Introdução

- A Programação Orientada a Objetos (POO) é um paradigma Baseado em objetos
  - Paradigma – Modelo, padrão para especificação de um problema
- Paradigmas existentes na programação de computadores:
  - Procedimental, Funcional, Lógico e Orientado a Objetos

Paradigma Procedimental

- Computação ocorre através da execução de instruções passo a passo.
  - Exemplos: Fortran, Cobol, C; Pascal

```
int fatorial(int n){
  int fat;
  fat = 1;
  if (n == 0)
    return 1;
  while (n >= 1){
    fat = fat * n;
    n = n -1;
  }
  return fat;
}
```

Paradigma Funcional

- Computação baseada em cálculo de funções.
  - Exemplo LISP; HASKELL

```
;LISP
(defun fatorial (n)
  (cond
    ((= n 0) 1)
    (t (* n (fatorial (- n 1))))
  )
)
-- Haskell
fatorial :: Integer -> Integer
fatorial 0 = 1
fatorial n = n * fatorial (n - 1)
```

Paradigma Lógico

- Computação baseada em fatos e regras.
  - Exemplo: Prolog

```
fatorial(0,1).
fatorial(N,X):- M is N - 1,
                fatorial(M,Y),
                X is N * Y.
```

Paradigma Orientado a Objetos

- Computação baseada em objetos que se intercomunicam.
- Objetos contém dados e métodos.
  - Exemplos: C++, Java

```
public class MathFunction {
  private long valor = 0;
  public static long Fatorial() {
    if (valor == 0) {
     return 1;
    } else {
    return valor*MathFunction.Fatorial(valor-1);
    }
  }
}
```

Por que a Orientação a Objetos?

- As abstrações podem corresponder às “coisas” do domínio do problema, facilitando o entendimento
- Esta abstração facilita a comunicação com os usuários
- Os mesmos objetos existem em todas as fases e uma notação única facilita a INTEGRAÇÃO ENTRE FASES de desenvolvimento
- A tecnologia de objetos facilita o entendimento do domínio do problema, permitindo o GERENCIAMENTO DA COMPLEXIDADE através da modularização
- Facilidade de mudanças através do ENCAPSULAMENTO de dados

Por que a Orientação a Objetos?

- Capacidade de aproveitar novas plataformas e ferramentas
- Facilidade de manutenção
- Economia de custos
- Encapsulamento das aplicações existentes
- Melhores interfaces
- Maior produtividade
- Participação no "futuro da computação“
- Rápido desenvolvimento de aplicações estratégicas
- Reuso de software

Por que a Orientação a Objetos?

- Domínio do Problema (Mundo Real)

- Domínio da Solução (Software)

Indústria de Software - Histórico

- A indústria de Software está em constante evolução
  - Década de 1960
    - Orientação Batch
    - Distribuição limitada
    - Software customizado
  - Década de 1970
    - Multiusuário
    - Tempo real
    - Bancos de Dados
    - Produto de Software

Indústria de Software - Histórico

- Década de 1980
  - Sistemas distribuídos
  - “Inteligência” embutida
  - Hardware Acessível (PCs)
  - Impacto de consumo
- Década de 1990 – Atual
  - Sistemas desktop poderosos
  - Tecnologias OO (Orientada a Objetos)
  - Sistemas Especialistas
  - Redes Neurais Artificiais
  - Computação Paralela

Panorama Atual Indústria de Software

- Maiores funcionalidades
  - Requisitos Funcionais e Não-Funcionais
- Maior complexidade
- Abrangência de um maior número usuários
- Especialização do trabalho, exigindo a participação de equipes em seu desenvolvimento
- Sistemas distribuídos baseados na WEB
- Mercado competitivo

Características do Software

- Software é desenvolvido e não produzido no sentido clássico (industrial)
  - Custo de Software é na engenharia e não na reprodução
- Software não se ‘gasta’
  - Custos em sua manutenção
- Software precisa se adaptar a novas tecnologias

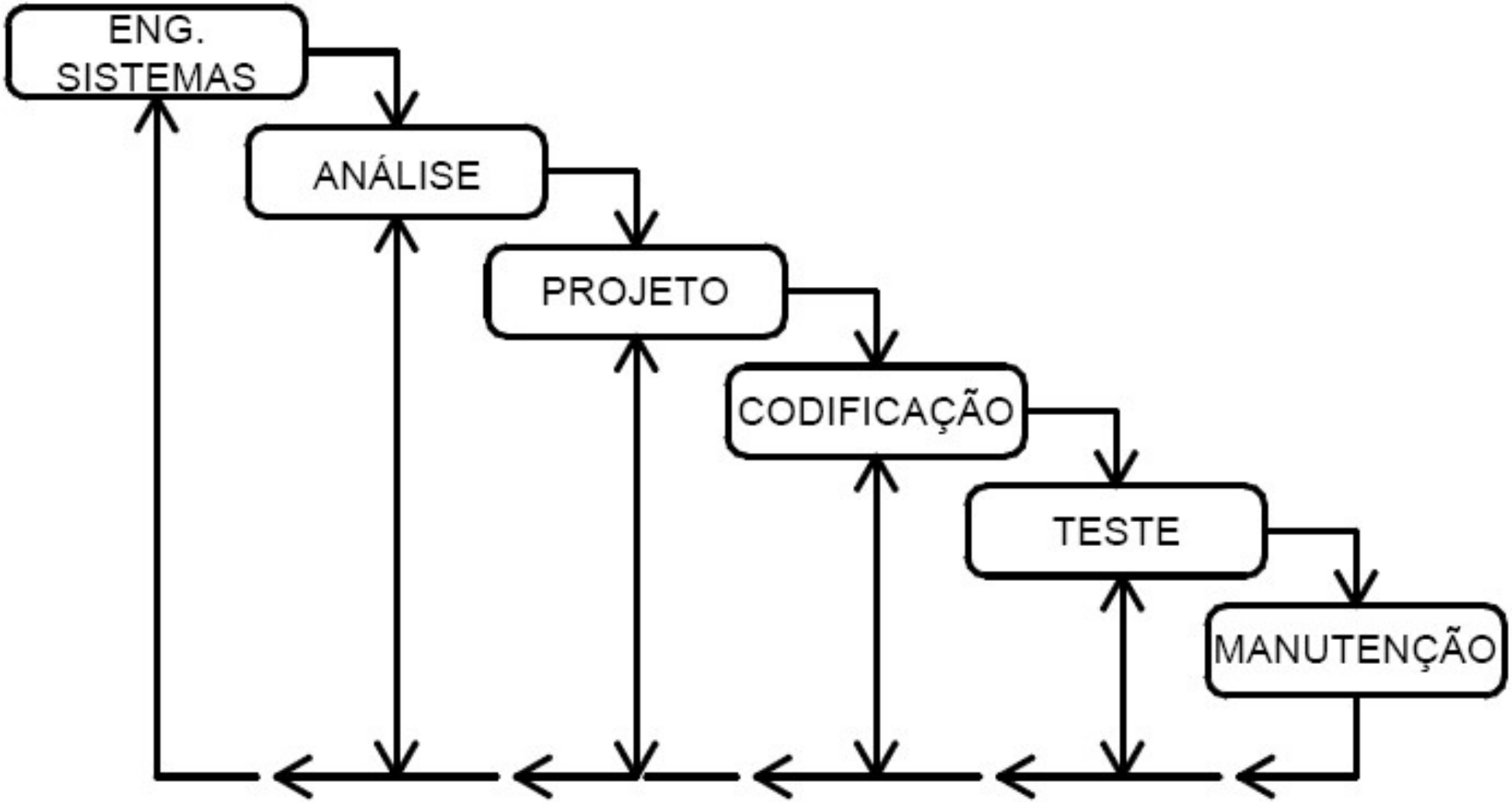
Engenharia de Software

- Estudo e aplicação de Métodos e Técnicas com o objetivo de tornar o desenvolvimento de software mais “eficiente”
- “O estabelecimento e uso de princípios de engenharia de forma a obter economicamente software confiável e que funcione eficientemente em máquinas reais.”
- Existe como disciplina há pouco tempo
- Estabelece um diferencial entre um engenheiro de software profissional e um “praticante da informática”
- Novos profissionais são agentes de mudanças (ou de problemas…)
  - Oportunidades, Desafios e Perigos...

Engenharia de Software

- Para o desenvolvimento de software uma linguagem de modelagem não é suficiente
- Precisamos também de um processo de desenvolvimento:
  - Linguagem de modelagem + processo de desenvolvimento = método (ou metodologia) de desenvolvimento
- O processo de desenvolvimento define quem faz o que, quando e como, para atingir os objetivos necessários.

Ciclo de Vida - Modelo Cascata



Engenharia de Software

- **ENGENHARIA DE SISTEMAS**
  - Levantamento dos requisitos
  - Inserir o sistema em um contexto maior –Hardware; Pessoas; Outros sistemas
  - Visão geral e ampla do sistema
  - Riscos; Custos; Prazos; Planejamento

Engenharia de Software

- **ANÁLISE**
  - Continua o processo de coleta de requisitos, porém concentra-se no âmbito do software
  - Modelos – Dados; Funções e comportamentos
  - Particionamento do problema
  - Documentação e Revisão dos requisitos
    - ANÁLISE ESTRUTURADA – DFD
    - ANÁLISE ORIENTADA A OBJETOS – Diagramas de Caso Uso

Engenharia de Software

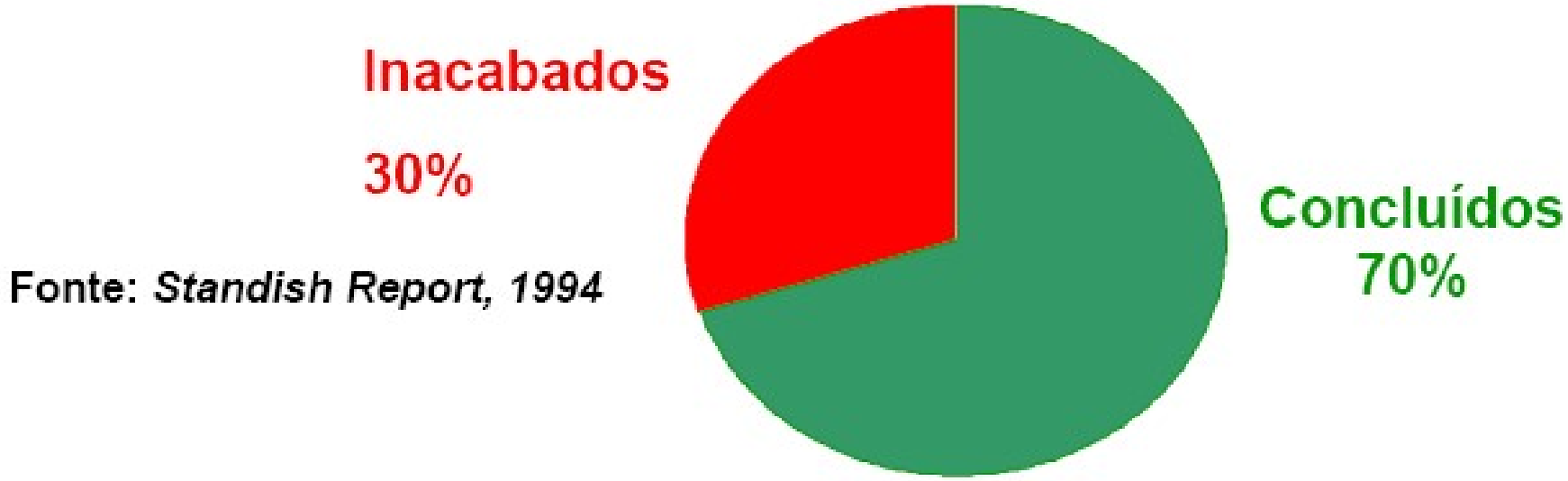
- **PROJETO**
  - "Como" o software irá executar os requisitos
  - Estrutura de dados; Arquitetura do Software;
  - Detalhes de execução; caracterização da
  - interface
  - Produzir um modelo que permita a sua construção posterior
    - PROJETO ESTRUTURADO – Módulos
    - PROJETO ORIENTADO A OBJETOS – Atributos;Especificação dos Métodos; Mensagens
    - Diagramas de Sequência; Diagrama de Classes

Engenharia de Software

- **CODIFICAÇÃO**
  - “Traduzir” o projeto para uma linguagem de
  - computador
  - Projeto detalhado pode levar a uma codificação
  - mecânica (Ferramenta CASE)
- **TESTES**
  - Verificação se o código atende aos requisitos
  - Aspectos lógicos e internos do software – Teste de todas as instruções
  - Aspectos funcionais externos – entrada produz o resultado esperado

Modelo Cascata - Crítica

- Resultado de projetos de software realizados nos EUA no início da década de 90 (Todos baseados no modelo Cascata)
  - 53% custaram até 200% acima da estimativa inicial.
  - Estimou-se que US$ 81 bilhões foram gastos em projetos fracassados só no ano de 1995



Fonte: *Standish Report, 1994*

Modelo Cascata – O que deu Errado?

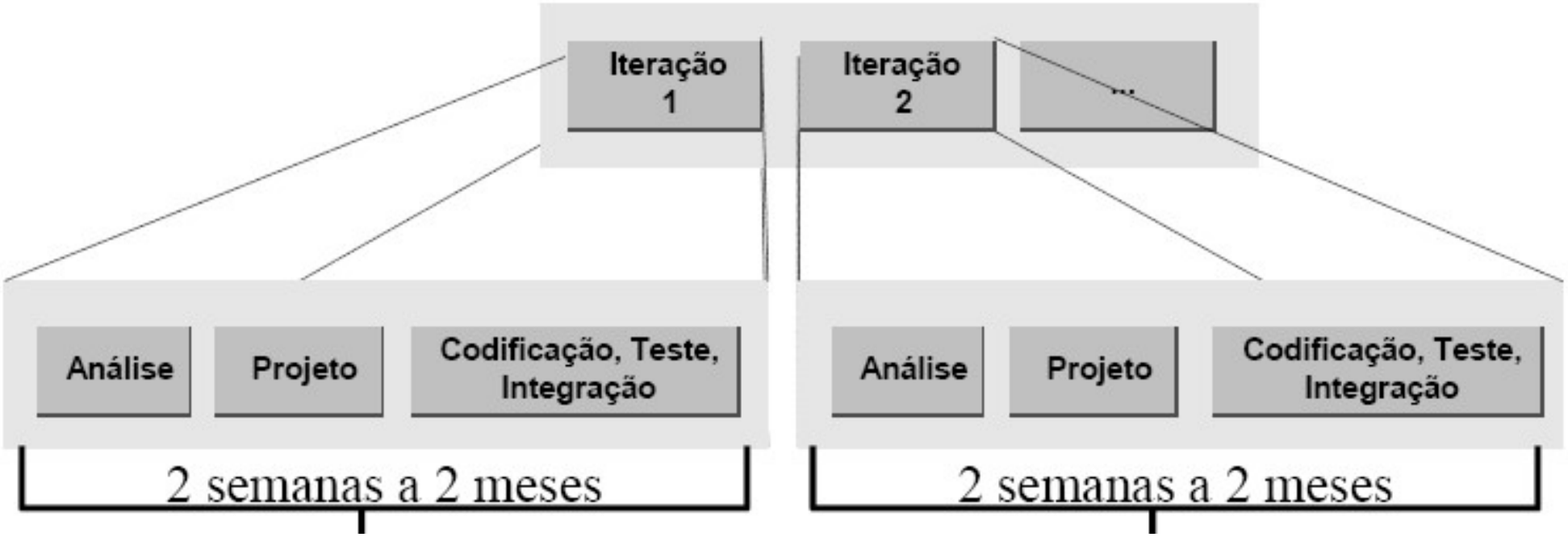
- O modelo cascata é fortemente baseado em suposições oriundas dos processos de engenharia convencionais
- Algumas dessas suposições não foram confirmadas na prática
  - Todos os requisitos podem ser precisamente identificados antes do desenvolvimento
  - Os requisitos são estáveis
  - O projeto pode ser feito totalmente antes da implementação

Modelo Cascata – O que deu Errado?

- **Instabilidade dos Requisitos**
  - O mercado está em mudança constante.
  - As tecnologias inevitavelmente mudam e evoluem
  - A vontade e objetivos dos usuários mudam, muitas vezes, de forma imprevisível.
- **Projeto Completo antes da Implementação?**
  - Pergunte a qualquer programador!
  - Uma especificação completa tem que ser tão detalhada quanto o próprio código
  - Desenvolver software é uma atividade intrinsecamente "difícil"

Ciclo de Vida - Modelo Iterativo

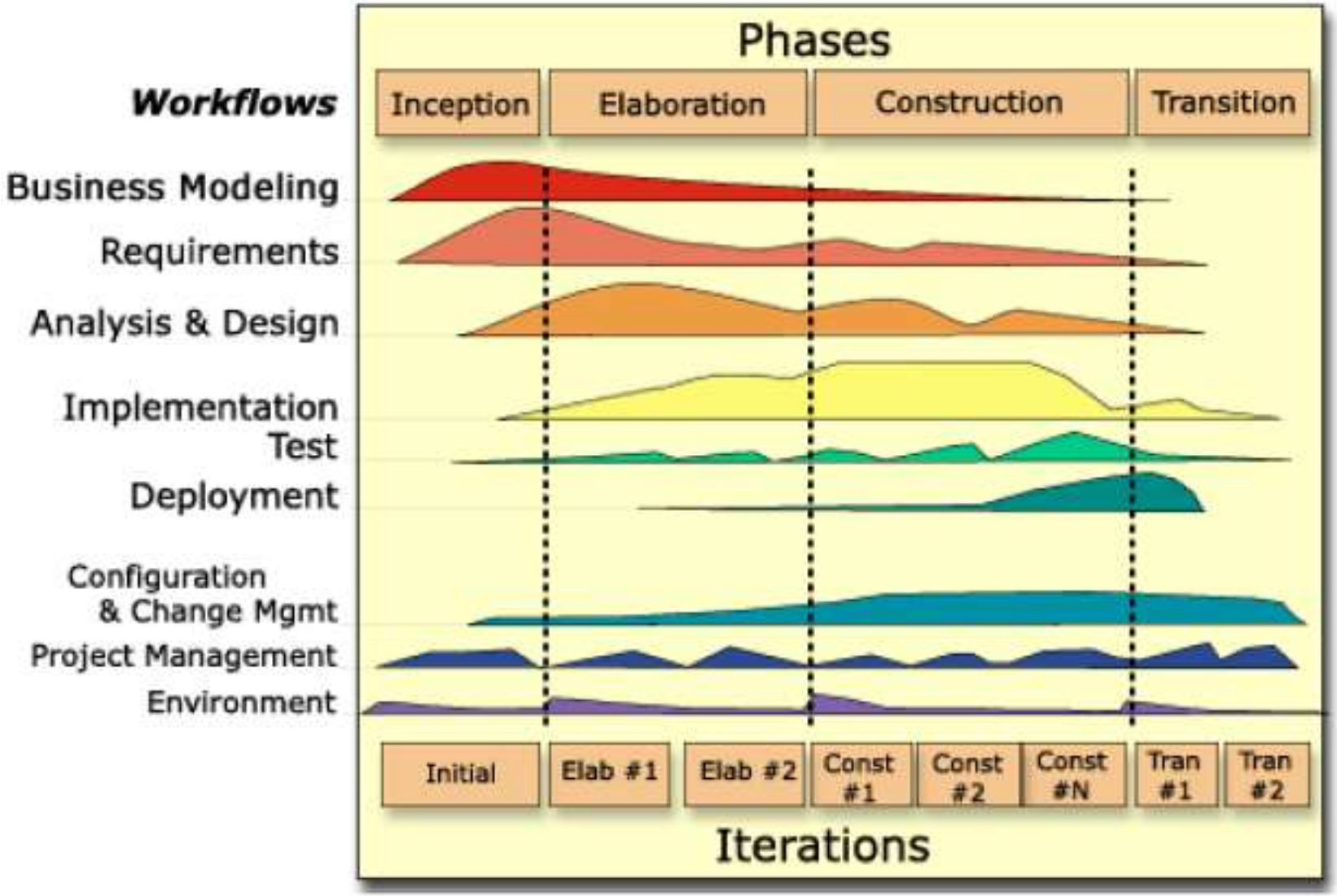
- Passos curtos, feedback e refinamento
- Iterativo, incremental, com intervalos de tempo (ciclos) pré-estabelecidos



Modelo Iterativo

- Baseia-se no fato de que não se deve ter o software inteiro funcionando por inteiro no primeiro release. Isto é um grande risco!
- Um processo de desenvolvimento deve ser:
  - Iterativo - Ter várias iterações no tempo. A iteração dura entre 2 semanas e 2 meses
  - Incremental - Gerar novas versões incrementadas a cada release.
- A cada iteração aumenta a compreensão do problema e são introduzidos aperfeiçoamentos sucessivos.

Modelo Iterativo - RUP

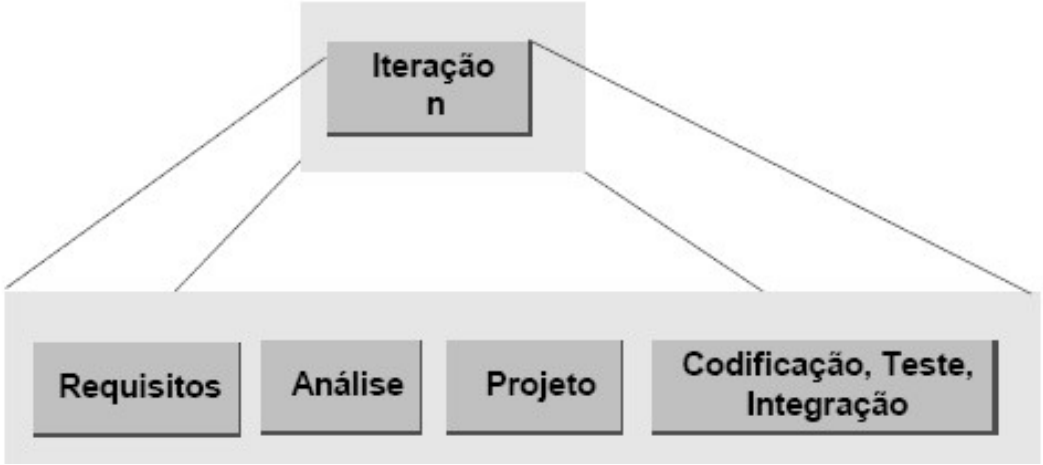


Modelo Iterativo - RUP

- No RUP (Rational Unifed Process) a ênfase é dada na criação de MODELOS (UML) ao invés de documentos.
- As atividades de desenvolvimento são orientadas por caso
- de uso.
- O RUP encoraja o controle de qualidade e o gerenciamento de riscos, contínuos e objetivos
- O desenvolvimento é divido em **FASES e ITERAÇÕES.**
  - FASE – Período de tempo entre marcos do processo, onde um conjunto bem definido de objetivos é alcançado.
  - ITERAÇÃO – Em cada fase, ocorrem várias iterações

Modelo Iterativo - RUP

- Em casa FASE acontece várias **ITERAÇÕES**
  - Uma iteração equivale a um ciclo completo de desenvolvimento
  - Cada iteração resulta em um projeto executável
  - Ao final de cada iteração é possível avaliar se as metas foram alcançadas e caso seja necessário é possível reestruturar o projeto



Modelo Iterativo – RUP - Fases

- **Concepção**
  - Estabelece os casos de negócio para o projeto e delimita o escopo do projeto.
  - Os Casos de negócio incluem: Critérios de Sucesso; Avaliação de Riscos; Recursos Necessários. Durante a concepção é comum a criação de um protótipo executável, utilizado como testes para concepção.
  - Ao final desta fase deve ser feita a decisão de continuar ou não o desenvolvimento
- **Elaboração**
  - Suas metas incluem: Análise do problema;Estabelecimento de uma arquitetura sólida; Eliminação de elementos de mais alto risco.
  - É necessário a maioria dos requisitos do sistema.
  - A implementação deve mostrar escolha da arquitetura.

Modelo Iterativo – RUP - Fases

- **Construção**
  - Desenvolvimento de maneira iterativa e incremental um produto completo.
  - Descrevendo os requisitos restantes e critérios de aceitação. Nesta fase o sistema ganha corpo, conclui-se a implementação e a realização de testes do software.
  - Ao final desta fase deve ser decidido se o software,ambientes e usuários estão prontos para se tornarem operacionais.
- **Transição**
  - Fornece o sistema a seus usuários finais.
  - Esta fase é tipicamente iniciada pelo fornecimento de uma versão beta.
  - Nesta fase são feitos desenvolvimentos adicionais a fim de ajustar o software.

RUP – Fluxos de Trabalho

- Modelagem do Negócio
  - Descreve a estrutura e a dinâmica da empresa
- Requisitos
  - Identificação dos requisitos a partir de casos de uso (Use Cases)
- Análise e Projeto
  - Descreve as várias visões da arquitetura através de modelos UML
- Implementação
  - Desenvolvimento do software; Teste Unitários e Integração
- Teste
  - Casos de teste; procedimentos e medidas para acompanhamento de erros
- Implantação
  - Configuração do sistema a ser entregue

RUP – Fluxos de Trabalho

- Gerenciamento da Configuração
  - Controle de modificações
- Gereciamento do Projeto
  - Descreve as estratégias para o trabalho com o processo iterativo
- Ambiente
  - Infra-estrutura necessária para o desenvolvimento do sistema

Orientação a Objetos - Conceitos

- OBJETO
- MÉTODO
- MENSAGEM
- CLASSE
- CLASSIFICAÇÃO
- GENERALIZAÇÃO
- ESPECIALIZAÇÃO
- HERANÇA
- POLIMORFISMO
- SOBRECARGA
- ENCAPSULAMENTO
- ABSTRAÇÃO
- MODULARIZAÇÃO

OBJETO

- Entidades que possuem **dados e instruções** sobre como manipular estes dados
- Os objetos estão ligado à solução do problema.
  - Software Gráfico – Objetos: Círculos; Linhas; etc.
  - Software BD – Objetos: Tabelas; Linhas; Campos; etc.
  - Software Comercial: Pedidos; Produtos; Clientes; etc.
- Na OO a solução do problema consiste em um primeiro momento estabelecer quais os objetos serão necessários.
- Um objeto representa uma entidade: física, conceitual ou de software

OBJETO

- Os dados mantidos pelo objeto são chamados de atributos(propriedades)
- Os atributos de um objeto representam seu ESTADO, ou seja, o valor de seus atributos em um determinado momento.
- Objetos possuem IDENTIDADE, ou seja, cada objeto é diferente do outro e cada um tem seu próprio tempo de vida
- Cada objeto tem uma identidade única, mesmo que o estado seja idêntico para ambos os objetos

OBJETO

- Dados ligados ao objeto – Exemplos:
  - Círculo – ponto_centro, raio
  - linha – ponto_inicio; ponto_final
  - Cliente – Nome;Data Nascimento; Telefone
  - Telefone – Numero; Modelo; Cor
  - Exemplos:

```
//criação de objetos (sintaxe C++)
Ponto p(3,4);
Circle c(p,5.4);
Cliente pessoa;
```

MÉTODOS

- Métodos são procedimentos que determinam como o objeto se comporta.
- Através dos métodos é possível manipular os dados contidos no objeto.
- Os métodos estão ligados ao comportamento do objeto
- Exemplos
  - Um círculo poderia possuir os métodos:

draw; move; getArea; getPerimeter; setCenter

  - Um cliente poderia possuir os métodos:

calculaIdade; getTelefone

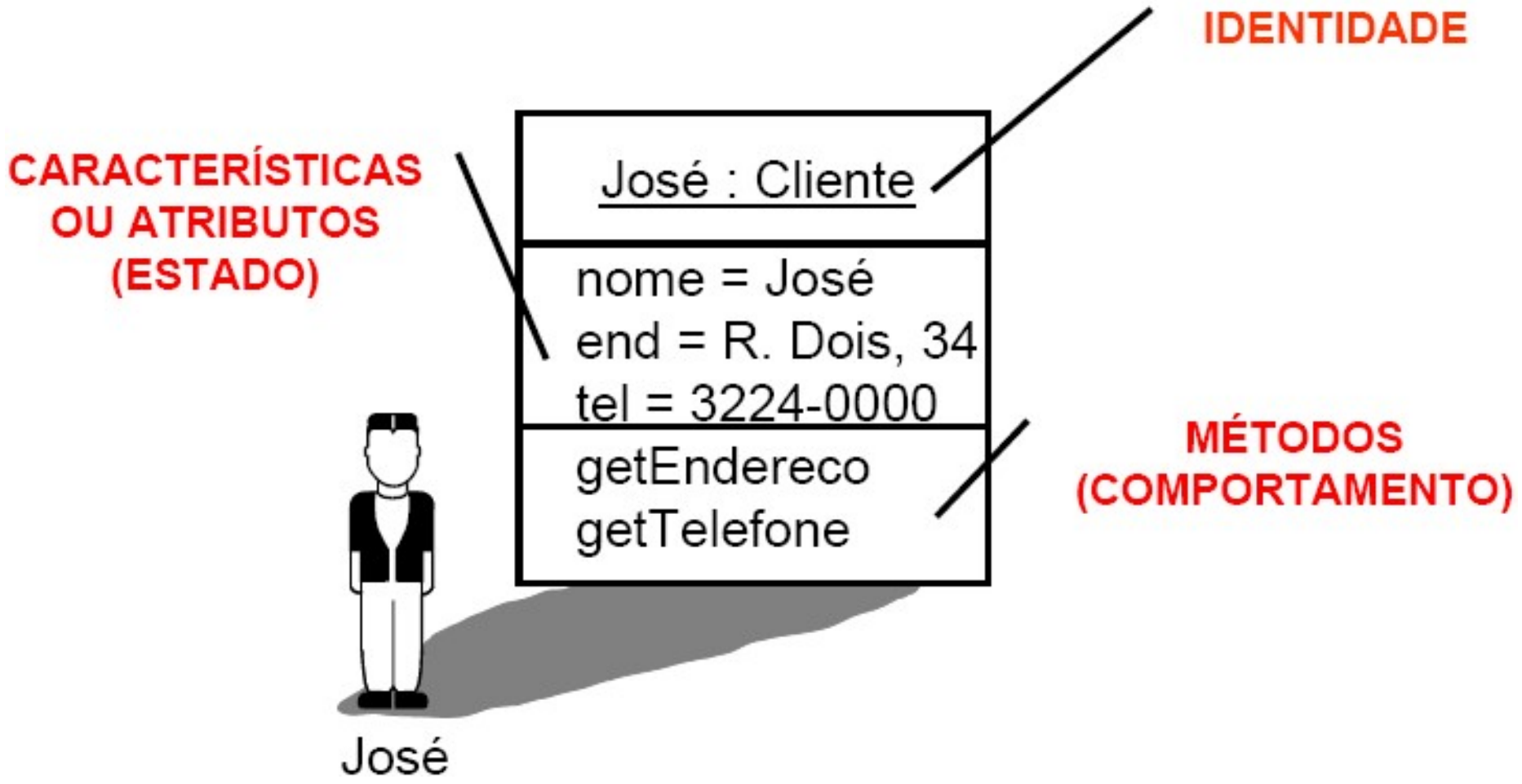
  - Um Telefone poderia possui os métodos:

tocar; discar

MÉTODOS

- Um método é a implementação de uma operação
- Métodos só tem acesso aos dados da classe para a qual foram definidos
- Métodos normalmente possuem argumentos, variáveis locais, valor de retorno, etc.
- Alguns métodos especiais:
  - Construtores – Criam objetos
  - Destrutores – Destroem objetos
  - Acessores – Recuperam o estado de um atributo (getNomeAtributo)
  - Modificadores – Alteram o estado de um atributo (setNomeAtributo)

OBJETOS - RESUMO



MENSAGEM

- Objetos se comunicam entre si através de mensagens.
- Uma mensagem é uma chamada de um método.
- A mensagem possui os seguintes componentes:
  - Receptor – nome do objeto que irá receber a mensagem
  - Método – Método do receptor que será utilizado
  - Argumentos – Informação adicional para a execução do método
  - **Exemplos**

```
Point p(0,0), pNewCenter(2,3);
Circle c(p,3);
c.getArea(); //Exemplo Mensagem
c.setCenter(pNewCenter); //Exemplo Mensagem
```

CLASSE

- Classe é um agrupamento de objetos
- A classe consiste nos métodos e nos dados que um determinado objeto irá possuir.
- Objetos são criados quando uma mensagem solicitando a criação é recebida pela sua classe.
- A programação orientada a objetos consiste em implementar as classes e na utilização das mesmas, através da sua intercomunicação.
- Um objeto é uma instância da classe.
- Os objetos de uma classe compartilham os mesmos atributos, operações, relacionamentos e semânticas
- Objetos só reagem a mensagens que fazem parte das ações do protocolo de sua classe

CLASSIFICAÇÃO

- Na POO classificação consiste em criar classes a partir dos objetos envolvidos em um determinado problema
- As classes podem ser criadas a partir do momento em que for possível isolar no domínio do problema objeto que possui atributos e métodos comuns
- Ex: Diferentes tipos de pessoas interagem com uma empresa

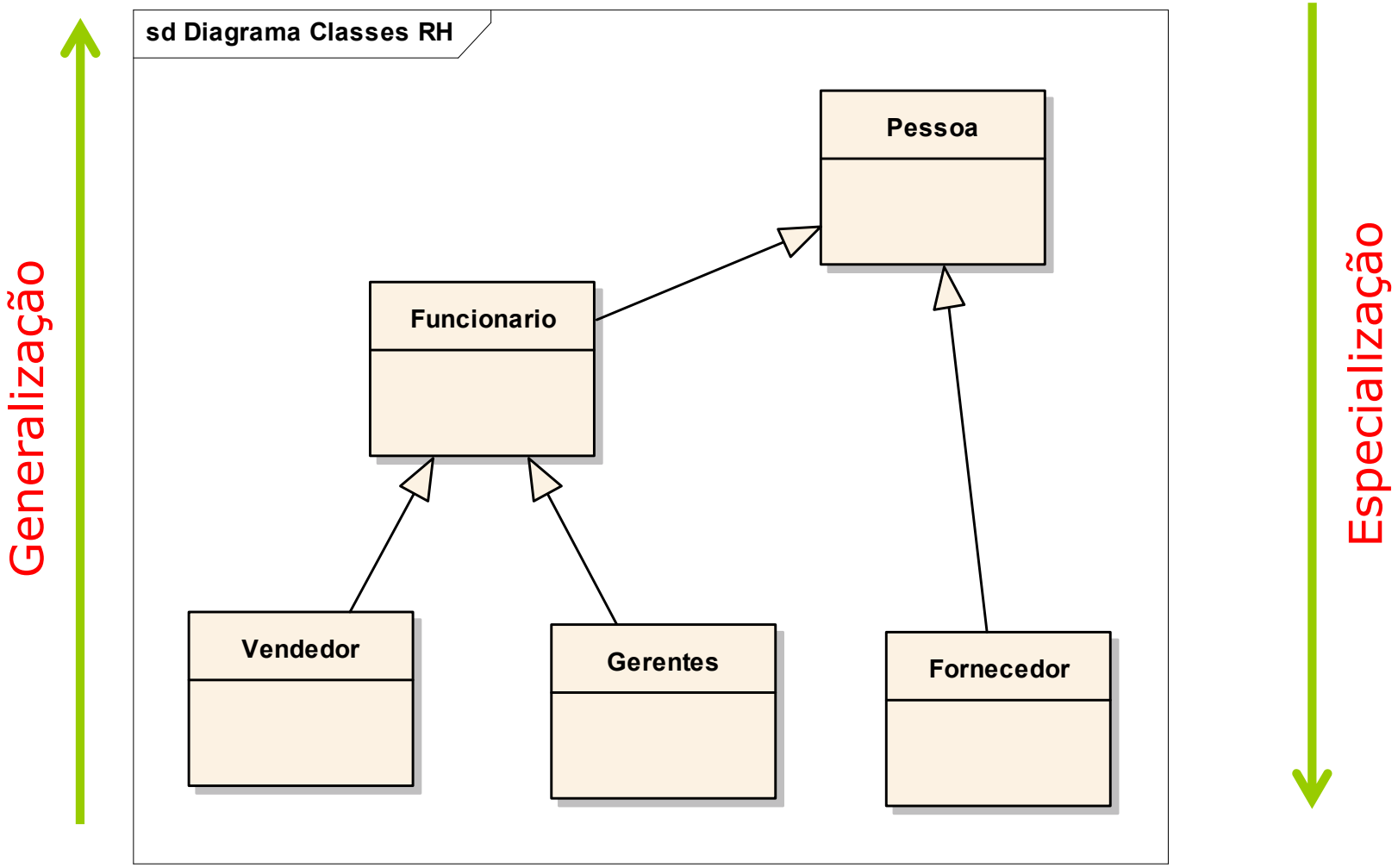
COMPORTAMENTO	CLASSE
Pessoas interessadas nos produtos	???
Pessoas que já compraram os produtos	???
Pessoas que são responsáveis por um grupo de trabalhadores	???
Pessoas responsáveis pela demonstração de produtos e sua venda	???
Trabalhadores da linha de produção	???

Generalização e Especialização

- GENERALIZAÇÃO
  - A generalização consiste em obter similaridades entre as várias classes e partir destas similaridades, novas classes são definidas.
  - Estas classes são chamadas **superclasses**
- ESPECIALIZAÇÃO
  - A especialização por sua vez consiste em observar diferenças entre os objetos de uma mesma classe e dessa forma novas classes são criadas.
  - Estas classes são chamadas **subclasses.**

Generalização e Especialização Exemplo

- Hierarquia de classes



HERANÇA

- Herança é a capacidade de uma subclasse de ter acesso as propriedades da superclasse a ela relacionada.
- Dessa forma as propriedades de uma classe são propagadas de cima para baixo em um diagrama de classes.
- Neste caso dizemos que a subclasse herda as propriedades e métodos da superclasse
- A relação de herança entre duas classes é uma relação da seguinte forma: A “e um tipo de” B, onde A e B são classes.
- Caso esta relação entre as classes não puder ser construída, em geral, também não se tem uma relação de herança entre a classe A a partir da classe B.

HERANÇA

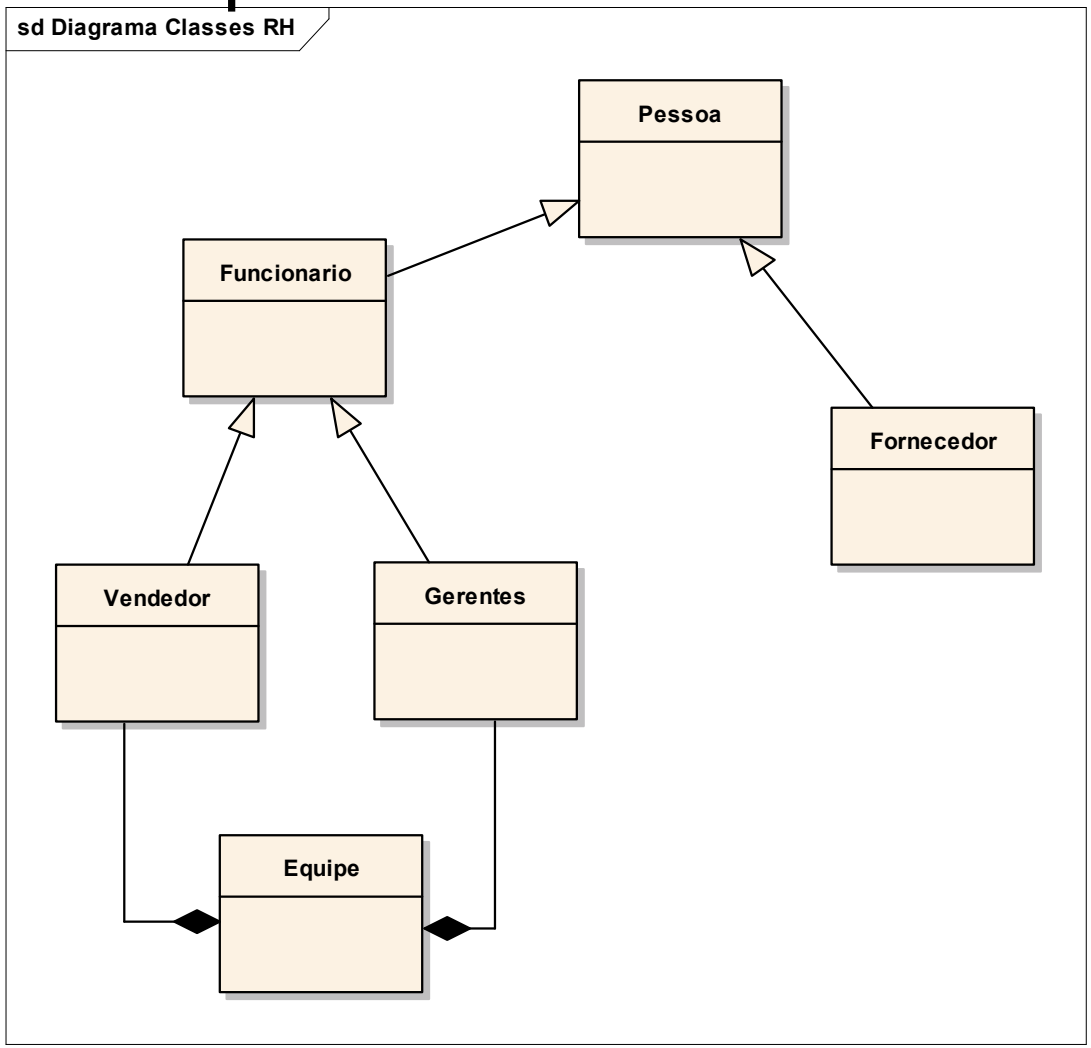
- Exemplos:
  - Um Carro de Passeio “é um tipo de ” veículo; Um caminhão “é um tipo de” veículo;
  - Um círculo “é um tipo de” uma figura geométrica; Um quadrado “é um tipo de” figura geométrica;
  - Um vendedor “é um tipo de” Empregado; Um empregado “é um tipo de” pessoa.
- Herança Múltipla
  - Uma subclasse herda características de mais uma classe
  - Exemplos
    - Um gerente de vendas “é um tipo” de vendedor e também “é um tipo de” gerente;

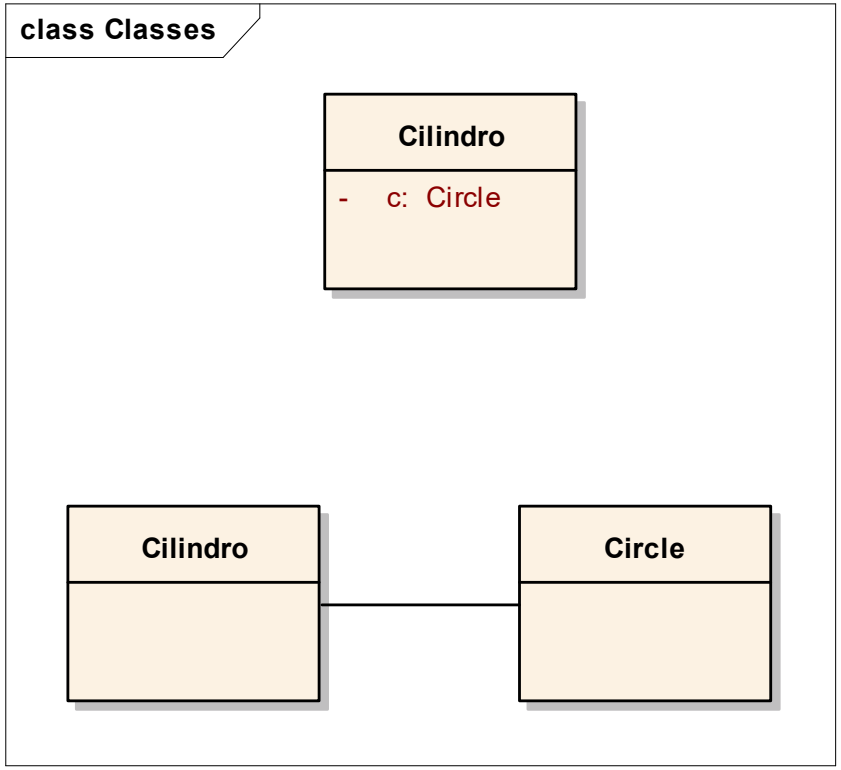
HERANÇA x USO

- Além da relação de herança entre as classes existe a relação de uso
- HERANÇA
  - classe A “é um tipo de” B
- USO / AGREGAÇÃO (Relação de Conteúdo)
  - classe D “contém” classe C”
  - classe D “usa” classe C”
  - classe C “é parte da” classe D
  - Exemplo: Uma **equipe contém um gerente e** um grupo de **vendedores**

Herança x Uso

- Exemplo

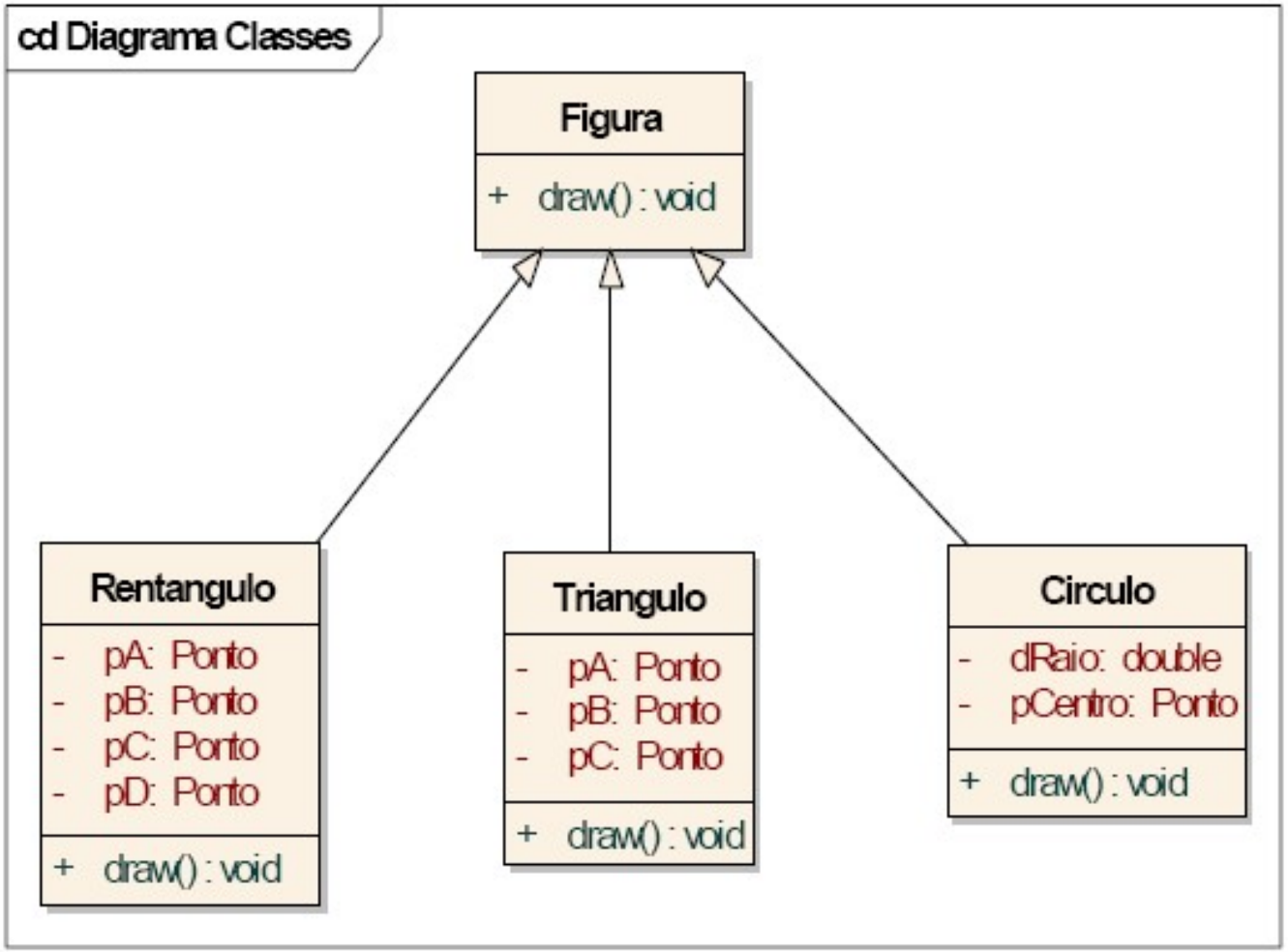




POLIMORFISMO (Override)

- Os objetos respondem às mensagens que eles recebem através dos métodos.
- A mesma mensagem pode resultar em diferentes comportamentos. Esta propriedade é chamada de **polimorfismo**
  - **Exemplo: Método getSalario()**
    - Para um empregado qualquer → getsalario() = Salario;
    - Para o gerente → getsalario() = salario + bonificacao;
  - **Exemplo: Método draw()**
    - Para uma figura qualquer desenha uma forma não definida
    - Para o retângulo, triângulo e círculo o mesmo método responde de uma forma diferente

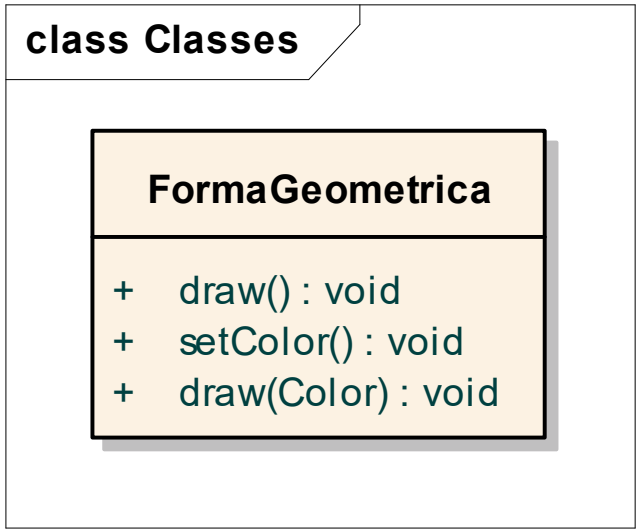
POLIMORFISMO - Exemplo

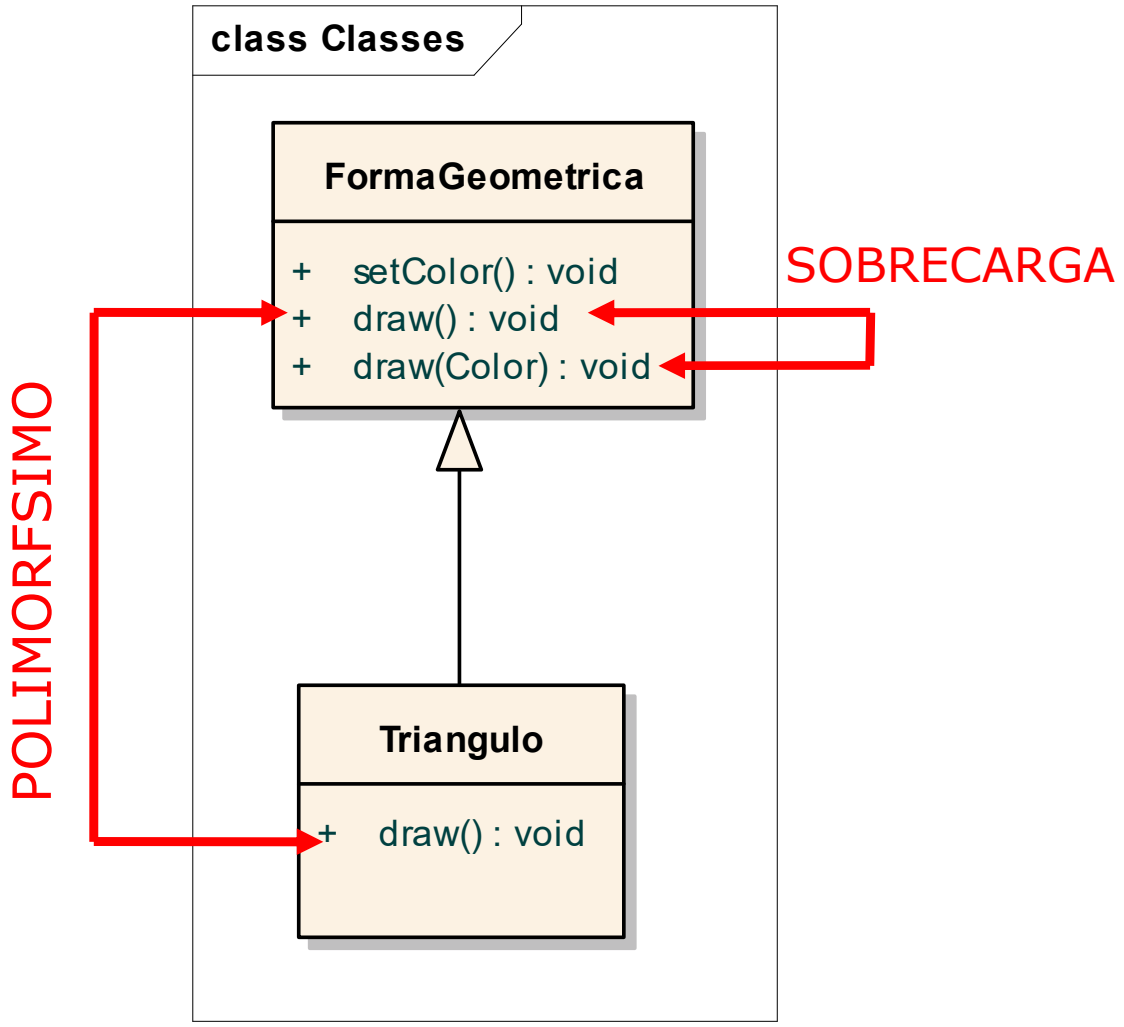


SOBRECARGA(Overload)

- Nas linguagens orientadas a objetos é possível, em uma classe, a existência de métodos que possuem o mesmo nome, porém com diferentes assinaturas.
- Este conceito é chamado de Sobrecarga (Overload)
  - **Exemplo: Em uma classe Figura, temos os seguintes métodos**
    - draw() → desenha o objeto e utiliza uma cor padrão
    - draw(Color) → desenha o objeto, porém recebe uma cor como parâmetro

SobreCarga - Exemplo





ENCAPSULAMENTO

- Conceito que indica que os dados contidos em um objeto somente poderão ser acessados e/ou modificados através de seus métodos.
- Dessa forma não é possível alterar os dados diretamente, somente através de métodos definidos no objeto
- Exemplo
  - O raio somente pode ser alterado/recuperado pelos métodos setCenter/getCenter.

ENCAPSULAMENTO

- O encapsulamento assegura que toda a comunicação com o objeto seja realizada por um conjunto pré-definido de operações
- O encapsulamento facilita as mudanças, visto que os objetos são isolados uns dos outros, reduzindo desta forma o acoplamento
- Além disso o encapsulamento facilita a manutenção de classes, bem como, garante a integridade dos atributos de um objeto em um determinado instante.

ABSTRAÇÃO

- Abstração é o processo de identificar as qualidades ou propriedades importantes do problema que está sendo modelado.
- Através de um modelo abstrato,pode-se concentrar nas características relevantes e ignorar as irrelevantes.
- Abstração é fruto do raciocínio.
- Através da abstração é possível controlar a complexidade. Isto é feito através da ênfase em características essenciais, fazendo-se uma supressão daquilo que não está ligado ao domínio do problema.

MODULARIZAÇÃO

- Consiste em decompor o problema em partes menores.
- Dessa forma o foco é mantido em itens(classes; pacotes; etc.) menores, coesos e fracamente acoplados.

Praticando os conceitos... Atividade

- Utilizando os conceitos
  - Herança
  - Polimorfismo
  - Sobrecarga
  - Classe
  - Atributo
  - Método
- Propor uma modelagem relacionada ao seu ambiente de trabalho
- Enviar para: flavio@facom.ufu.br até 23/08/2011