



Tópico 1 - Fundamentação

Luiz Antônio M. Pereira
lpereira@uninet.com.br
lpereira@luizantoniopereira.com.br



Conteúdo

- Software: definição e engenharia
- Qualidade de Software
- Processos de Software
- Análise e Modelagem Orientadas a Objetos (OOAD)
- UML – Breve História e Objetivos
- Minimundos para o Trabalho



O que é Software, Afinal?

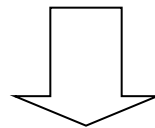
- Software

- Produtos projetados e construídos pelos Engenheiros de Software.
- Abrangem programas de computador, documentos (tangíveis e eletrônicos), dados e modelos (representação de conceitos por meio de figuras).



Eng^a. S/W - Motivação

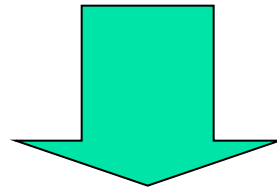
- Sociedade atual cada vez mais dependente do *software*; falhas podem gerar catástrofes.
- Empresas de TI buscam:
 - A manutenção e ampliação:
 - Dos seus nichos de atuação,
 - Dos níveis de satisfação dos seus usuários/clientes.
 - Diminuição de custos de produção de s/w.
 - Controle preciso sobre T e \$ envolvidos na produção de s/w.



Há cada vez menos espaço para improvisações



Eng^a. S/W - Motivação



Busca pela qualidade e confiabilidade do **produto** e no **processo** de produção de *software*



Desenvolvendo *Software*

Engenharia de *Software*

Estudo e aplicação de procedimentos sistemáticos, disciplinados e quantificáveis ao desenvolvimento, operação e manutenção de *software*.

IEEE/93

Desenvolvendo *Software*

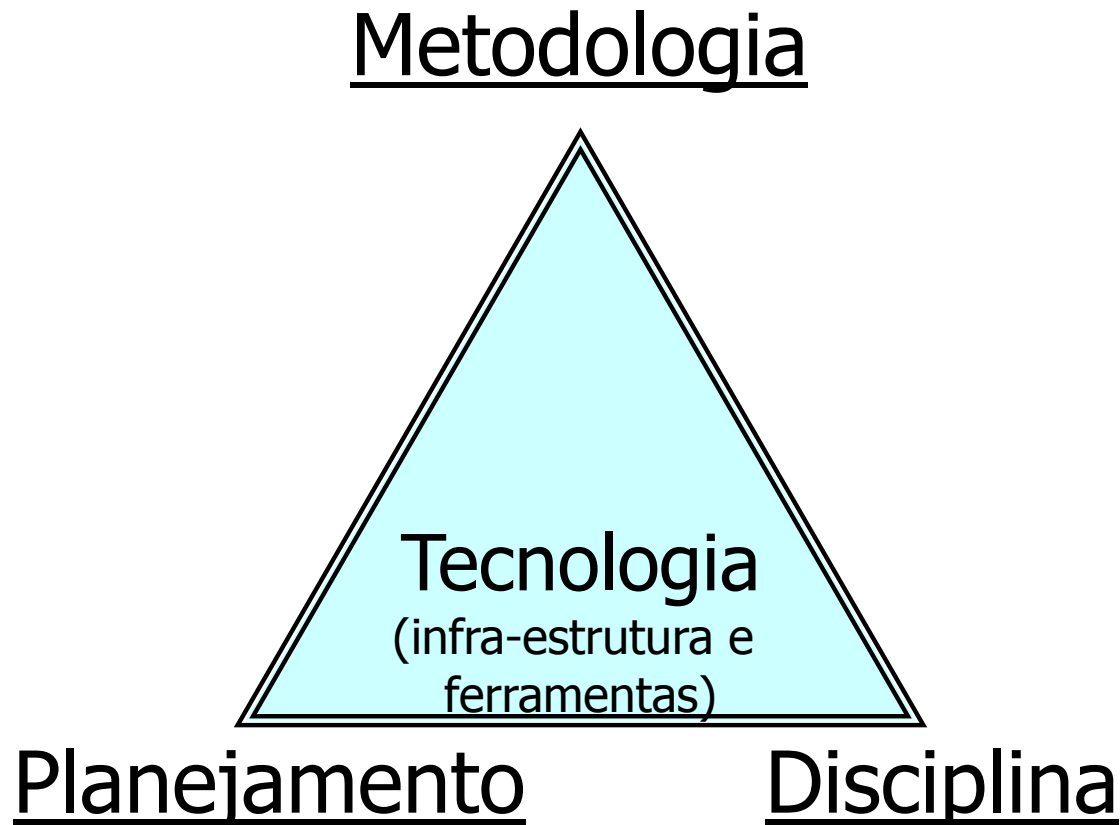
- Há uma chance bem maior de que uma boa solução surja em um *ambiente* organizado!
- Idealmente com o uso de processos bem comportados, envolvendo:
 - Etapas do processo,
 - Níveis de qualidade,
 - Custos e
 - Prazos

Definidos
a priori.



Desenvolvendo *Software*

- Desenvolvimento de s/w deve envolver:





Ciclo de vida do *Software*

- Fases Genéricas ()
 - Definição: **o que** fazer para atender às necessidades dos **clientes e usuários;**
 - Desenvolvimento: **como** fazer;
 - Manutenção: modificar
 - corrigir,
 - adaptar,
 - evoluir,
 - prevenir.

Qualidade em *Software*

- Ficar em conformidade com:
 - Requisitos funcionais e de desempenho explicitamente estabelecidos (são as bases para a medição da qualidade);
 - Padrões de desenvolvimento explicitamente documentados (definem um conjunto de critérios que guiam o processo de desenvolvimento);
 - Características implícitas que são esperadas em todo *software* desenvolvido profissionalmente (e.g. desejo de facilidade de uso e boa manutenibilidade se não forem atendidos colocam todo o s/w sob suspeição).



Qualidade em *Software*

- Diversos Modelos
 - ISO (IEC e 9000)
 - SPICE
 - (S/W) CMM
 - ...



Qualidade em *Software*

- No SW-CMM:
 - A maturidade é medida em níveis;
 - Um nível de maturidade é um patamar definido de evolução de processo;
 - Os níveis de maturidade estabelecem o estágio atual e as etapas necessárias para melhoria dos processos de *software*.



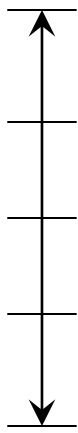
Qualidade em *Software*

- No SW-CMM (cont.):
 - Os níveis são patamares bem definidos conduzem a processos mais maduros de *software*.
 - Cada patamar compreende um conjunto de objetivos e compromimentos (KPAs – *Key Process Areas*) que devem ser satisfeitos.



Qualidade em *Software*

1

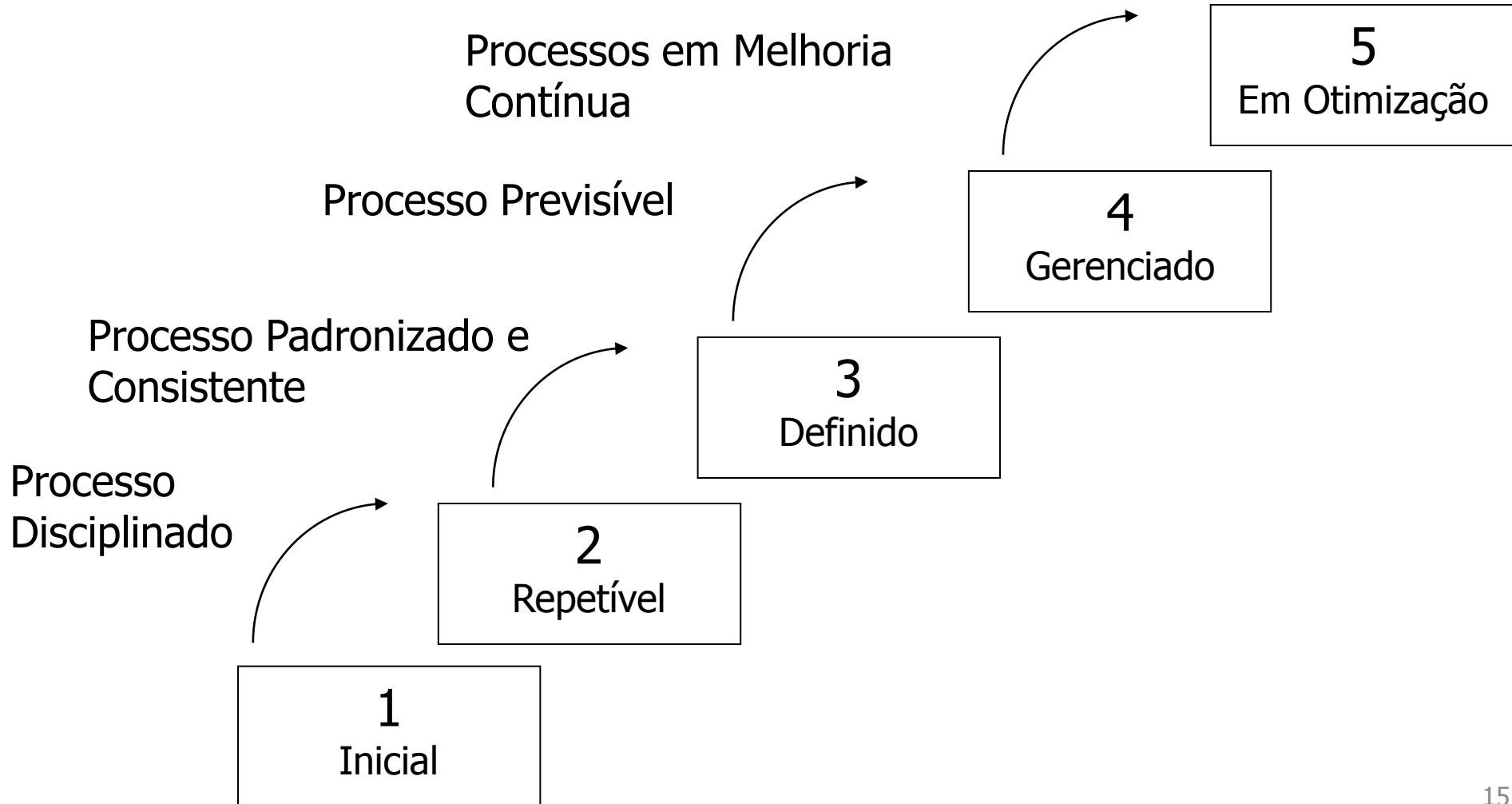


■ Organização imatura: o processo de *software* é improvisado. Mesmo que o processo seja especificado, ele não é seguido. São organizações reacionárias.

5

■ Organização madura: possui capacidade organizada para gerenciar o desenvolvimento e manutenção de *software*.

Qualidade em *Software*





Qualidade em *Software*

■ Níveis:

■ 1 - Inicial

- Processo *ad-hoc*, ocasional e até caótico. Poucos ou nenhum processo está definido. Sucesso depende do esforço individual.

■ 2 - Repetível

- Os processos básicos de gerência estão estabelecidos para acompanhamento de custo, prazos e funcionalidade. Existe a capacidade de repetir processos bem sucedidos em projetos anteriores.



Qualidade em *Software*

- Níveis (cont.):
 - 3 - Definido
 - O processo de software para as atividades de gerência e engenharia estão documentados, padronizados e integrados no processo de desenvolvimento de software da organização.
 - Todos os projetos usam uma versão documentada e aprovada do processo da organização para desenvolvimento e manutenção. Inclui o nível 2.



Qualidade em *Software*

- Níveis (cont.):
 - 4 - Gerenciado
 - Medições detalhadas do processo e do produto são coletadas. Ambos são quantitativamente compreendidos e controlados através de métricas minuciosas. Inclui o nível 3.
 - 5 - Em Otimização (ou Otimizante)
 - Um processo contínuo de melhoria baseado nos resultados quantitativos de outros projetos e em testes de novas idéias e tecnologias está estabelecido. Inclui o nível 4.



Qualidade em *Software*

- Exemplo:
 - KPAs para nível 2:
 - Gerência dos Requisitos
 - Planejamento do Projeto de Desenvolvimento
 - Controle do Projeto de Desenvolvimento
 - Gerência da Aquisição de Software
 - Garantia da Qualidade
 - Gerência de Configuração



Processo de *Software*

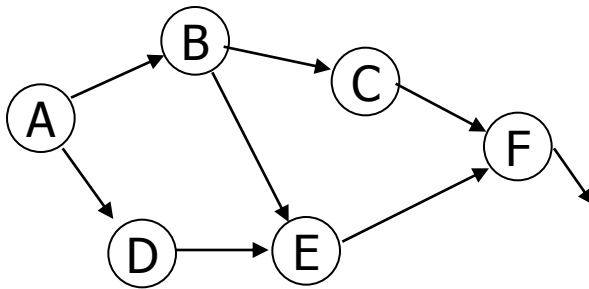
- Um conjunto de atividades, métodos, práticas e transformações que pessoas empregam para definir, desenvolver e manter *software* e produtos associados (plano do projeto, documentos do projeto, código, casos de teste, manuais do usuário ...)



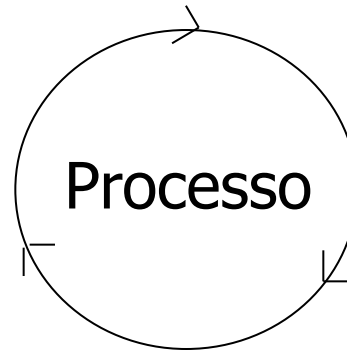
Processo de *Software*

Envolve:

Pessoas com habilidades, treinamento e motivação



Procedimentos e métodos definindo o relacionamento entre as tarefas



Ferramentas e equipamentos adequados



Processo de *Software*

- Possui uma série de tarefas definidas.
- Tarefas são organizadas de formas diferentes, de acordo com o *modelo de processo* adotado.
- Tarefas demandam marcos para verificação, documentação e garantia da qualidade

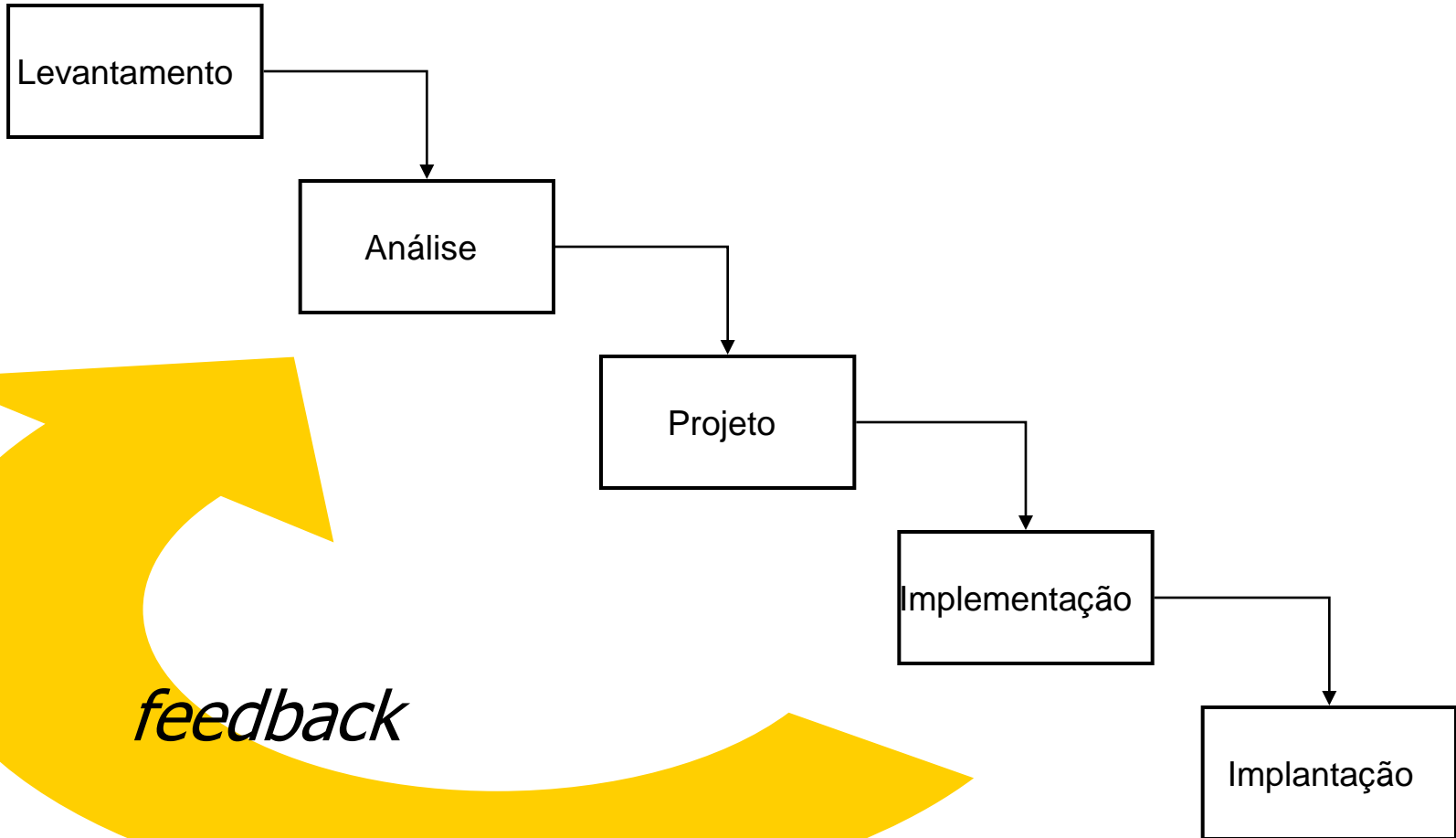


Modelos de Processos

- Cascata (Clássico)
- Prototipação
- Espiral
- Processo Unificado



Cascata





Cascata - Características

- Modelo linear e sequencial
- Pode usar *feedback* ou não

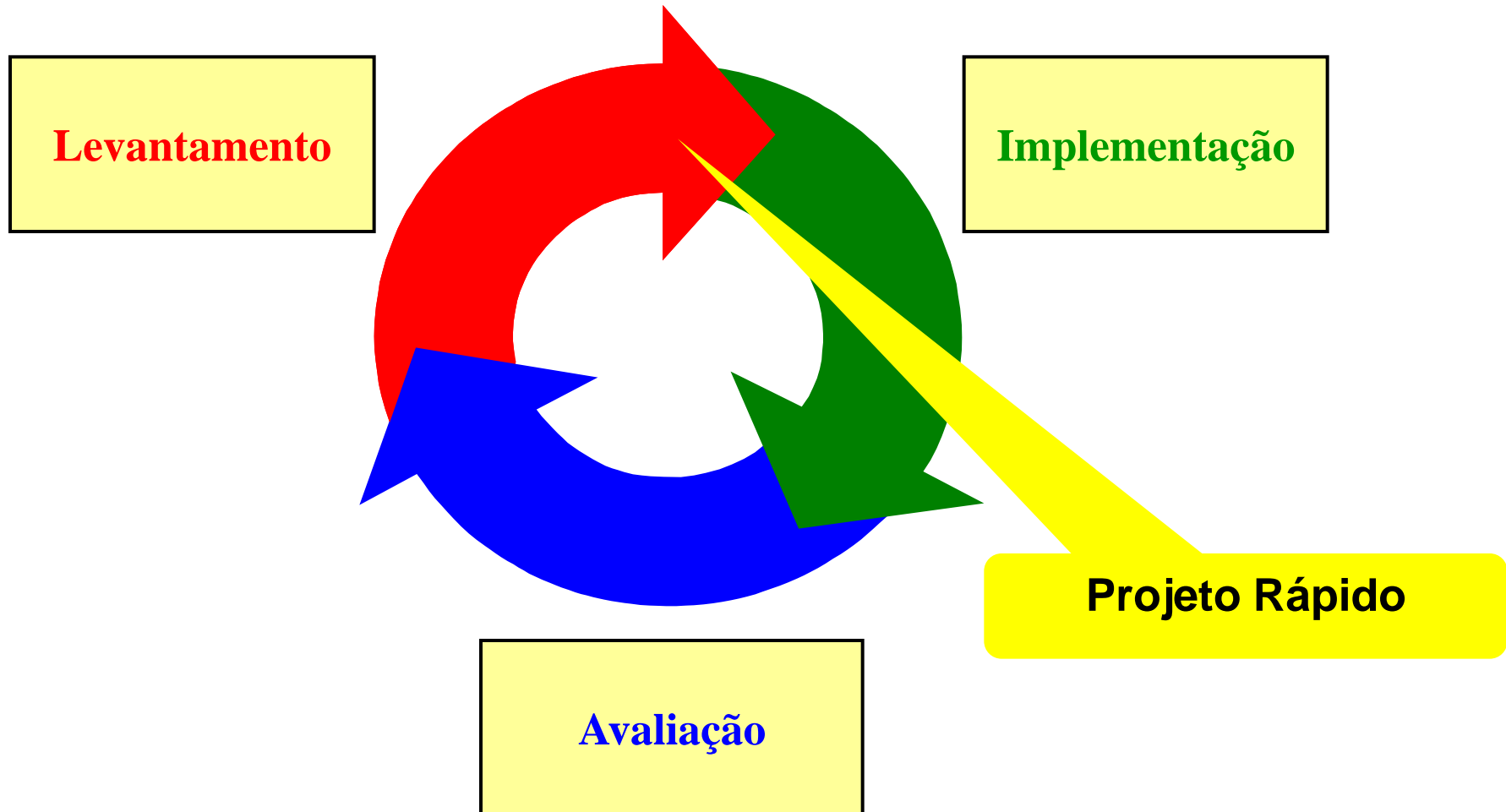


Cascata - Características

- Possui, em geral, um ciclo longo de desenvolvimento
- ⇒ Aplicável no desenvolvimento de sistemas pouco susceptíveis a mudanças de requisitos e em organizações estáveis
- Serve como base para outros modelos de processo



Prototipação



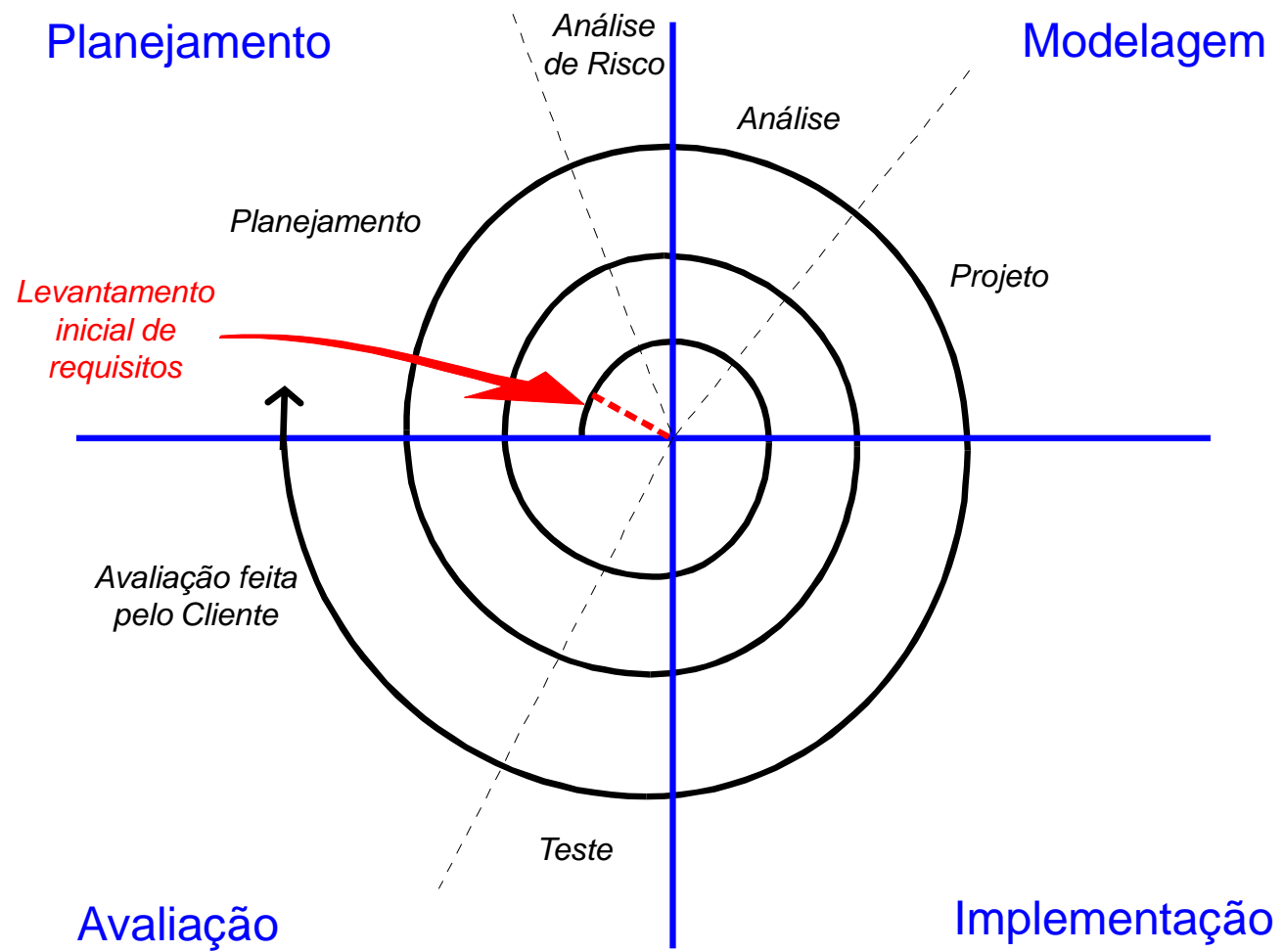


Prototipação - Características

- Construção de protótipos baseado nas informações do cliente
- Adequado para quando o “negócio” não é bem conhecido ou o cliente não sabe exatamente do que precisa
- Idealmente serve p/ identificar requisitos
- Protótipo = “1o. Sistema” (alguns autores recomendam que o joguemos fora)



C C E Espiral





Espiral - Características

- Inicialmente publicado por Barry Boehm(*)
- Reduz sensivelmente o risco de insucesso de um projeto
- Permite uma maior interação com o cliente
- Adequado à maioria dos tipos de projetos/sistemas

(*)"A Spiral Model for Software Development and Enhancement", 1988



Espiral - Características

- Incremental, evolutivo, iterativo
- Espiral é dividida em uma série de regiões
- Cada região contempla uma série de tarefas que são adaptadas às características do projeto a ser conduzido
- Um ciclo da espiral pode produzir, tanto uma especificação, como versões do *software*
- Pode ser adaptado para toda a vida do *software*

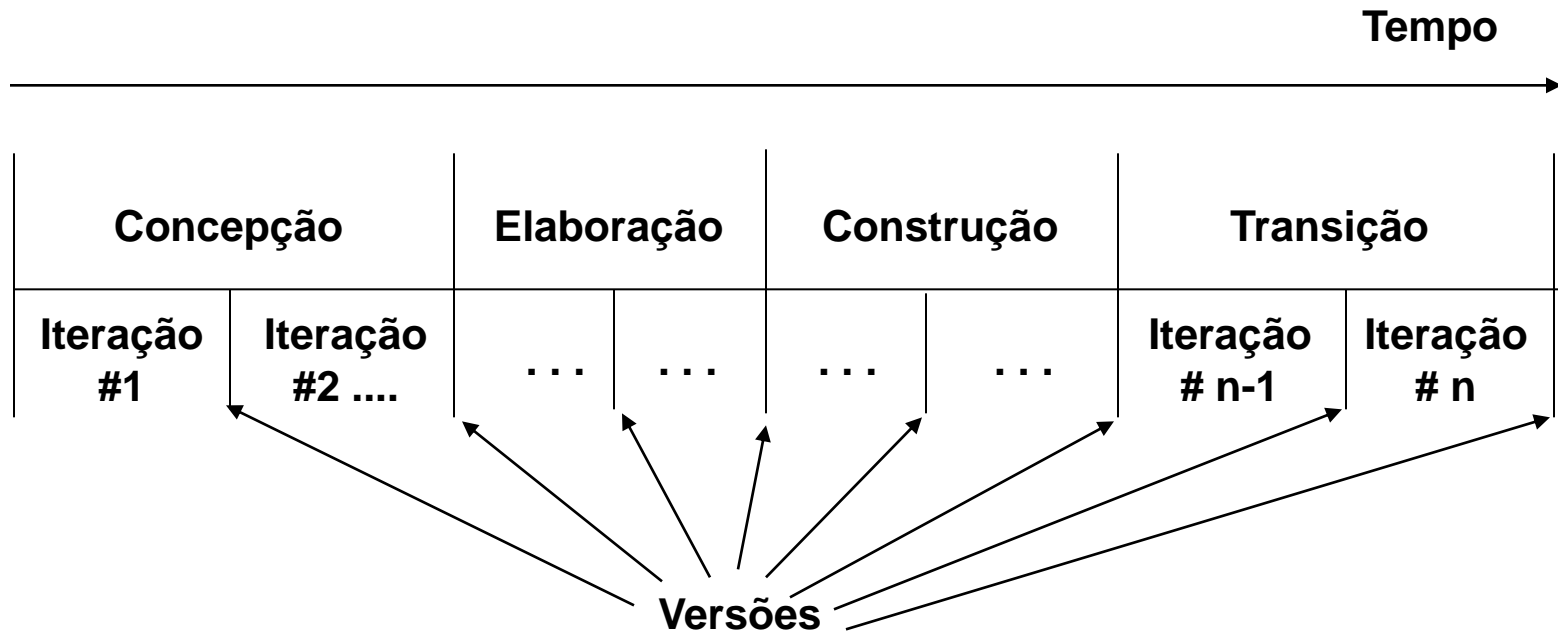


Espiral - Características

- Adaptável a mudanças de requisitos (cuidado com a convergência para uma solução final, no T e \$ definidos)
- Permite a redução dos riscos em projetos
- Permite o acúmulo gradativo de experiência e a homogeneização da equipe



Processo Unificado (UP)



Principal referência: "The Unified Software Development Process", Jacobson, Booch & Rumbaugh, Addison-Wesley.

UP - Características

- Orientado a caso de uso

Os casos de uso são utilizados como o principal recurso p/ o estabelecimento do comportamento desejado do sistema e p/ sua verificação e validação

- Centrado na Arquitetura

Arquitetura (última parte do curso) concebida p/ o novo sistema tem o papel importante no projeto

- Iterativo

- Gerenciamento de sequências de versões executáveis
- Divisão do trabalho em mini-projetos que se agregam ao longo do tempo



UP - Características

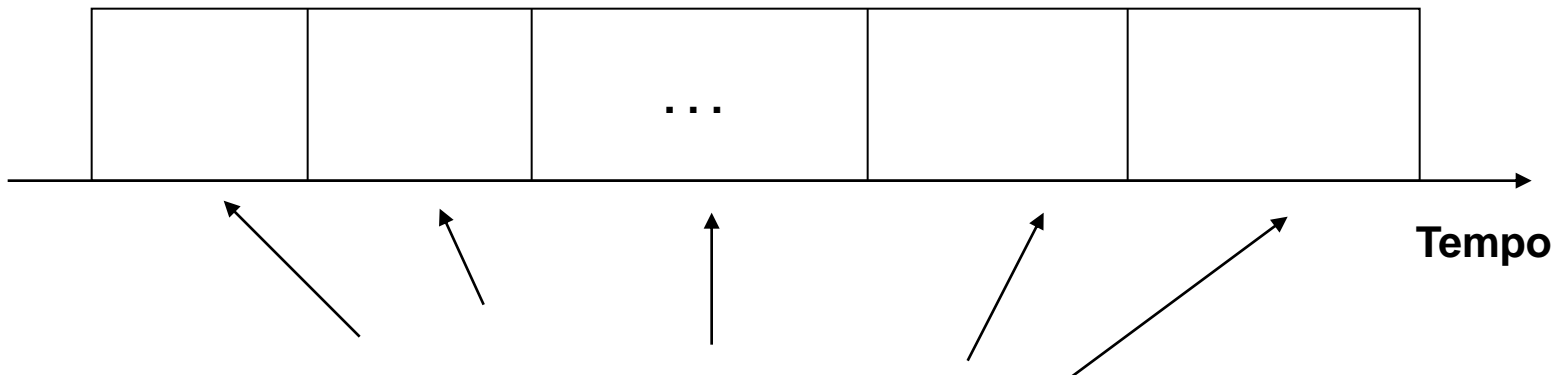
- Incremental
 - Cada nova versão incorpora os aprimoramentos incrementais em relação às anteriores.
 - Cada mini-projeto incrementa o produto.

UP- Ciclo

- Um ciclo possui 4 fases: concepção, elaboração, construção e transição
- Em cada fase são realizadas várias iterações
- **Ao final de cada iteração, temos uma versão executável, interna ou externa, que cresce de modo incremental**

Início

Fim



Fases concluídas com liberação de versão/artefatos



Concepção - Síntese

- O que o sistema deve fazer?
- Qual poderia ser a sua arquitetura?
- Qual o prazo e custo do desenvolvimento?



Elaboração - Síntese

- Análise do domínio do sistema
- Especificação da maioria dos requisitos
- Estabelecimento da arquitetura através de uma visão macro do sistema
- Detalhamento do plano do projeto
- Eliminação de riscos (pelo menos os de mais alto grau)
- Análise de resultados
- Construção de todos os modelos que não foram contemplados na Concepção



Construção - Síntese

- Requisitos restantes
- Descrição dos critérios de aceitação
- Desenvolvimento iterativo e incremental do produto completo
- Teste
- Avaliação para entrada em produção



Transição - Síntese

- Distribuição/implantação do software
- Software entendido como uma versão beta
- Ajustes e correções (quase sempre)
- Avaliação do produto (software e processo)
 - Permite a aquisição de experiência
 - Indica a possibilidade de outro ciclo

Desenvolvimento Ágil

- Contexto típico de desenvolvimento de S/W:
 - Negócio não é bem conhecido e/ou é dinâmico (necessidade constante de manutenção evolutiva e adaptativa);
 - Documentação externa permanece desatualizada em boa parte do ciclo de desenvolvimento;
 - As manutenções são feitas com base no código (*ninguém* consulta/atualiza a documentação externa);
 - Necessidade de disponibilidade rápida de código;

Continua...



Desenvolvimento Ágil

- Contexto típico de desenvolvimento de S/W (cont.):
 - Desenvolvimento centrado no código;
 - Demanda por maiores níveis de qualidade do produto (qualidades externa e interna);
 - Se há processo, ele é rígido e burocratizado;

Continua...



Desenvolvimento Ágil

- Contexto típico de desenvolvimento de S/W (cont.):
 - Assunções clássicas:
 - Usuários especificam exatamente o que querem;
 - Desenvolvedores entregam o que os usuários especificaram.
 - A situação real típica:
 - Usuários não sabem o que querem;
 - Desenvolvedores não entregam o que prometeram.



Desenvolvimento Ágil

- Algumas constatações importantes:
 - A entrega de *software* em prazos e custos estabelecidos quase nunca é conseguida;
 - Excesso de formalismo nos modelos de processo propostos nos últimos 30 anos;
 - Desenvolvimento de *software* é arriscado e difícil de ser gerenciado.



Desenvolvimento Ágil

- Necessita-se de novas metodologias que, dentre outras coisas:
 - Tratem adequadamente requisitos vagos e “mutantes”;
 - Mantenham a liberdade necessária para que os programadores trabalhem de forma efetiva e prazerosa;
 - Produzam *software* de qualidade;
 - Diminuem os encargos da equipe com documentação;
 - Mantenham as expectativas dos usuários atendidas e em níveis realizáveis;
 - Mantenham os projetos gerenciáveis;
 - Tenham base científica.



Desenvolvimento Ágil

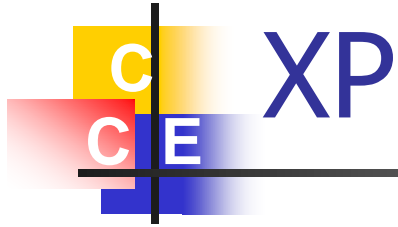
- Manifesto Ágil (*):
 - Assinado por 17 desenvolvedores (Kent Beck *et al*) em Utah – EUA - em fevereiro de 2001.
 - Descreve a essência de um conjunto de abordagens para desenvolvimento de *software* criadas ao longo da última década.

(*)<http://agilemanifesto.org>



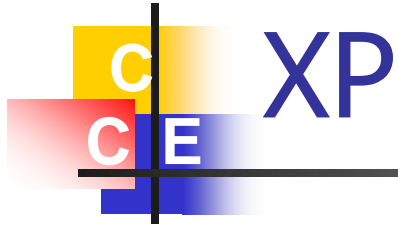
Desenvolvimento Ágil

- Manifesto Ágil – Assunções:
 - Indivíduos e interações são mais importantes que processos e ferramentas;
 - *Software* funcionando é mais importante que documentação completa e detalhada;
 - Colaboração com o cliente é mais importante que (re)negociação de contratos;
 - Adaptação a mudanças é mais importante que seguir o plano inicial.



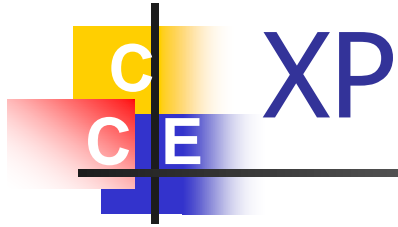
- XP = *Extreme Programming*;
- Metodologia de desenvolvimento de *software* sendo aperfeiçoada nos últimos anos(*);
- As práticas adotadas são “as melhores práticas de engenharia de *software* levadas a níveis extremos”;
- Originou-se das experiências de Kent Beck, Ward Cunningham e Ron Jeffries (também signatários do Manifesto Ágil) no projeto C3 (Chrysler, com Smalltalk e GemStone), 1996-1999.

(*). Correspondendo às edições 1 e 2 do livro “*Extreme Programming Explained: Embrace Change*”

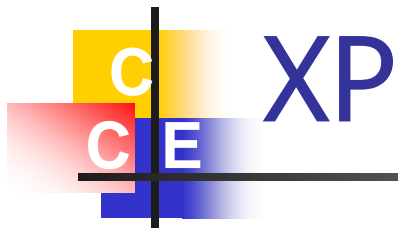


- XP é (*):
 - Uma tentativa de conciliação entre humanidade e produtividade;
 - Um mecanismo de mudança social;
 - Um caminho para a melhora;
 - Um estilo de desenvolvimento;
 - Uma disciplina de desenvolvimento de *software*.

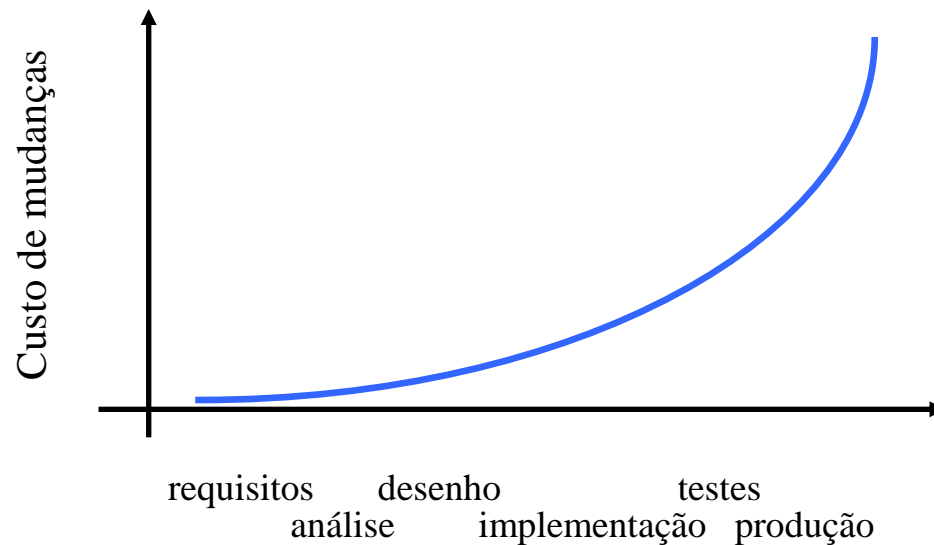
(*) Programação Extrema Explicada, Ed. 1 – Kent Beck

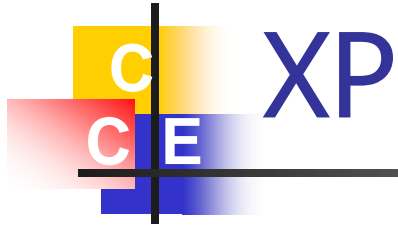


- O custo das modificações
 - Premissa *clássica*: Em processos tradicionais, os requisitos devem ser conhecidos *a priori*, já que...

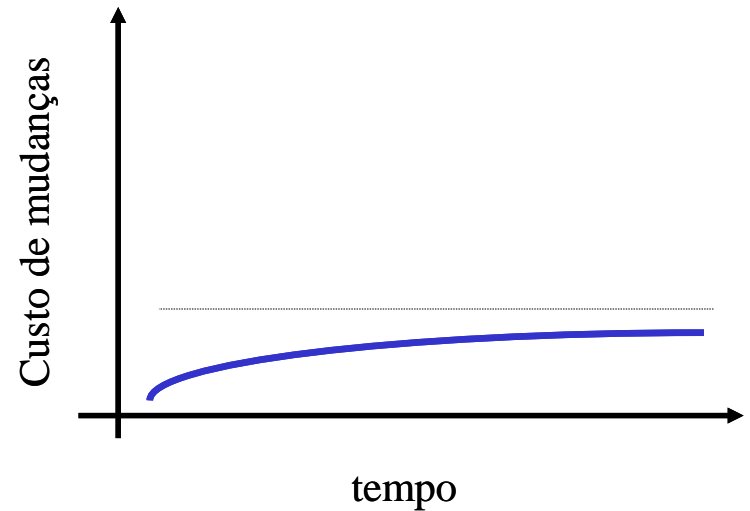


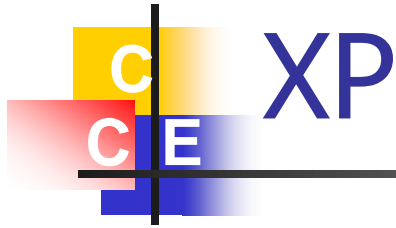
- ... segundo o entendimento tradicional, mudanças de requisitos têm um impacto no custo cada vez maior quanto mais tarde, no ciclo de desenvolvimento, as mesmas ocorrerem.



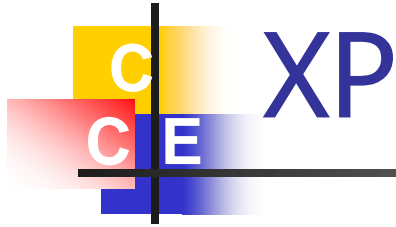


- O objetivo principal da XP é diminuir os custos de mudanças:
 - Com tecnologias e práticas apropriadas, é possível obter-se uma curva de custo de mudanças X tempo do projeto tendendo a um custo constante (máximo).

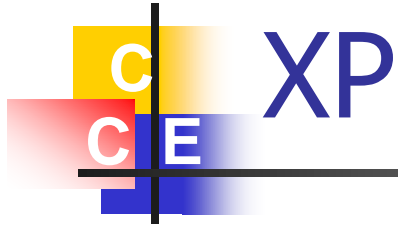




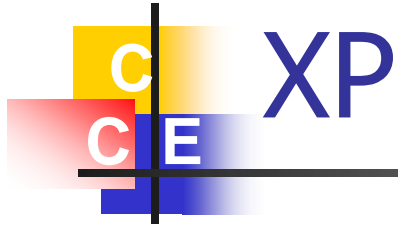
- E quais são essas tecnologias e práticas?
 - Objetos são a tecnologia chave
 - Encapsulamento provê manutenibilidade;
 - Modificações de comportamento, sem alteração de código existente, podem ser (mais facilmente) implementadas através de mudanças nas trocas de mensagens entre os objetos.
 - Projeto simples, sem elementos extras (antecipação de necessidades, flexibilidades não solicitadas, etc.);
 - Testes automatizados aplicados logo após as implementações das modificações;
 - Experiência prática desenvolvida na aplicação de modificações em projetos, provendo auto-confiança na hora em que elas são necessárias.



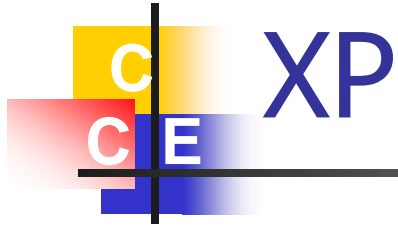
- Nesse contexto, as atitudes dos desenvolvedores ágeis são:
 - Implementam *agora* somente o que precisam *agora*;
 - Tomam as grandes decisões o mais tarde possível (quem sabe não será preciso tomá-las, de fato?);
 - Não implementam flexibilidade desnecessária num dado momento (não antecipam necessidades – quem sabe se serão mesmo necessárias?).



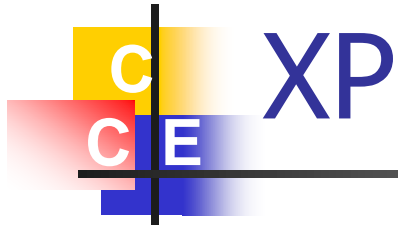
- Constatações:
 - O que muda em um *software*?
 - Os requisitos;
 - A tecnologia;
 - O projeto;
 - O time;
 - ...
 - Mudanças são inevitáveis. É importante saber lidar com elas;
 - Pequenas (às vezes grandes) e constantes mudanças são (quase inevitavelmente) necessárias – a metáfora de se dirigir em uma estrada reta – é o paradigma da XP.



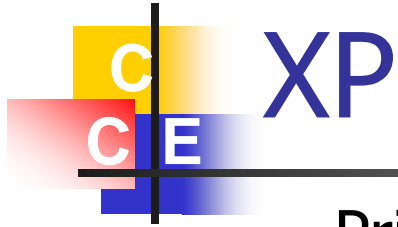
- XP se baseia (atualmente) em
 - 5 valores;
 - 14 princípios e
 - 24 (13+11) práticas.



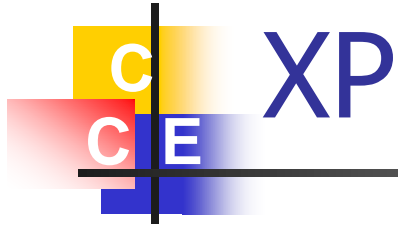
- Os cinco valores da XP são:
 - Comunicação
 - Simplicidade
 - *Feedback*
 - Coragem
 - Respeito



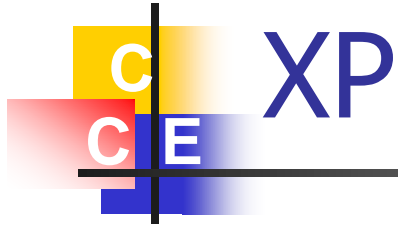
- Valores (vagos, subjetivos) precisam ser transformados em *princípios* (guias) bem definidos para que possam ser usados.
- Princípios são a *ponte* entre os valores e as práticas.



- **Princípios (alguns destaques)**
 - **A Economia**
 - Produzir *software* é agregar valor ao negócio.
 - **Melhora contínua**
 - A história do “está bom, mas pode melhorar”.
 - **Reflexão**
 - Análise constante dos sucessos e das falhas (lições aprendidas)...
 - ... entretanto deve-se fazer mais do que pensar.
 - **Fluxo (de *software*)**
 - Todas as atividades visam à produção de software; não é o caso de haver uma sequência de atividades com a produção do *software* somente ao final.
 - **Responsabilidade aceita...**
 - ... e não imposta;
 - “Obrigado, mas eu passo...” deve ser algo compreensível.



- *Práticas* são atitudes e atividades concretas que o programador XP faz no dia-a-dia. Elas garantem a aplicação dos valores.



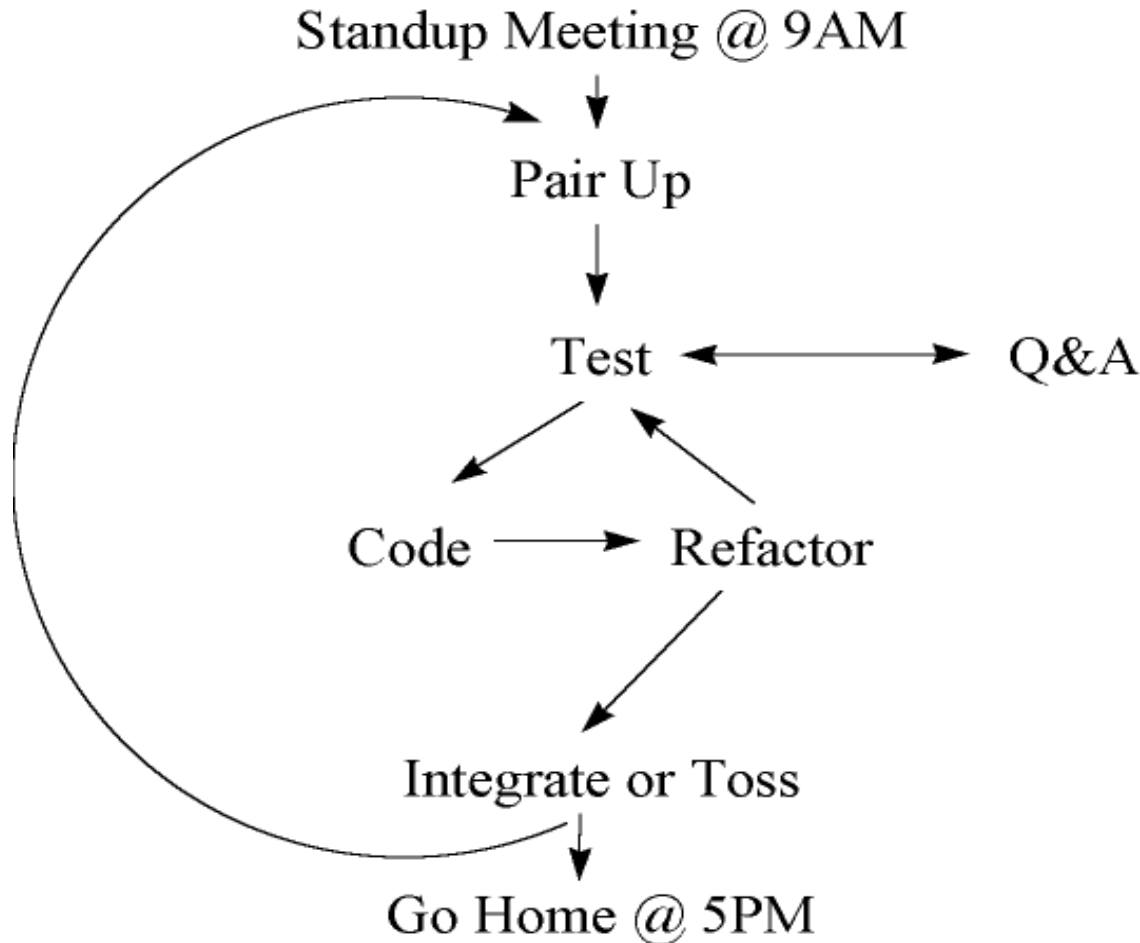
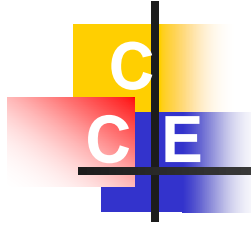
- Práticas (alguns destaques):
 - **Todos juntos:** trabalhar em um ambiente grande, para toda a equipe se ver (*War Room*).
 - **Transparência da informação:** manter as informações correntes do projeto em local de fácil visualização (e.g. um mural).
 - **Trabalho energizado:** manter ritmo de trabalho sustentável. Ter vida fora do trabalho (☺).
 - **Programação em pares:** duas pessoas programando juntas, num processo de interação constante.
 - **User Stories:** planejar e controlar o desenvolvimento através de “pedaços” de funcionalidades do negócio, tais como vistas pelos usuários.

Continua...



XP

O dia-a-dia de um programador XP





Modelo e modelagem de sistemas

- **Modelo** é uma representação de um sistema (ou de um objeto qualquer). É uma abstração da realidade e representa uma seleção de características do mundo real que são relevantes para o propósito com o qual o modelo foi construído.



Modelo e modelagem de sistemas

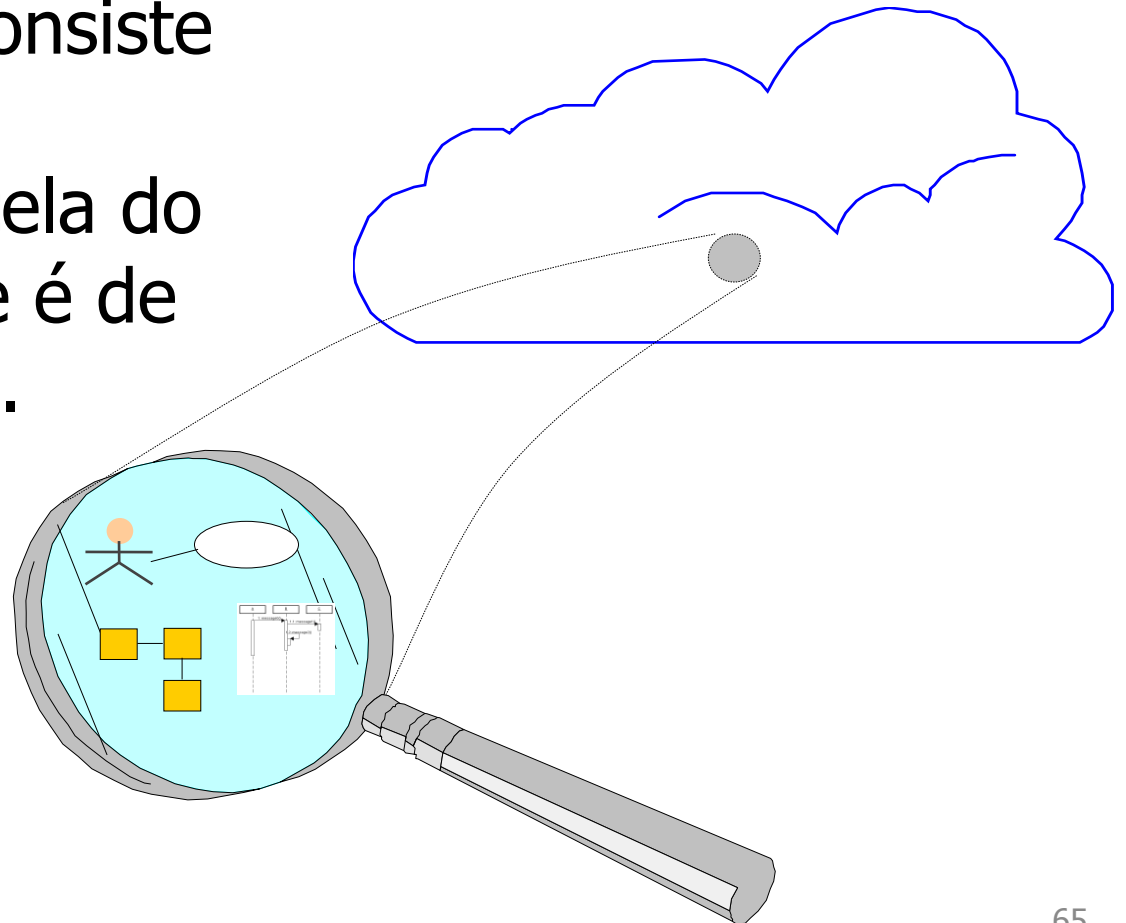
■ Detalhe:

- Todo bom modelo precisa representar
 - A estrutura dos dados (a dimensão de dados);
 - As funções que transformam os dados (a dimensão funcional);
 - As sequências de aplicação das funções (a dimensão temporal) e as demais restrições.



Modelo e modelagem de sistemas

- **Modelagem** consiste em se criar um modelo da parcela do mundo real que é de nosso interesse.





Modelo e modelagem de sistemas

- Por que modelar?
 - Possibilitar o estudo do comportamento do sistema;
 - Possibilitar a discussão de correções, modificações e validação com o usuário, a um custo baixo;
 - Facilitar a comunicação entre os membros da equipe;
 - Documentar o sistema, registrando todas as decisões tomadas durante o projeto.



OOA&D

- Décadas de 70-80 surgiram linguagens OO e híbridas:
 - Smalltalk;
 - C++;
 - Object-Pascal.
- POO usado no contexto acadêmico somente;
- Métodos de projetos OO passaram a ser pesquisados para dar suporte “organizado” ao desenvolvimento de sistemas com linguagens OO.

Continua...



OOA&D

- Resolveram as dificuldades dos paradigmas anteriores:
 - Ok quanto à separação entre dados e processos
 - Diagramas de classes identificam as entidades, os relacionamentos, as operações e responsabilidades.
 - Ok quanto à descontinuidade da análise para o projeto
 - Os diagramas são refinados ao longo do ciclo de vida pelo acréscimo de detalhes em um único nível hierárquico.
 - Ok quanto às diferenças de modelagem entre tipos de sistemas
 - As metodologias e linguagem de especificação adotadas permitem o tratamento de casos em qualquer domínio, indistintamente.



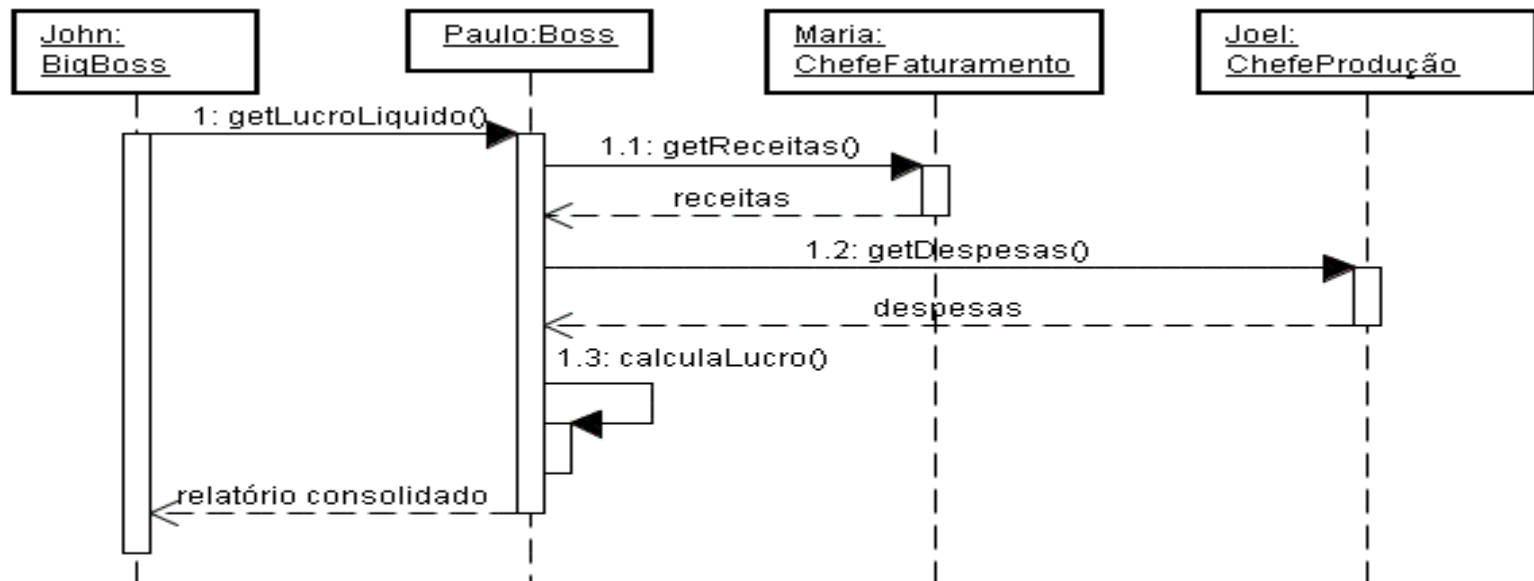
Objetos, o conceito central

Objetos são entidades:

- Que representam coisas concretas ou abstratas do mundo real (um carro, um processo químico);
- Que se categorizam em *classes*;
- Que possuem *estados*;
- Que mantêm relacionamentos entre si;
- Têm responsabilidades e executam operações...

Objetos, o conceito central

- ... participando *colaborativamente*, em sequências pré-definidas (programadas), da execução das funções do sistema;

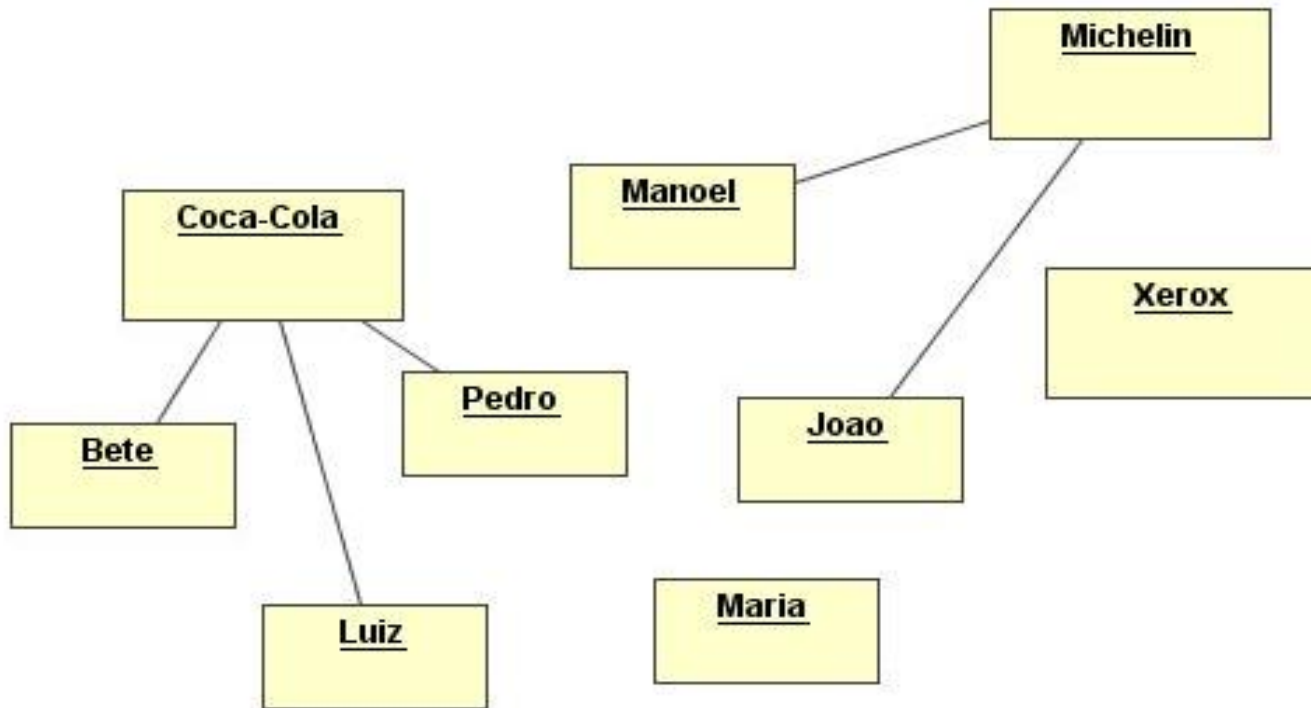




Objetos, o conceito central

- Objetos são entidades que os projetistas definem as características e como vão colaborar para a realização dos objetivos de um sistema.

Objetos, o conceito central





História da UML

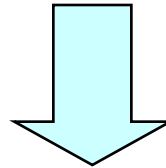
- Meados da década de 90:
 - Massa crítica de idéias produzidas pelas várias metodologias;
 - Necessidade de estabilizar o mercado OO para viabilizar o desenvolvimento de ferramentas CASE OO.



História da UML



Guerra dos métodos



Iniciativa da comunidade no sentido de juntar forças para criar uma linguagem unificada



História da UML

- Dentre os métodos mais importantes destacavam-se:
 - Booch (Grady Booch - Rational Software), bom nas fases de projeto e construção;
 - OOSE (Ivar Jacobson - Objectory), bom na captura de requisitos e análise (abordagem em alto nível de abstração).
 - OMT (Jim Rumbaugh - GE), bom na análise de SIs com uso intensivo de dados.
- Cada um dos três passa a usar também idéias dos outros dois.



História da UML

- Os três – RJB - agora (96/97) juntos na Rational:
 - Iniciaram o processo de padronização da UML, criaram uma proposta inicial e ...
 - ... “passaram a bola” para o OMG, que passou a considerar outras opiniões;
 - Desenvolveram a metodologia unificada e software de apoio à mesma (Objectory) e software de case (Rose)



História da UML

- UML está, atualmente, na versão 2.4 (beta), e o desenvolvimento é gerido pelo OMG;
- Especificação disponível em pdf;
- Versões anteriores (i.e. 1.3, 1.4, 1.5 e 2.X) ainda são bastante empregadas, incluindo CASEs.



Principais Propósitos

- Permitir a modelagem de sistemas, do conceito ao artefato executável, utilizando técnicas OO;
- Contemplar as necessidades de modelagem de sistemas pequenos e simples a grandes e complexos;
- Prover uma linguagem que permita o entendimento e utilização por humanos e por máquinas;
- Ser independente da linguagem de programação e do processo de desenvolvimento;
- Construir modelos precisos, sem ambiguidades e completos;
- Linguagem para visualização do modelo, facilitando o entendimento pela equipe de desenvolvimento e pelos clientes;
- Servir para construir código, embora não seja uma linguagem de programação;
- Servir para documentar sistemas (requisitos, arquitetura, projeto, etc.).



Referências

- Referências importantes:
 - Fowler & Scott, "UML Essencial", Bookman
 - Booch, Rumbaugh & Jabobson, "UML: Guia do Usuário", Campus
 - Larman, "Utilizando UML e Padrões-Uma Introdução à Análise e Projeto Orientados a Objetos, Bookman.
 - ...
- ...Referência Básica (centenas de páginas):
 - Infra-estrutura;
 - Superestrutura;
 - OCL.



Importante

- Aproximadamente 40% da linguagem cobre 98% das necessidades de um projeto comum (Fowler).

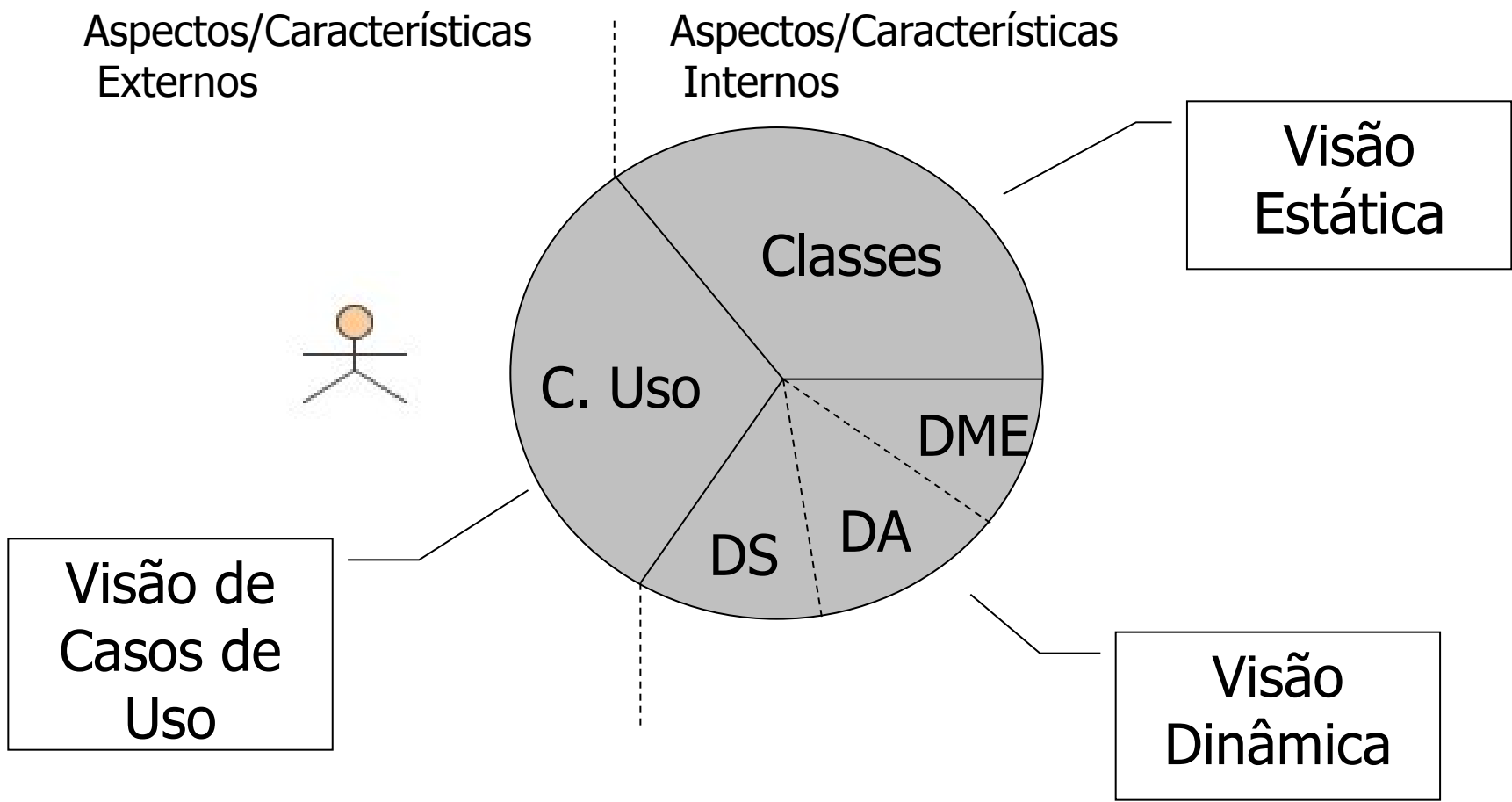


Diagramas da UML

- A UML define 13 diagramas, divididos em três categorias:
 - Diagramas estruturais
 - Classes, objetos, componentes, pacotes, estruturas compostas e implantação.
 - Diagramas comportamentais
 - Casos de uso (usados por algumas metodologias durante a captura de requisitos), atividade e máquina de estados.
 - Diagramas de interação
 - Sequência, comunicação, *temporização* (de tempo), visão geral de interação.



Modelo UML Básico





Minimundos Para o Trabalho

- Opções:
 - Em grupo(s)?
 - Mesmo tema?



Lembrete

- Próximas aulas:

UML – Diagramas de Casos de Uso