CONCEITOS E PILARES DE ANÁLISE E PROJETO ORIENTADOS A OBJETOS

PARADIGMA ORIENTADO A OBJETOS

Com o aumento do tamanho do código e da complexidade dos programas, o paradigma estruturado, que antecedeu o paradigma orientado a objetos, começou a apresentar limitações nos sistemas sob o ponto de vista da dificuldade de manutenção e reuso de programas e rotinas padronizadas.

Vamos, inicialmente, definir o termo paradigma como a maneira de abordar um problema.

A orientação a objetos surge como solução a esses problemas, permitindo − mediante propriedades como abstração, encapsulamento, herança e polimorfismo − maior organização, reaproveitamento e extensibilidade de código e, consequentemente, programas mais fáceis de serem escritos e mantidos.

O principal foco do paradigma orientado a objetos é permitir o desenvolvimento de sistemas de forma mais rápida e confiável.

Um dos criadores do paradigma orientado a objetos, Alan Kay, imaginou um sistema como um conjunto de agentes autônomos, os objetos, que interagem entre si.

Ele definiu os princípios centrais da orientação a objetos:

Qualquer coisa do mundo real é um objeto
Objetos realizam tarefas requisitando serviços a outros objetos
Os objetos similares são agrupados em classes e cada objeto pertence a uma classe
A classe determina o comportamento possível a um objeto
Classes são organizadas em hierarquias

O paradigma da orientação a objetos visualiza um sistema de software como uma coleção de agentes interconectados chamados objetos. Cada um deles é responsável por realizar tarefas específicas e, para cumprir com algumas das tarefas sob sua responsabilidade, um objeto pode ter que interagir com outros. É pela interação entre eles que uma tarefa computacional é executada. Um sistema de software orientado a objetos consiste, portanto, em objetos colaborando para realizar as funcionalidades desse sistema. É graças à cooperação entre os objetos que a computação do sistema se desenvolve.

(BEZERRA, 2015)

Atenção
Observação: Ao longo do texto, abreviaremos o termo "orientação a objetos" como O.O.

CONCEITOS FUNDAMENTAIS DA ORIENTAÇÃO A OBJETOS

A orientação a objetos enfatiza a identificação, a representação e a organização dos objetos necessários ao funcionamento de um sistema. Tem por base os conceitos de objetos, classes, operação, mensagem e estado, e está calcada em quatro pilares fundamentais: abstração, encapsulamento, herança e polimorfismo, discutidos na sequência.

Neste tópico, adentraremos nesses conceitos fundamentais da orientação a objetos.

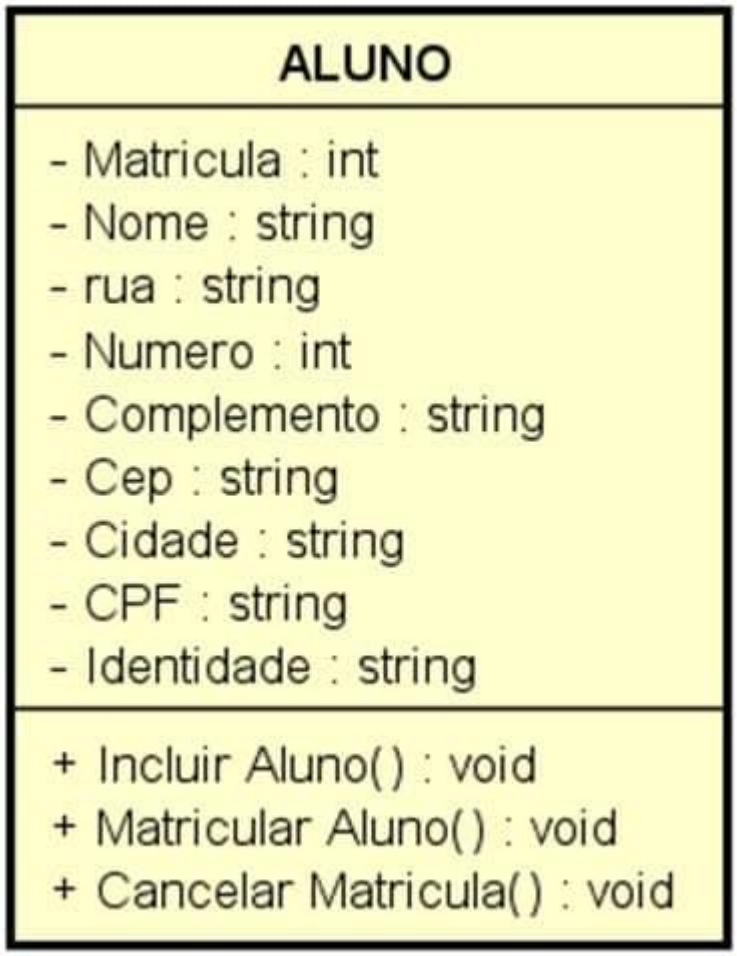
Objetos e classes

Um objeto pode referenciar qualquer coisa do mundo real: um aluno, uma disciplina, um curso, um professor, entre outros, considerando um sistema acadêmico como contexto. Ou seja, um objeto é qualquer coisa do mundo real, de interesse no contexto em estudo.

Quando analisamos os objetos pertinentes a um contexto, não estamos preocupados com um objeto específico como, por exemplo, o aluno “José Carlos Aragão”, e sim com todos os alunos envolvidos no estudo. Surge então o conceito de classe que, conceitualmente, reúne (agrupa) um conjunto de objetos com as mesmas propriedades. Ou seja, estamos interessados em todos os alunos e não apenas em “José Carlos Aragão”. Usando o princípio da abstração (que detalharemos mais adiante), temos que uma classe agrupa objetos com as mesmas características ou propriedades, que são seus dados (atributos) e seus procedimentos (métodos) que implementam os serviços que essa classe vai prestar. A classe ALUNO agrupa “José Carlos Aragão” e os demais alunos envolvidos. **Um objeto é um elemento específico de uma classe, ou ainda uma instância de uma classe.**

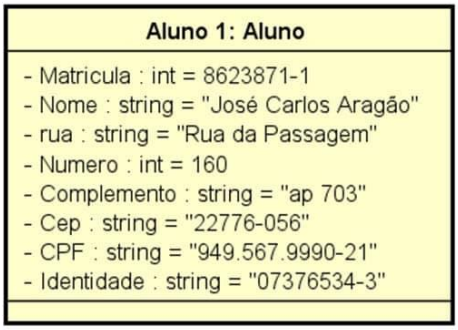
As classes são, portanto, abstrações que definem uma estrutura que encapsula dados (chamados de atributos) e um conjunto de operações possíveis de serem usados, chamados métodos. Por exemplo, a classe ALUNO encapsula um conjunto de dados que identifique os alunos − matrícula, nome, endereço (rua, número, complemento, estado e CEP), CPF e Identidade − e um conjunto de métodos: Incluir Aluno, Matricular Aluno, Cancelar Matrícula, dentre outros.

Resumindo
Classe: abstração das características de um grupo de coisas do mundo real.
Objeto: um elemento específico de uma classe ou uma instância de uma classe.
A seguir, temos a representação de uma classe em três compartimentos: o nome da classe (ALUNO), seus atributos (Matrícula... Identidade) e métodos (Incluir Aluno... Cancelar Matrícula).

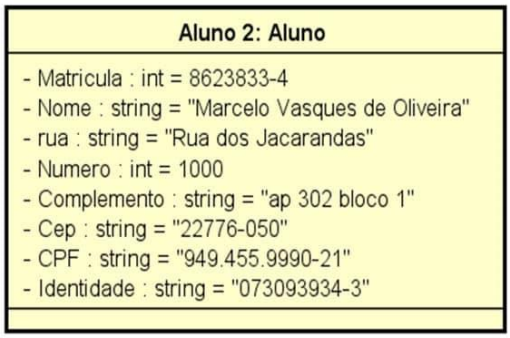


Classe ALUNO

A seguir, temos a representação de dois objetos da classe ALUNO:



Objeto − Aluno 1 da classe ALUNO.



Objeto Aluno 2 da classe ALUNO

Operação, mensagem e estado

Um sistema orientado a objetos consiste na cooperação entre seus objetos. Cada um tem uma responsabilidade no sistema, correspondendo à parte das funcionalidades que lhes são atribuídas. Em outras palavras, uma tarefa computacional é realizada pela interação entre seus objetos, cada um executa parte da tarefa.

OPERAÇÃO

Operação é o nome dado a cada ação (função) que o objeto sabe realizar. Mas um objeto não realiza nenhuma ação sem uma motivação, sem um estímulo.

MENSAGEM

Chamamos esse estímulo de **mensagem**, que chega a um objeto e solicita que ele realize uma de suas operações. Uma operação, por sua vez, pode ser implementada por meio de pelo menos um método.

Em outras palavras, cada objeto presta um serviço. Quando um objeto precisa de um serviço da responsabilidade de outro, ele precisa enviar uma **mensagem** a ele. Cada mensagem ativa uma das operações do objeto.

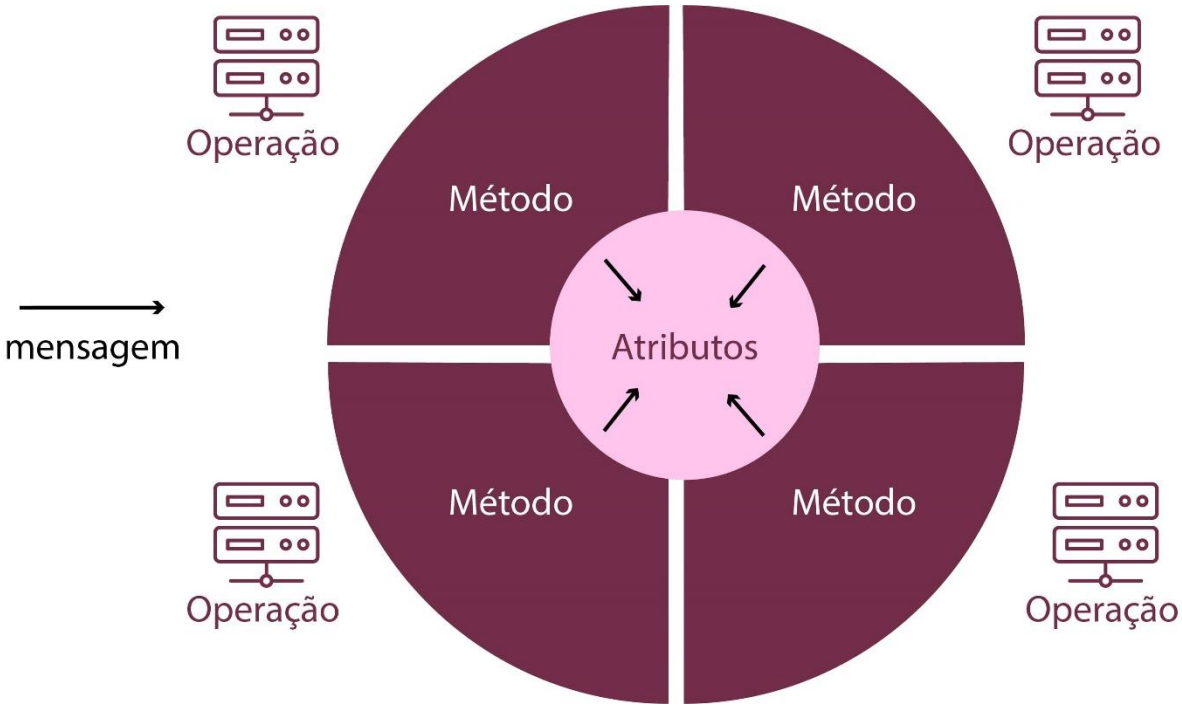
ESTADO

Já sabemos que um objeto contém atributos, que são dados necessários para prestar os serviços que cabem a esse objeto. Por definição, chama-se **estado do objeto** o conjunto de valores de seus atributos em dado momento. Uma mensagem enviada a um objeto pode (ou não) alterar o seu estado, na medida em que pode alterar um ou mais valores de seus atributos.

Objeto

Imagine, por exemplo, que o objeto Notas_Aluno contenha as notas do aluno em determinada disciplina. Em dado momento, é recebida uma mensagem informando uma nova nota a ser armazenada. O estado desse objeto foi alterado, pois um de seus atributos recebeu a nova nota. Agora suponha que determinado objeto enviou uma

mensagem a Notas_Aluno solicitando que seja exibida a média atual; nesse caso, não houve alteração de estado, pois as notas foram consultadas, e a média foi calculada (não fica armazenada) e exibida.



Partes de um objeto

OS PILARES DA ORIENTAÇÃO A OBJETOS

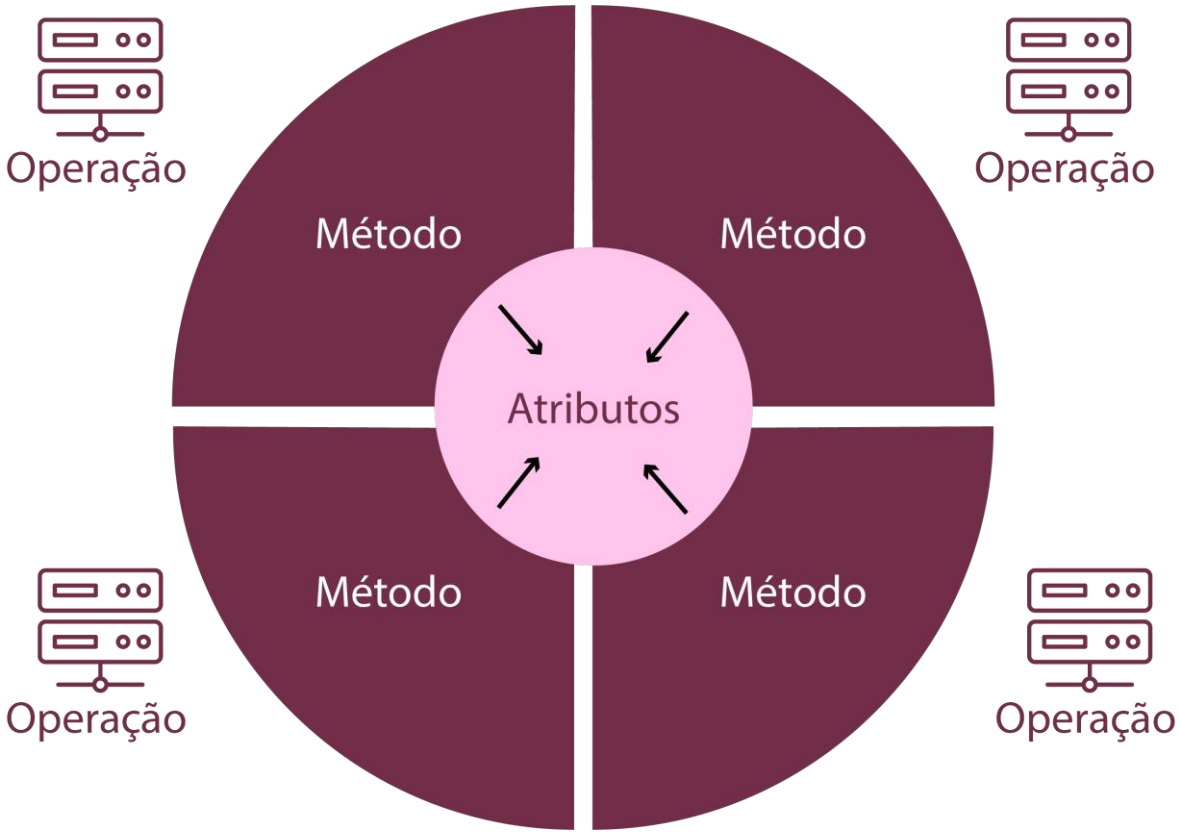
Dentre as principais características do paradigma orientado a objeto (O.O.), destacamos:

ABSTRAÇÃO

É um processo mental que permeia toda a orientação a objetos, como um princípio básico que serve de base aos demais princípios. A abstração permite que, ao estudar algo, ponhamos nossa concentração nos aspectos relevantes e deixemos de lado os menos importantes. Permite, portanto, gerenciar a complexidade de um objeto para que possamos nos ater às suas propriedades essenciais. E os aspectos essenciais de um objeto dependem, claro, do contexto no qual os analisamos, podendo variar. Ou seja, uma propriedade de um objeto pode ser relevante em um contexto e não ser em outro.

ENCAPSULAMENTO

O objeto esconde (encapsula) seus dados (atributos) do acesso indevido por outros objetos e somente permite que eles sejam acessados por operações implementadas pelos seus próprios métodos (funcionalidades que implementam o comportamento do objeto). Com isso, o encapsulamento protege os dados do objeto do uso arbitrário ou não intencional, como pode ser visualizado na figura seguinte. O encapsulamento é uma técnica para minimizar a interdependência entre as classes, pois apenas os métodos da respectiva classe podem alterar seus dados (atributos), facilitando a identificação de erros e a alteração dos programas. Em outras palavras, garante que apenas os métodos da própria classe possam alterar o estado de um objeto.



Encapsulamento

HERANÇA

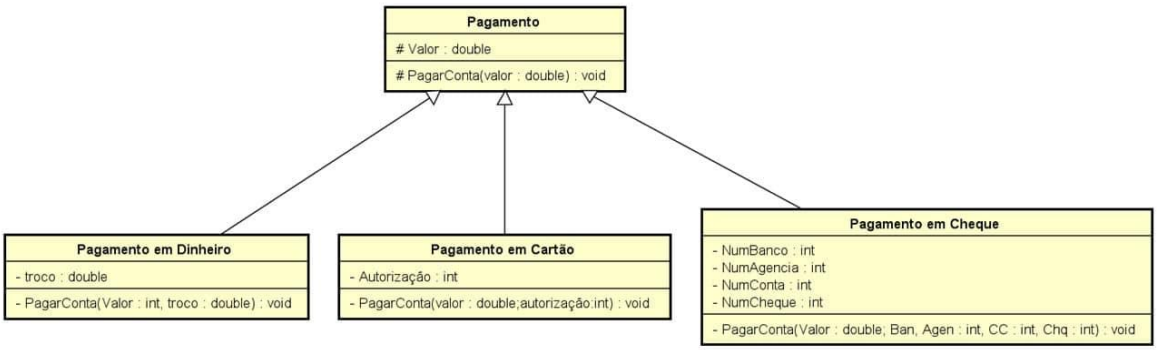
Mecanismo para derivar novas classes a partir da definição de classes existentes, com base em um processo de refinamento. Uma classe derivada ou descendente herda os dados (atributos) e o comportamento (métodos) da classe base ou ancestral ou ascendente. A implementação da herança garante a reutilização de código, que, além de economizar tempo e dinheiro, propicia mais segurança ao processo de desenvolvimento, posto que as funcionalidades da classe base podem ter sido usadas e testadas.

POLIMORFISMO

A palavra polimorfismo deriva do grego e significa “muitas formas”. A partir do momento em que uma classe herda atributos e métodos de uma (herança simples) ou mais (herança múltipla) classes base, ela tem o poder de alterar o comportamento de cada um desses procedimentos (métodos). Isso amplia o poder do reaproveitamento de código promovido pela herança, permitindo que se aproveite alguns métodos e se altere (redefina) outros. Dessa forma, um método com mesmo nome em classes distintas pode ter diferentes comportamentos.

Exemplificando herança e polimorfismo

Acompanhe o exemplo da imagem a seguir, na qual identificamos uma herança: Pagamento em Dinheiro, Pagamento em CC (Cartão de crédito) e Pagamento em Cheque herdam da classe Pagamento, o atributo Valor e o método Pagar.



Herança e polimorfismo

Observe que, em cada classe filha (Pagamento em Dinheiro, Pagamento em Cartão e Pagamento em Cheque), o método PagarConta é escrito de forma diferente, com distintos parâmetros e códigos internos, conforme exige a respectiva forma de pagamento. Essa possibilidade ocorre pelo princípio do polimorfismo.

ORIENTAÇÃO A OBJETOS COMO ELEMENTO DE REUSABILIDADE E EXTENSIBILIDADE

A orientação a objetos minimiza a gestão da complexidade na medida em que permite uma organização mais adequada de seus componentes, além de seu reaproveitamento. Um dos principais motivos para a baixa produtividade na construção de sistemas computacionais é a dificuldade de reutilização de código.

As hierarquias de classes (herança) são estruturas que permitem o seu reaproveitamento entre aplicações que, se bem projetadas, podem ser reutilizadas em vários sistemas, otimizando tempo e dinheiro.

Além disso, tais estruturas podem ser estendidas (usando o princípio do polimorfismo) sem corromper o que já existe. Dessa forma, pode-se concluir que a orientação a objetos traz em si os seguintes benefícios:

Reusabilidade

O uso de componentes já escritos pode ser a base para outros softwares (através da herança).

Extensibilidade

Novos componentes podem ser desenvolvidos a partir de outros, já desenvolvidos, sem afetar o comportamento do componente de origem (mediante o princípio do polimorfismo) e permitindo que esse comportamento seja alterado, estendido para um novo contexto.

ANÁLISE DE SISTEMAS ORIENTADA A OBJETOS

Antes de adentramos no universo da análise sob o enfoque do paradigma orientado a objetos, vamos tecer rápidos comentários acerca do que seja a atividade de análise no contexto do desenvolvimento de software.

De forma simples, pode-se dizer que a atividade de análise visa identificar **o que** os usuários e os demais interessados (que juntos formam os *stakeholders*) precisam que o sistema faça.

Análise de sistemas implica numa investigação dos problemas e dos requisitos (necessidades dos usuários) de um contexto, em particular, visando a construção de um sistema automatizado.

Atenção

O foco da atividade de análise é estudar e entender o funcionamento de um sistema sob pelo menos alguns pontos de vista: da estrutura que sustenta o sistema (os dados); dos procedimentos e processos intervenientes no sistema; da dinâmica de funcionamento do fluxo de informações e dados, dentre outros aspectos que podem ser adicionados, dependendo da especificidade do sistema em construção.

A atividade de análise, por ser muito abrangente, costuma ser dividida em: **levantamento de requisitos** (investigação dos requisitos) e **análise dos requisitos.**

Inicialmente, entendemos a realidade, identificamos a abrangência do sistema e capturamos as necessidades dos usuários desse sistema, usando técnicas específicas (levantamento de requisitos).

Posteriormente, analisamos e entendemos essas necessidades e o funcionamento e a dinâmica da organização (análise dos requisitos), visando à construção de modelos que representem o sistema a ser desenvolvido, em sua concepção lógica, sem preocupação com qualquer recurso tecnológico que venha sustentar o seu funcionamento.

A atividade de análise não leva em consideração nenhum recurso tecnológico que possa ser utilizado pelo sistema em construção. A ideia é construir uma estratégia de solução sem considerar como (com que tecnologia) essa estratégia será construída.

A preocupação da atividade de análise é identificar: O QUE o sistema deve fazer para atender às necessidades de seus usuários.

A sua finalidade é construir a melhor solução, que possa ser implementada em qualquer tecnologia (plataforma operacional, linguagem de programação e banco de dados), de acordo com as disponíveis no mercado, naquele momento.

No contexto da orientação a objetos, foca-se na identificação, no entendimento e na definição dos objetos de software e em como eles irão colaborar para que os requisitos dos usuários sejam respondidos satisfatoriamente.

Levantamento de requisitos

Nesta fase, o foco é conversar com as pessoas envolvidas com o sistema (patrocinadores, gestores, usuários atuais e futuros e demais envolvidos) e compreender as necessidades e desejos de cada um. O foco, portanto, é na compreensão do problema.

As necessidades e os desejos que os usuários têm são, tecnicamente, chamados de **requisitos**. Os desenvolvedores e as demais pessoas envolvidas se reúnem, visando

à identificação dos requisitos do sistema, considerando o contexto específico e o domínio do problema (área do conhecimento ou atividade específica) a que o sistema se aplica.
Para que os desenvolvedores possam identificar as necessidades dos usuários do sistema, é usado um conjunto de técnicas de levantamento de dados, desde as tradicionais entrevistas, reuniões, observação do ambiente do usuário e questionários até técnicas mais sofisticadas como *brainstorm*.

O produto gerado por essa fase é o **documento de requisitos**, que contém todos os requisitos necessários ao sistema, classificados em :

Requisitos funcionais

Declaram as funcionalidades necessárias ao sistema.
Requisitos não funcionais

Apresentam algumas características associadas a uma, algumas ou todas as funcionalidades, e dizem respeito a aspectos de qualidade, confiabilidade, desempenho, portabilidade, segurança e usabilidade do sistema.

Exemplo
Imagine um sistema financeiro, no qual haja a necessidade das seguintes funcionalidades: **Cadastramento dos pagamentos, Quitação dos pagamentos, Cadastramento das receitas, Quitação das receitas e Emissão das faturas.**

Cada uma das cinco funcionalidades anteriores representa um requisito funcional do sistema. Imagine, então, que o sistema precise ser *touch screen*. Tal característica da funcionalidade **Emissão das faturas** é um requisito não funcional de usabilidade (relacionada com uma interface de qualidade). Outro requisito não funcional de segurança seria a necessidade de um *backup* diário da base de dados.
A correta identificação e seu registro no **documento de requisitos** são cruciais para a precisão e a qualidade do processo de desenvolvimento do sistema. Um requisito faltante ou outro mal definido pode ser fatal para seu sucesso.

Dica
O **documento de requisitos** seguirá como base da comunicação entre os desenvolvedores e os usuários, devendo ser validado por estes, uma vez que servirá de norte para as atividades subsequentes do desenvolvimento do sistema. É, portanto, fundamental a participação ativa e efetiva dos usuários do sistema na fase de levantamento de requisitos.

No documento de requisitos estará definido, também, o escopo do sistema.
O principal ponto da fase de levantamento de requisitos é compreender profundamente no sistema antes de iniciar a sua construção.

Análise de requisitos

A fase de análise de requisitos é a transposição dos registros dos requisitos funcionais e não funcionais para os modelos que a equipe pretende usar.

A principal atividade é desdobrar os requisitos funcionais em funcionalidades do sistema, uma vez que não necessariamente há uma relação de 1 para 1, ou seja, um requisito funcional pode demandar mais de uma funcionalidade e uma funcionalidade pode agregar mais de um requisito funcional.

A análise de requisitos tem, minimamente, duas perspectivas ou visões:

ANÁLISE DO DOMÍNIO (OU DO NEGÓCIO)

Visa a identificar ou modelar os objetos que serão usados na aplicação. Por exemplo, **Pagamento** é um objeto no contexto do sistema financeiro, usado para exemplificar os requisitos funcionais. Logo, **Pagamento** é um objeto do domínio, assim como Recebimento e Fatura.

ANÁLISE DA APLICAÇÃO

Em geral, sucede a análise do domínio. Nela, são identificados os objetos de análise que não fazem sentido para os analistas de domínio, mas que são fundamentais para atender às funcionalidades necessárias ao sistema, a aplicação em si (daí seu nome). Por exemplo, uma **interface de cadastramento de pagamentos** é um objeto de aplicação do sistema de controle financeiro. Nesse momento, o foco é apenas a identificação desse objeto, sem especificar sua implementação, o que será detalhado na fase de Projeto (descrita adiante).

Análise do domínio

Objetos do domínio (relacionado ao problema)

!=

Análise da aplicação

Objetos da aplicação (relacionado a aspectos computacionais de alto nível)

Os principais diagramas UML usados nas fases de análise são: diagramas de casos de uso e diagrama de classes. Além desses, ajudam também diagramas de interação e de estados, em alguns casos.

Análise pode ser traduzida em ”faça a coisa certa”.

PROJETO (DESENHO) DE SISTEMAS ORIENTADO A OBJETOS

A atividade de projeto denota uma solução, voltada a atender aos requisitos identificados na fase de análise, considerando os recursos tecnológicos necessários para implementar os objetos do domínio e os objetos da aplicação.

O objetivo é decidir “**como o sistema funcionará**” para atender aos requisitos. Em geral, o projeto enfoca a definição da arquitetura do sistema, determinando suas partes e a relação entre elas, o padrão de interface gráfica, o ambiente computacional (sistema operacional e computacional), a linguagem de programação o gerenciamento do banco de dados.

O projeto, portanto, deriva da análise e produz uma descrição dos recursos computacionais e de como o sistema deve executar no dia a dia. Muitas vezes algumas restrições tecnológicas podem ser impostas como, por exemplo, desenvolver na linguagem X, pelo fato de os sistemas da organização estarem nessa linguagem (claro, se ainda existir no mercado), ou usar o sistema gerenciador de banco de dados Y, pelo fato de a base corporativa da organização estar nele.

A fase de projeto pode ser dividida em:

PROJETO DA ARQUITETURA

Ato de distribuir as classes de análise em subsistemas com seus componentes, bem como distribuir os componentes pelos recursos de hardware disponíveis. Dentre os diagramas da UML, usamos aqui os diagramas de pacotes, componentes e implantação.

PROJETO DETALHADO

Compreende atividades como modelagem das colaborações entre os objetos, projeto da interface, projeto do banco de dados e aspectos computacionais de alto nível (concorrência e distribuição de processamento e dados). O projeto de algoritmos pode ser considerado aqui também, detalhando aspectos do que for relevante para o projeto. Dentre os diagramas UML usados aqui, destacam-se: diagrama de interação (sequência ou comunicação), atividades (se demandar detalhamento de um método ou método com processamento paralelo), detalhamento do diagrama de classes, de estados, dentre outros detalhados adiante.

Observação: Projeto pode ser traduzido em “faça certo a coisa”.

DESENVOLVIMENTO DE SISTEMAS EM CAMADAS

Abordaremos, inicialmente, a arquitetura de projeto de software em camadas de uma forma geral, sem nos atermos a um modelo em especial. O foco é separar o desenvolvimento do código em camadas, diminuindo sua complexidade e facilitando a sua manutenção. Camadas separam as responsabilidades e gerenciam as dependências.

No início da computação, os sistemas eram **monolíticos**, ou seja, todo o código ficava confinado numa única camada, onde misturavam-se comandos de processamento, de construção e manipulação de interface, bem como de acesso e persistência de dados em SGBD. “**Tudo junto e misturado**”.

Quando era preciso fazer manutenção no código, havia dificuldade no entendimento e na separação do que de fato precisava ser alterado, sem contar a interferência em uma parte do código quando se alterava outra, em princípio sem relação entre si.

À medida que os sistemas cresceram e se tornaram complexos, a manutenção ficou mais difícil e a divisão em camadas foi uma das soluções encontradas para o projeto de arquitetura de um software.

Num primeiro momento, a rede cliente-servidor, naturalmente, dividiu o software em duas camadas: a camada de código que roda no cliente (camada de interface com usuário) e a camada servidor (camadas de lógica do negócio e persistência dos dados).
Posteriormente, com o advento da web, separou-se em três e depois em quatro camadas. Atualmente, pode-se criar tantas camadas quantas sejam necessárias, em função do tipo de aplicação.

As camadas em geral:

• Possuem alta coesão e baixo acoplamento, ou seja, concentram atividades afins (coesão) e são independentes umas das outras.
• Possuem propósito bem definido.
• A camada superior tem conhecimento apenas da imediatamente inferior, que fornece os serviços, por uma interface.

No caso da orientação a objetos, as classes são organizadas em módulos maiores, as chamadas camadas. Uma camada somente pode usar serviço (de outras classes) da camada imediatamente inferior.

A seguir, as vantagens e desvantagens do desenvolvimento de software em camadas:

Clique nas barras para ver as informações.

VANTAGENS

- Torna o código mais organizado e legível.
- Permite o desenvolvimento, o teste e a manutenção das camadas isoladamente.
- Permite melhor reuso do código ou dos objetos.
- Pode substituir uma tecnologia que implemente uma camada, de forma simples, sem interferir nas demais. Por exemplo, para trocar o SGBD de SQL Server para PostgreSQL, basta alterar a camada de persistência. As demais permanecem como estavam.
- Disciplina as dependências entre as camadas.
- Mais adaptável a uma quantidade maior de usuários.

DESVANTAGENS

- Aumenta o número de classes do sistema.
- A adição de camadas torna o sistema mais complexo.
- Potencialmente, reduz o desempenho do software.

Como exemplo, podemos citar o modelo de camadas mais usado nos últimos anos, o de três camadas, que engloba as camadas de:

Apresentação

Compreende as classes do sistema que permitem a interação com o usuário, as chamadas classes de fronteira.

Negócio

Compreende as classes responsáveis pelos serviços e pelas regras do negócio, ou seja, reúne as classes de controle e negócio.

Dados

Responsável pelo armazenamento e pela recuperação dos dados persistentes do sistema, ou seja, as classes de persistência de dados.