UNIVERSIDADE DO SUL DE SANTA CATARINA

CIÊNCIA DA COMPUTAÇÃO

ENGENHARIA DE SOFTWARE III

Prof. Clávison M. Zapelini – clavison.zapelini@unisul.br

Processo de desenvolvimento de software

Processo de desenvolvimento de software

Um **processo de desenvolvimento de software** é um conjunto de atividades, parcialmente ordenadas, com a finalidade de obter um produto de software.

É estudado dentro da área de Engenharia de software, sendo considerado um dos principais mecanismos para se obter um software de qualidade e cumprir corretamente os contratos de desenvolvimento.

Processo de desenvolvimento de software

Os principais processos no desenvolvimento de software são:

Análise de requisitos de software: A extração dos requisitos de um desejado produto de software é a primeira tarefa na sua criação. Embora o cliente, provavelmente, acredite saber o que o software deva fazer, esta tarefa requer habilidade e experiência em engenharia de software para reconhecer a incompletude, ambigüidade ou contradição nos requisitos de um ser abstrato.

Processo de desenvolvimento de software

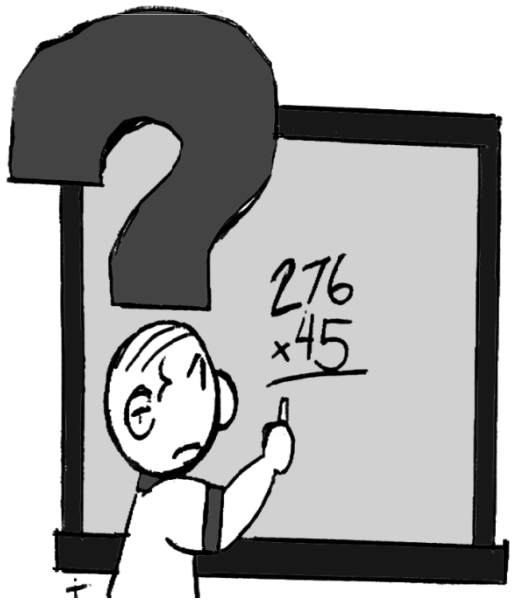
Projeto: se encarrega de transformar os resultados da Análise de requisitos em um documento ou conjunto de documentos capazes de serem interpretados diretamente pelo programador e cliente.

Processo de desenvolvimento de software

Implementação: é a fase do Ciclo de Vida de um software (programa computacional, documentação e dados), no contexto de um Sistema de Informação, que corresponde à elaboração e preparação dos módulos necessários à sua execução.

Processo de desenvolvimento de software

Testes: é a investigação do sofware a fim de fornecer informações sobre sua qualidade em relação ao contexto em que ele deve operar. Isso inclui o processo de utilizar o produto para encontrar seus defeitos



Análise de Requisitos

Análise de Requisitos

Análise de requisitos é um processo de descoberta e refinamento.

ATORES: Cliente e Desenvolvedor

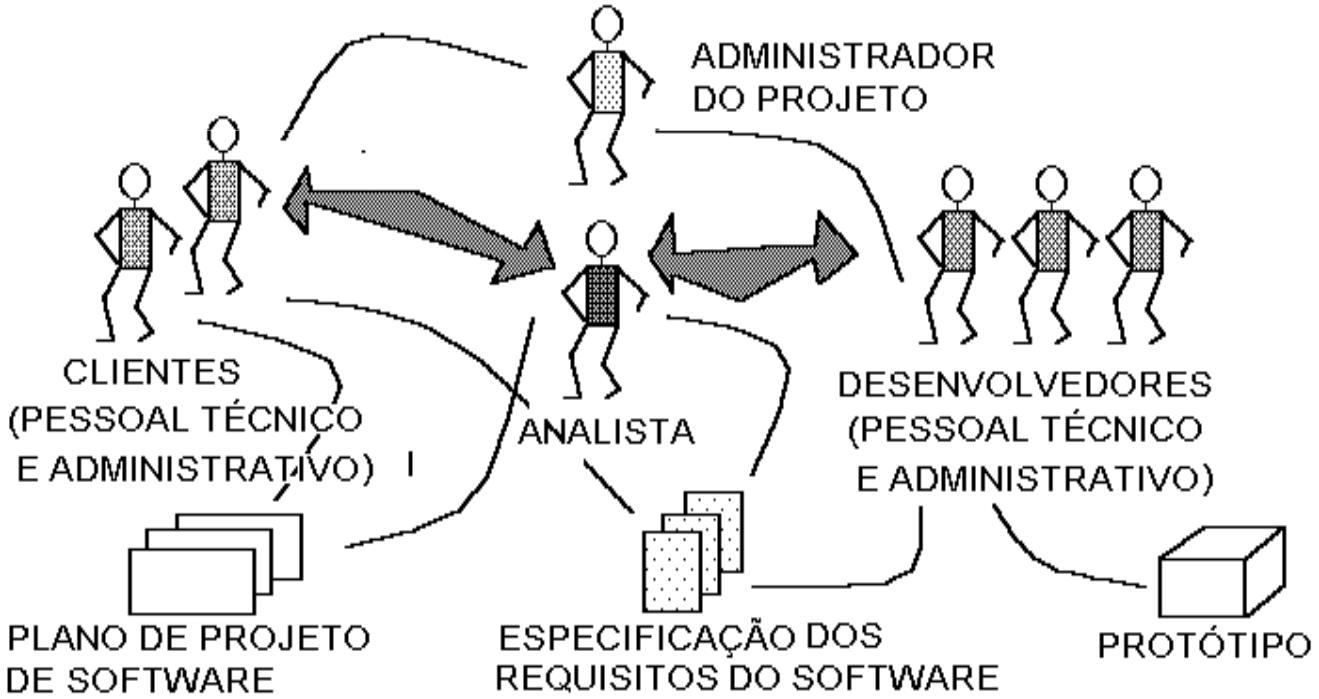
PROBLEMA: Grande propensão a mal entendidos
"*atividade aparentemente simples torna-se complexa*"

ATIVIDADES DE ANÁLISE:
1- Reconhecimento do Problema
2. Avaliação do Problema e Síntese da Solução (Modelagem)
3. Especificação dos Requisitos do Software
4. Revisão

Análise de Requisitos

Reconhecimento do problema

A meta é o reconhecimento dos elementos básicos do problema, conforme percebidos pelo cliente.



Análise de Requisitos

Avaliação do problema e síntese da solução

Nesta tarefa o analista deve avaliar os problemas com a situação atual

Para o novo sistema:

- definir e elaborar todas as funções do sistema
- identificar dados que o sistema produz e consome
- entender o comportamento do sistema
- estabelecer características de interface
- descobrir restrições do projeto

Sintetizar uma ou mais soluções (dentro do alcance delineado no *Plano de Projeto do Software*)

Análise de Requisitos

Avaliação do problema e síntese da solução

O processo de avaliação e síntese continua até que o analista e o cliente concordem que o software pode ser adequadamente especificado.

É a maior área de esforço

Análise de Requisitos

Avaliação do problema e síntese da solução

Durante a atividade de avaliação e síntese devem ser criados modelos do sistema para se compreender melhor o fluxo de dados e de controle, o processamento funcional e a operação comportamental, além do conteúdo da informação.

O modelo serve como um fundamento para o projeto de software e como base para a criação de sua especificação

Análise de Requisitos

Especificação dos requisitos

Envolve:

- A descrição do fluxo e estrutura da informação
- O refinamento detalhado de todas as funções do software
- O estabelecimento das características de interface
- A identificação das restrições de projeto
- A especificação dos critérios de validação

Análise de Requisitos

Problemas no processo

1- Aquisição da Informação

Que informação deve ser coletada e como ela deve ser representada?

Quem fornece as informações?

Que técnicas e ferramentas estão disponíveis para facilitar a coleta de informações?

Análise de Requisitos

Problemas no processo

2- Tamanho do Sistema

Como eliminar inconsistências na especificação de grandes sistemas?

É possível detectar omissões?

Pode um grande sistema ser efetivamente particionado para que se torne intelectualmente administrável?

Análise de Requisitos

Problemas no processo

3- Alterações

Como as alterações efetuadas em outros elementos do software são coordenadas com os requisitos do software?

Como se determina o impacto de uma alteração em outras partes do software aparentemente não relacionadas?

Como se corrige erros na especificação para que não se gere efeitos colaterais?

Análise de Requisitos

Causas dos problemas

1- Comunicação Ineficiente

2- Técnicas e Ferramentas Inadequadas

3- Tendências de eliminar a tarefa de Especificação dos Requisitos

4- Falha de considerar alternativas antes que o software seja especificado.

Análise de Requisitos

FAST – Técnica facilitada para especificação de aplicações

1 - Preparação

ENCONTROS INICIAIS: Perguntas e respostas básicas para ajudar a estabelecer o escopo dos problemas e a percepção global de uma solução.

REQUISIÇÃO DE PRODUTO: Documento elaborado pelo cliente e desenvolvedor no final dos encontros iniciais.

Análise de Requisitos

Entrevista preliminar

PERGUNTAS PARA UM PRIMEIRO ENCONTRO	
(Sobre o Cliente)	Quem está por trás do pedido deste trabalho? Quem usará a solução? Qual o benefício econômico de uma solução bem sucedida? Há outra fonte para a solução exigida?
(Sobre o Problema)	Como você caracteriza um "bom" resultado que seria gerado por uma solução bem sucedida? Qual problema (s) essa solução resolverá? Você poderia mostrar-me o ambiente em que a solução será usada? Existem questões de desempenho ou restrições especiais que afetarão a maneira pela qual a solução é abordada.
(Efetividade do Encontro)	Você é a pessoa certa para responder a essas perguntas? Suas respostas são "oficiais"? Minhas perguntas são pertinentes ao problema que você tem? Estou fazendo perguntas demais? Há mais alguém que possa fornecer informações adicionais? Existe algo mais que eu deva perguntar-lhe?

Análise de Requisitos

2 – Tarefas dos participantes (antes do encontro)

Elaborar LISTA DE OBJETOS que fazem parte do ambiente que circunda o sistema, que são produzidos pelo sistema e que são usados pelo sistema para executar suas funções.

Elaborar LISTA DE OPERAÇÕES que manipulam ou interagem com o objeto

Elaborar LISTA DE RESTRIÇÕES - custo, tamanho

Elaborar CRITÉRIOS DE DESEMPENHO - velocidade, precisão

Análise de Requisitos

3 – Encontro FAST

1o Tópico de Discussão: NECESSIDADES e JUSTIFICATIVA do novo sistema.

APRESENTAÇÃO DAS LISTAS para posterior crítica e discussão.

Criação de uma LISTA COMBINADA de cada área de assunto (objetos, operações, restrições e desempenho)

DISCUSSÃO (coordenada pelo moderador) das listas combinadas

Análise de Requisitos

3 – Encontro FAST

Elaboração de uma LISTA CONSENSUAL de cada área de assunto

Divisão da equipe em SUB-EQUIPES menores

Criação de uma MINI-ESPECIFICAÇÃO pelas sub-equipes. A Mini-Especificação é uma elaboração das palavras ou frases contidas numa lista.

Apresentação das mini-especificações a todos os participantes do encontro

Análise de Requisitos

3 – Encontro FAST

Criação, pelas sub-equipes, de uma LISTA DE CRITÉRIOS DE VALIDAÇÃO

Criação de uma LISTA de CONSENSO dos CRITÉRIOS DE VALIDAÇÃO

Escrita do ESBOÇO DE ESPECIFICAÇÃO COMPLETO, usando todas as entradas do encontro FAST

Análise de Requisitos

Formato da especificação de requisitos

I. INTRODUÇÃO
Declara as metas e os objetivos do software, descrevendo-os no contexto do sistema baseado em computador

II. DESCRIÇÃO DA INFORMAÇÃO
Apresenta uma descrição detalhada do problema que o software deve resolver

III. DESCRIÇÃO FUNCIONAL
Engloba uma descrição de cada função exigida para resolver o problema

Análise de Requisitos

Formato da especificação de requisitos

IV. DESCRIÇÃO COMPORTAMENTAL
Examina a operação do software como uma seqüência de eventos

V. CRITÉRIOS DE VALIDAÇÃO
Designam classes de testes que devem ser efetuadas para validar a função, o desempenho e as restrições. Seção muito importante, porém negligenciada.

VI. BIBLIOGRAFIA
Contém referências a todos os documentos que se relacionam com o software.

VII. APÊNDICE
Traz informações que complementam a especificação.

Análise de Requisitos

Conclusão

Logo que a Revisão for concluída, a Especificação de Requisitos de Software é "assinada" pelo cliente e pelo desenvolvedor.

A especificação torna-se um "contrato" de desenvolvimento de software.

Mudanças solicitadas depois que a especificação for concluída serão consideradas, porém cada mudança posterior pode aumentar o custo e/ou alongar o prazo de entrega.

Mesmo com os melhores procedimentos de revisão em andamento, uma série de problemas de especificação ainda persiste.

Análise de Requisitos

Projeto:

Software para gerenciar um consultório odontológico

Software para gerenciar um salão de beleza

Software para gerenciar uma lan-house

Software para gerenciar uma escola de cursos de idiomas

Software para gerenciar uma locadora de veículos

Projeto de Software

PROJETO DE SOFTWARE

É o processo pelo qual os requisitos do software são traduzidos numa representação do software que permite sua realização física.

O projeto é o lugar onde a qualidade é fomentada durante o processo de desenvolvimento

O projeto fornece representações do software que podem ser avaliadas quanto à qualidade

É a única maneira pela qual pode-se traduzir com precisão os requisitos de um cliente num produto de software acabado

Projeto de Software

Existem muitos métodos de projeto de software, contudo eles têm uma série de características em comum:

1- Um mecanismo para a tradução da representação do domínio de informação numa representação de projeto

2- Uma notação para representar os componentes funcionais e suas interfaces

3- Heurísticas para refinamento e divisão em partições

4- Diretrizes para a avaliação da qualidade

Ferramentas Case

FERRAMENTAS CASE

CASE (*Computer-Aided Software Engineering*)
é uma classificação que abrange todas ferramentas baseada em computadores que auxiliam atividades de engenharia de software, desde análise de requisitos e modelagem até programação e testes.

Podem ser consideradas como ferramentas automatizadas que tem como objetivo auxiliar o desenvolvedor de sistemas em uma ou várias etapas do ciclo de desenvolvimento de software

FERRAMENTAS CASE

Categorização:

a) ***Front End ou Upper CASE:*** apóiam as etapas iniciais de criação dos sistemas (planejamento, análise e projeto do software)

b) ***Back End ou Lower CASE:*** dão apoio à parte física, isto é, a codificação testes e manutenção da aplicação.

c) ***I-CASE ou Integrated CASE:*** classifica os produtos que cobrem todo o ciclo de vida do software, desde os requisitos do sistema até o controle final da qualidade.

JUDE

JUDE

O JUDE é uma ferramenta de modelagem gratuita (open source) na sua versão **astah community**, que suporta desenho de sistemas orientados a objeto.

É baseada nos diagramas e na notação da UML 2.0 (Unified Modeling Language)

Gera código em Java.

JUDE

Fazer o cadastro e download da versão mais recente em:

http://jude.change-vision.com/jude-web/download/index.html

UML

UML

A UML é uma linguagem de modelagem com as seguintes características:

Absorveu influências de outras técnicas de modelagem :
- Diagrama de Entidade e Relacionamento - DER
- Modelagem de Negócio - WorkFlow
- Modelagem de Objetos e Componentes

Incorporou idéias de diversos autores :
- Peter Coad, Derek Coleman, Ed Yourdon,...

Criada a partir de outras ferramentas de modelagem
- Booch´93
- OMT-2
- OOSE

UML

Linguagem-padrão para a elaboração da estrutura de projetos de software.

Enfoque Orientado a Objetos

Utilizada para Visualizar, Especificar, Construir e Documentar artefatos que modelem sistemas de software.

UML é apenas uma linguagem de modelagem, não uma metodologia para desenvolvimento de sistemas.

UML

Baseada em Diagramas (Ênfase Visual), onde vários aspectos fundamentais na modelagem de Sistemas são abordados, tais como :

Funcional (estrutura estática e interação dinâmica),
Não funcional (tempo de processamento, confiabilidade, produção)
Organizacional (organização do trabalho, mapeamento e código).

Cada visão é descrita em um número de diagramas que contém informações enfatizando um aspecto em particular. Analisando o Sistema através de visões diferentes, é possível se concentrar em um aspecto de cada vez.

DIAGRAMAS UML

Casos de uso (Use Cases) : Modelam o comportamento geral do Sistema, através dos relacionamentos com atores externos.

Classes : Modela classes, interfaces e seus relacionamento, representando uma visão estática da estrutura do Sistema.

Interação : modelam uma especificação comportamental representando a troca de mensagens entre objetos, em UML os Diagramas de Interação são representados por:

Sequência : modela a interação entre objetos através de seus relacionamentos e troca de mensagens. Demonstram a dinâmica do Sistema com ênfase na ordenação temporal das mensagens.

Colaboração : Semelhante ao diagrama de Sequência, sendo que sem observar a ordenação temporal das mensagens.

DIAGRAMAS UML

Estados : Modela uma máquina de estados, formado por : estados, transições e eventos

Atividades : Tipo especial de Diagrama de Gráfico de Estado

Diagramas de Implementação : Modelam a arquitetura lógica e física do hardware e software que implementam o Sistema.

Componentes : Exibe a organização e dependências existentes em um conjunto de componentes do Sistemas (Programas fontes, objetos, executáveis e bibliotecas)

Implantação : Mostra a configuração dos nós de processamento em tempo de execução e os componentes neles existentes.

Casos de Uso (use Case)

Um caso de uso é um documento narrativo que descreve a seqüência de eventos de um ator (agente externo) que usa um Sistema para completar um processo[Jacobson92].

Eles são histórias ou casos de utilização de um Sistema.

Casos de uso não são exatamente especificações de requisitos ou especificações funcional, mas ilustram e implicam requisitos na história que eles contam.

Casos de Uso (use Case)

Caso de Uso

Caso de Uso	Comprar Itens
Atores	Cliente, Caixa
Descrição	Um Cliente chega a um ponto de pagamento, com vários itens que deseja comprar. O Caixa registra os itens de compra, informa o total da compra ao cliente e recebe o pagamento.

Os detalhes dos casos de uso não são formalizados pela UML e podem ser adequados para satisfazer as necessidades e o espírito de documentação necessários (clareza de comunicação).

Os casos de uso podem ser detalhados através de uma Sequência Típica de Eventos

Casos de Uso (use Case)

Sequência Típica de Eventos	
Ação do Ator	**Resposta do Sistema**
1. Este caso de uso começa quando um Cliente chega a um ponto de pagamento (Equipado com um Post) com vários itens que deseja comprar.	
2. O Caixa registra o código de cada item Se houve mais de um exemplar do mesmo item, o Caixa também pode entrar a quantidade	3. Determina o preço do item e acrescenta a informação sobre o item à transação corrente de venda.
4. Ao término da entrada de itens, o Caixa indica ao POST que a entrada de itens está completa	5. Calcula e apresenta o total da venda
6. O Caixa informa ao Cliente o total da compra	
7. O Cliente dá um pagamento em dinheiro, possivelmente maior que o total da venda	
8. O Caixa registra o montante de dinheiro recebido	9. Exibe o valor do troco a ser devolvido ao Cliente
10. O Caixa deposita o dinheiro recebido e retira o troco	11. Registra a venda completada
12. O Cliente sai com os itens comprados	

Diagramas de caso de Uso

O Diagrama de *Casos de Uso* tem o objetivo de auxiliar a comunicação entre os analistas e o cliente graficamente.

Um diagrama de Caso de Uso descreve um cenário que mostra as funcionalidades do sistema do ponto de vista do usuário.

O cliente deve ver no diagrama de Casos de Uso as principais funcionalidades de seu sistema.

Diagramas de caso de Uso

O diagrama de Caso de Uso é representado por:

- atores;
- casos de uso;
- relacionamentos entre estes elementos.

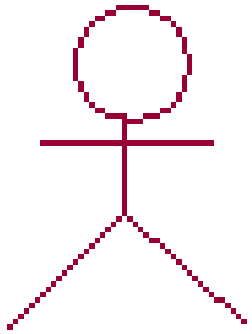
Estes relacionamentos podem ser:

- associações entre atores e casos de uso;
- generalizações entre os atores;
- generalizações, *extends* e *includes* entre os casos de uso.

Diagramas de caso de Uso

Atores

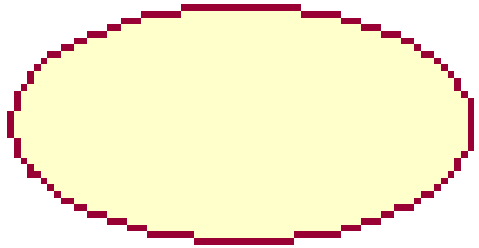
Um ator é representado por um boneco e um rótulo com o nome do ator. Um ator é um usuário do sistema, que pode ser um usuário humano ou um outro sistema computacional.



Diagramas de caso de Uso

Caso de Uso

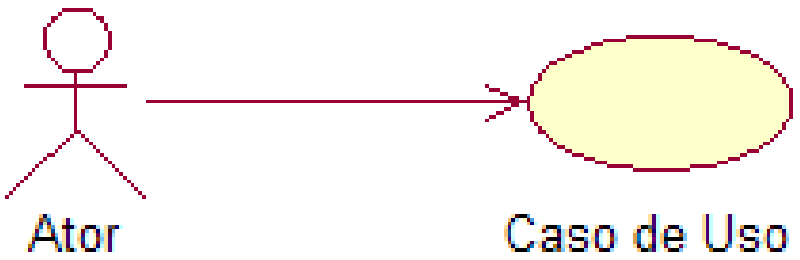
Um *caso de uso* é representado por uma elipse e um rótulo com o nome do *caso de uso*. Um *caso de uso* define uma grande função do sistema. A implicação é que uma função pode ser estruturada em outras funções e, portanto, um *caso de uso* pode ser estruturado.



Diagramas de caso de Uso

Relacionamentos: Ajudam a descrever casos de uso

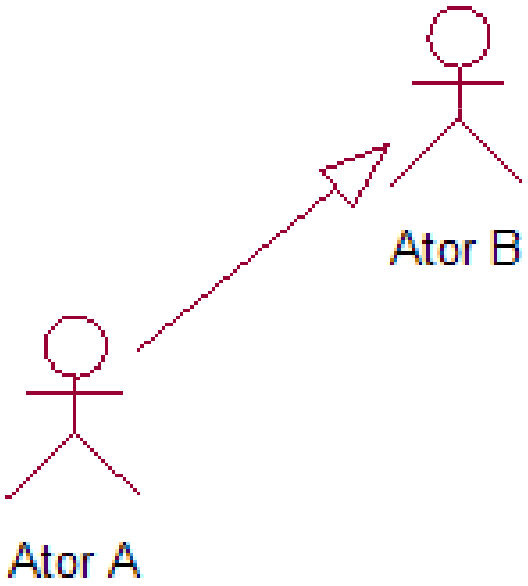
Entre um ator e um caso de uso:



Define uma funcionalidade do sistema do ponto de vista do usuário.

Diagramas de caso de Uso

Entre atores:



Os *casos de uso* de B são também *casos de uso* de A
- A tem seus próprios *casos de uso*

Diagramas de caso de Uso

Entre casos de uso:

Include

Um relacionamento *include* de um *caso de uso* A para um *caso de uso* B indica que B é essencial para o comportamento de A. Pode ser dito também que B *is_part_of* A.

Diagramas de caso de Uso

Extend

Um relacionamento *extend* de um *caso de uso* B para um *caso de uso* A indica que o *caso de uso* B pode ser acrescentado para descrever o comportamento de A (não é essencial).

Quando se especifica B *extends* A, B é inserido em A.

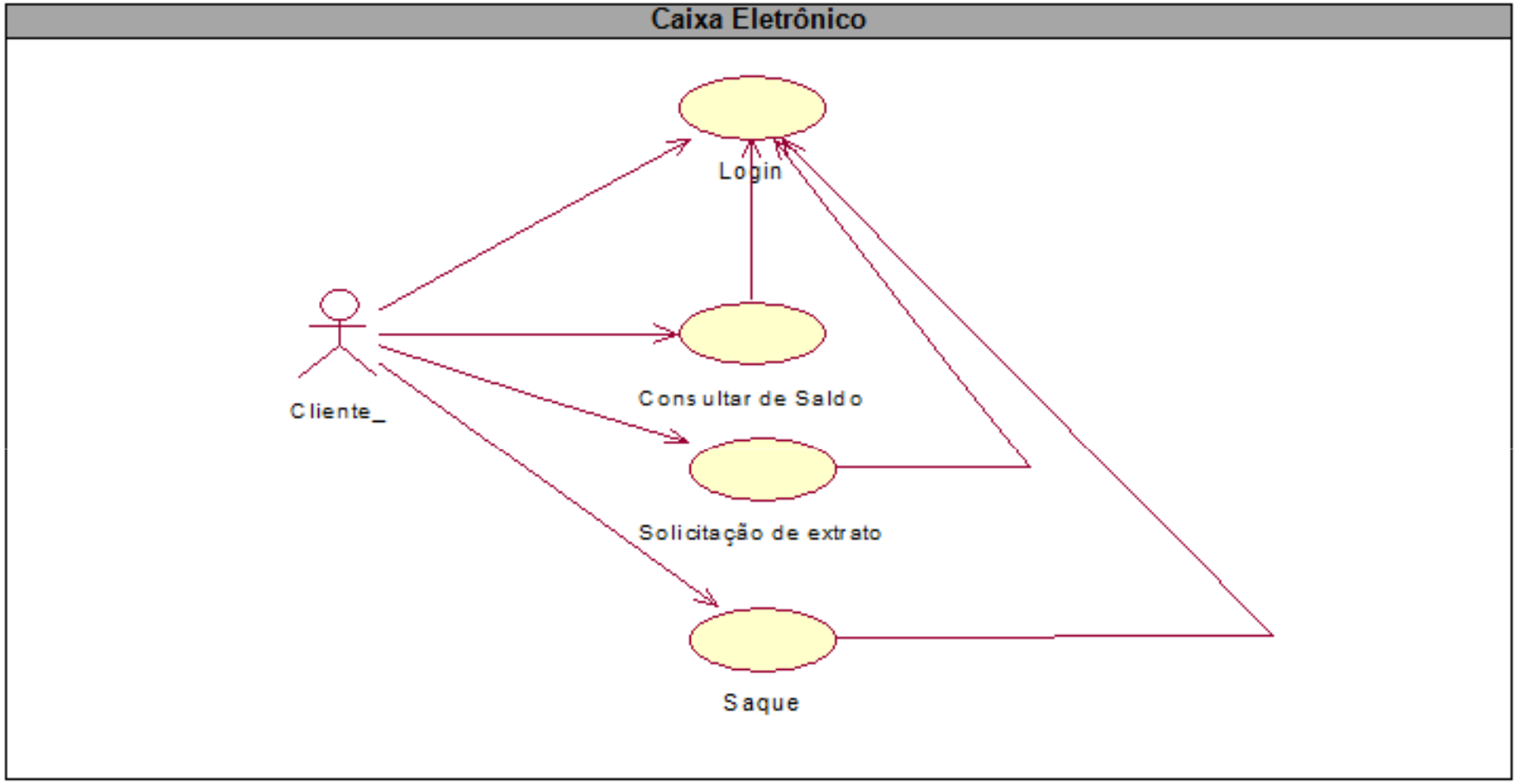
Diagramas de caso de Uso

Generalização ou Especialização (é_um)

caso de uso B é_um *caso de uso* A (A é uma generalização de B, ou B é uma especialização de A).

Um relacionamento entre um caso de uso genérico para um mais específico, que herda todas as características de seu pai.

Diagramas de caso de Uso



Projeto proposto

Implementar os diagramas de caso de uso

Diagrama de Classes

Em POO , os problemas de programação são pensados em termos de objetos, propriedades e métodos.

Um objeto é um termo que usamos para representar uma entidade do mundo real (Fazemos isto através de um exercício de abstração.)

Atributos físicos de um Cachorro	Ações de um Cachorro
Cor do corpo Cor dos olhos Altura Comprimento Largura	Balançar o rabo Latir Deitar Sentar

Diagrama de Classes

O conjunto de atributos e métodos caracterizam um objeto.

Em termos de POO para poder tratar os objetos se inicia criando classes: classe chamada **Cachorro**.

Uma classe representa um conjunto de objetos que possuem comportamentos e características comuns.

"Na UML o nome de uma classe é um texto contendo letras , dígitos e algumas marcas de pontuação. Na realidade, é melhor guardar os nomes curtos com apenas letras e dígitos. UML sugere capitalizar todas as primeiras letras de cada palavra no nome (ex.: ``Lugar", ``DataReserva"). É melhor também manter nomes de classe no singular". [Nicolas Anquetil]

Diagrama de Classes

Uma classe descreve como certos tipos de objetos se parecem do ponto de vista da programação , pois quando definimos uma classe precisamos definir duas coisas:

Propriedades - Informações específicas relacionadas a uma classe de objeto. São as características dos objetos que as classes representam. *Ex Cor , altura , tamanho , largura , etc...*

Métodos: São ações que os objetos de uma classe podem realizar. *Ex: Latir , correr , sentar , comer, etc.*

Diagrama de Classes

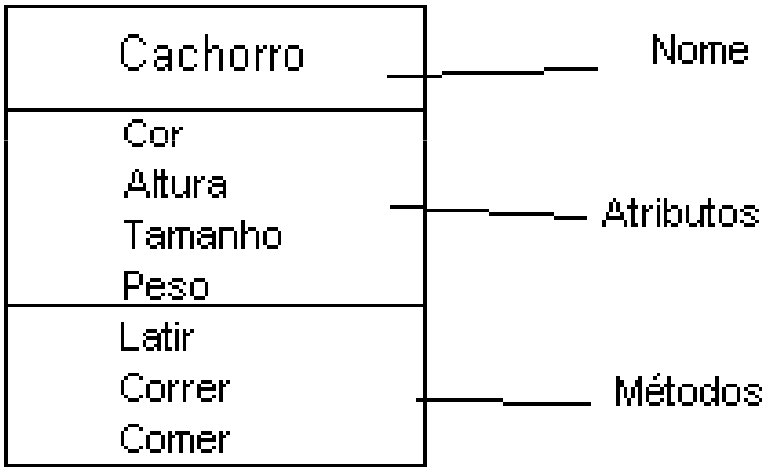
Geralmente em um sistema de médio porte serão identificados diversas classes que compõem o sistema.

UML é uma proposta de ser uma linguagem para modelagem de dados com artefatos para representar o modelo de negócio.

Um destes artefatos é o **diagrama de classes**.

Diagrama de Classes

A representação de uma classe usa um retângulo dividido em três partes:



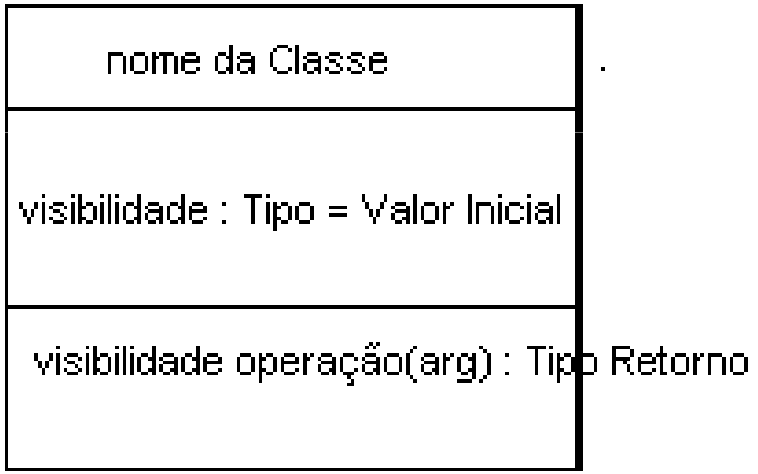


Diagrama de Classes

Os diagramas de classes são os principais diagramas estruturais da UML pois ilustram as classes , interfaces e relacionamentos entre elas.

Os objetos tem relações entre eles: um professor *ministra* uma disciplina *para* alunos *numa* sala, um cliente *faz* uma reserva *de* alguns lugares *para* uma data, etc. Essas relações são representadas também no diagrama de classe. [Nicolas Anquetil]

A UML reconhece três tipos mais importantes de relações: dependência, associação e generalização (ou herança).

Diagrama de Classes

Associação : São relacionamentos estruturais entre instâncias e especificam que objetos de uma classe estão ligados a objetos de outras classes.

A associação pode existir entre classes ou entre objetos. Uma associação entre a classe Professor e a classe disciplina (um professor ministra uma disciplina) significa que uma instância de Professor (um professor específico) vai ter uma associação com uma instância de Disciplina. Esta relação significa que as instâncias das classes são conectadas, seja fisicamente ou conceitualmente.[Nicolas Anquetil]

Diagrama de Classes

Dependência - São relacionamentos de utilização no qual uma mudança na especificação de um elemento pode alterar a especificação do elemento dependente. A dependência entre classes indica que os objetos de uma classe usam serviços dos objetos de outra classe.

Generalização - Relacionamento entre um elemento mais geral e um mais específico. Onde o elemento mais específico herda as propriedades e métodos do elemento mais geral (Herança).

Diagrama de Classes

Agregação - tipo de associação (é parte de , todo/parte) onde o objeto parte é um atributo do todo ; onde os objetos partes somente são criados se o todo ao qual estão agregados seja criado. Pedidos é composto por itens de pedidos.

Composição - Relacionamento entre um elemento (o todo) e outros elementos (as partes) onde as parte só podem pertencer ao todo e são criadas e destruídas com ele.

Diagrama de Classes

Representação dos relacionamentos:

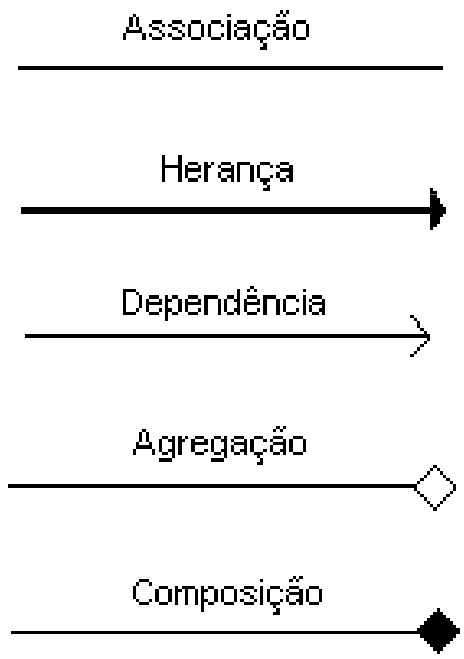


Diagrama de Classes

Atributos:

Na UML, o nome de um atributo é um texto contendo letras e dígitos e algumas marcas de pontuação. UML sugere de capitalizar todas as primeiras letras de cada palavra no nome menos a primeira palavra (ex.: "nome", "nomeCachorro").

Num modelo, os atributos devem ser de um tipo simples (inteiro, texto, talvez data), não podem conter outros objetos.

Os objetos são representados pelo relacionamernto.

Diagrama de Classes

Métodos:

Uma classe pode ter qualquer número de métodos e dois métodos em duas classes podem ter o mesmo nome.

Todos os métodos que vão implementar a operação tem que respeitar exatamente a assinatura dela (mesmo nome, mesmo número de atributo , com os mesmo tipos e o mesma ordem). *[Nicolas Anquetil]*

Projeto proposto

Implementar os diagramas de classes

Diagrama de Sequência

Consiste em um diagrama que tem o objetivo de mostrar como as mensagens entre os objetos são trocadas no decorrer do *tempo* para a realização de uma operação.

Em um diagrama de seqüência, os seguintes elementos podem ser encontrados:

Diagrama de Sequência

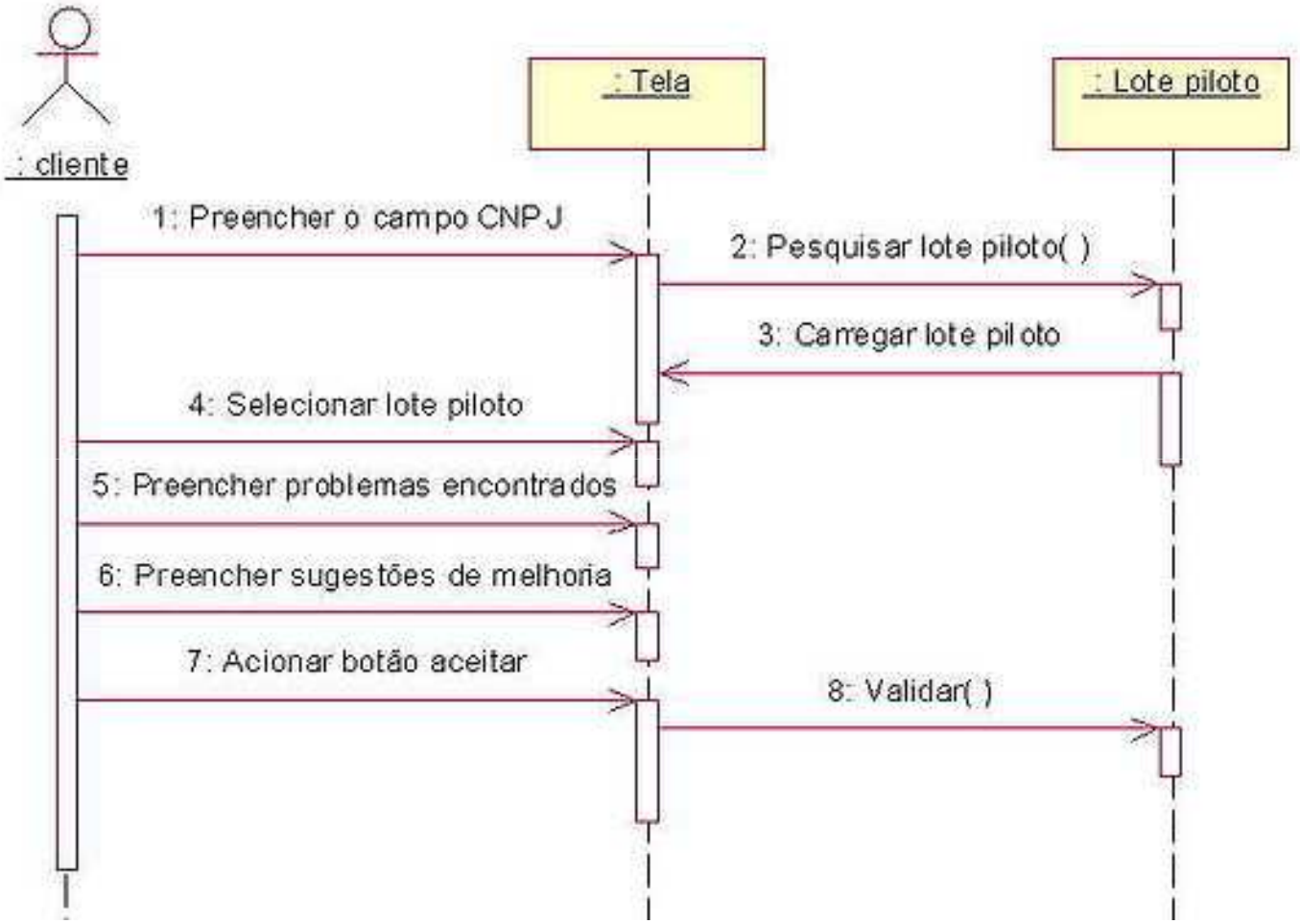
Linhas verticais representando o tempo de vida de um objeto (*lifeline*);

Estas linhas verticais são preenchidas por barras verticais que indicam exatamente quando um objeto passou a existir. Quando um objeto desaparece, existe um "X" na parte inferior da barra;

Linhas horizontais ou diagonais representando mensagens trocadas entre objetos. Estas linhas são acompanhadas de um rótulo que contém o nome da mensagem e, opcionalmente, os parâmetros da mesma.

Também podem existir mensagens enviadas para o mesmo objeto, representando uma iteração;

Diagrama de Sequência



Projeto Proposto

Criar os diagramas de seqüência para cada caso de uso

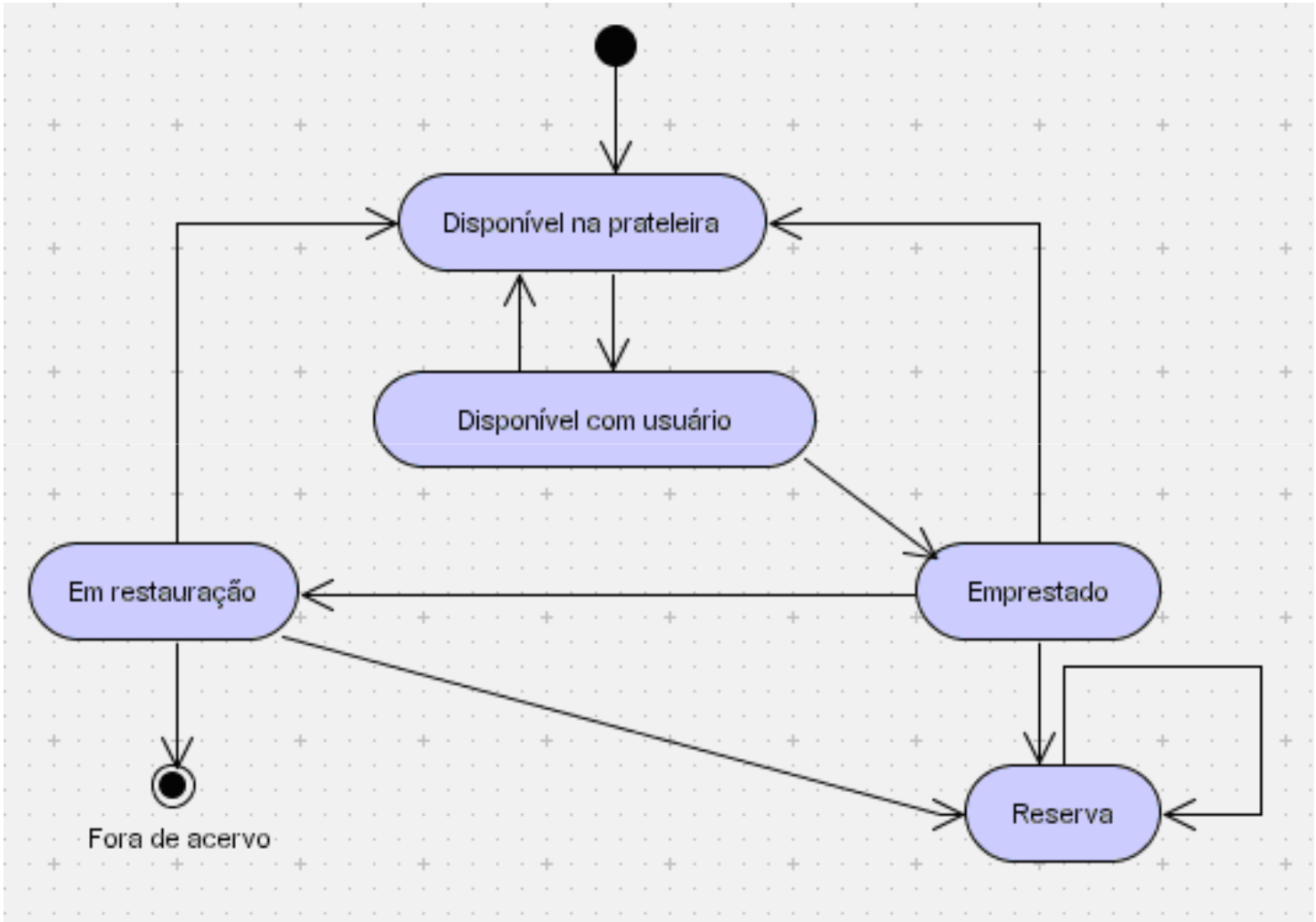
Diagrama de Estados

Um diagrama de gráfico de estados mostra uma máquina de estados, dando ênfase ao fluxo de controle de um estado para outro.

Uma *máquina de estados* é um comportamento que especifica as sequências de estados pelos quais um objeto passa durante seu tempo de vida em resposta a eventos, juntamente com suas respostas a esses eventos.

Usado normalmente para modelar o **ciclo de vida** dos objetos de uma classe

Diagrama de Estados



IMPLEMENTAÇÃO

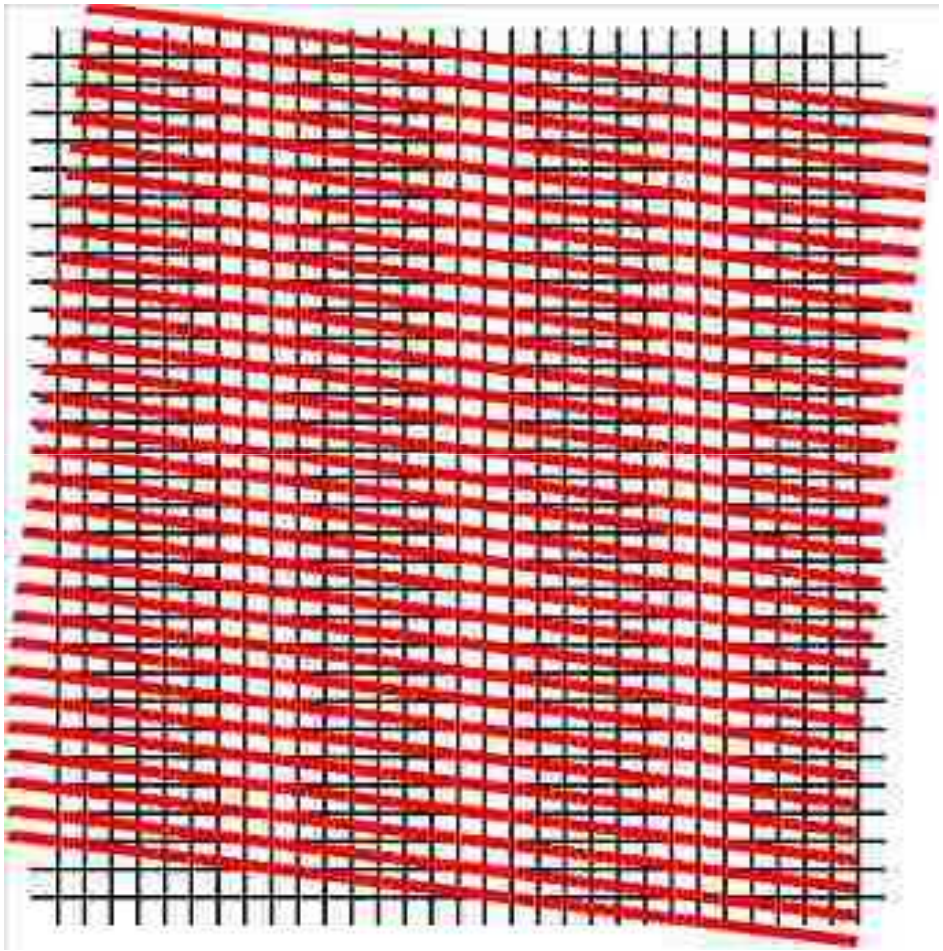
IMPLEMENTAÇÃO

A implementação se caracteriza pelo desenvolvimento (em linhas de código) do sistema proposto.

Esta etapa deve levar em consideração desde os layouts de tela, as interfaces de acesso e as regras de negócio.

Vários padrões para o desenvolvimento podem ser utilizados

Implementação: Padrões de Projeto



Implementação: Padrões de Projeto

O que é um padrão?

Maneira **testada** ou **documentada** de alcançar um objetivo qualquer.

Padrões são comuns em várias áreas da engenharia.

Design Patterns,ou Padrões de Projeto

Padrões para alcançar objetivos na engenharia de software usando classes e métodos em linguagens orientadas a objeto. Existem 23 padrões de projetos considerados úteis.

Inspirado em "A Pattern Language" de Christopher Alexander, sobre padrões de arquitetura de cidades, casas e prédios.

Implementação: Padrões de Projeto

"Os padrões de projeto são descrições de objetos que se comunicam e classes que são customizadas para resolver um problema genérico de design em um contexto específico“

Gamma, Helm, Vlissides & Johnson, sobre padrões em software

IMPLEMENTAÇÃO: PADRÕES DE PROJETO

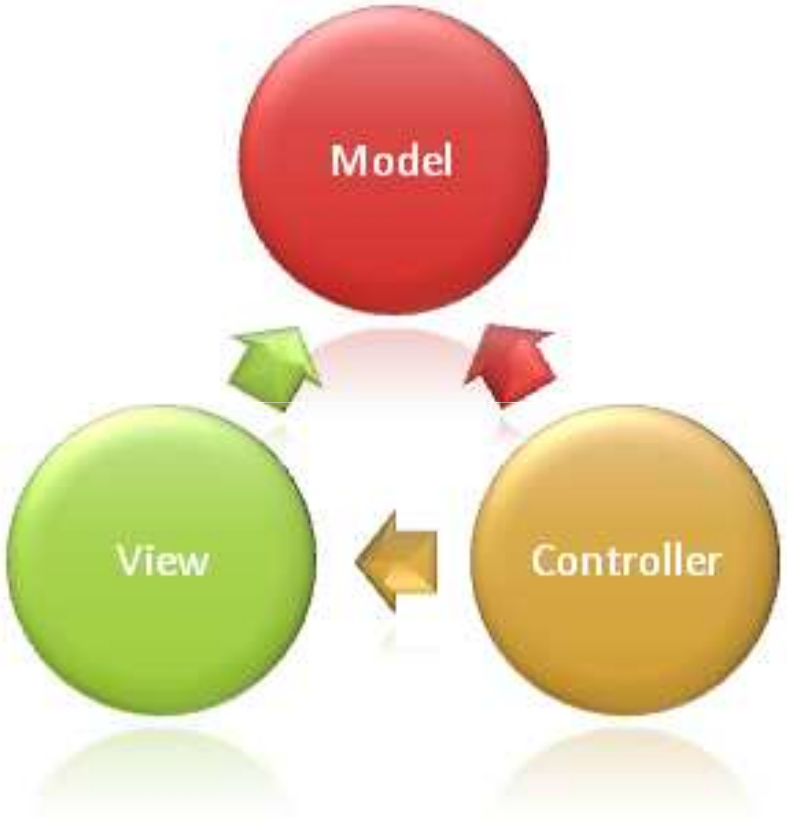
Por que aprender padrões?

- Aprender com a experiência dos outros
  Identificar problemas comuns em engenharia de software e utilizar soluções testadas e bem documentadas

- Aprender a programar bem com orientação a objetos

- Desenvolver software de melhor qualidade
  Os padrões utilizam eficientemente polimorfismo, herança, modular idade, composição, abstração para construir código reutilizável, eficiente, de alta coesão e baixo acoplamento

MVC



MVC

Model-view-controller (**MVC**) é um padrão de arquitetura de software.

Com o aumento da complexidade das aplicações desenvolvidas torna-se fundamental a separação entre os dados (**Model**) e o layout (**View**).

Desta forma, alterações feitas no layout não afetam a manipulação de dados, e estes poderão ser reorganizados sem alterar o layout.

O model-view-controller resolve este problema através da separação das tarefas de acesso aos dados e lógica de negócio, lógica de apresentação e de interação com o utilizador, introduzindo um componente entre os dois: o **Controller**.

MVC

A partir do momento em que dividimos os nossos componentes em Camadas podemos aplicar o MVC nestas.

Geralmente isto é feito definindo a Camada de Negócios como o Model, a Apresentação como a View.

O componente Controller exige um pouco mais de controle. Logo, cuidado para não confundir MVC com separação de camadas.

Camadas dizem como agrupar os componentes. **O MVC diz como os componentes da aplicação interagem.**

O MVC baseia-se em 2 princípios fortes. - O Controller Despacha as Solicitações ao Model; - A View observa o Model;

MVC

Model

Nesta camada são definidas as regras de acesso e manipulação dos dados, que normalmente são armazenados em bases de dados ou arquivos, mas nada indica que sirva só para alojamento persistente dos dados.

Pode ser usado para dados em memória volátil (memória RAM) apesar não se verificar tal utilização com muita frequência.

Todas as regras relacionadas com tratamento, obtenção e validação dos dados devem ser implementados nesta camada.

MVC

View:

Esta camada é responsável por gerar a forma como a resposta será apresentada: página web, formulário, relatório, etc...

Ou ainda como o usuário irá interagir com as possíveis entradas ao sistema (interface gráfica com o usuário)

MVC

Controller:

É a camada responsável por responder aos pedidos por parte do utilizador. Sempre que um utilizador faz um pedido ao servidor esta camada é a primeira a ser executada.

Processa e responde a eventos e pode invocar alterações no Model.

É lá que é feita a validação dos dados e também é onde os valores postos pelos usuários são filtrados.

MVC

Cadeia de funcionamento

1 - O usuário interage com a interface de alguma forma (por exemplo, o usuário aperta um botão)

2 - O Controller manipula o evento da interface do usuário através de uma rotina pré-escrita.

3 - O Controller acessa o Model, possivelmente atualizando-o de uma maneira apropriada, baseado na interação do usuário (por exemplo, atualizando os dados de cadastro do usuário).

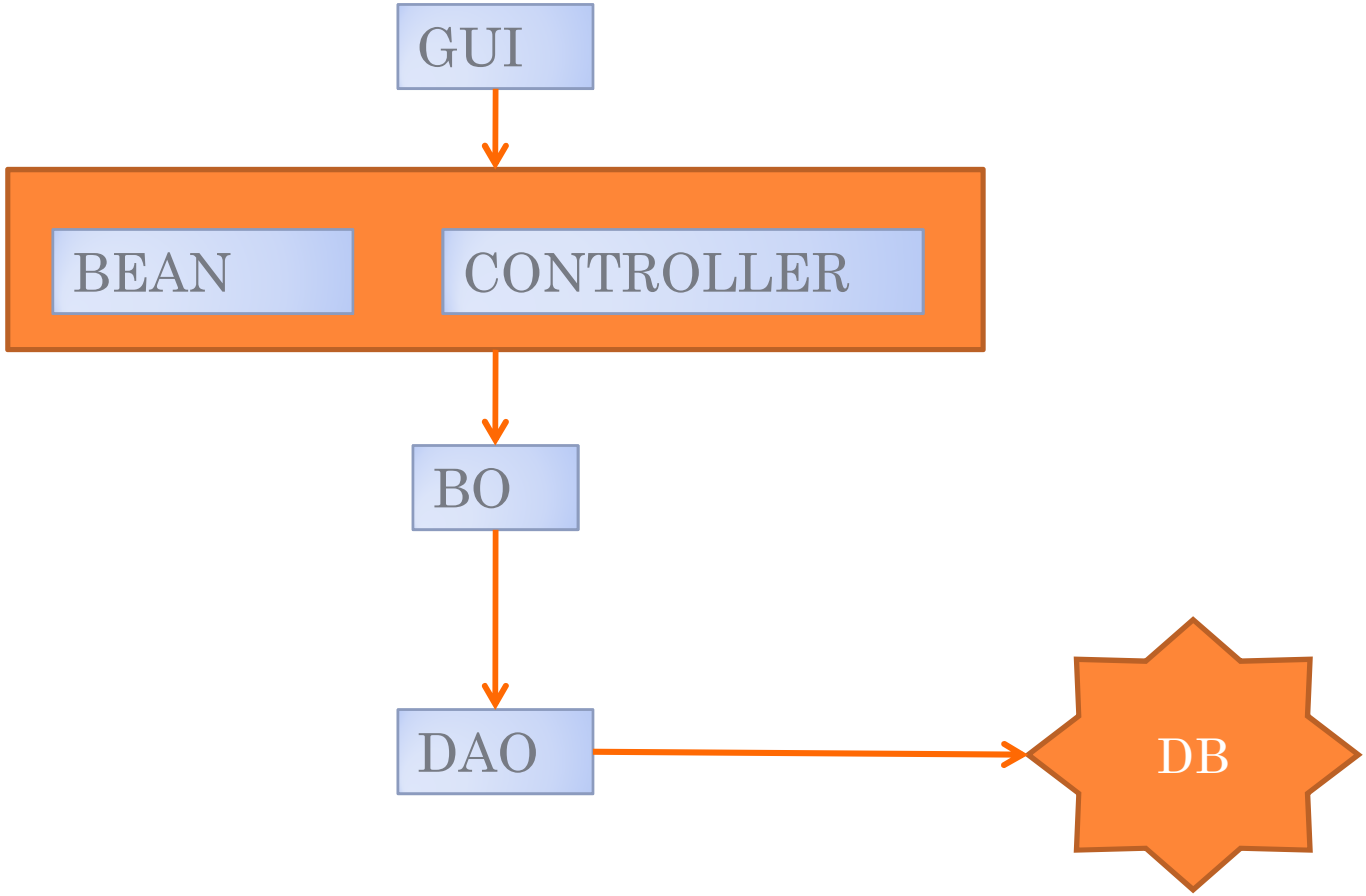
MVC

Cadeia de funcionamento

4 - Algumas implementações de View utilizam o Model para gerar uma interface apropriada (por exemplo, mostrando na tela os dados que foram alterados juntamente com uma confirmação). O View obtém seus próprios dados do Model. O Model não toma conhecimento direto da View.

5 - A interface do usuário espera por próximas interações, que iniciarão o ciclo novamente.

MVC



TESTES

Testes de Software

Para a grande maioria dos desenvolvedores, testar os programas desenvolvidos, é a parte mais chata de todo o processo.

Por outro lado, a entrega de um sistema sem passar pelos devidos testes, torna propenso a erros graves e prejuízos tanto para os usuários quanto para o desenvolvedor.

Com o objetivo de auxiliar e automatizar testes, existem ferramentas facilitadoras.

Iremos utilizar JUNIT.

JUNIT

O JUnit é um framework open-source, criado por Eric Gamma e Kent Beck, com suporte à criação de testes automatizados na linguagem de programação Java.

Esse framework facilita a criação de código para a automação de testes unitários com apresentação dos resultados.

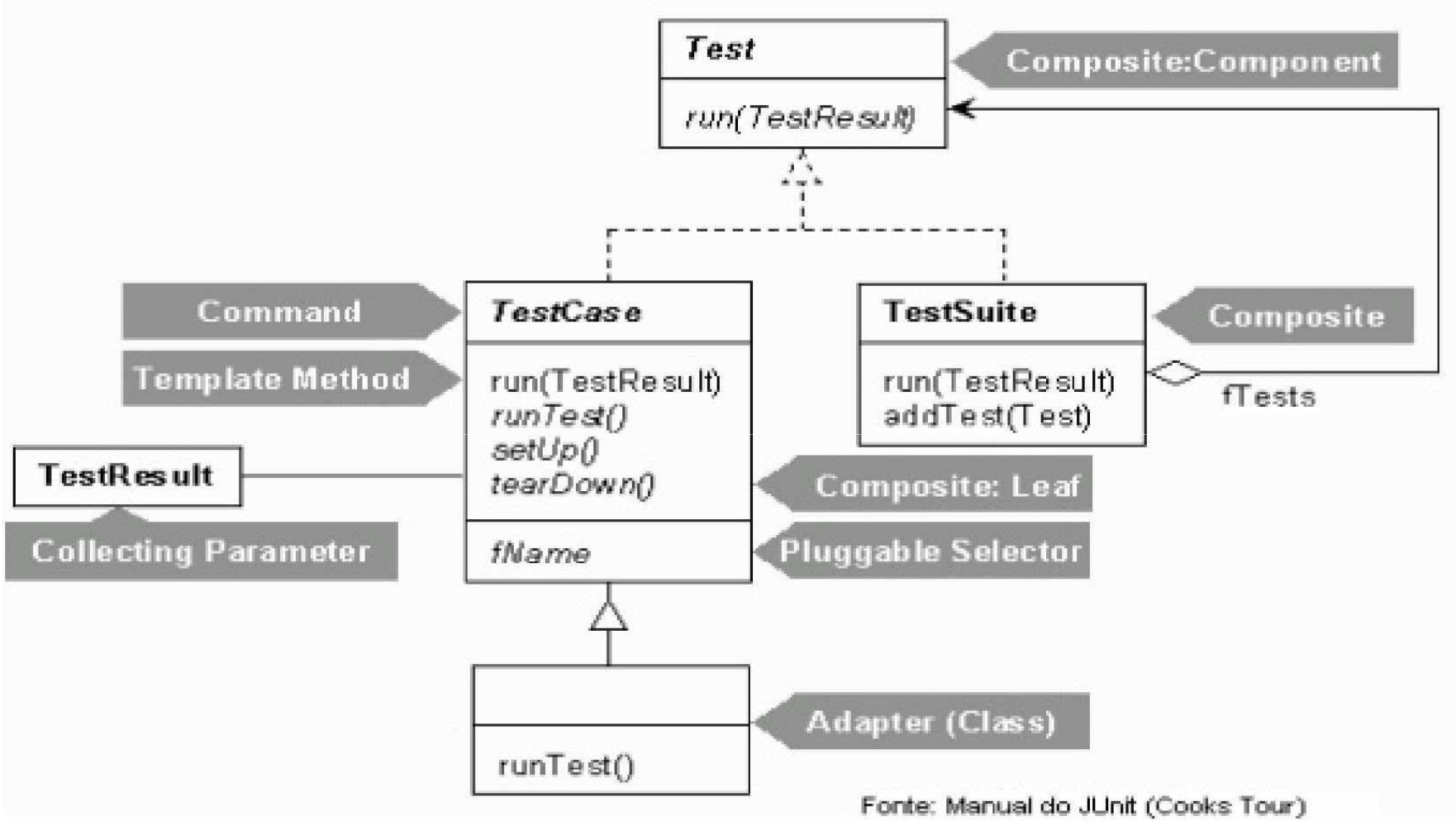
Com ele, pode ser verificado se cada método de uma classe funciona da forma esperada, exibindo possíveis erros ou falhas podendo ser utilizado tanto para a execução de baterias de testes como para extensão.

JUNIT

Para utilizar o JUnit, é necessário a utilização do .jar do JUnit que pode ser encontrado na página principal do próprio framework (http://www.junit.org).

Para configurar o Junit em seu ambiente, basta adicionar o .jar do JUnit ao classpath do projeto.

JUNIT


Test
run(TestResult)
Composite:Component
Command
Template Method
TestCase
run(TestResult)
runTest()
setUp()
tearDown()
fName
TestSuite
run(TestResult)
addTest(Test)
Composite
fTests
TestResult
Collecting Parameter
Composite: Leaf
Pluggable Selector
runTest()
Adapter (Class)
Fonte: Manual do JUnit (Cooks Tour)

JUNIT

Classe TestCase:

command – O padrão (pattern) permite encapsular um pedido (de teste) como objeto e fornece um método run().
run() – Cria um contexto (método setUp); em seguida executa o código usando um contexto e verifica o resultado (método runTest); e por fim, limpa o contexto (método tearDown).
setUp() – Método chamado antes de cada método, pode ser utilizado para abrir uma conexão de banco de dados.
tearDown() – Método chamado depois de cada método de teste, usado para desfazer o que setUp() fez, por exemplo fechar uma conexão de banco dedados.
runTest() – Método responsável por controlar a execução de um teste particular.

JUNIT

Classe TestSuite:

Com esta classe, o desenvolvedor executa um único teste com vários métodos de testes e registra os resultados num TestResult.

composite – O padrão (pattern) permite tratar objetos individuais e composições de objetos de forma uniforme. Este padrão requer os seguintes participantes:

addTest() – Método responsável em adicionar um novo teste a rotina.

JUNIT

Pesquisa de Testes com JUNIT