectos físicos e dependentes de implementação).

Projeto

Aos modelos construídos na fase de análise são adicionadas as denominadas “restrições de tecnologia”. Exemplos de aspectos a serem considerados na fase de projeto: arquitetura física do sistema, padrão de interface gráfica, algoritmos específicos, o gerenciador de banco de dados a ser utilizado

Projeto

A fase de projeto consiste em duas atividades principais: ***projeto da arquitetura*** (também conhecido como projeto de alto nível) e ***projeto detalhado*** (também conhecido como projeto de baixo nível).

Projeto

Projeto da arquitetura consiste em distribuir as classes de objetos relacionadas do sistema em subsistemas e seus componentes. Consiste também em distribuir esses componentes fisicamente pelos recursos de hardware disponíveis. Os diagramas da UML normalmente utilizados nessa fase do projeto são os diagramas de **implementação.**

Projeto

No **projeto detalhado, s**ão modeladas as colaborações entre os objetos de cada módulo com o objetivo de realizar suas funcionalidades. Também são realizados o projeto da interface com o usuário e o projeto de banco de dados, bem como são considerados aspectos de concorrência e distribuição do sistema. Os diagramas da UML que podem ser utilizados nesta fase de projeto são: **diagrama de classes**, **diagrama de casos de uso**, **diagrama de interação**, **diagrama de estados** e **diagrama de atividades**.

Projeto

Embora a **análise** e o **projeto** sejam descritos separadamente neste, é importante notar que, durante o desenvolvimento, não há uma distinção assim tão clara entre essas duas fases. As atividades dessas duas fases frequentemente se misturam, principalmente no desenvolvimento de sistemas orientados a objetos.

Implementação

Na fase de implementação, o sistema é codificado, ou seja, ocorre a tradução da descrição computacional obtida na fase de projeto em código executável mediante o uso de uma ou mais linguagens de programação.

Teste

Diversas atividades de teste são realizadas para verificação do sistema construído, levando-se em conta a especificação feita nas fases de análise e de projeto. Um possível produto dessa fase são os **relatórios de testes**, que apresentam informações sobre erros detectados no software.

Teste

Idealmente, testes devem ser realizados de forma hierárquica. Isso quer dizer que partes pequenas do software podem ser testadas em separado e, a seguir, unidas a outras partes, para formar um componente maior.

Teste

Testes de unidades são realizados sobre elementos do código-fonte do sistema. No contexto do desenvolvimento orientado a objetos, esses elementos correspondem a classes ou mesmo a métodos de uma classe.

Teste

Os **testes de integração** são realizados após os testes de unidades. No contexto de um sistema orientado a objetos, o elemento de uma atividade de teste de integração pode ser uma operação de sistema.

Implantação

Na fase de implantação, o sistema é empacotado, distribuído e instalado no ambiente do usuário. Os manuais do sistema são escritos, os arquivos são carregados, os dados são importados para o sistema e os usuários treinados para utilizar o sistema corretamente. Em alguns casos, nesse momento também ocorre a migração de sistemas de software e de dados preexistentes

Participantes do processo

Componentes humanos

Uma equipe de desenvolvimento de software típica consiste em um gerente, analistas, projetistas, programadores, clientes e grupos de avaliação de qualidade. Esses participantes do processo de desenvolvimento são descritos a seguir

Gerente de projeto

O **gerente de projetos** é o profissional responsável pela gerência ou coordenação das atividades necessárias à construção do sistema.

- orçamento do projeto de desenvolvimento,
- estimar o tempo necessário para o desenvolvimento do sistema,
- definir qual o processo de desenvolvimento,
- o cronograma de execução das atividades,
- a mão de obra especializada,
- os recursos de hardware e software etc.

Gerente de projeto

O **gerente de projetos** é o profissional responsável pela gerência ou coordenação das atividades necessárias à construção do sistema.

- orçamento do projeto de desenvolvimento,
- estimar o tempo necessário para o desenvolvimento do sistema,
- definir qual o processo de desenvolvimento,
- o cronograma de execução das atividades,
- a mão de obra especializada,
- os recursos de hardware e software etc.

Analistas

O analista de sistemas é o profissional que deve ter conhecimento do domínio do negócio e entender seus problemas para que possa definir os requisitos do sistema a ser desenvolvido. Analistas devem estar aptos a se comunicar com especialistas do domínio para obter conhecimento acerca dos problemas e das necessidades envolvidas na organização empresarial

Projetista

O projetista de sistemas é o integrante da equipe de desenvolvimento cujas funções são:

- Avaliar as alternativas de solução (da definição) do problema resultante da análise;
- Gerar a especificação de uma solução computacional detalhada. A tarefa do projetista de sistemas é muitas vezes chamada de projeto físico

Projetista

Na prática, existem diversos tipos de projetistas. Pode-se falar em:

- projetistas de interface
- de redes
- de bancos de dados

O ponto comum a todos eles é que todos trabalham nos modelos resultantes da análise para adicionar os aspectos tecnológicos a tais modelos.

Arquitetos de software

Um profissional encontrado principalmente em grandes equipes reunidas para desenvolver sistemas complexos é o arquiteto de software. O objetivo desse profissional é elaborar a arquitetura do sistema como um todo. É ele quem toma decisões sobre quais subsistemas compõem o sistema como um todo e quais são as interfaces entre esses subsistemas.

Arquitetos de software

Além de tomar decisões globais, o arquiteto também deve ser capaz de tomar decisões técnicas detalhadas (p. ex., decisões que têm influência no desempenho do sistema). Esse profissional trabalha em conjunto com o gerente de projeto para priorizar e organizar o plano de projeto.

Programadores

São os responsáveis pela **implementação do sistema**. É comum haver vários programadores em uma equipe de desenvolvimento. Um programador pode ser proficiente em uma ou mais linguagens de programação, além de ter conhecimento sobre bancos de dados e poder ler os modelos resultantes do trabalho do projetista.

Programadores

Na verdade, a maioria das equipes de desenvolvimento possui analistas que realizam alguma programação e programadores que realizam alguma análise

Especialistas do domínio

O **especialista do domínio**, também conhecido como **especialista do negócio**. Esse componente é o indivíduo, ou um grupo deles, que possui conhecimento acerca da área ou do negócio em que o sistema em desenvolvimento estará inserido. Um termo mais amplo que especialista de domínio é cliente.

Especialistas do domínio

Podem-se distinguir dois tipos de clientes: o **cliente usuário** e o **cliente contratante.**

- **O cliente usuário** é o indivíduo que efetivamente utilizará o sistema. Ele normalmente é um especialista do domínio. É com esse tipo de cliente que o analista de sistemas interage para levantar os requisitos do sistema.

Especialistas do domínio

Podem-se distinguir dois tipos de clientes: o **cliente usuário** e o **cliente contratante.**

- **cliente contratante** é o indivíduo que solicita o desenvolvimento do sistema. Ou seja, é quem encomenda e patrocina os custos de desenvolvimento e manutenção.

Avaliadores de qualidade

O desempenho e a confiabilidade são exemplos de características que devem ser encontradas em um sistema de software de boa qualidade. Avaliadores de qualidade asseguram a adequação do processo de desenvolvimento e do produto de software sendo desenvolvido aos padrões de qualidade estabelecidos pela organização.

Modelo de Ciclo de Vida

Exercícios

Desenvolva os exercícios da lista 1

Bibliografia

BEZERRA, Eduardo. Princípios de Análise e Projeto de Sistemas com UML. 2. ed. Rio de Janeiro: Elsevier, 2006.

LARMAN, Craig. Utilizando UML e padrões. 3. ed. Porto Alegre: Bookman, 2007.

WAZLAWICK, Raul. Análise e Projetos de Sistemas de Informação orientados a objetos. 2. ed. Rio de Janeiro: Campus, 2011

Bibliografia

BLAHA, Michael; RUMBAUGH, James. Modelagem e projetos baseados em objetos com UML 2. 2. ed. Rio de Janeiro: Elsevier, 2006

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. UML: guia do usuário. 2. ed. Rio de Janeiro: Campus, 2006.

GAMMA, Erich. Padrões de projeto: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

HUMPHREY, Watts S. A discipline for software engineering. 7. ed. Massachusetts: Addison-Wesley, 1997.

SOMMERVILLE, Ian. Engenharia de Software. 8. ed. São Paulo: Prentice Hall, 2007.