

**Disciplina:**

# **Sistemas de Informação I**

Curso: Sistemas de Informação  
Prof. Cristiano Vieira  
3º Período - 2019

# Apresentação



# Plano de Ensino

- Introdução ao Conteúdo
- Especificação de SI;
- Sistemas de informações Gerenciais e de apoio à decisão;
- Soluções de Problemas organizacionais;
- Exemplos de sistemas de informações comerciais.

# Plano de Ensino

## Objetivos

- Esta disciplina tem por objetivo desenvolver habilidades específicas relativas ao estudo de Sistemas de Informação Gerenciais.
- Trabalhar com uma visão profunda de como as empresas utilizam os SI e as tecnologias de Informação para atingir seus objetivos.

# Plano de Ensino

## **RECURSOS METODOLÓGICOS:**

Aulas teóricas expositivas em sala ou no LTI, apresentação de modelos práticos e aplicáveis, exercícios propostos e dinâmicas em grupos.

## **PROCESSO DE AVALIAÇÃO:**

- Trabalho proposto compondo 30% da nota.
- Prova teórica bimestral compondo 65% da nota.
- Pontos extras até 5%, para incentivo ao estudo e dedicação.
- A entrega de trabalhos ou dos projetos fora do prazo acarretará em perda de pontos.

# Programação da Disciplina

1. Especificação de SI;
2. Sistemas de informações Gerenciais e de apoio à decisão;
3. Tomada de Decisão e a Gestão do Conhecimento;
4. Soluções de Problemas organizacionais;
5. Exemplos de sistemas de informações comerciais;
6. Estudo de Problemas Organizacionais;

# Referências Bibliográficas

## BIBLIOGRÁFIA BÁSICA

1. LAUDON, Kenneth C.; LAUDON, Jane P. Sistemas de informação gerenciais: Administrando a empresa digital. 5<sup>a</sup> ed. São Paulo: Prentice Hall, 2004. 562 p.
2. STAIR, Ralph M.; REYNOLDS, George W. Princípios de sistema de informação: uma abordagem gerencial. 4 ed. Rio de Janeiro: LTC, 2002.
3. O'BRIEN, James A. Sistemas de Informação e as Decisões Gerenciais na Era da Internet. São Paulo: Saraiva, 2003.

# Referências Bibliográficas

## BIBLIOGRÁFIA COMPLEMENTAR

1. Turban Efraim; Rainer Jr., R. Kelly; Potter, Richard E. Administração de tecnologia da informação: teoria e prática. Rio de Janeiro: Elsevier, 2005. 618 p. ISBN 85-352-1571-9.
2. POLLONI, Enrico Giulio Franco. Administrando sistemas de informação. São Paulo: Futura, 2000. 272 p. ISBN 85-7413-028-1.
3. LAUDON, Kenneth C.; LAUDON, Jane P. Gerenciamento de sistemas de informação. 3 ed. Rio de Janeiro: LTC, 2001.
4. CHEN, Peter. Modelagem de Dados: a abordagem entidade-relacionamento para projeto lógico. São Paulo: Makron Books, 1990.
5. MONTEIRO, Mário A. Introdução à organização de computadores. Rio de Janeiro: LTC, 1996.

# Capítulo 1



# Introdução ao Software

# Software

Primeiro programador: Ada Lovelace (1843) escreveu um código rudimentar para máquina analítica criada por Charles Babbage (1827)

Lógica Booleana: George Boole (1815 – 1864) Matemático britânico que provou a relação da lógica com a matemática e não com a filosofia como então era pensado.

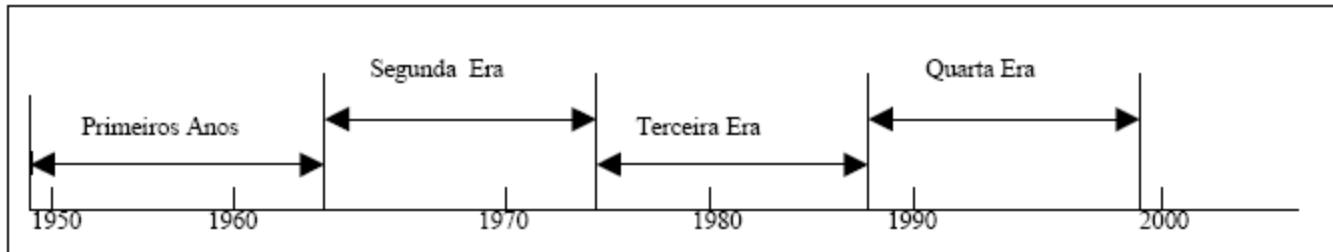
John Von Neumann (1945) elaborou a técnica de *shared-program* e depois a linguagem de programação.

# Software

O software é um elemento de sistema lógico, e não físico. Portanto, o software tem características que são consideravelmente diferentes das do hardware:

1. O software é desenvolvido e projetado por engenharia, não manufaturado no sentido clássico.
2. O software não se desgasta.
3. A maioria dos softwares é feita sob medida em vez de ser montada a partir de componentes existentes.

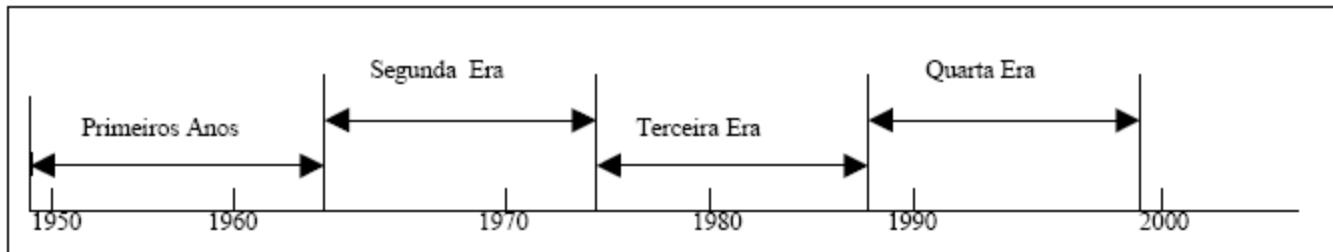
# Evolução da Programação



**Primeiros Anos :** Nos primeiros anos, a programação era uma arte “secundária” para qual havia poucos métodos sistemáticos. O desenvolvimento do software era feito virtualmente sem administração. Usado a orientação batch (em lote) para a maioria dos sistemas.

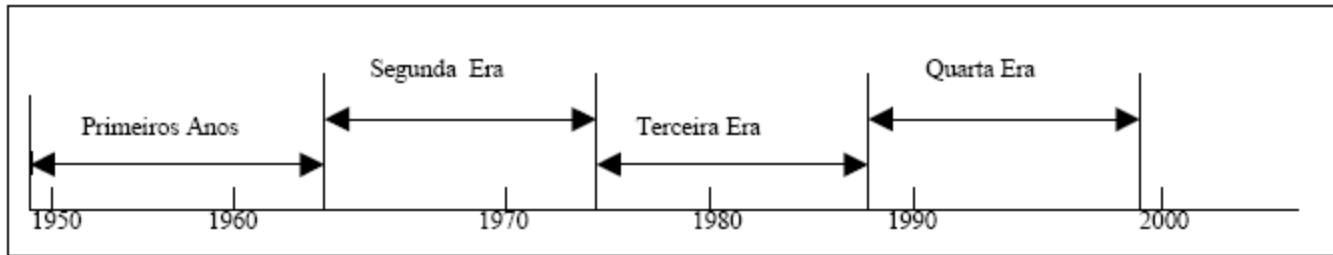
- Software projetado para cada aplicação, distribuição limitada
- Desenvolvimento na própria organização – individualidade
- Sem documentação

# Evolução da Programação



- Segunda Era:
- \_ Multiprogramação e multiusuário
  - \_ Sistemas de tempo-real e bancos de dados
  - \_ Desenvolvimento para ampla distribuição no mercado – software house
  - \_ Manutenção – crise de software – detectadas falhas necessidade de correção ou adaptados a um novo hardware que fosse comprado. O esforço despendido na manutenção de software começou a absorver recursos em índices alarmantes. E, ainda pior, a natureza personalizada de muitos programas tornava-os virtualmente impossíveis de sofrer manutenção.

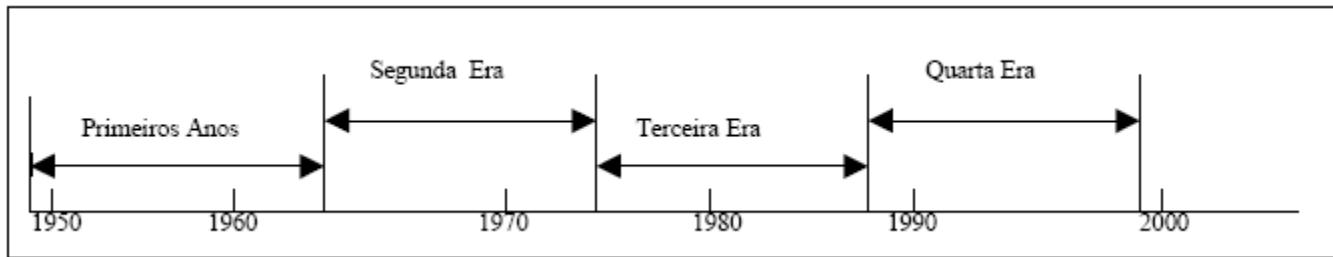
# Evolução da Programação



Quarta Era: A Quarta era do software de computador está apenas começando.

- \_ Novas abordagens – desenvolvimento de software
- \_ Sistemas especialistas
- \_ Redes Neurais Artificiais

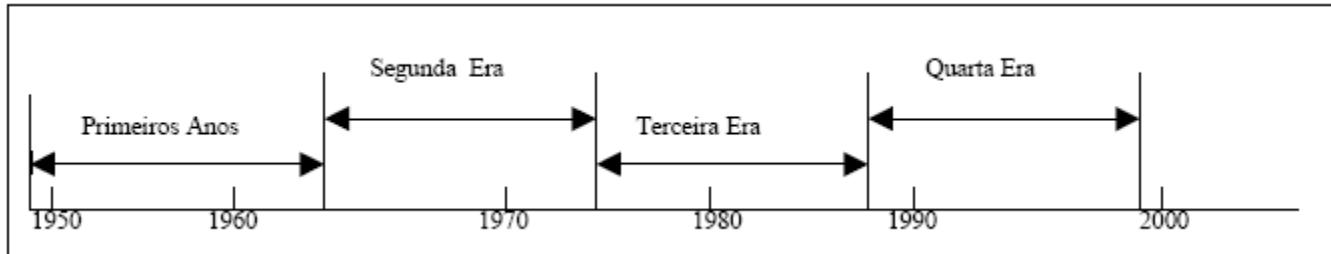
# Evolução da Programação



Terceira Era: Começou em meados da década de 70 e continua até hoje.

- \_\_ Sistemas distribuídos
- \_\_ Redes Locais e Remotas
- \_\_ Aumento da complexidade
- \_\_ Computador pessoal
- \_\_ Produtos inteligentes

# Evolução da Programação



Quinta Era:

- \_ Softwares mais complexos?
- \_ Atendimento de demanda?
- \_ Manutenção eficiente?

# Aplicações de Software

- Software Básico: apoio. Componentes de sistemas operacionais,..
- Software de Tempo Real: ambiente. ...
- Software Comercial: acesso à informação administrativa,..
- Software Científico e de Engenharia: algoritmos de processamento numérico,...
- Software Embutido
- Software de Computador Pessoal: processadores de texto, planilhas, ...
- Software de inteligência Artificial: técnicas de IA

# Linguagens e suas gerações

1<sup>a</sup>G → linguagem Assembler – o mais baixo nível de abstração

2<sup>a</sup>G → Cobol, Fortran, Algol

3<sup>a</sup>G → Programação Estruturada

Uso geral – PL1, Pascal, C, Ada

Orientada a Objeto: C ++, Objective-C, CLOS, VisualBasic

Visual Smalltalk, C#, Power Builder, Delphi, Visual Object

4<sup>a</sup>G → Linguagens com um nível elevado de abstração

Linguagens de consulta

Geradores de Programas – em linguagem 3<sup>a</sup> G

Linguagens de apoio a decisões

Linguagens de prototipação

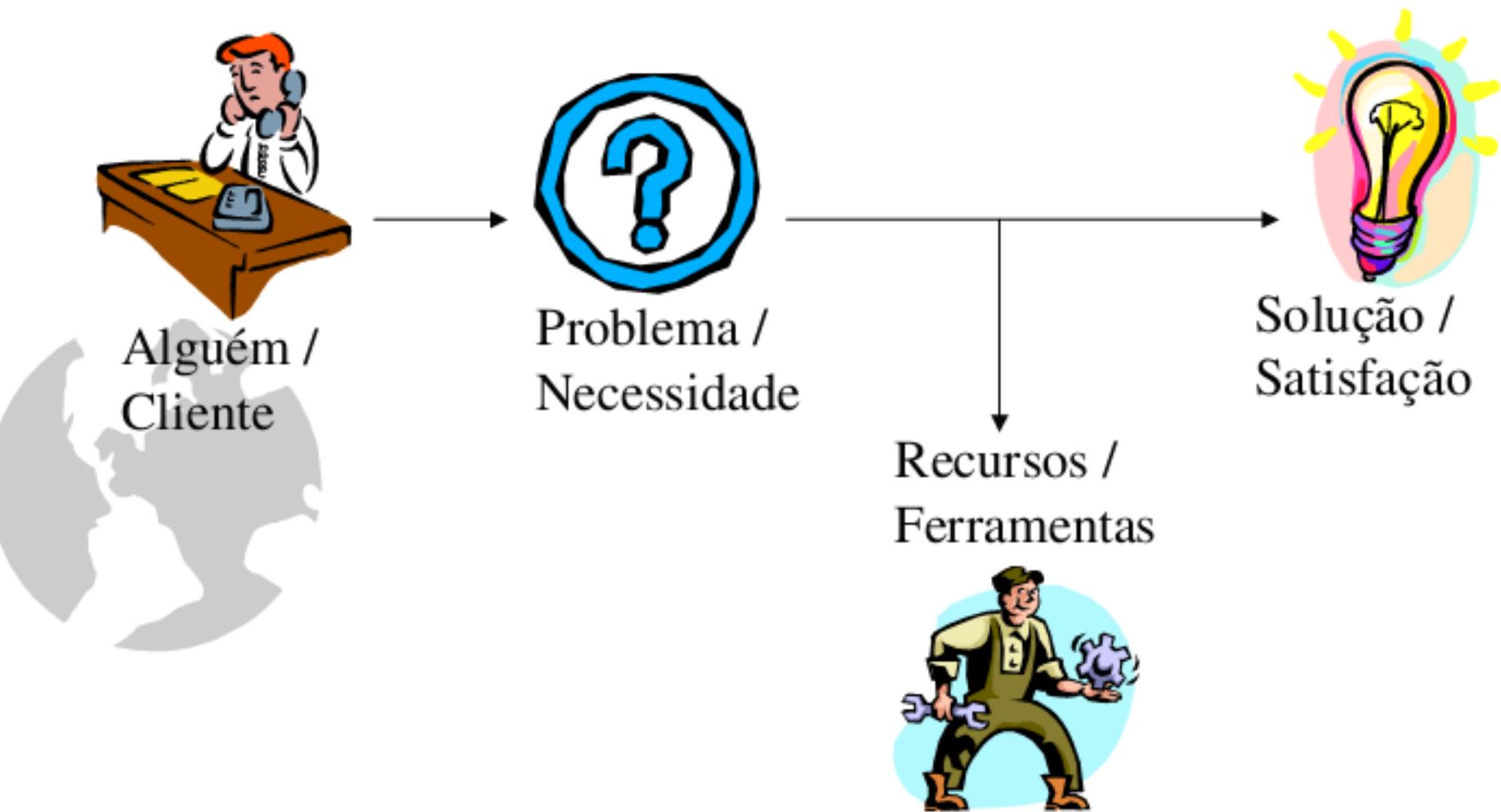
Linguagens formais de especificações - Prolog, PL1, Lisp

Ferramentas no ambiente PC – Planilhas, Sistemas de B.D.

CASE → UsDesigner, Designer,

IBPI – o mais alto nível de abstração

# Relação entre Softwares e a Informação



O QUE É  
INFORMAÇÃO?

# É muito importante estar informado!



Você sairia com ela?

Você pode sair com ela ou não. Entretanto a informação sobre quem ela é (ou quem ela foi) é importante para uma melhor decisão.

# Bases dos Sistemas de Informação

# Definição de Sistema

É um conjunto de componentes que, segundo um plano, é sujeito a restrições, procurando atingir objetivos interagindo com elementos naturais, evoluindo no tempo e sendo controlado.

# ABACO

inventado 2000 ac



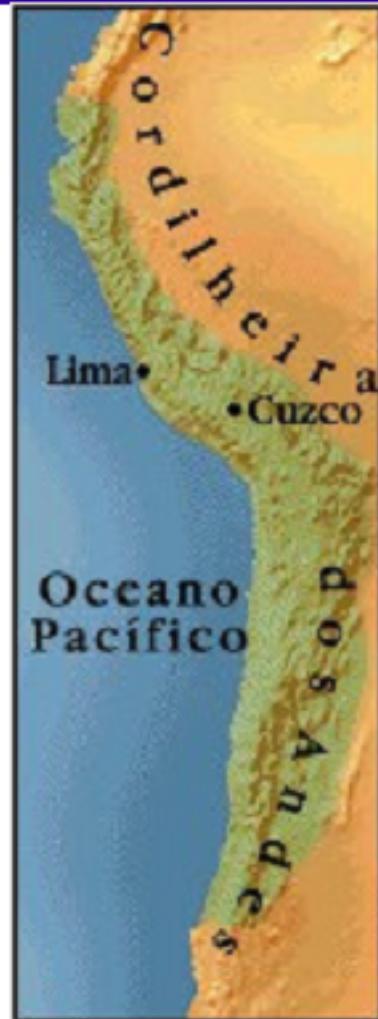
# Sistema de Informação Inca

**Capital: Cuzco**

**Período: 1100 – 1572**

## VIDA INCA

O IMPÉRIO INCA tinha mais de 10 milhões de habitantes. No auge, cobria o atual Peru, o sul da Colômbia, Bolívia, Equador, norte do Chile e norte da Argentina. Chefes incas administravam este vasto território sem um sistema de escrita e sem veículos com rodas.



# Sistema de Informação Inca

- Não possuíam linguagem escrita ou números;
- Usavam o *Quipu*:
  - Barbante comprido com outros barbantes pendurados de várias cores e espessuras;
  - Este barbantes recebiam nós, que representavam as quantidades de coisas.
- Os registros em *Quipu* ficavam em Cuzco;
- O governo sempre tinha informações precisas sobre o estado do império:
  - Quantos viviam em cada área;
  - Registros das colheitas;
  - Registros das exatas quantidades de lã, armas e metais preciosos.

# Bases dos Sistemas de Informação

- Era do Descobrimento
- Era do Aprendizado
- Era do Armazenamento e Perpetuação
- Era do Conhecimento
- Era da Inteligência Artificial
- Era da Informação
- Era do(a) ????????

# Sistemas de Informação

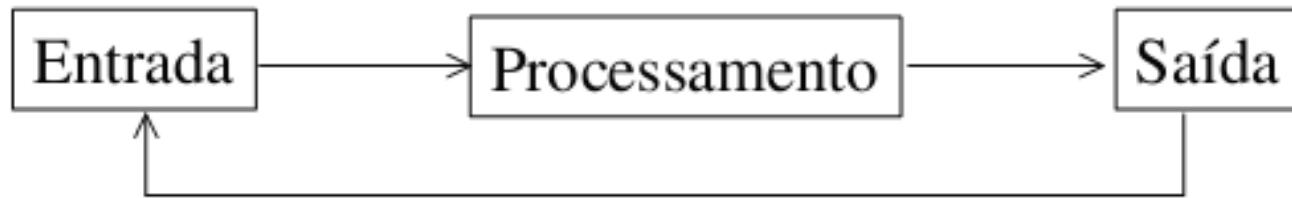
## Capacidades de um SI?

- Processar transações de forma rápida e precisa.
- Armazenar e acessar rapidamente grandes massas de dados.
- Comunicação rápida.
- Reduzir sobrecarga de informações.
- Fornecer suporte para a tomada de decisões.

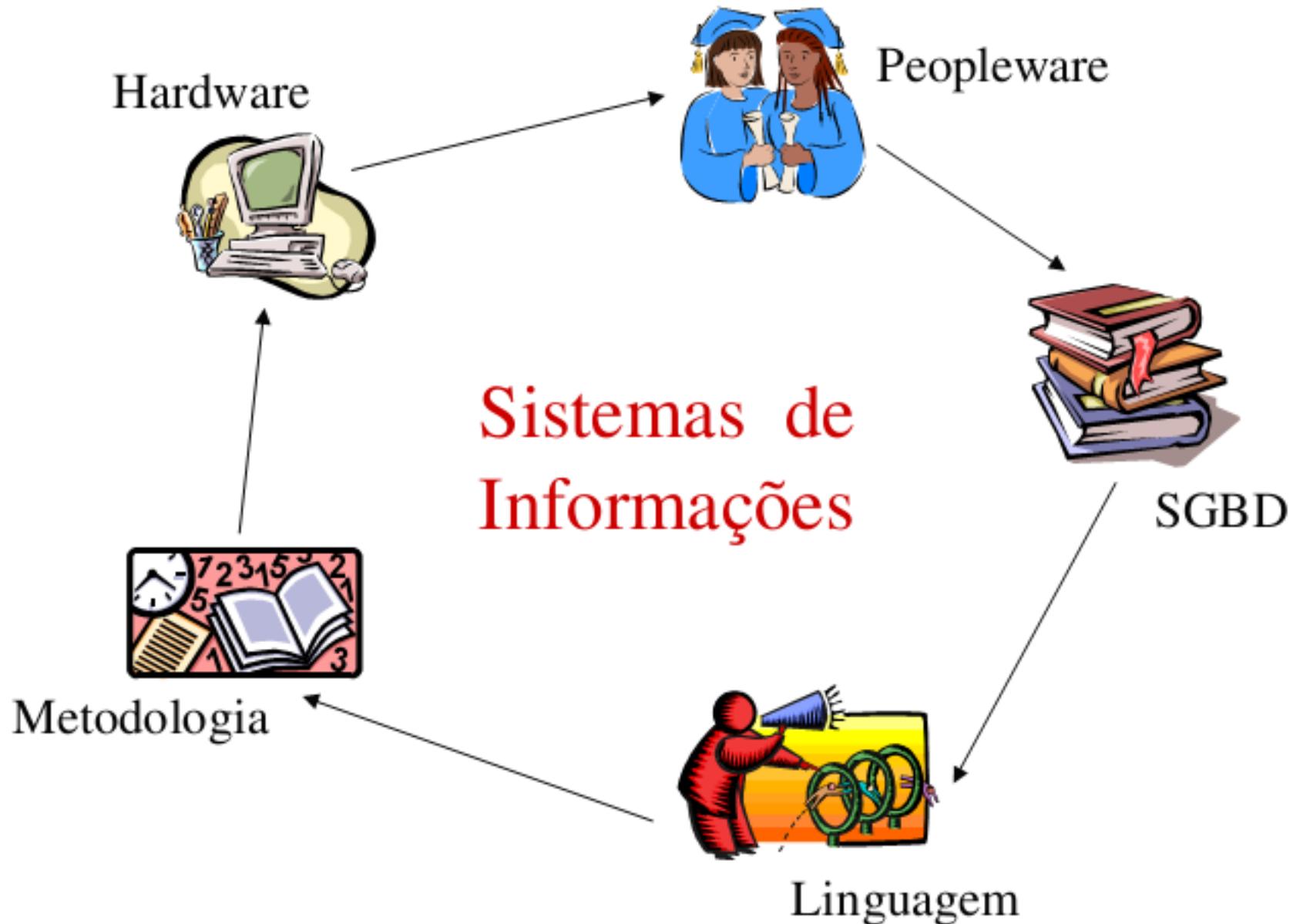
# Bases de Sistemas de Informação

## Atividade básicas de um SI?

- Entrada (input),
- Processamento,
- Saída (output),
- Realimentação (feedback)



## Recursos / Ferramentas:



# Hardware



Equipamentos elétricos/eletrônicos/mecânicos responsáveis por todas atividades de interpretação, armazenamento, processamento, entradas e saídas das informações.

- CPU;
- Periféricos de entrada;
- Periféricos de Saída;
- Unidades de armazenamento;
- Unidades de comunicação.

# Peopleware

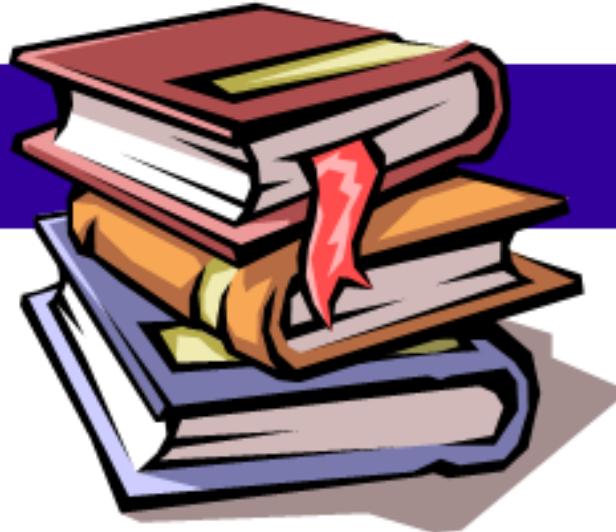


Pessoas capacitadas a entender e utilizar o hardware.

- Programadores;
- DBA;
- Técnicos de manutenção;
- Técnicos de suporte;
- Analistas de Sistemas;
- Operadores.

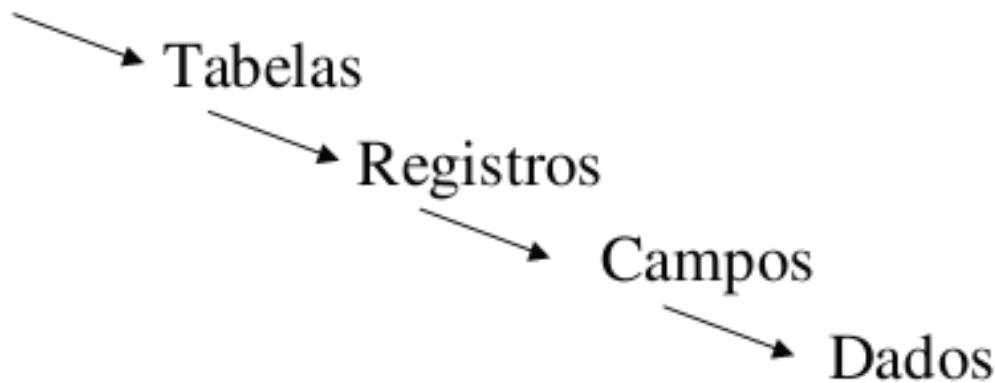
# S.G.B.D.

Sistema de Gerenciamento de  
Banco de Dados



Responsáveis pela criação e manutenção dos bancos de dados e consequentemente pelas informações lá armazenadas.

Banco de Dados



# Linguagem

Forma de escrita padronizada que é interpretada pelo processador e age como ordens ao funcionamento do sistema. É o meio de comunicação homem-máquina.



- Visual Basic
- C
- Delphi
- Clipper
- Pascal
- Cobol
- Fortran

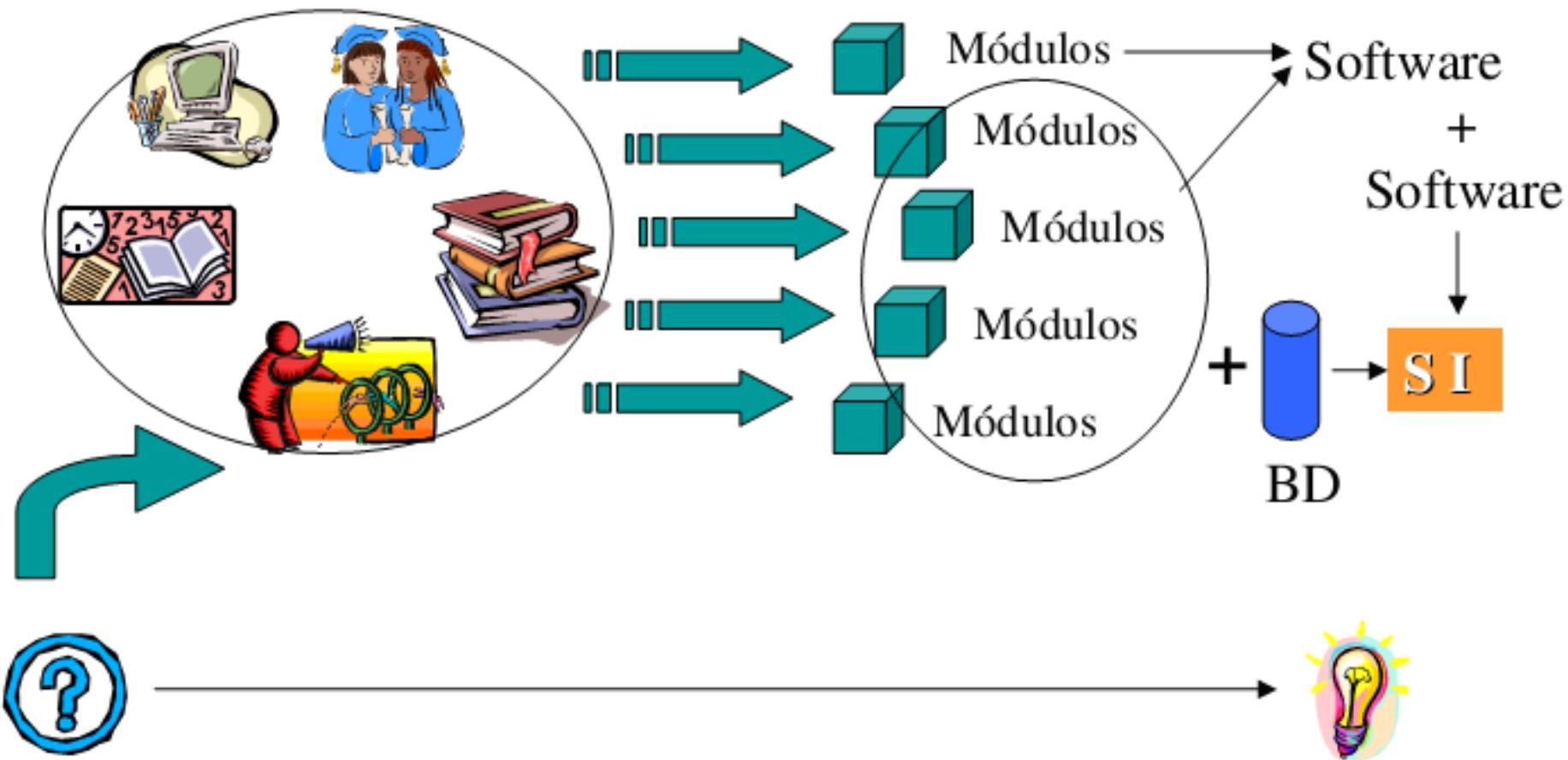
# Metodologia



## Técnicas de Engenharia de Software

- Diferença entre sistema e softwares
- Tipos de sistemas de informações
- Ciclo de vida de softwares
- Adises de desenvolvimento
- Poka-Yoki
- Projeto de Desenvolvimento...

# Não confundir Software e Sistemas



# Perspectiva Empresarial em SI

Do ponto de vista empresarial, um sistema de informação é uma solução organizacional e administrativa, baseada na tecnologia da informação, para um desafio imposto pelo ambiente.

Observe a definição acima ...

Existe uma ênfase na natureza organizacional e administrativa dos SI

Um Gerente precisa ter um entendimento amplo da Organização, de Administração e dimensões de Tecnologia de Informação e do seu poder de fornecer solução para desafios e problemas no ambiente empresarial

# Perspectiva Empresarial em SI



Usar Sistemas de Informação eficientemente requer uma compreensão de como a organização, a administração e a tecnologia da informação moldam os sistemas

# Perspectiva Empresarial em SI

## Organizações

- Funções empresariais (Vendas e Marketing; Fabricação e Produção; Finanças; Contabilidade; Recursos Humanos);
- Procedimentos operacionais padrões (regras formais para realização de tarefas);
- Cada empresa tem uma cultura peculiar e um conjunto fundamental de premissas.

# Perspectiva Empresarial em SI

## **Administração**

- Estabelecer uma estratégia organizacional;
- Formular planos de ação para a resolução de problemas.

# Perspectiva Empresarial em SI

## Tecnologias

- Infra-estrutura em tecnologia de informação:
  - Hardware, software, tecnologia de Armazenagem, tecnologia de comunicações
- Provê a fundação da qual a empresa pode montar seu sistema de informação específico.

## Capítulo 2

# Engenharia de Software

# Software e Engenharia de Software

- Relação entre Hardware e Software
- Evolução histórica
  - A partir 1980, o Software se ultrapassa o Hardware em importância.
- Capacidade de adaptação
- Elemento fundamental para gestão de negócios
- Baixo custo e grande variabilidade

# Software

O software é um elemento de sistema lógico, e não físico. Portanto, o software tem características que são consideravelmente diferentes das do hardware:

1. O software é desenvolvido e projetado por engenharia, não manufaturado no sentido clássico.
2. O software não se desgasta.
3. A maioria dos softwares é feita sob medida em vez de ser montada a partir de componentes existentes.

# A importância do Software

Durante as três primeiras décadas da era do computador, o principal desafio era desenvolver um hardware que reduzisse o custo de processamento e armazenagem de dados. Ao longo da década de 1980, avanços na microeletrônica resultaram em maior poder de computação a um custo cada vez mais baixo. Hoje, o problema é diferente. O principal desafio durante a década de 1990 é melhorar a qualidade (e reduzir o custo) de soluções baseadas em computador - soluções que são implementadas com software.

# Problemas associados ao Software

Existem um conjunto de problemas associados ao software que vários autores chamam de “crise do software”. Este conjunto de problemas não se limitam ao software que não funciona adequadamente, mas abrange também problemas associados a forma de desenvolvimento destes softwares, a forma como é efetuada a manutenção destes softwares, como atender a demanda por novos softwares e como desenvolver novos softwares cada vez mais rapidamente.

- Estimativas de prazos e de custos que são frequentemente imprecisos;
- Produtividade dos profissionais da área que não tem acompanhado a demanda por novos serviços;
- A qualidade do software desenvolvido que é insuficiente.

# A importância do Software

Durante as três primeiras décadas da era do computador, o principal desafio era desenvolver um hardware que reduzisse o custo de processamento e armazenagem de dados. Ao longo da década de 1980, avanços na microeletrônica resultaram em maior poder de computação a um custo cada vez mais baixo. Hoje, o problema é diferente. O principal desafio durante a década de 1990 é melhorar a qualidade (e reduzir o custo) de soluções baseadas em computador - soluções que são implementadas com software.

# Problemas associados ao Software

Existem um conjunto de problemas associados ao software que vários autores chamam de “crise do software”. Este conjunto de problemas não se limitam ao software que não funciona adequadamente, mas abrange também problemas associados a forma de desenvolvimento destes softwares, a forma como é efetuada a manutenção destes softwares, como atender a demanda por novos softwares e como desenvolver novos softwares cada vez mais rapidamente.

- Estimativas de prazos e de custos que são frequentemente imprecisos;
- Produtividade dos profissionais da área que não tem acompanhado a demanda por novos serviços;
- A qualidade do software desenvolvido que é insuficiente.

# A Crise de Software...

25% dos projetos são cancelados  
o tempo de desenvolvimento é bem maior do que o estimado

75% dos sistemas não funcionam como planejado

a manutenção e reutilização são difíceis e custosas

os problemas são proporcionais a complexidade dos sistemas

# Causas da Crise de Software

## Essências

- Complexidade dos sistemas
- Dificuldade de formalização

## Acidentes

- Má qualidade dos métodos, linguagens, ferramentas, processos, e modelos de ciclo de vida
- Falta de qualificação técnica

# Softwares no “Mundo Real”

- Fazer software no “mundo real” deve considerar fatores como:
  - Escopo
  - Custo
  - Prazo
  - Qualidade
- Em função do tamanho do software, esses fatores se tornam difíceis de garantir!



# Cenários

## Cenário 1: Agenda Pessoal

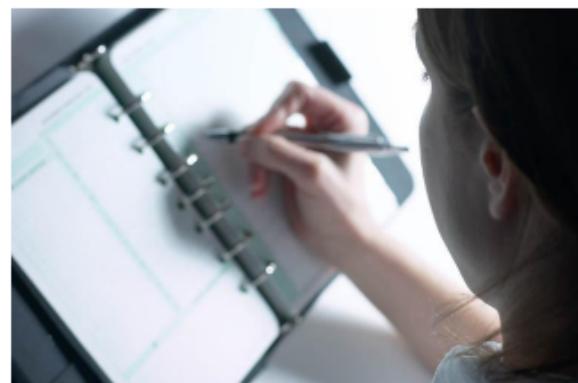
### Objetivo

- Guardar o nome e o aniversário de até 50 pessoas

Quanto custa para fazer?

Quanto tempo vai levar para ficar pronto?

Qual a consequência no caso de defeito?



# Cenários

## Cenário 2: Boeing 777

### Objetivo

- Controlar todo o hardware do Boeing 777

Quanto custa para fazer?

Quanto tempo vai levar para ficar pronto?

Qual a consequência no caso de defeito?



# Cenários

## Tamanho

- Mais de 4 milhões de linhas de código
- Linguagem dominante (>99%): Ada

## Documentação

- De 100 a 10.000 páginas por sub-sistema
- Total de 79 sub-sistemas integrados

## Duração

- 4,5 anos de desenvolvimento

## Ampla utilização de Engenharia de Software

## Em operação desde 1995

- Zero acidentes graves até 2006

# Cenários

## Outros cenários extremos...

Toyota Lexus LS460: > 7 MLOCs



Eclipse Galileo: 24MLOCs



Windows XP: 40 MLOCs



- 1800 desenvolvedores
- 2200 testadores

SAP: 250 MLOCs



Debian GNU/Linux 4: 283 MLOCs



- 1000 desenvolvedores

# Qual a dimensão dos códigos?

Assuma que uma folha A4 tem em torno de 50 linhas.

Assuma que uma pilha de 1000 folhas A4 tem em torno de 10 centímetros de altura

Assim, 1.000.000 de LOCs, caso impresso, seria uma pilha de 2 metros de altura!

Caso todo o código do Debian GNU/Linux fosse impresso, teria a altura de um prédio de 188 andares!!!

# Caso real: Therac-25

Máquina de radioterapia controlada por computador

Problema:

- Doses indevidas de radiação emitidas

Causa:

- Interface com usuário inapropriada
- Documentação deficiente
- Software reutilizado sem ser adaptado para o novo hardware
- Software de sensores de falha com defeito

Conseqüências

- Ao menos 5 mortes entre 1985 e 1987



<http://sunnyday.mit.edu/papers/therac.pdf>

# Caso real: Ariane 5

Foguete lançador de satélites

Problema:

- O foguete se auto-destruiu 40 segundos após o lançamento



Causa:

- Software reutilizado sem ser adaptado para o novo hardware
- Ausência de testes em solo deste software
- Defeito apresentado em vôo

Conseqüências

- Prejuízo de mais de US\$ 370.000.000,00 em 1996

# Fazer software é arte?

Parte arte, parte engenharia...

- Se o cantor/ator/pintor errar, a audiência fica chateada
- Se o engenheiro civil errar o prédio pode cair
- Se o médico errar o paciente pode morrer

Se o desenvolvedor de software errar, o que pode acontecer?

# Definição de Eng. Software

“O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais”.

“Engenharia de Software é uma disciplina que trata do desenvolvimento e da aplicação de metodologias, métodos, técnicas e ferramentas para desenvolver e manter sistemas de software com qualidade pre ditível e controlável, operando em máquinas e ambientes reais de modo econômico”.

**Métodos + Ferramentas + Procedimentos = Paradigma de ES**

# Definição de Eng.Software

Estudo ou aplicação de abordagens sistemáticas, econômicas e quantificáveis para o desenvolvimento, operação e manutenção de software de qualidade.

Engenheiros de software devem adotar uma abordagem sistemática e organizada para seu trabalho e usar ferramentas e técnicas/métodos apropriados dependendo do problema a ser solucionado, das restrições de desenvolvimento e dos recursos disponíveis

# Definição de Eng. Software

Programas de computador e documentação associada

Produtos de software podem ser desenvolvidos para um cliente particular ou podem ser desenvolvidos para um mercado geral

# Sites de Interesse em Engenharia de Software

## Sociedades e Institutos

[Softex](#) - Sociedade Brasileira para a exportação de software.

[Software Engineering Institute](#).

[SIGSOFT](#) - Grupo de Interesse em ES da ACM. Tem links para revistas e anais de conferências.

## Revistas especializadas

[IEEE Software](#).

[Software Development Magazine](#)

[C/C++ Users Journal](#)

[Dr. Dobb's Journal](#)

[MSDN Magazine](#)

[Sys Admi](#)

[SD Expo](#)

[Unixreview](#)

[Windows Developer's Journal](#)

## Software Metrics

[IFPUG](#)

[COCOMO II](#)

# Objetivos da Engenharia de Software

Controle sobre o desenvolvimento de software dentro de **custos, prazos** e níveis de **qualidade** desejados

Produtividade no desenvolvimento, operação e manutenção de software

Qualidade versus Produtividade

Permitir que profissionais tenham controle sobre o desenvolvimento de software dentro de custos, prazos e níveis de qualidade desejados

# Objetivos da Eng. de Software

Saber como estimar um projeto (tamanho, custo, cronograma)

Saber como monitorar o andamento de um projeto de desenvolvimento e a outros elementos vinculados ao software

Saber como testar efetivamente

Saber como controlar a evolução do Software

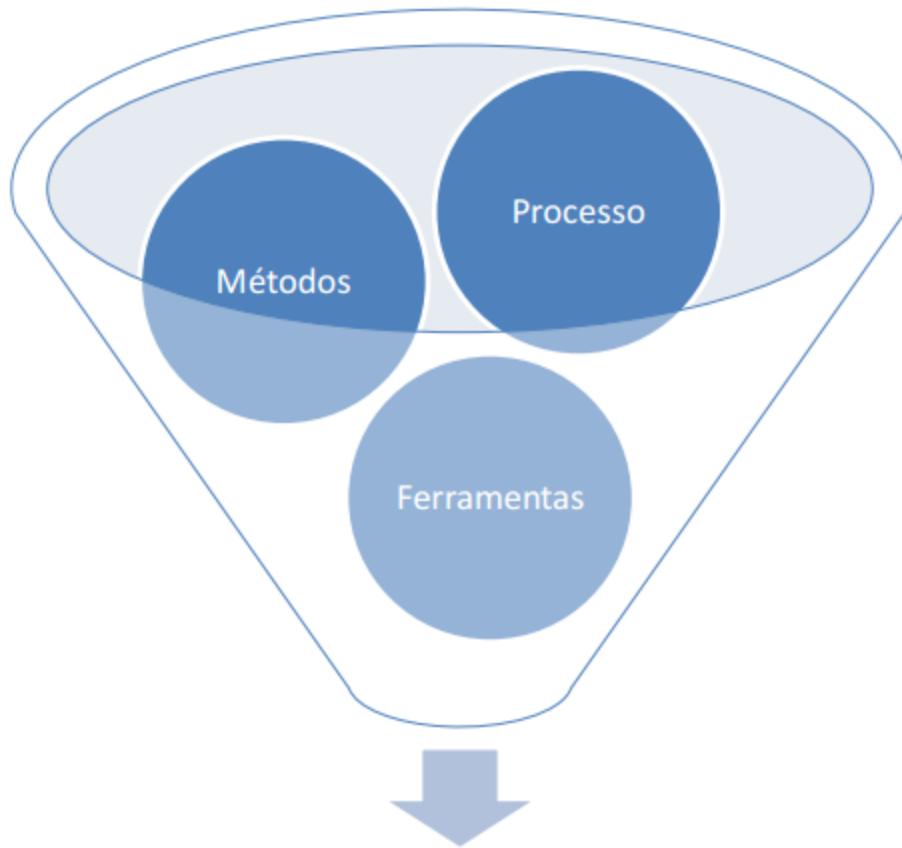
# Características da Engenharia de Software

A Engenharia de Software se refere a software (sistemas) desenvolvidos por grupos ao invés de indivíduos

usa princípios de engenharia ao invés de arte,  
e

inclui tanto aspectos técnicos quanto não  
técnicos

# Elementos da Eng. Software



Engenharia de Software

# Elementos da Eng. Software

## Processo

- Define os passos gerais para o desenvolvimento e manutenção do software
- Serve como uma estrutura de encadeamento de métodos e ferramentas

## Métodos

- São os “how to’s” de como fazer um passo específico do processo

## Ferramentas

- Automatizam o processo e os métodos

# Elementos da Eng. Software

Cuidado com o  
“desenvolvimento guiado  
por ferramentas”

- É importante usar a ferramenta certa para o problema
- O problema não deve ser adaptado para a ferramenta disponível

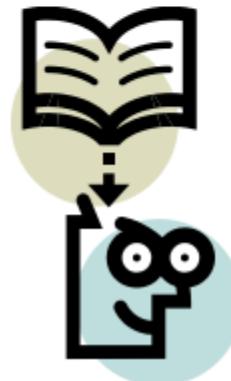


“Para quem tem um martelo,  
tudo parece prego”

# Processos implícitos x explícitos

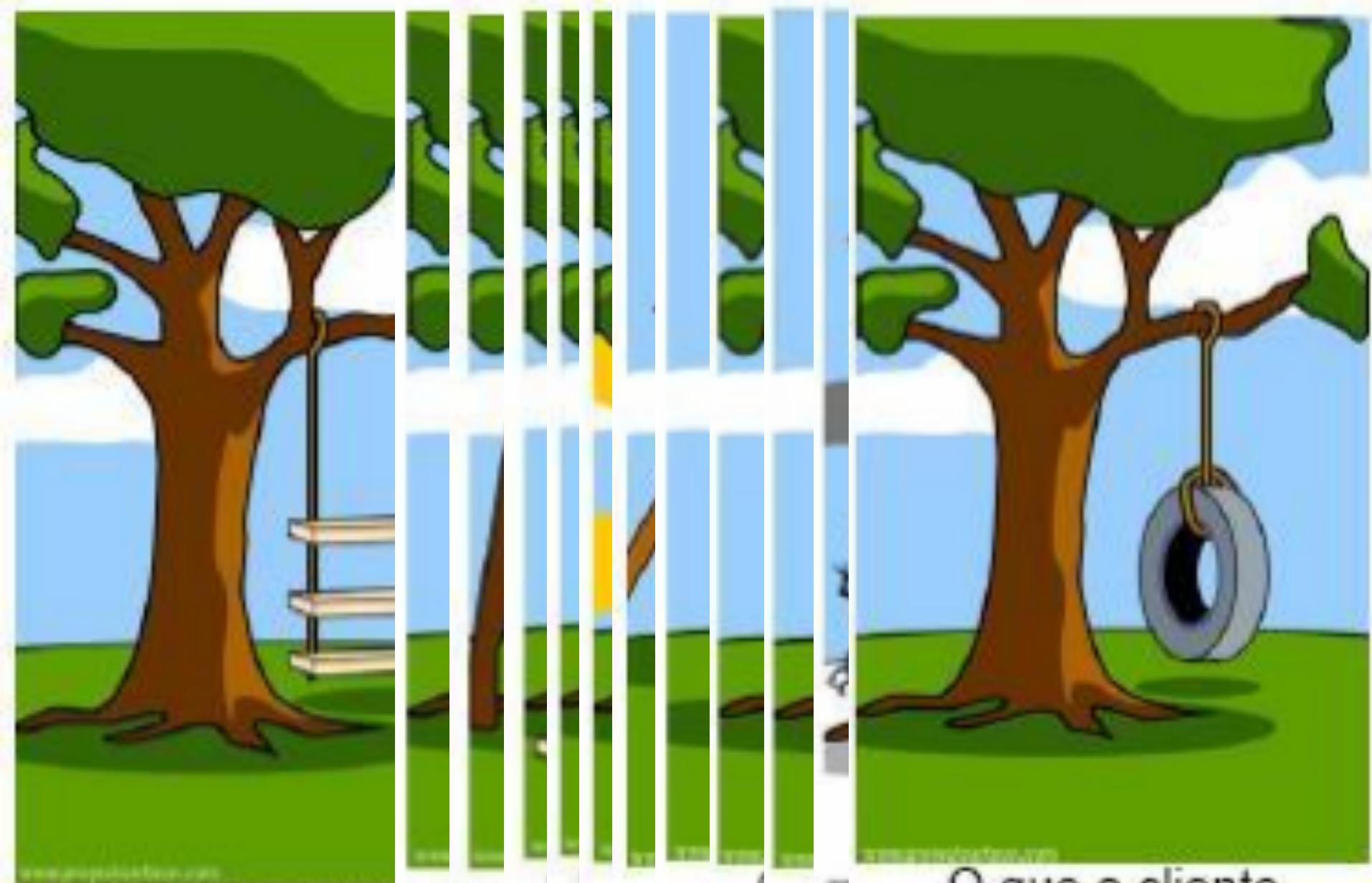
Lembrem-se: Processos sempre existem, seja de forma implícita ou explícita!

- Processos implícitos são difíceis de serem seguidos, em especial por novatos
- Processos explícitos estabelecem as regras de forma clara



# Escopo da Eng. Software

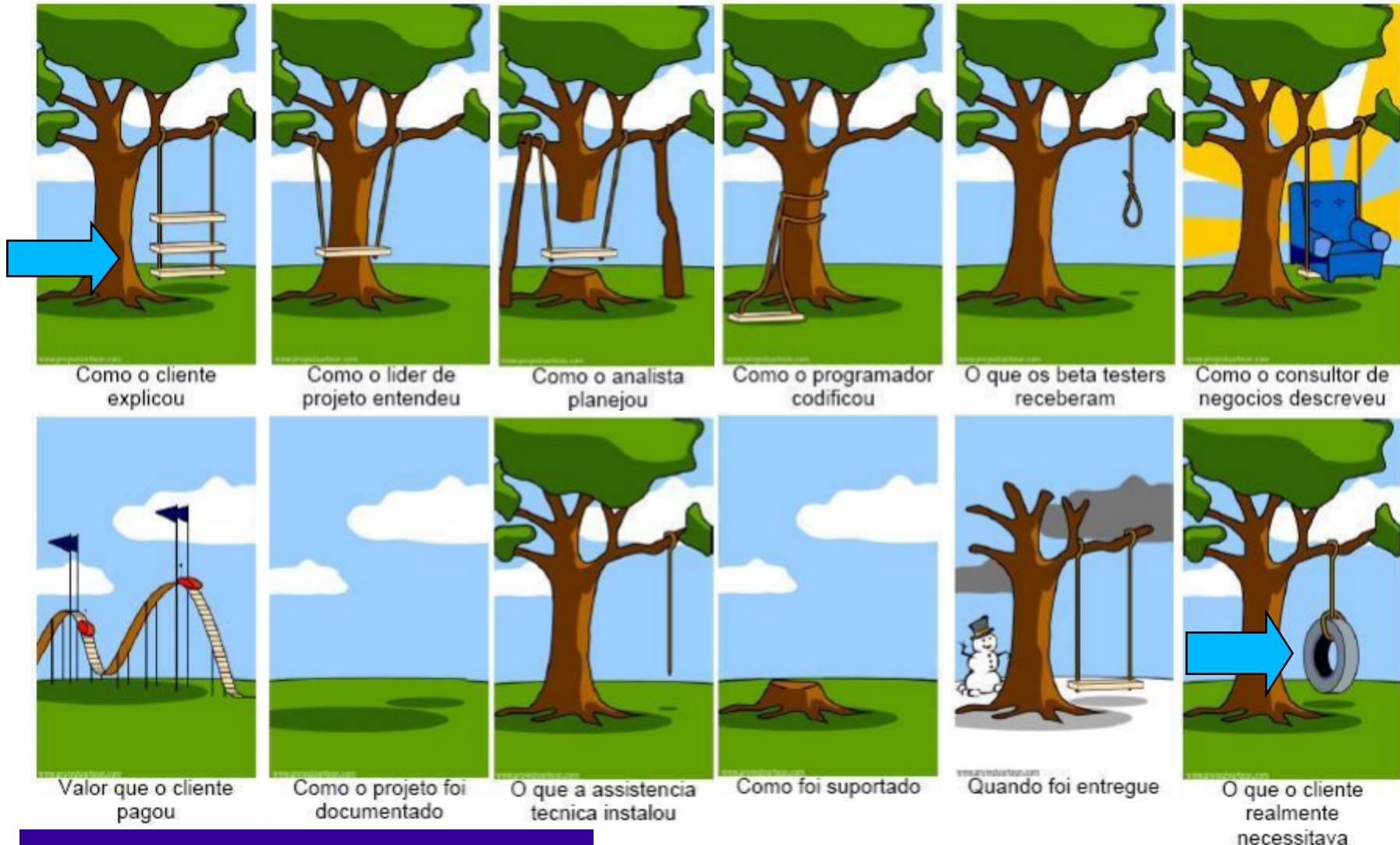




Como o cliente  
explicou

O que o cliente  
realmente  
necessitava

# Comunicação em projetos de software



# Comunicação em projetos de software



Como o cliente explicou



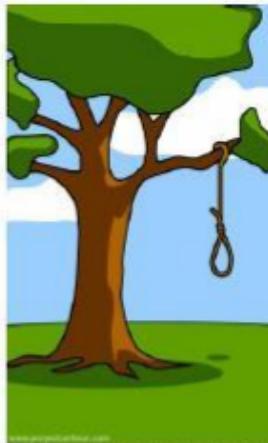
Como o líder de projeto entendeu



Como o analista planejou



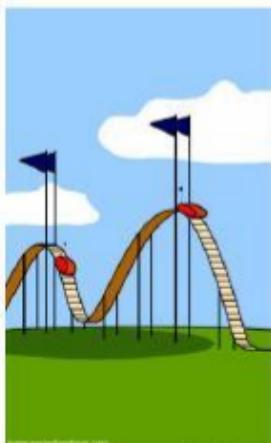
Como o programador codificou



O que os beta testers receberam



Como o consultor de negócios descreveu



Valor que o cliente pagou



Como o projeto foi documentado



O que a assistencia técnica instalou



Como foi suportado



Quando foi entregue



O que o cliente realmente necessitava

# Evolução dos requisitos



O que é necessário fazer



O que os usuários querem



O que os usuários pedem



O que os desenvolvedores entendem



O que acaba sendo feito...

# Tabela de requisitos

Número de ordem	Caso de uso	Descrição
1	Abertura do Caixa	Passagem para o Modo de Venda, liberando assim o caixa da mercearia para a Operação de Venda. O Gerente da mercearia deve informar o valor inicial deste caixa.
2	Emissão de Nota Fiscal	Emissão de Nota Fiscal para o cliente da mercearia (extensão da Operação de Venda).
3	Emissão de Relatórios	Emissão de relatórios com as informações das bases de dados do Merci.
4	Fechamento do Caixa	Totalização das vendas do dia e mudança para o Modo de Gestão.
5	Gestão de Fornecedores	Processamento de inclusão, exclusão e alteração de fornecedores.
6	Gestão de Mercadorias	Processamento de inclusão, exclusão e alteração de mercadorias.
7	Gestão de Pedidos de Compra	Processamento de inclusão, exclusão e alteração de pedidos de compra de mercadorias.
8	Gestão de Usuários	Controle de usuários que terão acesso ao Merci.
9	Gestão Manual de Estoque	Controle manual de entrada e saída de mercadorias.
10	Operação de Venda	Operação de venda ao cliente da mercearia.

# Exercício em Grupo:

Baseado no modelo de desenvolvimento do software para uma biblioteca, identifique a tabela de requisitos para as seguintes etapas:

1)Processo de cadastro de usuários

1)Processo de empréstimo de livros

1)Processo de devolução de livros e colocação do mesmo na posição correta na prateleira.

1)Consulta de disponibilidade de livros via internet.

# Importância da Eng. de Software

Qualidade de software e produtividade garantem:

- Disponibilidade de serviços essenciais
- Segurança de pessoas
- Competitividade das empresas
  - Produtores
  - Consumidores

# Produtividade

Custo de desenvolvimento reduzido

- A empresa consumidora quer investir pouco em software
- A empresa produtora tem que oferecer “software barato”

Tempo de desenvolvimento reduzido

- Suporte rápido às necessidades do mercado

# Software barato?

*Nem tanto resultado de baixos custos de desenvolvimento, mas principalmente da distribuição dos custos entre vários clientes.*

*Reuso, extensibilidade e adaptabilidade são essenciais para viabilizar tal distribuição.*

# Processo de Qualidade

Última palavra para medir a qualidade de um processo: **Satisfação do Cliente**

Outros indicadores importantes

- Qualidade dos produtos gerados
- Custo real do projeto
- Duração real do projeto

# O que é um software de qualidade?

O software que satisfaz os requisitos solicitados pelo usuário. Deve ser fácil de manter, ter boa performance, ser confiável e fácil de usar

Alguns atributos de qualidade

- Manutenibilidade
  - O software deve evoluir para atender os requisitos que mudam
- Eficiência
  - O software não deve desperdiçar os recursos do sistema
- Usabilidade
  - O software deve ser fácil de usar pelos usuários para os quais ele foi projetado

# Qualidade de Software

Correto

- A loja não pode deixar de cobrar por produtos comprados pelo consumidor

Robusto e altamente disponível

- A loja não pode parar de vender

Eficiente

- O consumidor não pode esperar
- A empresa quer investir pouco em recursos computacionais (CPU, memória, rede)

# Qualidade de Software

Amigável e fácil de usar

- A empresa quer investir pouco em treinamento

Altamente extensível e adaptável

- A empresa tem sempre novos requisitos (para ontem!)
- A empresa quer o software customizado do seu jeito (interface, teclado, idioma, moeda, etc.)

Reusável

- Várias empresas precisam usar partes de um mesmo sistema

# Qualidade de Software

Aberto, compatível, de fácil integração com outros sistemas

- A empresa já tem controle de estoque, fidelização, etc.

Portável e independente de plataforma (hw e sw)

- A empresa opta por uma determinada plataforma

Baixo custo de instalação e atualização

- A empresa tem um grande número de PDVs

# A Crise de Software...

25% dos projetos são cancelados  
o tempo de desenvolvimento é bem maior do que o estimado

75% dos sistemas não funcionam como planejado

a manutenção e reutilização são difíceis e custosas

os problemas são proporcionais a complexidade dos sistemas

# Causas da Crise de Software

## Essências

- Complexidade dos sistemas
- Dificuldade de formalização

## Acidentes

- Má qualidade dos métodos, linguagens, ferramentas, processos, e modelos de ciclo de vida
- Falta de qualificação técnica

# Elementos e Atividades

## Elementos

- Modelos do ciclo de vida do software
- Linguagens
- Métodos
- Ferramentas
- Processos

## Atividades

- Modelagem do negócio
- Elicitação de requisitos
- Análise e Projeto
- Implementação
- Testes
- Distribuição
- Planejamento
- Gerenciamento
- Gerência de Configuração e Mudanças
- Manutenção

# Atividades e Artefatos

## Atividades

- Modelagem do negócio
- Elicitação de requisitos
- Análise e Projeto
- Implementação
- Testes
- Distribuição
- Planejamento
- Gerenciamento
- Gerência de Configuração e Mudanças
- Manutenção

## Artefatos

- Plano de Negócios
- Plano de Projeto
- Plano de Riscos
- Documento de Requisitos
- Mapeamentos A&P
- Documento de Caso de Uso
- Documento de Arquitetura
- Classes
- Documento de Testes
- Documento de Validação
- Manual do Sistema

# Linguagem

Notação com sintaxe e semântica bem definidas

- com representação gráfica ou textual

Usada para descrever os artefatos gerados durante o desenvolvimento de software

Exemplos: UML, Java

# Método

Descrição sistemática de como deve-se realizar uma determinada atividade ou tarefa

A descrição é normalmente feita através de padrões e guias

Exemplos: Método para descoberta das classes de análise no RUP.

# Processo

Conjunto de atividades

- bem definidas
- com responsáveis
- com artefatos de entrada e saída
- com dependências entre as mesmas e ordem de execução
- com modelo de ciclo de vida

# Ferramenta CASE

Provê suporte computacional a um determinado método ou linguagem

Ambiente de desenvolvimento: conjunto de ferramentas integradas (CASE)

Exemplos: Rational Rose, JBuilder

# Processo de software

Um conjunto de atividades cujo objetivo é o desenvolvimento ou a evolução do software

Conjunto coerente de atividades para especificação, projeto, implementação e teste de sistemas de software

# O ciclo de vida do software

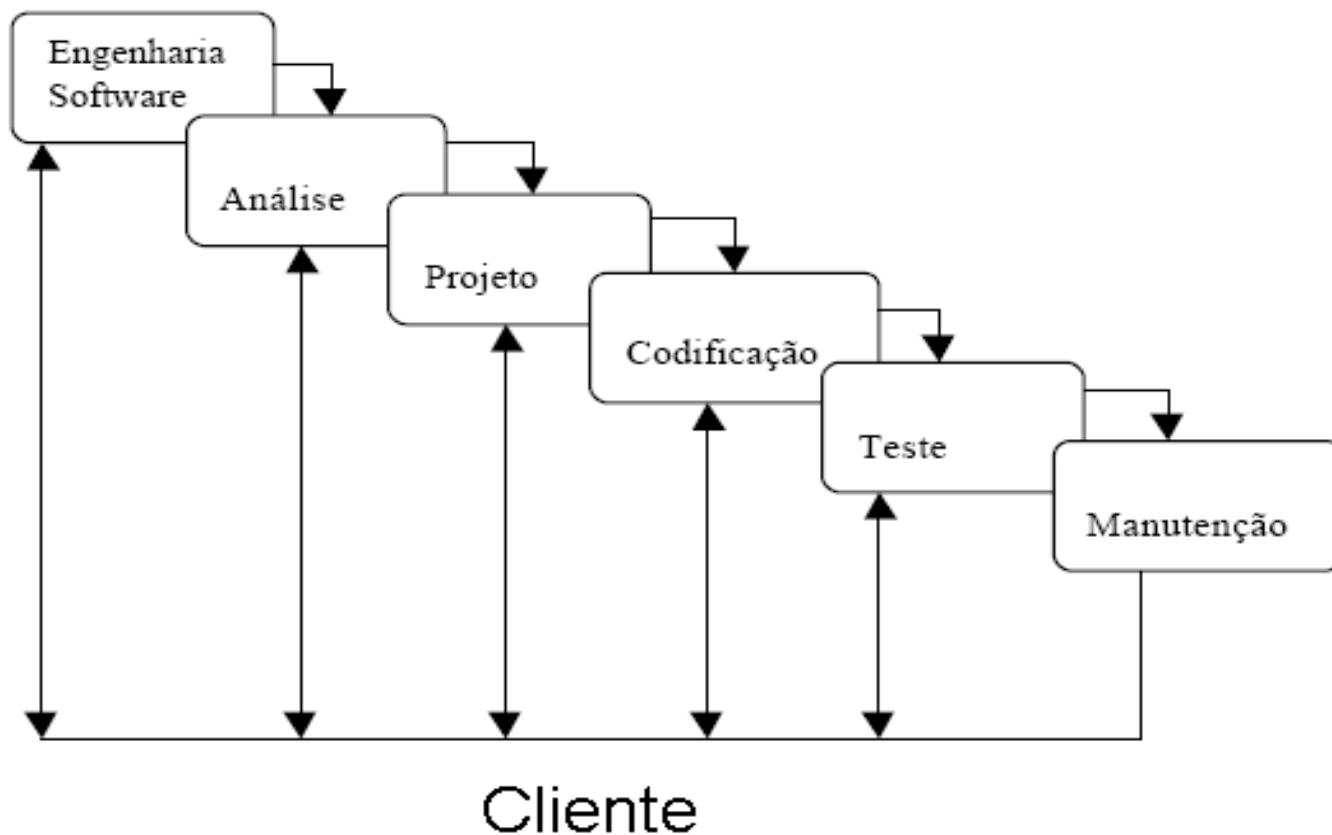
Objetivos de se definir o ciclo de vida do software:

- Definir atividades;
- Definir consistência entre vários projetos da empresa, todos os projetos executados da mesma forma;
- Introduzir pontos de verificação para controle gerencial de decisões;
- Atingir níveis mais apropriados de maturidade na gerência do processo de desenvolvimento do software.

O processo definido pelo Ciclo de Vida é conhecido como Paradigma da Engenharia de Software.

# O ciclo de vida do software

## O CICLO DE VIDA CLÁSSICO



# O ciclo de vida do software

Percepção da necessidade																
Cliente	Projeto	Concepção														
		Elaboração														
		Desenvolvimento	Desenho inicial		Desenho detalhado											
			Construção	Codificação												
				Testes de unidade												
			Testes alfa													
		Implantação														
Adquiação ao ambiente																
Aplicação dos resultados																

*Esquema simplificado do ciclo de vida do software*

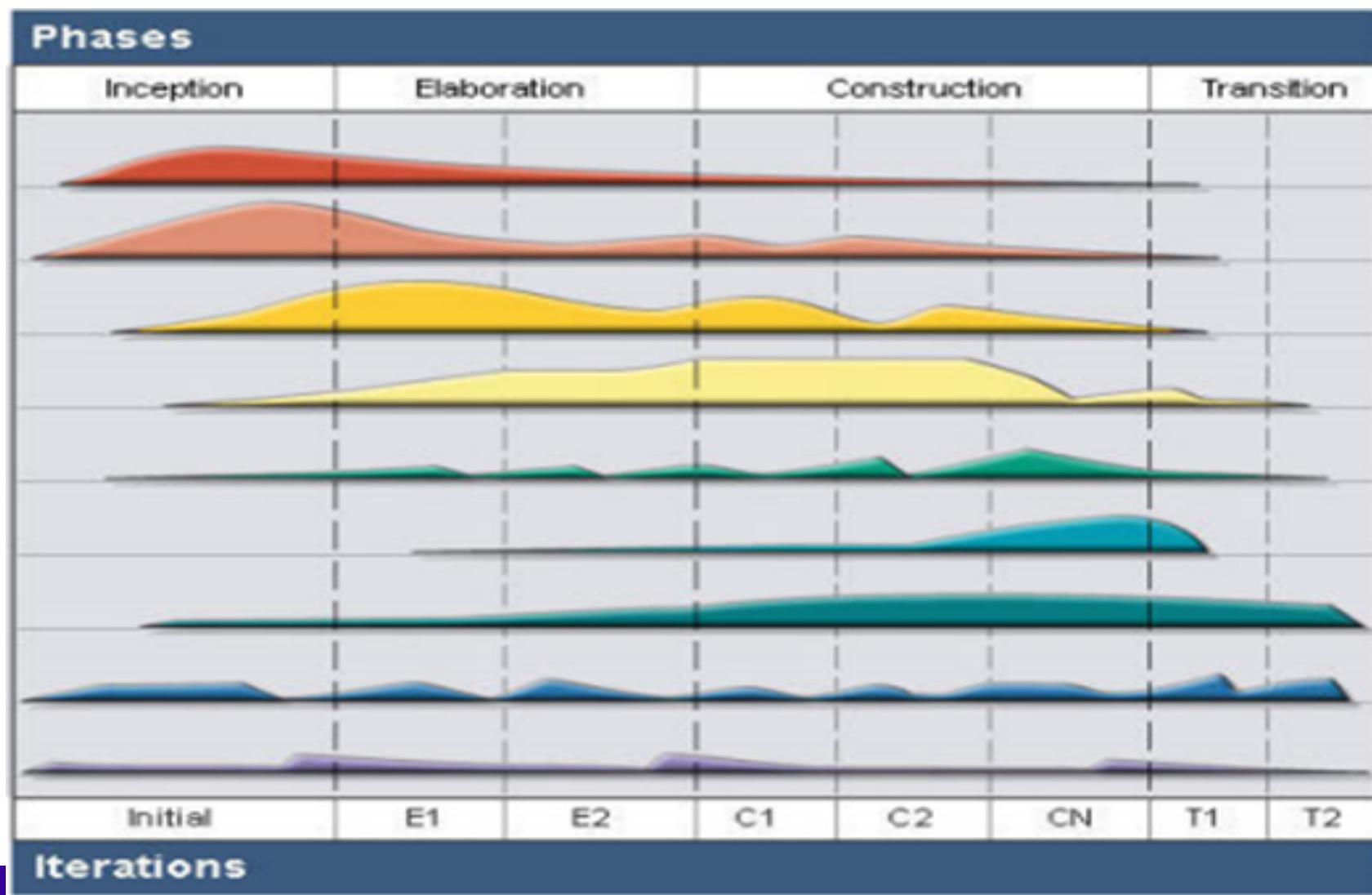
# Metodologia RUP

IBM (2006)

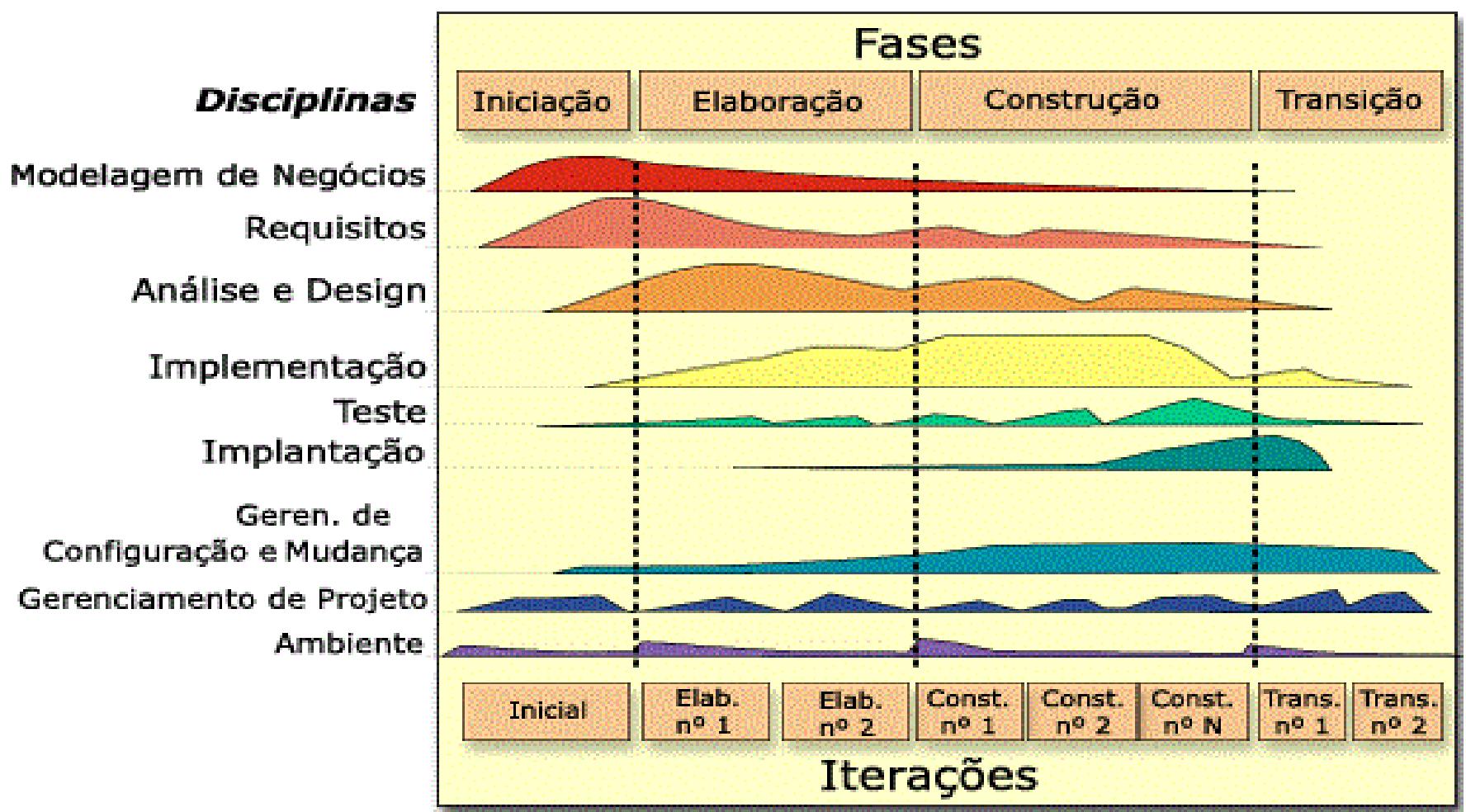
# RUP (Rational Unified Process)

- Ele é composta por nove disciplinas, as quais estão dispostas em quatro fases (IBM, 2004).
- Além de possuir processos que podem ser adaptados conforme necessidade, o RUP é composto pelas melhores práticas de desenvolvimento de *software*, as quais são (KRUCHTEN, 2001):
  - Desenvolvimento do *software* de forma iterativa;
  - Gerenciamento de requisitos;
  - Utilização de arquiteturas baseadas em componentes;
  - Constante verificação da qualidade do *software*;
  - Controle das mudanças de *software*.

# Diagrama do RUP



# Disciplinas X Fases



# Fases do RUP

- Segundo Martins (2007), as fases são constituídas de objetivos específicos:
  - Concepção;
  - Elaboração;
  - Construção;
  - Transição.

# 1<sup>a</sup> Fase - Concepção

- Para Piske (2003) nesta etapa é realizada a concepção inicial do sistema, sendo organizadas discussões sobre o problema, a definição do escopo do projeto, estimativas de recursos e ferramentas necessárias para a execução do projeto, *etc.* Com o intuito de obter um resultado satisfatório para ambas as partes e é necessário que a fase de concepção seja bem definida e documentada para a continuidade do desenvolvimento do projeto.

## 2<sup>a</sup> Fase - Elaboração

- Segundo Piske (2003), o propósito desta fase é analisar o domínio do problema, desenvolver o plano de projeto, estabelecer a fundação arquitetural e eliminar os elementos de alto risco.
- Os elementos de risco a serem analisados, nesta fase, são os riscos de requerimentos, tecnológicos (referentes à capacidade das ferramentas disponíveis), de habilidades (dos integrantes do projeto) e políticos.

## 3<sup>a</sup> Fase - Construção

- De acordo com Sommerville (2007) a fase de construção está essencialmente relacionada ao projeto, programação e testes de sistema. As partes do sistema são desenvolvidas paralelamente e integradas durante essa fase.
- Para Piske (2003) é constituída pelo o desenvolvimento de telas, lógicas, relações, banco de dados e testes e utiliza alguma notação definida pela UML. É a fase em que o *software* é efetivamente programado através de códigos fontes.

## 4<sup>a</sup> Fase - Transição

- Segundo Laudon e Laudon (2004), na fase de transição devem ser feitos testes exaustivos e minuciosos para se assegurar que o sistema está produzindo resultados sólidos e satisfatórios.
- Para Sommerville (2007), a fase de transição é a fase final do RUP e está relacionada à transferência do sistema da comunidade de desenvolvimento para a comunidade dos usuários e com a entrada do sistema em funcionamento no ambiente real.

## 4<sup>a</sup> Fase - Transição

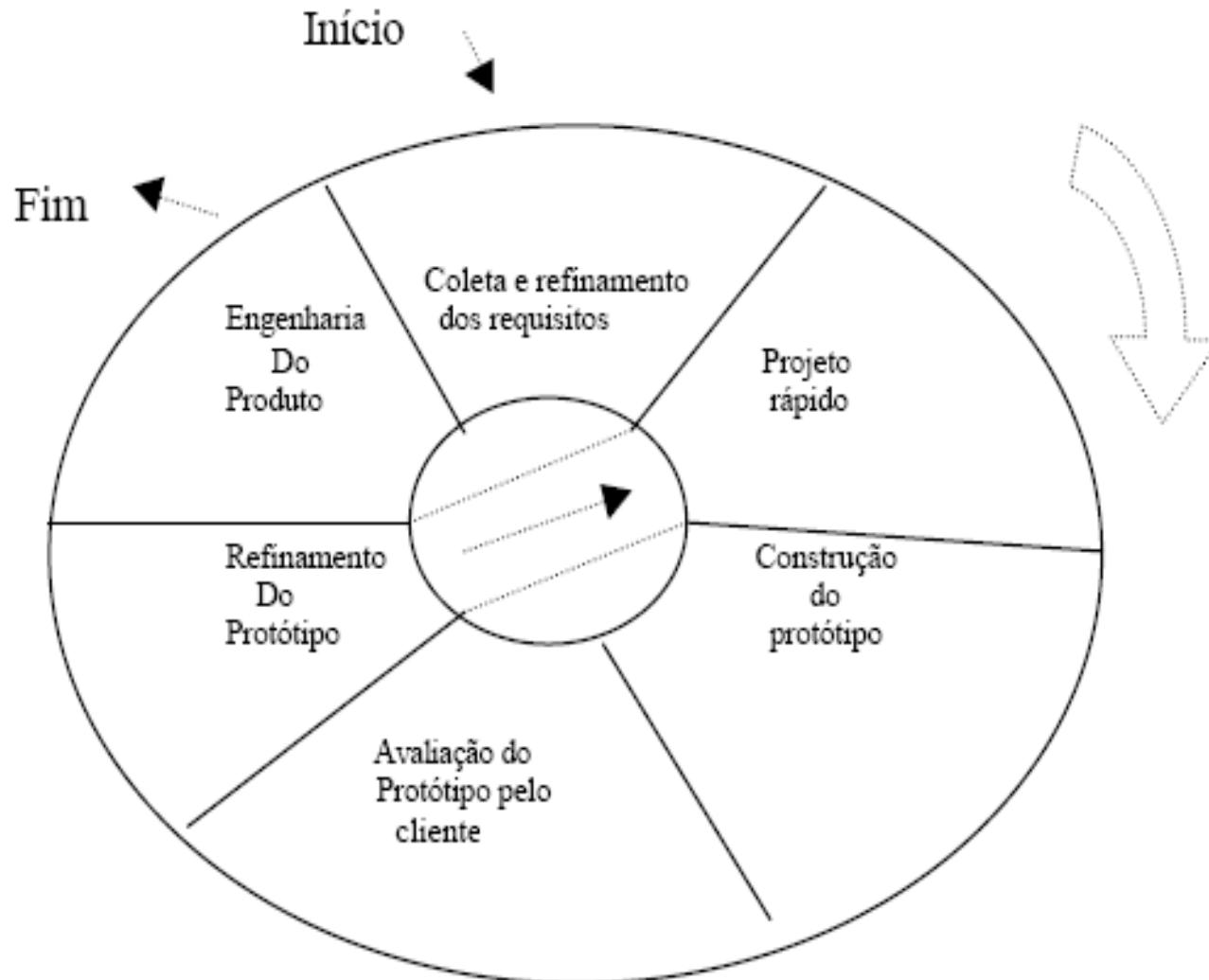
- De acordo com Martins (2010) o objetivo deste processo é disponibilizar o sistema para os usuários, o que inclui:
  - Testes do sistema no ambiente de produção;
  - Empacotamento de *software* para distribuição;
  - Distribuição do *software*;
  - Instalação do *software*;
  - Treinamento dos usuários e equipe comercial;
  - Migração de dados para novo sistema.

# Prototipação

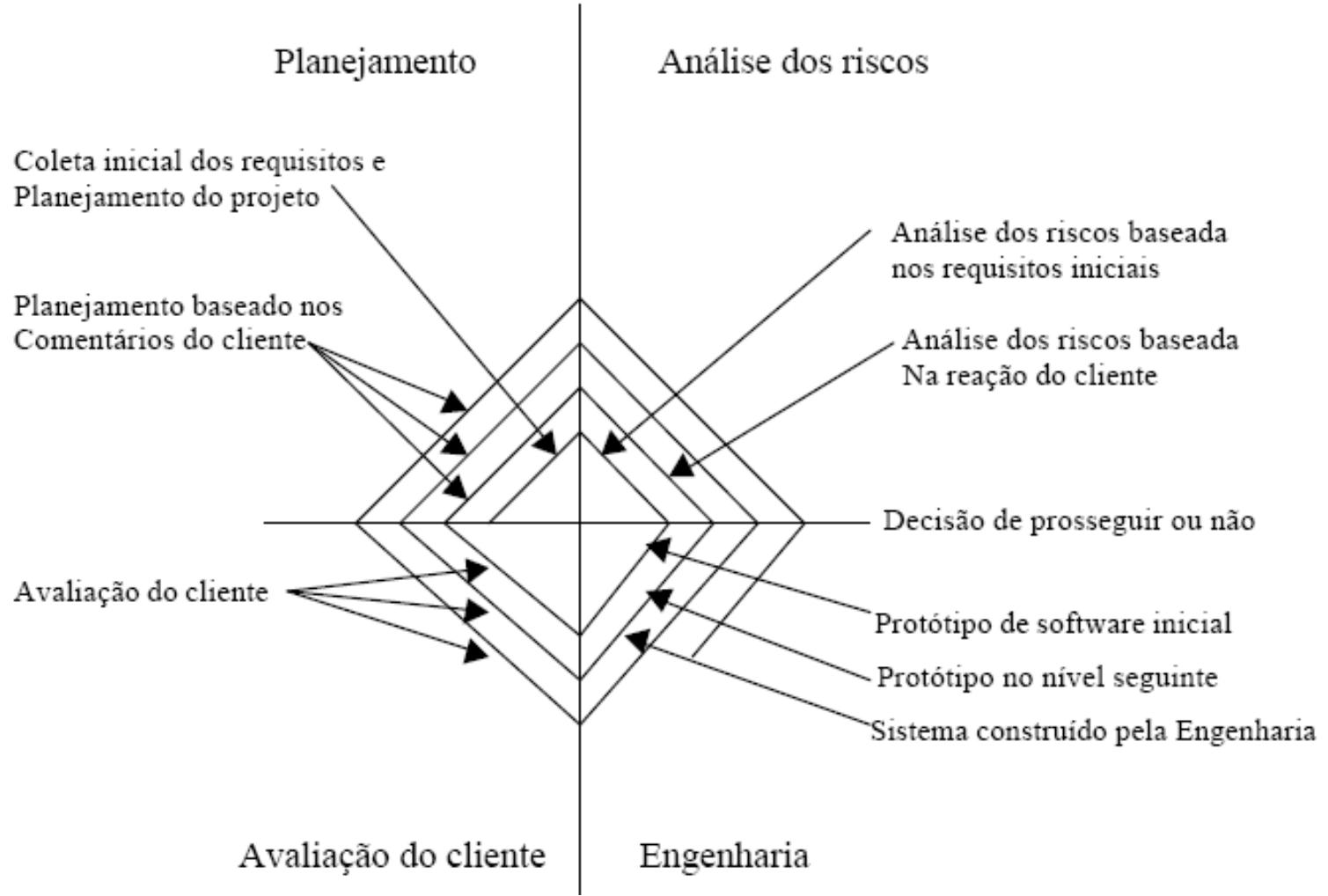
Quando logo no início do projeto o cliente não sabe definir com muita clareza os requisitos para o novo sistema, torna-se benéfico o uso da prototipação.

A prototipação permite que seja criado um modelo do software que será implementado. Este modelo pode ser um protótipo em papel ou em PC, que mostra a interação homem-máquina: um programa que implementa um conjunto de funções exigidas pelo sistema ou um programa existente que executa parte das funções requeridas. A seqüência de eventos para o paradigma de prototipação é apresentada conforme diagrama a seguir.

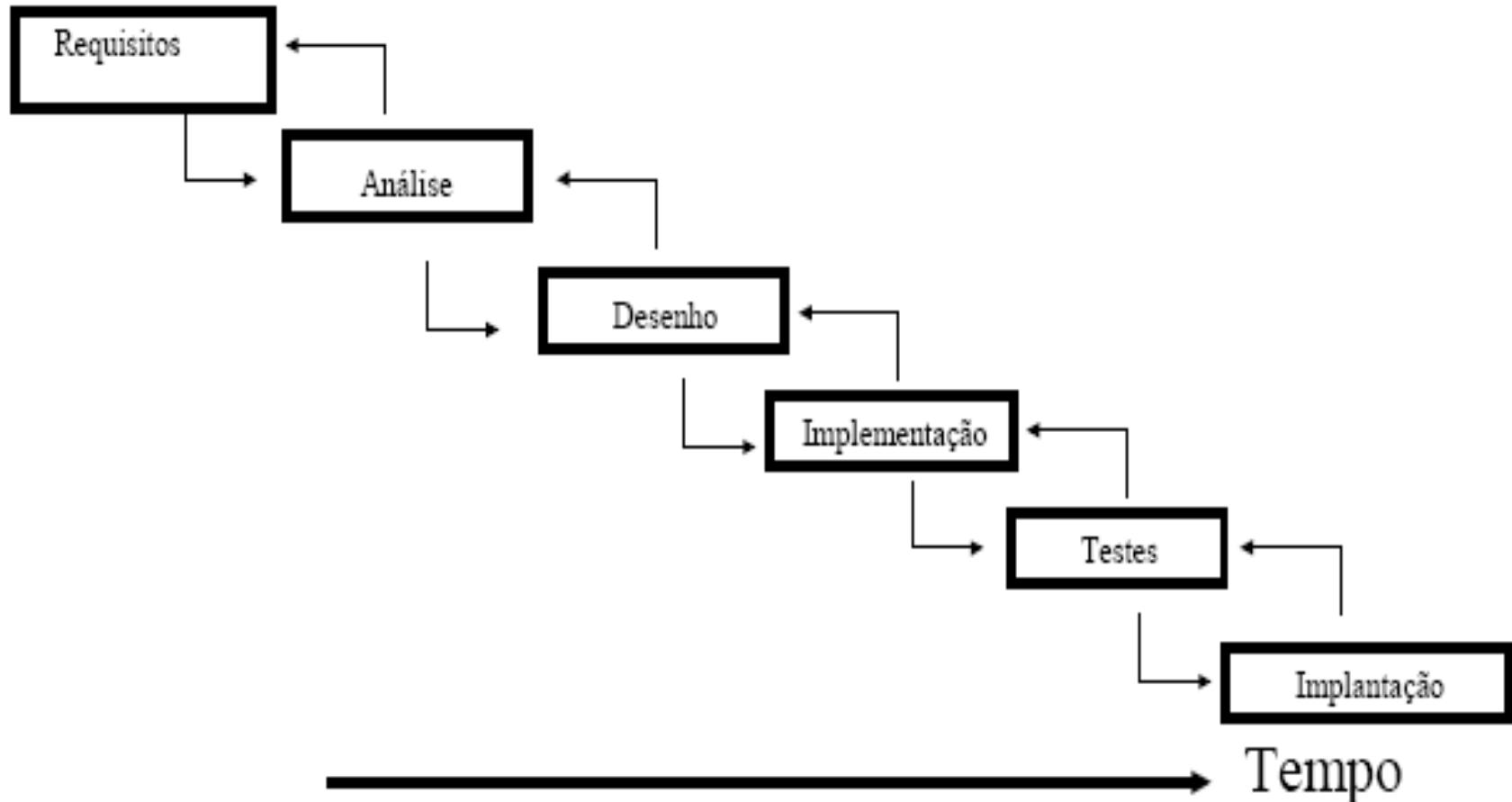
# Prototipação – Modelo padrão



# Prototipação - Modelo Espiral



# Processo de desenvolvimento de software



# Níveis do desenvolvimento do software

- **Planejamento:** na etapa de planejamento é onde deve ser desenvolvido um plano inicial de desenvolvimento, levando em consideração questões como definição da abrangência do sistema, missão e objetivos do sistema, cronogramas de desenvolvimento, análise custo X benefício, levantamento inicial de informação etc.
- **Análise:** também chamada de análise de requisitos, é onde deve se obter um claro entendimento sobre o sistema. A análise proporciona a base para uma boa implementação do software. Nesta etapa são construídos os modelos do sistema.

# Níveis do desenvolvimento do software

- **Projeto:** também chamada de especificação do projeto, é onde propomos uma arquitetura de implementação para o software, que atenda aos requisitos do sistema identificados na análise. Aqui passamos a nos preocupar com o software. Os algoritmos dos programas a serem implementados são construídos nesta fase.
- **Implementação:** a etapa de implementação é onde os programas são efetivamente construídos, a partir da arquitetura feita na etapa anterior. Nesta etapa é onde a atividade de codificação ocorre de forma massiva.

# Níveis do desenvolvimento do software

- **Teste:** nesta etapa todos os programas construídos serão testados de forma exaustiva. Existe uma grande variedade de testes que são realizados, indo desde o teste unitário dos módulos de programas até o teste de integração de todo o sistema de software.
- **Manutenção:** é onde ocorre ajustes do software implementado, que podem ser ocasionados por vários motivos: erros de projetos identificados após a implementação e o teste do software, inovações tecnológicas, evolução do sistema etc.

# Atividades do desenvolvimento

## ATIVIDADE 1: O ESTUDO INICIAL

Também conhecido como ESTUDO DE VIABILIDADE ou LEVANTAMENTO.

- Identificar usuários responsáveis e montar o escopo do sistema – entrevistas;
- Identificar deficiências do Sistema Atual;
- Estabelecer metas e objetivos para um novo sistema – reutilização.
- Determinar se é possível automatizar o sistema e sugerir alternativas – (adquirir ou desenvolver?);
- Preparar previsão inicial para o projeto.

# Atividades do desenvolvimento

## ATIVIDADE 2: ANÁLISE DE SISTEMAS

- Transformar critérios do usuário e previsão em ESPECIFICAÇÃO DE REQUISITOS;
- Modelagem Essencial do Sistema:
  - Modelo Ambiental
  - Modelo Comportamental
    - . Eventos
    - . Componente Domínio do Problema

# Atividades do desenvolvimento

## ATIVIDADE 3: PROJETO

- Componente Interface Humana
- Componente Gerenciador de Tarefas
- Componente Gerenciador de Dados

## ATIVIDADE 4: CONVERSÃO DE BANCO DE DADOS

- Reaproveitar base de dados já existente, se possível.

# Atividades do desenvolvimento

## ATIVIDADE 5: IMPLEMENTAÇÃO

- Programação

## ATIVIDADE 6: GERAÇÃO DE TESTES DE ACEITAÇÃO

- Baseado nos requisitos especificados durante a análise, estabelecer casos de testes para o sistema.

## ATIVIDADE 7: DESCRIÇÃO DE PROCEDIMENTOS

- Manual do usuário

## ATIVIDADE 8: INSTALAÇÃO

# Desenvolvimento de Software e Reusabilidade

Fatores como o aumento do tamanho e complexidade, bem como, a necessidade de adaptação rápida a novos ambientes têm contribuído para a elevação do custo de desenvolvimento e manutenção de software. A técnica de construção de sistemas a partir de componentes reusáveis é a solução indicada para tratar este problema.

Um componente de software reusável é aquele produzido com a finalidade de ser utilizado em diferentes aplicações. Componentes reusáveis podem ser gerados a partir de qualquer combinação de produtos de engenharia de software, incluindo requisitos, especificações, projetos e produtos de implementação (código, testes e documentação).

# Desenvolvimento de Software e Reusabilidade

*Reuso de software é a aplicação de um componente reusável em mais de um Sistema de aplicação.*

As melhores organizações de PD estão conseguindo níveis de reusabilidade de 70% a 80%. Desta forma, quando se defronta com a tarefa de construir um sistema que exige 10.000 linhas de código, apenas 2.000 a 3.000 são escritas e o restante é proveniente de uma biblioteca de componentes reusáveis.

# Benefícios da Reusabilidade

A) Custos Reduzidos: A redução significativa de custos pode-se sentir em todas as fases do ciclo de vida do software. Por exemplo, durante a manutenção, os custos são reduzidos porque os engenheiros estão familiarizados com requisitos comuns, projetos, arquiteturas e componentes.

A) Qualidade Ampliada: Não importa quão extensivamente um sistema é testado e analisado, pois alguns erros só serão identificados quando o mesmo entra em processo de operação. Desde que, componentes reusáveis façam parte de muitos sistemas, erros identificados em um sistema podem ser utilizados para corrigir outros sistemas desenvolvidos ou em desenvolvimento.

# Benefícios da Reusabilidade

- C) Tarefas Reduzidas: Sistematicamente, o reuso em domínios específicos significa o reuso de requisitos, especificações, projetos e produtos de implementação. Isso permite o desenvolvimento mais rápido que o convencional.
- D) Aumento de Produtividade: Desde que existam ferramentas apropriadas para localizar componentes reusáveis em uma biblioteca, esta técnica permite construir sistemas com esforços reduzidos.

# Exercício em Grupo:

- 1) O software é o fator de diferenciação de muitos produtos e sistemas baseados em computador. Apresente exemplos de dois ou três produtos e de pelo menos um sistema em que o software, não o hardware, é o elemento que faz a diferença.
- 2) Nas décadas de 1950 e 1960, a programação de computadores era uma forma de arte aprendida num ambiente semelhante ao de aprendizes. Como os primórdios afetaram as práticas de desenvolvimento de software atuais?
- 3) A engenharia de software auxiliada por computador é uma indústria crescente. Pesquise um produto CASE comercialmente disponível e apresente uma comparação usando critérios que você desenvolve.
- 4) Pesquise a mídia popular (jornais, revistas e televisão que estejam dirigidos para as massas) ao longo dos últimos 12 meses e descubra pelo menos uma grande reportagem em que o software era o tema em questão. Anote quaisquer erros na apresentação.

# **Diagrama de Fluxo de Dados (DFD)**

# Introdução

É amplamente usado para definir entradas, processos e saídas de sistemas

- Elabora uma modelagem lógica do sistema, englobando os diversos processos que envolvem o sistema, os diversos dados e informações que circulam pelo sistema e as diversas entidades envolvidas com o sistema.

# Componentes

- Entidade Externa
- Fluxo de Dados
- Processo
- Depósito de Dados

# Entidade Externa

- Agente que envia ou recebe dados do sistema, também chamada de fonte de dados e/ou destino de informações, sendo considerada um ser externo ao estudo do projeto.
- Deve ser nomeada com um substantivo.

# Fluxo de Dados

- Representa a troca de dados ou informações entre uma **entidade externa e um processo**, ou entre um **processo e um depósito de dados**.
- Deve ser nomeada com um substantivo.

# Processo

- Representa um processo ou trabalho que é executado internamente ao sistema.
- Deve ser numerado (1,2,..., n)
- Deve ser nomeado com um verbo no infinitivo.

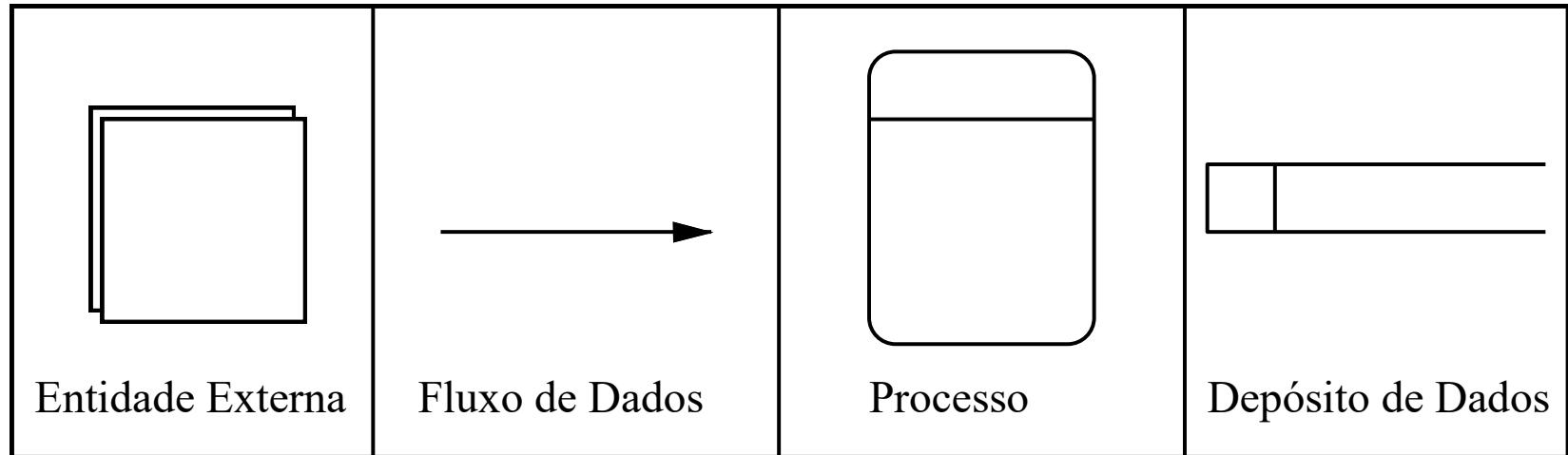
# Depósito de Dados

- Armazena informações e dados do sistema, sendo futura referência para a modelagem de dados do sistema.
- Deve ser numerado ( $d_1, d_2, \dots, d_n$ )
- Deve ser nomeada com um substantivo.

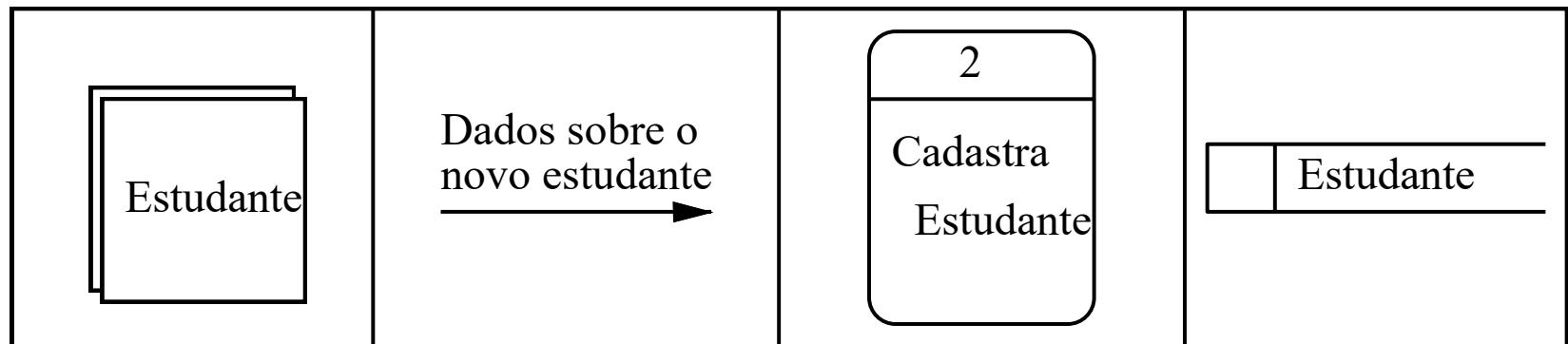
# Regras Básicas

- Todo processo e todo depósito de dados devem possuir ao menos um fluxo de entrada **e** um fluxo de saída.
- Toda entidade externa deve possuir um fluxo de entrada **ou** um fluxo de saída.

# Simbologia



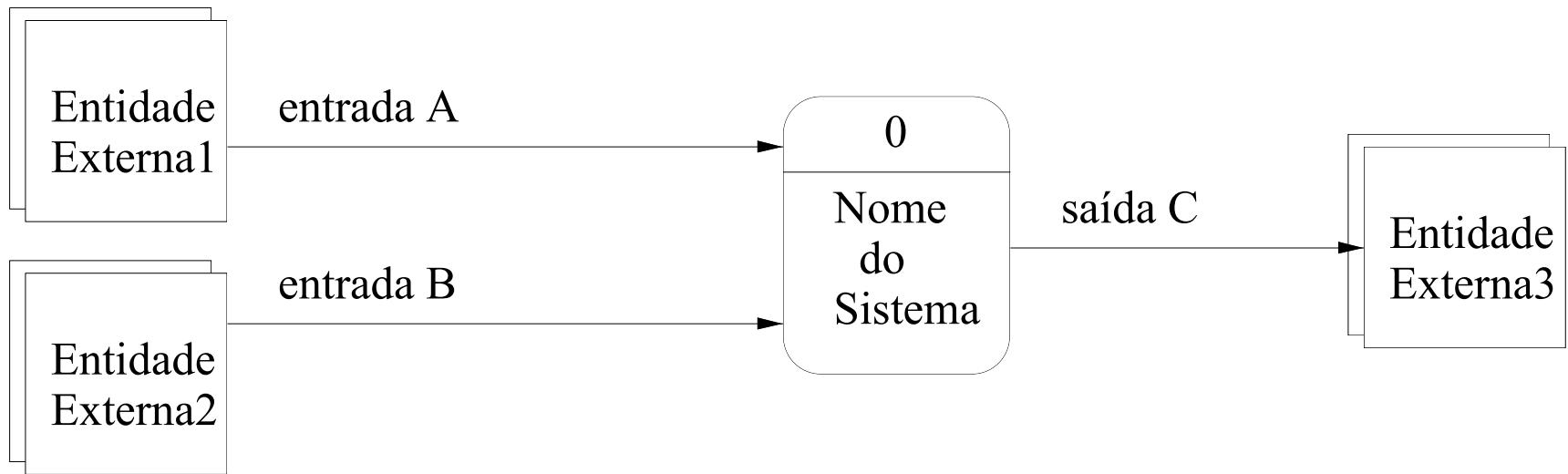
Exemplo



# Diagrama de Contexto

- É o nível mais alto do DFD contendo um único processo, as entidades externas e todos os fluxos que fluem entre as entidades externas e os diversos processos do DFD. Não incluem os depósitos de dados.
- Os fluxos do Diagrama de Contexto devem ter o mesmo nome dos fluxos no DFD.

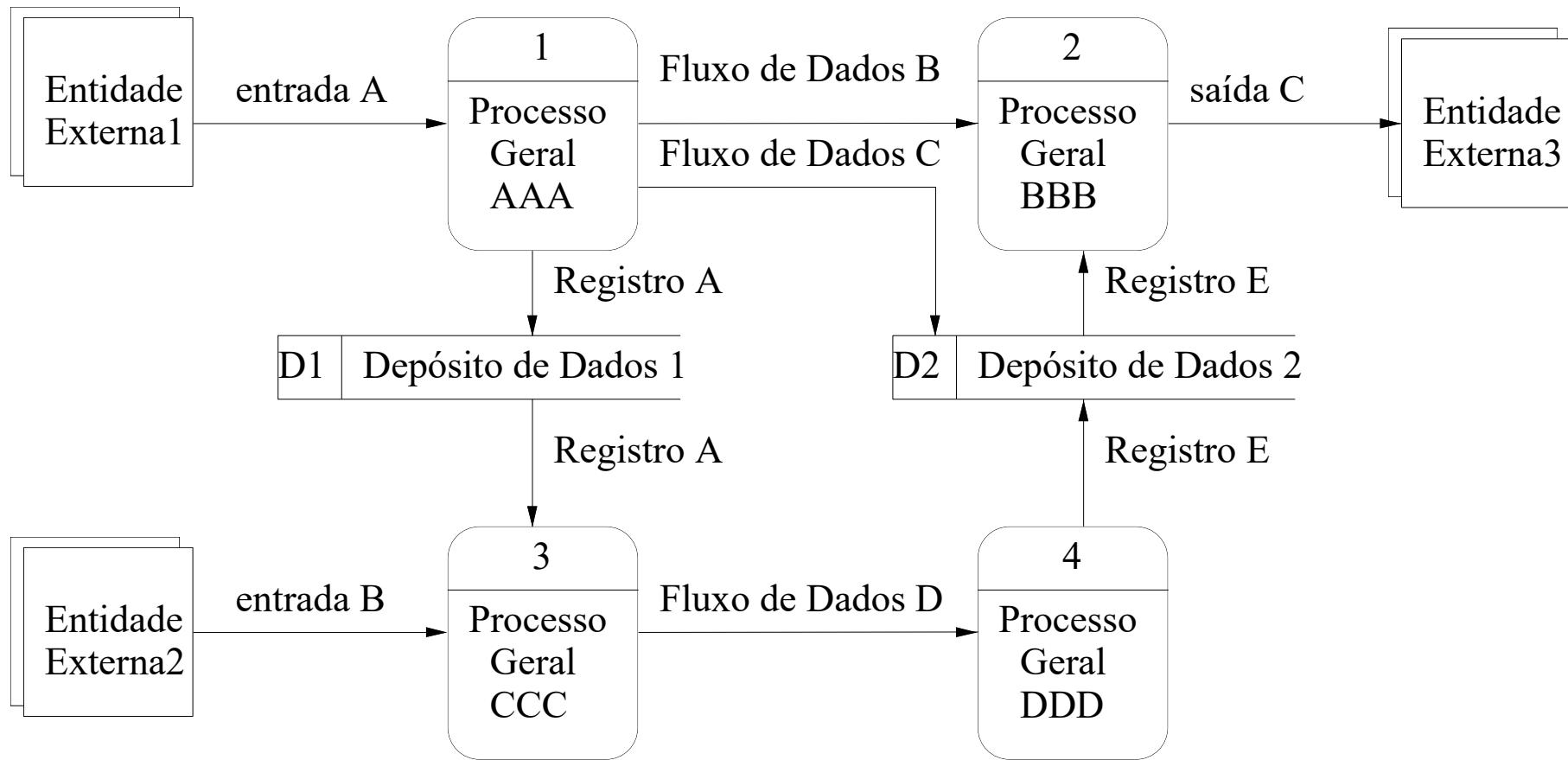
# Diagrama de Contexo (Exemplo)



# Diagrama de Nível 0

- Diagrama detalhando o diagrama de contexto.
- Deve conter de 3 a 9 processos (facilitando o processo de leitura do mesmo).
- Todos os depósitos de dados e entidades externas devem estar inclusas neste nível.

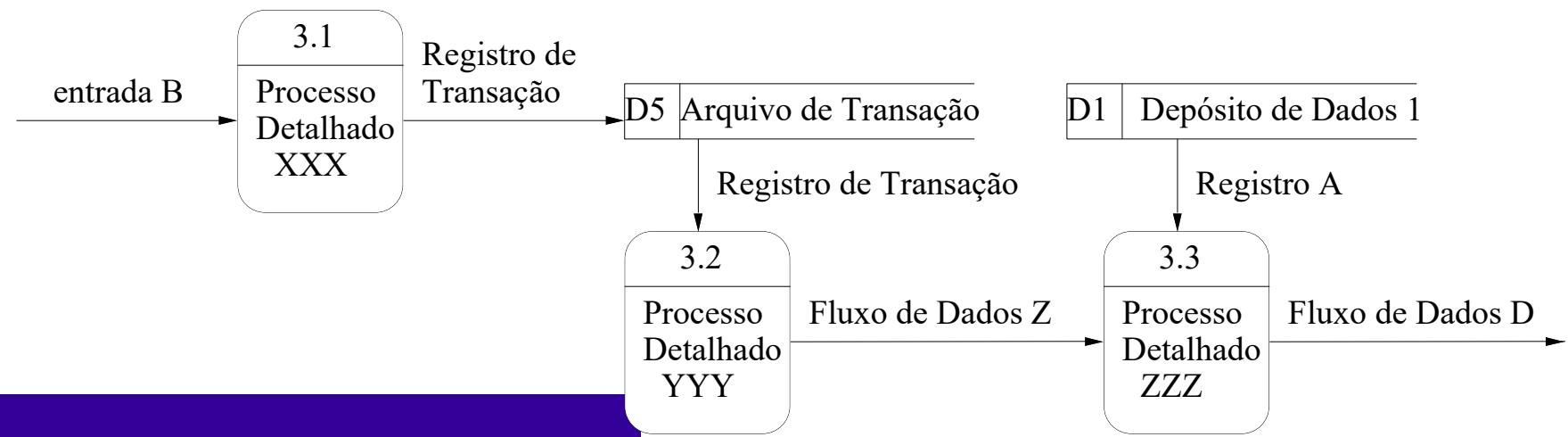
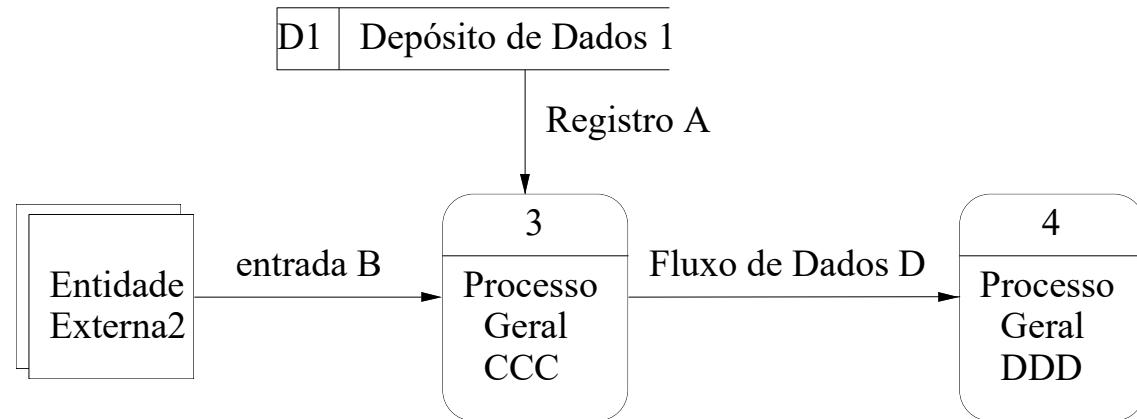
# Diagrama de Nível 0 (Exemplo)



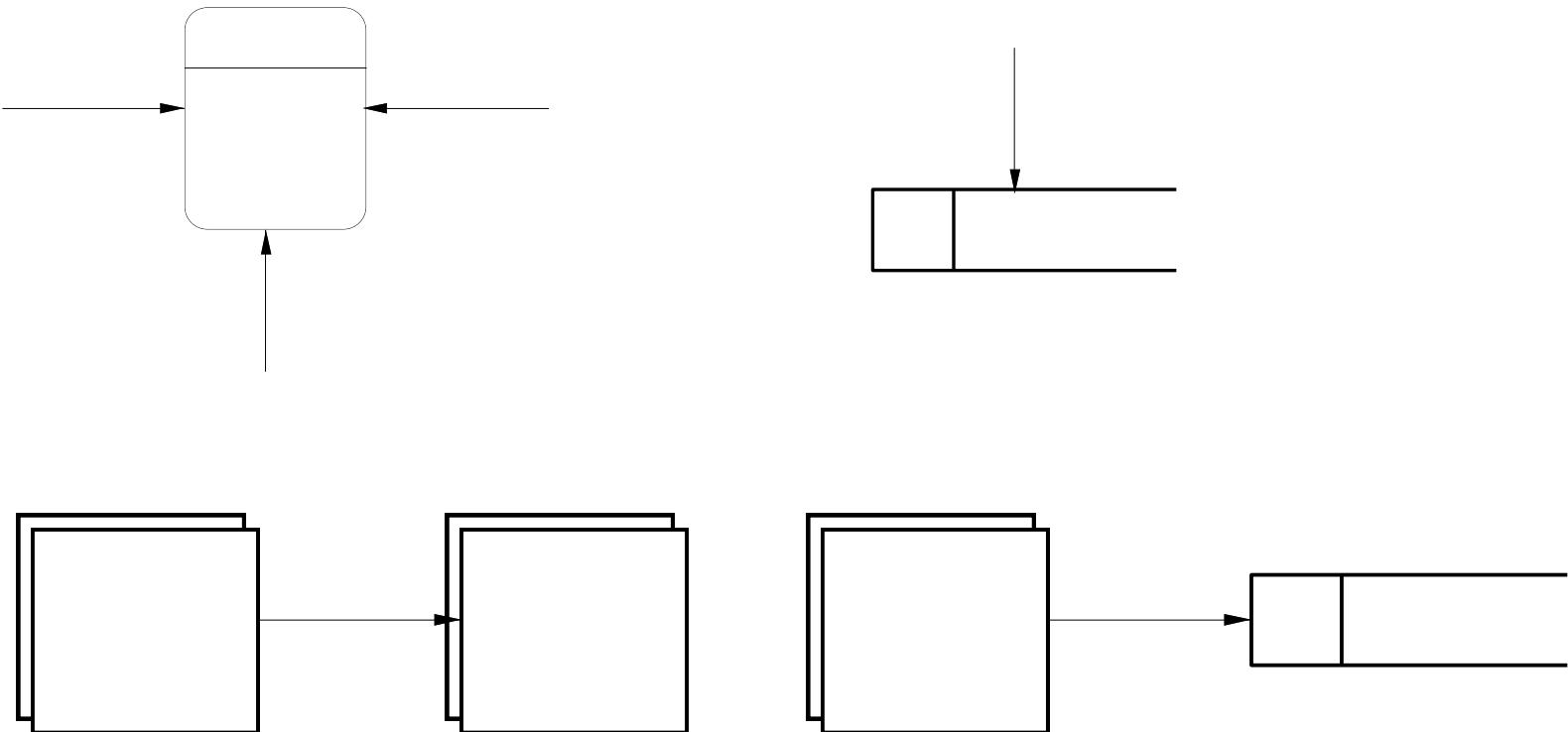
# Diagrama Filho

- Um diagrama filho é um detalhamento de um processo do Diagrama Nível 0, devendo ser gerado quando um processo for complexo.
- Deve englobar todos os fluxos que entram ou saem do processo pai, além de seus próprios fluxos.

# Diagrama Filho (Exemplo)



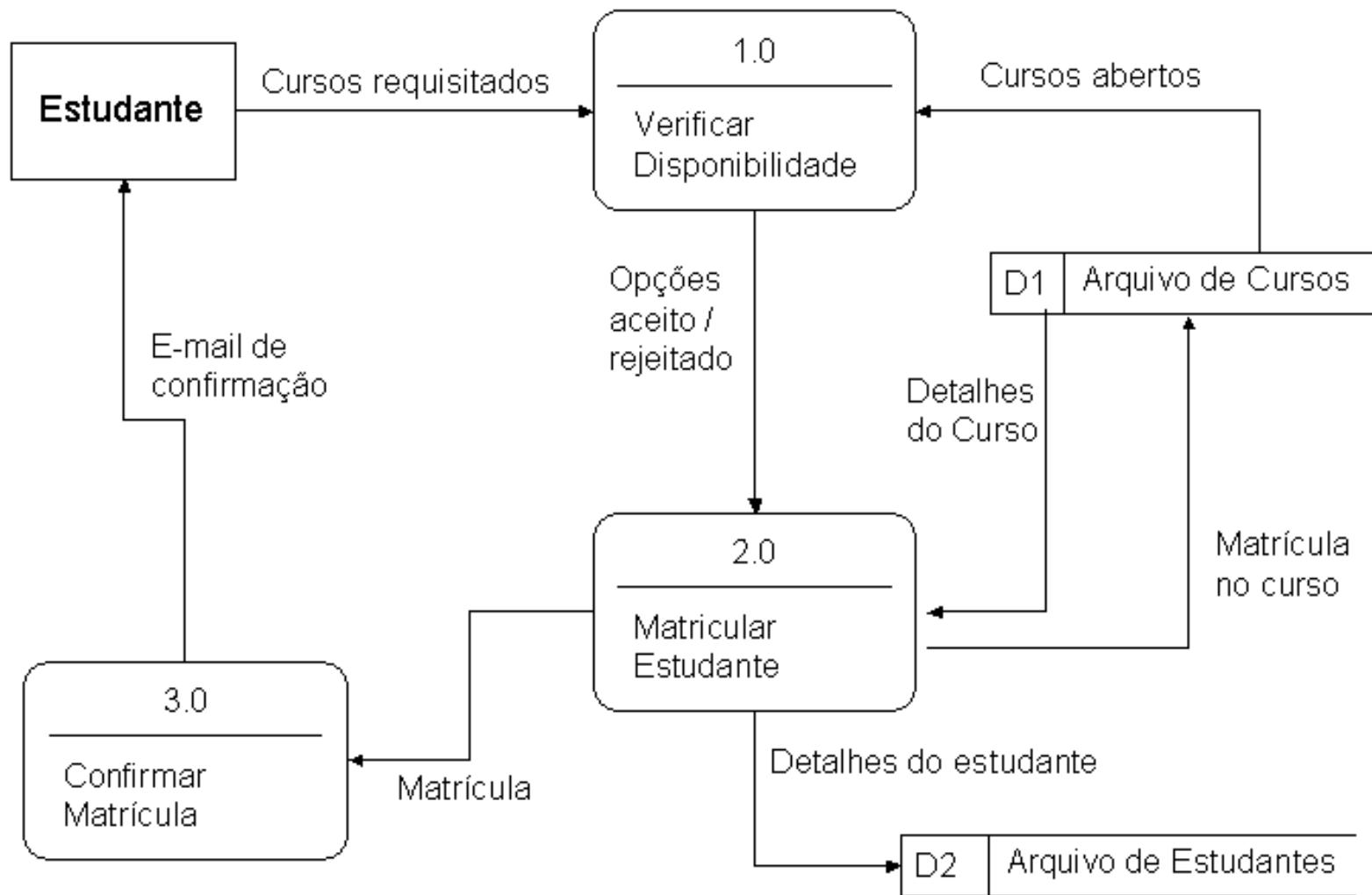
# Checagem de Regras



# Exemplo

Os estudantes apresentam formulários de matrícula com seus nomes, números de identificação e os números dos cursos que querem freqüentar. No processo 1.0, o sistema verifica se cada curso selecionado ainda está aberto consultando o arquivo de cursos da universidade. O arquivo diferencia cursos que estão abertos daqueles que foram cancelados ou já estão lotados. O processo 1.0 então determina quais das opções do estudante podem ser aceitas ou rejeitadas. Em seguida, o processo 2.0 matricula o estudante nos cursos para os quais foi aceito. Também atualiza o arquivo de cursos da universidade com o nome e o número de identificação do estudante e recalcula a lotação da classe. Quando o número máximo de matrículas é alcançado, o número do curso recebe um rótulo indicando que está fechado. O processo 2.0 atualiza, ainda, o arquivo mestre de alunos da universidade com informações de novos estudantes ou mudanças de endereço. O processo 3.0 então envia a cada estudante pretendente uma carta de confirmação de matrícula, listando os cursos para os quais ele está matriculado e citando as opções que não puderam ser atendidas.

# Solução



# Capítulo 3

UML  
Linguagem Unificada de  
Modelagem

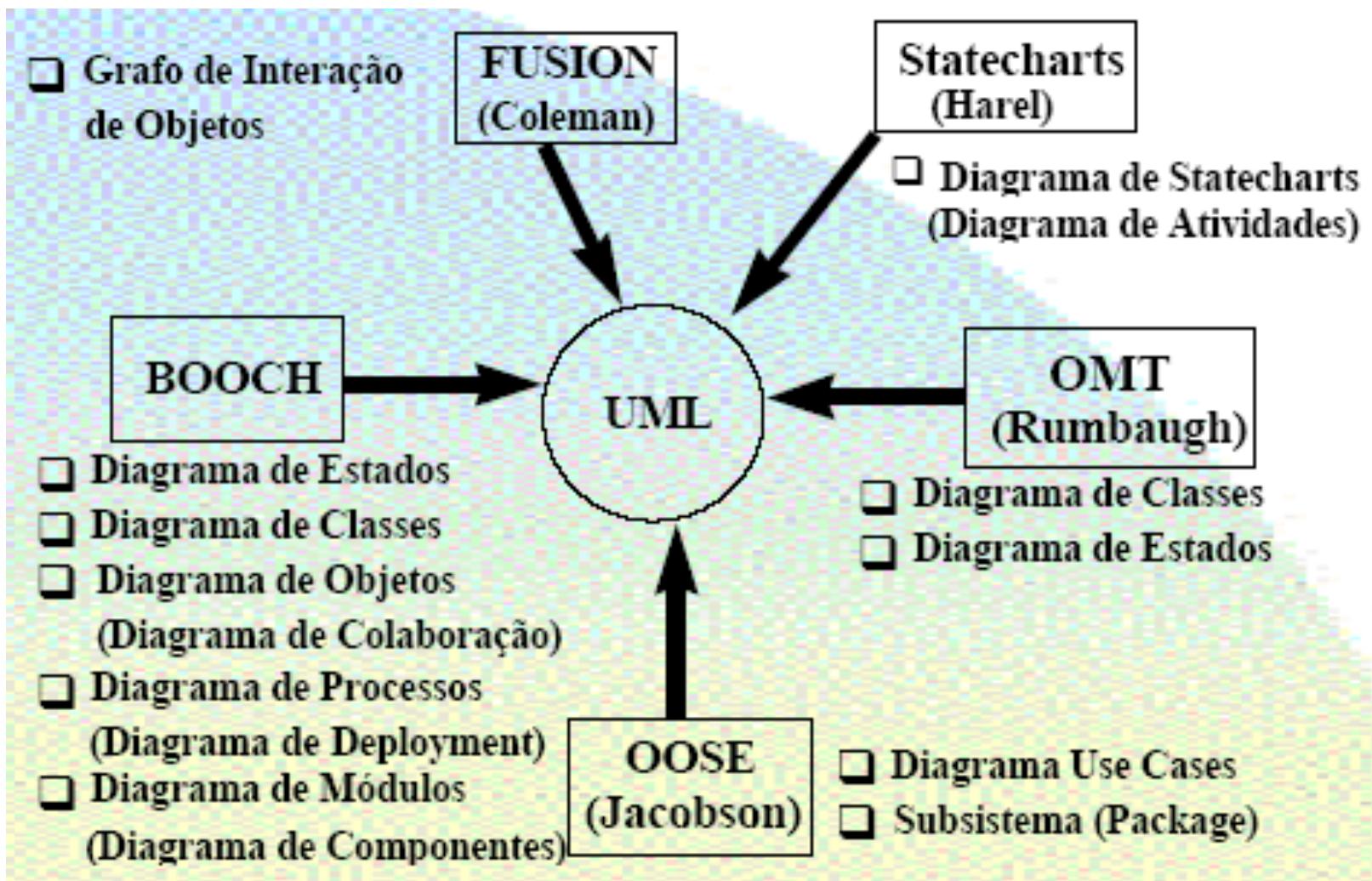
# UML - Definição

A UML (*Unified Modeling Language*) é  
uma linguagem para especificação,  
construção, visualização e documentação  
de sistemas de **software**.

# UML - Definição

A UML é uma evolução das linguagens para especificação dos conceitos de Booch, OMT (*Object Modeling Technique- Rumbaugh*), OOSE (*Object-Oriented Software Engineering - Jacobson*) e também de outros métodos de especificação de requisitos de software orientados a objetos ou não. A notação UML é uma união de sintaxe gráfica de vários métodos, com certo número de símbolos removidos (porque são confusos, supérfluos ou pouco usados) e com outros símbolos adicionados.

# UML - Associação



# Ferramenta CASE

Provê suporte computacional a um determinado método ou linguagem

Ambiente de desenvolvimento: conjunto de ferramentas integradas (CASE)

Exemplos: Rational Rose, JBuilder

# Desenvolvimento de Software e Reusabilidade

Fatores como o aumento do tamanho e complexidade, bem como, a necessidade de adaptação rápida a novos ambientes têm contribuído para a elevação do custo de desenvolvimento e manutenção de software. A técnica de construção de sistemas a partir de componentes reusáveis é a solução indicada para tratar este problema.

Um componente de software reusável é aquele produzido com a finalidade de ser utilizado em diferentes aplicações. Componentes reusáveis podem ser gerados a partir de qualquer combinação de produtos de engenharia de software, incluindo requisitos, especificações, projetos e produtos de implementação (código, testes e documentação).

# Desenvolvimento de Software e Reusabilidade

*Reuso de software é a aplicação de um componente reusável em mais de um Sistema de aplicação.*

As melhores organizações de PD estão conseguindo níveis de reusabilidade de 70% a 80%. Desta forma, quando se defronta com a tarefa de construir um sistema que exige 10.000 linhas de código, apenas 2.000 a 3.000 são escritas e o restante é proveniente de uma biblioteca de componentes reusáveis.

# Benefícios da Reusabilidade

A) Custos Reduzidos: A redução significativa de custos pode-se sentir em todas as fases do ciclo de vida do software. Por exemplo, durante a manutenção, os custos são reduzidos porque os engenheiros estão familiarizados com requisitos comuns, projetos, arquiteturas e componentes.

A) Qualidade Ampliada: Não importa quão extensivamente um sistema é testado e analisado, pois alguns erros só serão identificados quando o mesmo entra em processo de operação. Desde que, componentes reusáveis façam parte de muitos sistemas, erros identificados em um sistema podem ser utilizados para corrigir outros sistemas desenvolvidos ou em desenvolvimento.

# Benefícios da Reusabilidade

- C) Tarefas Reduzidas: Sistematicamente, o reuso em domínios específicos significa o reuso de requisitos, especificações, projetos e produtos de implementação. Isso permite o desenvolvimento mais rápido que o convencional.
- D) Aumento de Produtividade: Desde que existam ferramentas apropriadas para localizar componentes reusáveis em uma biblioteca, esta técnica permite construir sistemas com esforços reduzidos.

# Exercício em Grupo:

- 1) O software é o fator de diferenciação de muitos produtos e sistemas baseados em computador. Apresente exemplos de dois ou três produtos e de pelo menos um sistema em que o software, não o hardware, é o elemento que faz a diferença.
- 2) Nas décadas de 1950 e 1960, a programação de computadores era uma forma de arte aprendida num ambiente semelhante ao de aprendizes. Como os primórdios afetaram as práticas de desenvolvimento de software atuais?
- 3) A engenharia de software auxiliada por computador é uma indústria crescente. Pesquise um produto CASE comercialmente disponível e apresente uma comparação usando critérios que você desenvolve.
- 4) Pesquise a mídia popular (jornais, revistas e televisão que estejam dirigidos para as massas) ao longo dos últimos 12 meses e descubra pelo menos uma grande reportagem em que o software era o tema em questão. Anote quaisquer erros na apresentação.