Engenharia de Software

Parte 2

Histórico (era pré-ES)

- 1940s: Primeiro computador eletrônico de uso geral – ENIAC
  - Custo estimado de US$ 500.000,00
  - Início da programação de computadores
- 1950s: Primeiros compiladores e interpretadores
- 1960s: Primeiro grande software relatado na literatura – OS/360
  - Mais de 1000 desenvolvedores
  - Custo estimado de US$ 50.000.000,00 por ano
- 1968: Crise do software – nasce a Engenharia de Software

Crise do Software

- Conjunto de problemas que são encontrados no desenvolvimento de software, como:
  - software que não funcionam
  - manutenção de um volume crescente de software existente
  - atendimento a uma demanda cada vez maior, etc.

Crise do Software

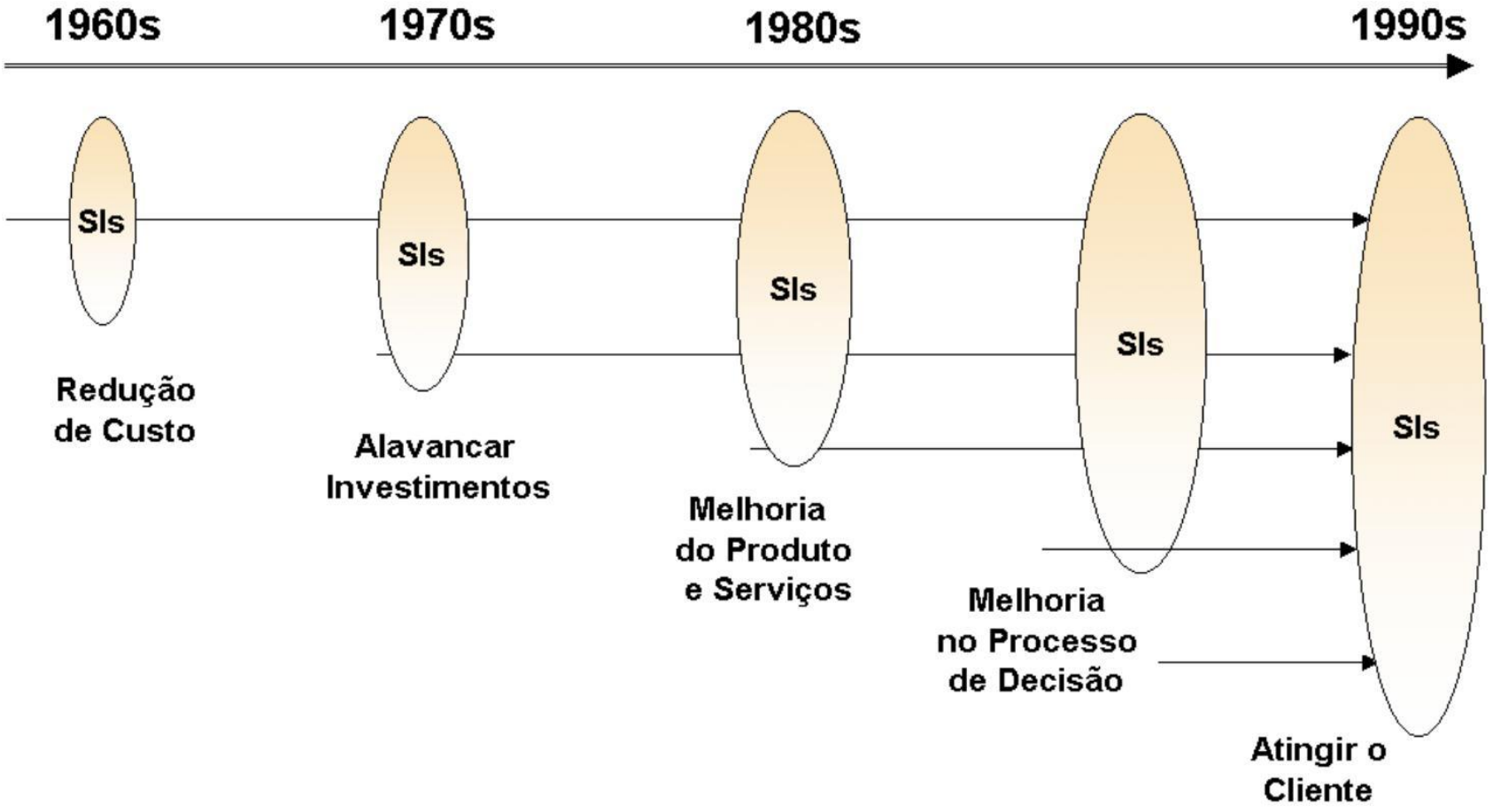
- Relatório do Standish Group (Caos - 1995)
  - Em 1995 os Estados Unidos gastaram $81 milhões em projetos de software que foram cancelados
  - 31% dos projetos foram cancelados antes de estarem concluídos
  - 53% excederam mais de 50% da estimativa de custo
  - Somente 9% dos projetos das grandes empresas foram entregues em tempo e estimativa
  - Em pequenas empresas os números são de 28% e 16% respectivamente

Histórico (era pós-ES)

- 1970s:
  - Lower-CASE tools (programação, depuração, colaboração)
  - Ciclo de vida cascata
  - Desenvolvimento estruturado
- 1980s:
  - Ciclo de vida espiral
  - Desenvolvimento orientado a objetos
- 1990s: Upper-CASE tools
  - Processos
  - Modelagem
- Atualmente:
  - Métodos ágeis
  - Desenvolvimento dirigido por modelos
  - Linhas de produto
  - Experimentação

A Importância do Software

- A complexidade do software vem crescendo ao longo do tempo



Mitos do Software

- Antigas atitudes e hábitos são difíceis de serem modificados, e os remanescentes dos mitos de sw ainda merecem crédito quando nos movimentamos em direção a mais uma década de existência do sw

Mitos Administrativos

- Mito:
  - Temos um manual repleto de padrões e procedimentos para a construção de sw. Isso não oferecerá ao meu pessoal tudo o que eles precisam saber?
- Realidade:
  - O manual pode existir, mas ele será usado? Os profissionais de sw tem conhecimento de sua existência? Ele reflete a moderna prática de desenvolvimentos de sw? É completo?

Mitos Administrativos

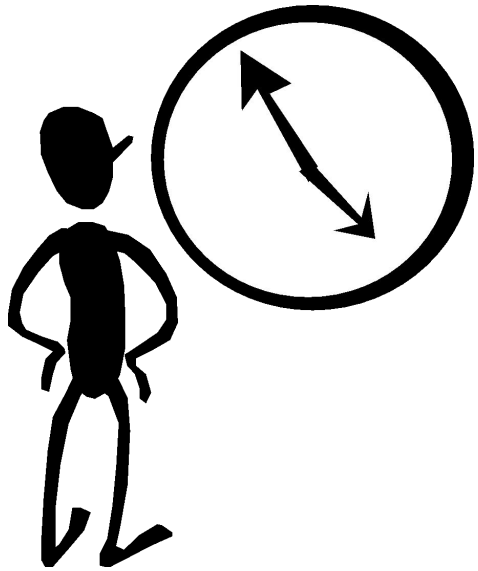
- Mito:
  - Meu pessoal tem ferramentas de desenvolvimento de sw de última geração, afinal de contas compramos os mais novos computadores

Mitos Administrativos

- Realidade:
  - É preciso muito mais do que o último modelo de computador para se fazer um desenvolvimento com alta qualidade. Ferramentas CASE por exemplo, são mais importantes do que o hw para se conseguir boa qualidade e produtividade, mas muitos desenvolvedores ainda não as utilizam

Mitos Administrativos

- Mito:
  - Se nós estamos atrasados nos prazos, podemos adicionar mais programadores e tirar o atraso



Mitos Administrativos

- Realidade:
  - O desenvolvimento de sw não é mecânico como a manufatura. Acrescentar pessoas em projetos atrasados torna-o ainda mais atrasado
    - Quando novas pessoas são acrescentadas, as pessoas que estavam trabalhando devem gastar tempo educando os recém-chegados, o que reduz o tempo despendido num esforço de desenvolvimento produtivo. Pessoas podem ser acrescentadas mas de forma planejada e coordenada

Mitos do Cliente

- Mito:
  - Uma declaração geral dos objetivos é suficiente para se começar a escrever programas – podemos preencher o detalhes mais tarde

Mitos do Cliente

- Realidade:
  - Uma definição inicial ruim é a principal causa de fracasso dos esforços de desenvolvimento de sw. Uma definição formal e detalhada do domínio da informação, função, interfaces, validação, ..., é fundamental. Essas características podem ser determinadas somente depois de cuidadosa comunicação entre o cliente e o desenvolvedor

Mitos do Cliente

- Mito:
  - Os requisitos de projeto modificam-se continuamente, mas as mudanças podem ser facilmente acomodados pois o sw é flexível

Mitos do Cliente

- Realidade:
  - Os requisitos de sw se modificam sim, mas o impacto de mudança varia de acordo com o tempo em que ela é introduzida
    - Se uma séria atenção for dada à definição inicial, os primeiros pedidos de mudança podem ser acomodados facilmente. O cliente pode rever as exigências e recomendar modificações sem causar grande impacto

Mitos do Profissional

- Mito:
  - Assim que escrevermos o programa e o colocarmos em funcionamento, nosso trabalho estará completo
- Realidade:
  - Alguns dados indicam que entre 50 e 70% de todo o esforço gasto num programa serão despendidos depois que ele for entregue pela primeira vez ao cliente

Mitos do Profissional

- Mito:
  - Enquanto não tiver o programa “funcionando”, eu não terei nenhuma maneira de avaliar sua qualidade

Mitos do Profissional

- Realidade:
  - Um dos mecanismos mais efetivos de garantia de qualidade pode ser aplicado desde o começo de um projeto – a revisão técnica formal, que tem sido considerada mais eficiente do que a realização de testes para a descoberta de certas classes de defeitos de sw

Mitos do Profissional

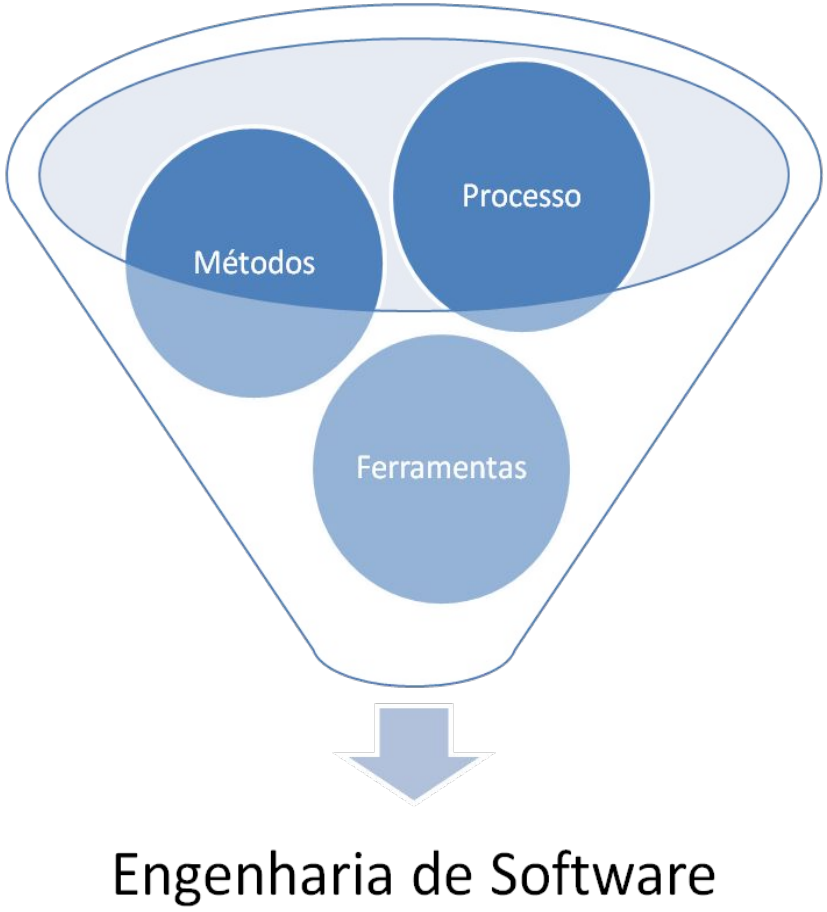
- Mito:
  - A única coisa a ser entregue em um projeto bem sucedido é o programa funcionando

Mitos do Profissional

- Realidade:
  - Um programa funcionando é somente uma parte de uma configuração de sw que inclui vários elementos: plano, especificação de requisitos, projeto, estrutura de dados, teste, ...
  - A documentação forma os alicerces para um desenvolvimento bem sucedido e, o que é mais importante, fornece um guia para a tarefa de manutenção de sw.

Elementos da ES

- A ES compreende um conjunto de etapas que envolve métodos, ferramentas e procedimentos, chamadas Modelo de Processo de Software.



Elementos da ES

- 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 ES

- 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”

O Supermercado de ES

- ES fornece um conjunto de métodos para produzir software de qualidade
- Pense como em um supermercado...
  - Em função do problema, se escolhe o processo, os métodos e as ferramentas
- Cuidado
  - Menos do que o necessário pode levar a desordem
  - Mais do que o necessário pode emperrar o projeto

Modelo de Processo de Sw

- Estes modelos são escolhidos considerando:
  - domínio da aplicação
  - os métodos e as ferramentas a serem usadas
  - os controles e os produtos que precisam ser entregues
  - as características do grupo desenvolvedor
  - o grau de conhecimento sobre o problema

Processo de Software

- Definições (Sommerville)
  - Processo de Software
    - Conjuntos de atividades para especificação, projeto, implementação e teste de sistemas de software
  - Modelo de Processo Software
    - Um modelo de processo software é uma representação abstrata de um processo
    - Apresenta a descrição de um processo a partir de uma perspectiva particular

Fases Genéricas do Processo de Sw

- Fase de Definição: o quê
  - A ES tenta identificar
    - que informação deve ser processada
    - que função e desempenho são desejados
    - que comportamento deve ser esperado do sistema
    - que interfaces devem ser estabelecidas
    - quais as restrições de projeto
    - Quais os critérios de validação
  - Os requisitos chave do sistema e do software são identificados

Fases Genéricas do Processo de Sw

- Fase de Desenvolvimento: como
  - Definição de
    - como os dados devem ser estruturados
    - como a função deve ser implementada dentro da arquitetura do software
    - como os detalhes procedimentais devem ser implementados
    - como as interfaces devem ser caracterizadas
    - como o projeto deve ser traduzido em linguagem de programação
    - como o teste deve ser realizado

Fases Genéricas do Processo de Sw

- Fase de Manutenção: modificações
  - Tipos: Corretiva, Adaptativa, Perfectiva e Preventiva
  - Reengenharia é um tipo de manutenção que normalmente implica ou deriva da reengenharia dos processos de negócios da organização usuária

Fases Genéricas do Processo de Sw

- Atividades Guarda Chuva: tranversais às demais etapas
  - Acompanhamento e controle do projeto de software
  - Revisões técnicas formais
  - Garantia de qualidade de software
  - Gestão de configuração de software
  - Preparação e produção de documentos
  - Gestão de reutilização
  - Medição
  - Gestão de riscos

Modelos de Processo de Sw

- Registro histórico da prática passada e roteiro para o futuro
- Todo processo pode ser caracterizado como um ciclo:
  - Situação atual: o estado atual das “coisas”
  - Definição dos problema: Identifica o problema específico a ser desenvolvido
  - Desenvolvimento técnico: resolve o problema
  - Integração da solução: entrega os resultados

1- Ciclo de Vida Clássico (CVC)

- Às vezes chamado cascata ou linear seqüencial

- CVC requer uma abordagem sistemática, seqüencial ao desenvolvimento de sw que se inicia no nível do sistema e avança ao longo da análise, projeto, codificação, teste e manutenção

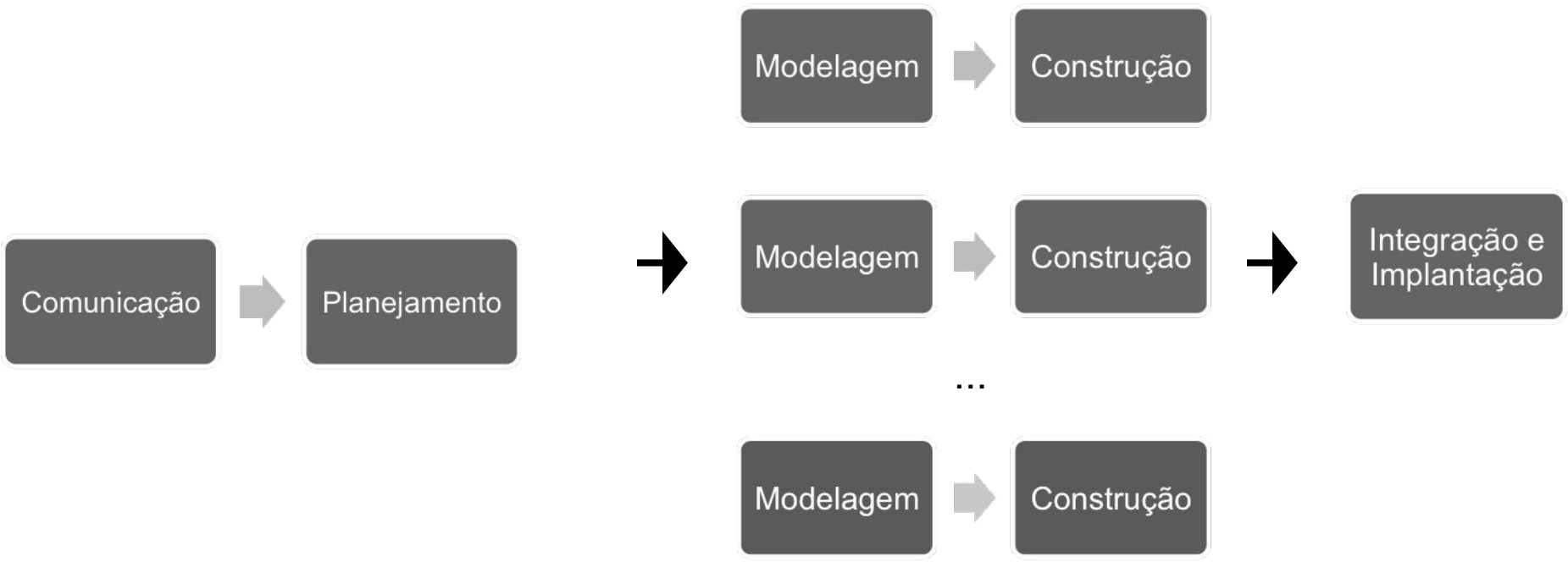
CVC

- Paradigma mais antigo e muito usado
  - Problemas:
    - Versão do trabalho disponível em um ponto tardio do cronograma
    - Erros detectados nesta fase causam problemas maiores
    - Projetos reais raramente seguem o fluxo seqüencial que o modelo propõe
    - Muitas vezes é difícil para o cliente declarar todas as exigências explicitamente

CVC

- Pontos Positivos
  - Tem lugar definido e importante na ES
  - É significativamente melhor do que uma abordagem casual ao desenvolvimento de sw
  - Produz um padrão onde métodos para análise, projeto, codificação, testes, etc podem ser usados
  - Etapas do CVC são muito semelhantes as etapas genéricas aplicadas a todos os paradigmas
  - Continua sendo um modelo muito usado pela ES

2 - Ciclo de vida RAD



Ciclo de vida RAD

- Funcionamento equivalente ao cascata
- Principais diferenças
  - Visa entregar o sistema completo em 60 a 90 dias
  - Múltiplas equipes trabalham em paralelo na modelagem e construção
  - Assume a existência de componentes reutilizáveis e geração de código
- Difícil de ser utilizado em domínios novos ou instáveis

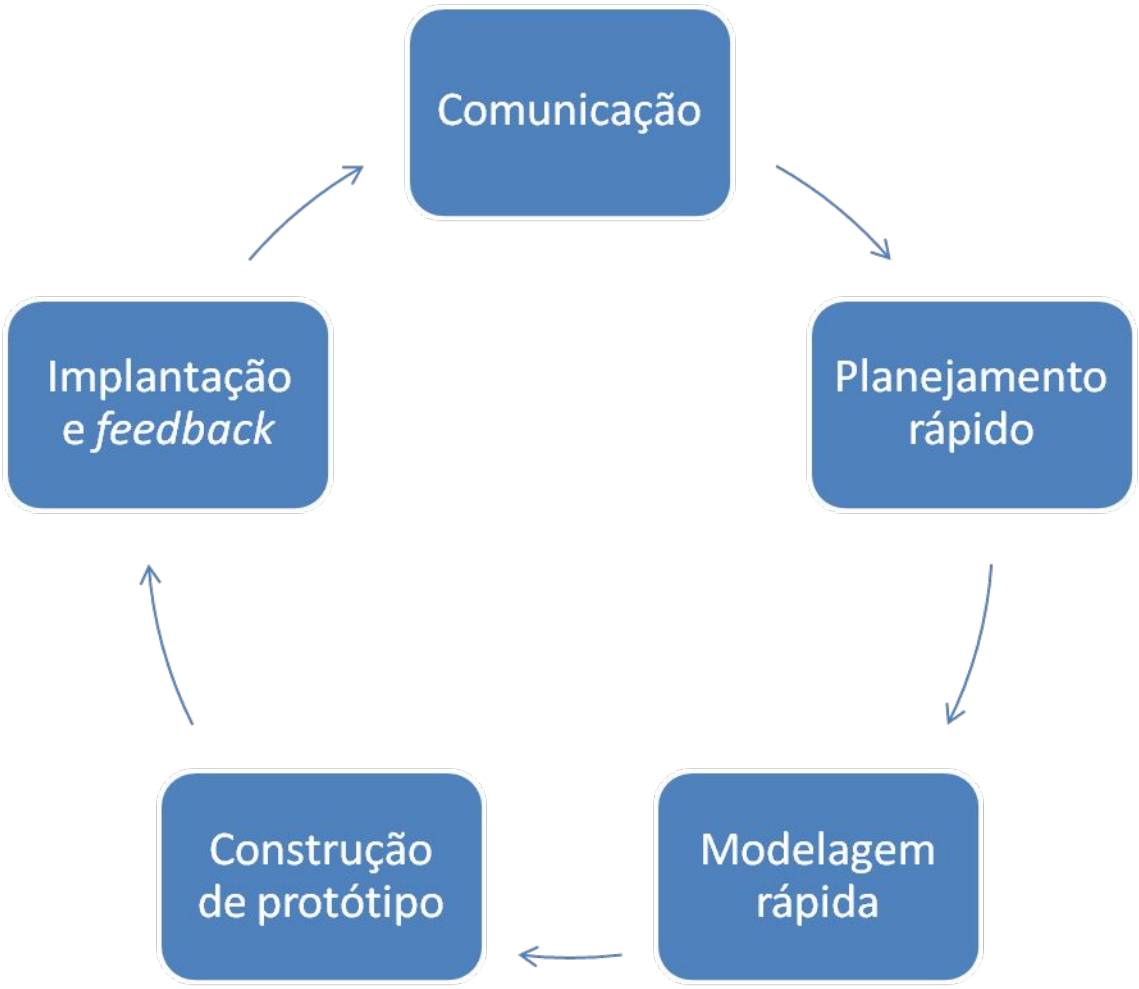
3 - Prototipação

Processo que capacita o desenvolvedor a criar um modelo do sw que será implementado (similar a uma maquete em arquitetura)

Prototipação

- Motivação:
  - O cliente definiu um conjunto de objetivos gerais para o software, mas não identificou requisitos de entrada, processamento e saída detalhados
  - O desenvolvedor pode não ter certeza da eficiência de um algoritmo, da adaptabilidade a um SO...
  - Ou seja…
    - Em situações em que a incerteza está presente, a prototipagem pode representar uma abordagem interessante

Prototipação

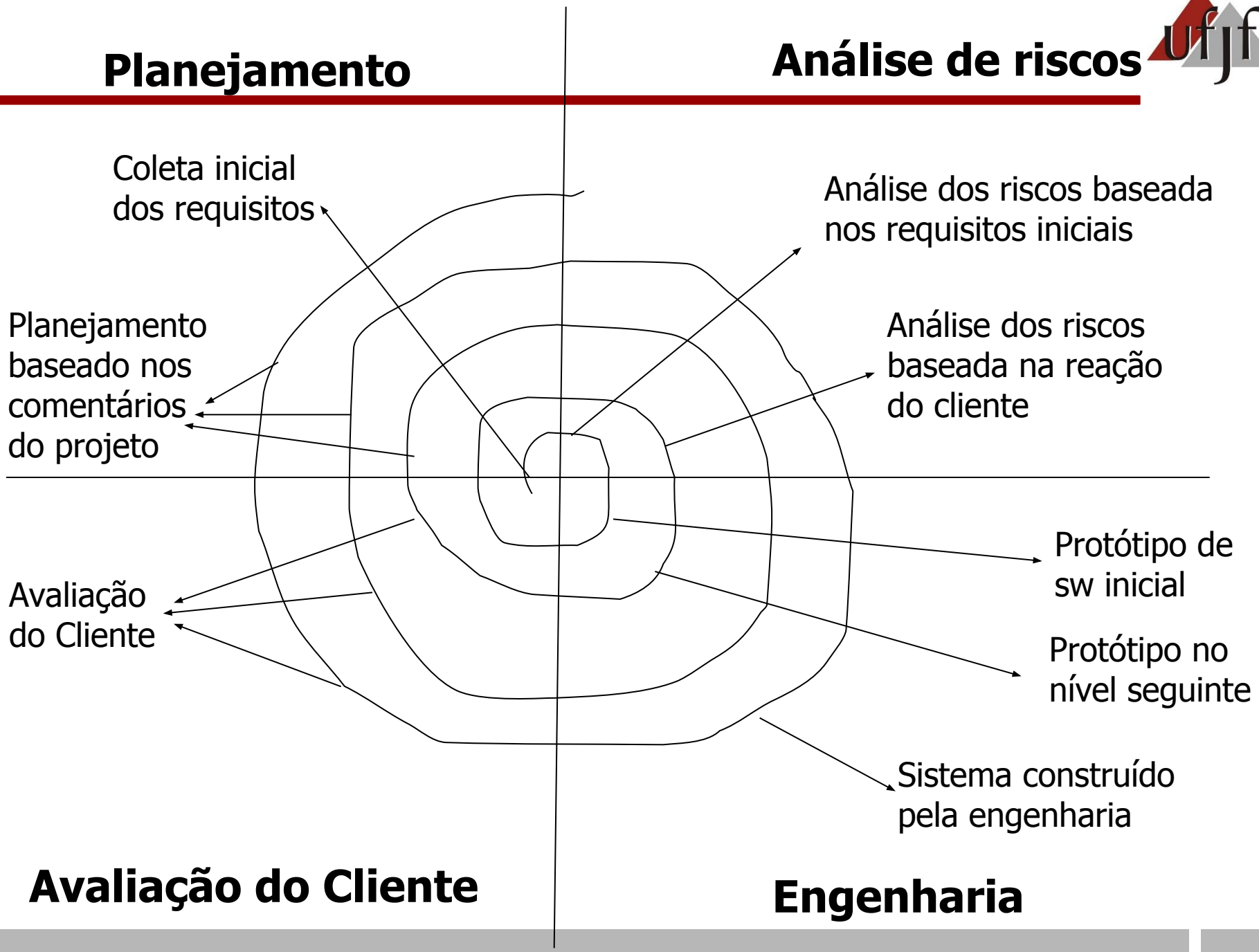


Prototipação

- É um paradigma eficiente da ES
  - Cliente e desenvolvedor devem concordar que o protótipo seja construído para servir como um mecanismo a fim de definir os requisitos
  - Depois será descartado (pelo menos em parte) e o sw real será projetado
- Problemas
  - Cliente vê o protótipo como uma versão do software e exige a sua adequação para o produto, pensando no prazo e não considerando as questões de qualidade e manutenibilidade
  - Desenvolvedor faz concessões de implementação a fim de colocar um protótipo em funcionamento

4 - Modelo Espiral

- Foi desenvolvido para abranger as melhores características tanto do CVC como da prototipação, acrescentando, ao mesmo tempo, um novo elemento – a análise dos riscos


Planejamento
Análise de riscos
Coleta inicial dos requisitos
Análise dos riscos baseada nos requisitos iniciais
Planejamento baseado nos comentários do projeto
Análise dos riscos baseada na reação do cliente
Protótipo de sw inicial
Avaliação do Cliente
Protótipo no nível seguinte
Sistema construído pela engenharia
Avaliação do Cliente
Engenharia

Modelo Espiral

- O modelo espiral é uma abordagem realística para o desenvolvimento de sistemas e de sw em grande escala
- Usa uma abordagem “evolucionária” à ES, capacitando o desenvolvedor e o cliente a entender e reagir aos riscos em cada etapa
- Usa a prototipação como um mecanismo de redução de risco mas, o que é mais importante, possibilita que o desenvolvedor aplique a abordagem de prototipação em qualquer etapa da evolução do produto

Modelo Espiral

- Mantém a abordagem de passos sistemáticos sugerida pelo CVC, mas incorpora-a numa estrutura iterativa que reflete melhor o mundo real
- Exige uma consideração direta dos riscos técnicos em todas as etapas do projeto e se adequadamente aplicado, deve reduzir os riscos antes que eles se tornem problemáticos

Modelo Espiral

- Como os outros paradigmas, apresenta problemas:
  - Pode ser difícil de convencer grandes clientes de que a abordagem evolutiva é controlável
  - Exige considerável experiência na avaliação dos riscos
  - Se um grande risco não for descoberto, ocorrerão problemas

5 - Desenv. Orientado a Reuso

- O desenvolvimento de aplicações hoje em dia sofre grandes pressões:
  - Escalabilidade e complexidade estão em constante crescimento
  - A tecnologia da informação se tornou estratégica no contexto das empresas
  - Necessidade de garantia de qualidade das aplicações
  - Mudanças tecnológicas crescentes
  - Necessidade de interoperação com aplicativos de terceiros

Projeto com reuso

- Construir software a partir de componentes reutilizáveis

Desenvolvimento Orientado a Reuso

- Incorpora as características do modelo espiral e compõe aplicações a partir de componentes de software previamente desenvolvidos ou desenvolvidos durante o processo
- Enfatiza a reutilização, isto é desenvolve para e com reuso

ES baseado no reuso de sw

- Reuso de sistemas de aplicações
  - Todo o sistema pode ser reutilizado pela sua incorporação, sem mudança, em outros sistemas
- Reuso de componentes
  - Componentes de uma aplicação que variam desde subsistemas até objetos isolados podem ser reutilizados
- Reuso de funções
  - Componentes de sw que implementam uma única função podem ser reutilizados

Benefícios de reuso de sw

- Maior confiabilidade
  - Os componentes já foram experimentados e testados em sistemas que já estão funcionando
- Redução dos riscos de processo
  - Menos incertezas sobre as estimativas de custo de desenvolvimento
- Diminuição de custos e tempo na construção de aplicações
- Uso efetivo de especialistas
  - Reuso de componentes ao invés de pessoas

Benefícios de reuso de sw

- Conformidade com padrões
  - Os padrões são embutidos ao reutilizar componentes.
- Desenvolvimento acelerado
  - Evita o desenvolvimento e validação, acelerando a produção
- Maior flexibilidade na manutenção
- Maior qualidade dos produtos

Problemas com o reuso

- Identificação e recuperação de componentes
- Compreensão dos componentes
- Qualidade de componentes
- Paradigmas psicológicos, legais e econômicos
- Aumento nos custos de manutenção
  - Código fonte não disponível

6 - DBC

- Demanda abordagem iterativa para a construção do sw, como o modelo espiral
- No entanto, compõe as aplicações a partir de componentes de sw previamente desenvolvidos
- É muito aplicado em desenvolvimento OO

DBC - Motivação

- Necessidade de melhoria no processo de desenvolvimento de sw: Maior produtividade e menor custo
- Antes: Desenvolvimento de sw em blocos monolíticos
- Solução: Uso de Técnicas de Reutilização de Sw para criação de componentes interoperáveis

DBC

- **Objetivo:** quebra de blocos monolíticos em componentes interoperáveis
- Componentes são construídos/ empacotados com o objetivo de serem reutilizados em diferentes aplicações
- Um componente provê um conjunto de serviços acessíveis através de uma interface bem definida

DBC

- Definição
  - Técnica de desenvolvimento de software onde todos os artefatos, desde códigos executáveis até especificações de interfaces, arquiteturas e modelos de negócio, podem ser construídos pela combinação, adaptação e união de componentes
- Características
  - Segue o princípio de dividir para conquistar
  - Um software construído a partir de componentes é menos complexo que os softwares tradicionais
  - Enfatiza a reutilização em todas as fases

DBC

- **Dificuldade**
  - O que é de fato um componente?
  - Que tecnologias estão envolvidas?
  - Como desenvolver componentes?
  - A síndrome do “não foi inventado aqui”

DBC

- Vantagens do DBC
  - Maior facilidade em solucionar problemas
  - Reduz custos na construção de aplicações
  - Maior confiabilidade (Grau de Maturidade)
  - Facilidade em realizar manutenções
  - Possibilita uma melhor gerência de complexidade: divisão em partes

7 - Desenvolvimento Ágil

- O que é agilidade?
  - “Habilidade de criar e responder a mudanças de maneira a aproveitar as mudanças no ambiente”
- “Linha de pensamento” revolucionária
  - Precisamos parar de tentar evitar mudanças

Agilidade

- Metodologias ágeis
  - Coleção de práticas organizadas para modelagem e desenvolvimento de software
  - “Filosofia” onde muitas “metodologias” se encaixam
    - Definem um conjunto de atitudes e não processo prescritivo
  - Completam alguns métodos existentes

Reação às metodologias tradicionais

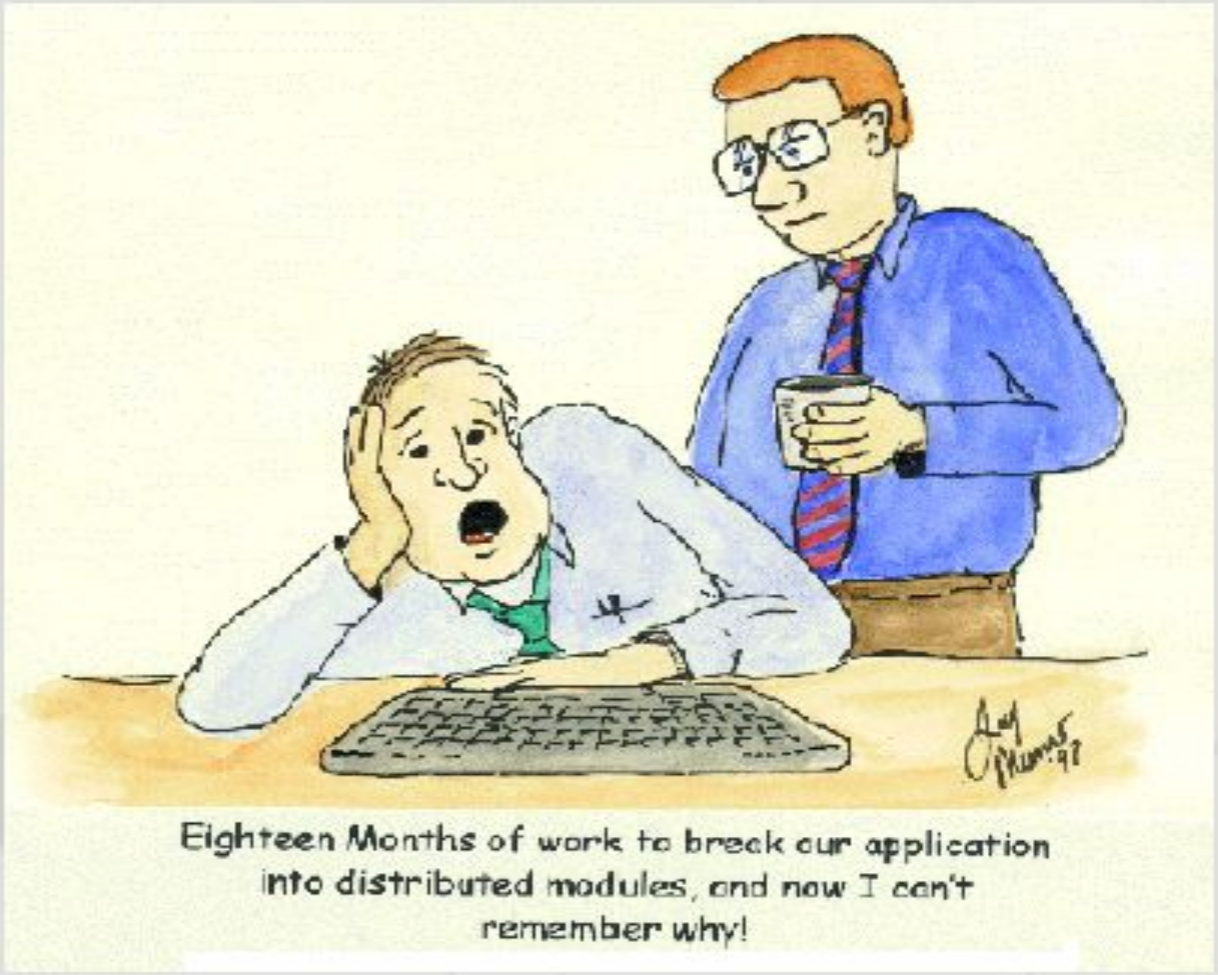
- “Manifesto ágil“ (2001)
  - Princípios
    - Indivíduos e interações são mais importantes que processos e ferramentas
    - Software funcionando é mais importante que documentação completa
    - Colaboração com o cliente é mais importante que negociação com contratos
    - Adaptação às mudanças é mais importante que seguir um plano

Extreme Programming – XP

- Metodologia de desenvolvimento de software aperfeiçoada nos últimos 7 anos
- Ganhou notoriedade a partir da OOPSLA'2000
- Nome principal: Kent Beck

Valores - Simplicidade

Design Simples



Valores - Simplicidade

- O XP utiliza o conceito de simplicidade em inúmeros aspectos do projeto
  - Para assegurar que a equipe se concentre em fazer, primeiro, apenas aquilo que é claramente necessário e
  - Evite fazer o que poderia vir a ser necessário, mas ainda não se provou essencial

Valores - Coragem

Valores - Coragem

- Costuma-se dizer que a única constante em um projeto de software é a mudança
- Clientes mudam de idéia com freqüência
  - Mudam porque aprendem durante o projeto e descobrem problemas mais prioritários a serem solucionados ou formas mais apropriadas de resolvê-los
  - Embora seja natural, gera uma preocupação para a equipe de desenvolvimento que precisa alterar partes do sistema que já estavam prontas, correndo o risco de se quebrar o que já vinha funcionando

Valores – Feedback

- Projetos XP estabelecem formas de encurtar o tempo entre o momento em que uma ação é executada e o seu resultado é observado
  - Assim, desenvolvedores procuram entregar novas funcionalidades no menor prazo possível, para que o cliente compreenda rapidamente as conseqüências daquilo que pediu
  - Os clientes, por sua vez, procuram se manter próximos dos desenvolvedores para prover informações precisas sobre qualquer dúvida que eles tenham ao longo do desenvolvimento

Valores – Respeito

Valores – Respeito

- Dá sustentação a todos os demais valores
  - Membros de uma equipe só se preocuparão em comunicar-se melhor, por exemplo, se importarem uns com os outros
  - Respeito é o mais básico de todos os valores
  - Se ele não existir em um projeto, não há nada que possa salvá-lo
  - Saber ouvir, saber compreender e respeitar o ponto de vista do outro é essencial para que um projeto de software seja bem sucedido

Valores – Comunicação

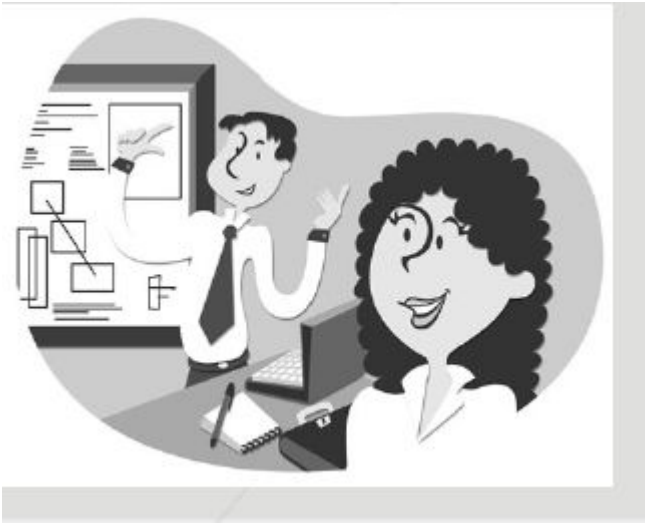
Valores – Comunicação

- O cliente tem problemas que deseja solucionar com o sistema em desenvolvimento e possui algumas idéias sobre que funcionalidades podem resolvê-los
- Por sua vez, desenvolvedores possuem conhecimento sobre aspectos técnicos que influenciam a forma de solucionar o problema do cliente
- Para que os desenvolvedores compreendam o que o cliente deseja e este último entenda os desafios técnicos que precisam ser vencidos, é preciso que haja comunicação entre as partes

XP – Alguns Princípios

Trabalhe com seus clientes

- Talvez o mais extremo dos princípios da XP seja sua insistência em fazer um cliente real trabalhar diretamente no projeto
  - Essa pessoa dever ser um dos eventuais usuários do sistema
    - Fator essencial na XP: Comunicação e Feedback
    - Viabiliza a simplicidade da metodologia
    - Ambiente aberto com quadro branco



Metáforas

- Use metáforas para descrever os conceitos difíceis
  - Uma metáfora é uma forma poderosa de relacionar uma idéia difícil em uma área desconhecida
  - Funcionalidades são informadas através de estórias
    - Estórias devem ser simples

Metáforas

- Cada projeto da XP deve usar um ou mais metáforas para orientar e fornecer um contexto compartilhado
- Exemplo:
  - Help Desk – Outlook

Planejamento

- Projetos de desenvolvimento de sw de qualquer tamanho significativo precisam ser planejados
  - Planejamento serve para fornecer uma compreensão mútua, para informar as partes quanto tempo, aproximadamente, levará o projeto

Stand-up meeting

- Mantenha as reuniões curtas
  - XP usa as reuniões em pé
  - O conceito é simples: não são permitidas cadeiras
  - Se todos estiverem em pé, a reunião será rápida e objetiva
  - A melhor hora para uma reunião em pé é todos os dias pela manhã após todos terem chegado
  - Coloque a equipe em um círculo e faça com que cada membro da equipe faça uma breve atualização do status: o que eles fizeram ontem, o que eles farão hoje

User Story

- Descreve um comportamento geral do sistema
  - Devem se concentrar no comportamento externo do sistema, não devemos ter histórias sobre como armazenar registros em banco de dados, etc...

Teste

- Teste primeiro
  - O teste é outra peça central da filoso O teste ocorre em diversos níveis
    - O primeiro nível inclui os testes de unidade, escritos pelos desenvolvedores de software
      - Verificação feita sobre cada classe
      - Quando uma nova classe ou método entra no sistema, todos os testes são executados

Teste

- Teste de Aceitação
  - Gerado pelo cliente
  - Um teste de aceitação é uma situação concreta que o sistema pode encontrar e que exibe aquele comportamento
  - Testa cada funcionalidade
    - Para cada user story, deve haver pelo menos um teste de aceitação que mostre que o nosso sistema demonstrou aquele comportamento

Programação em pares

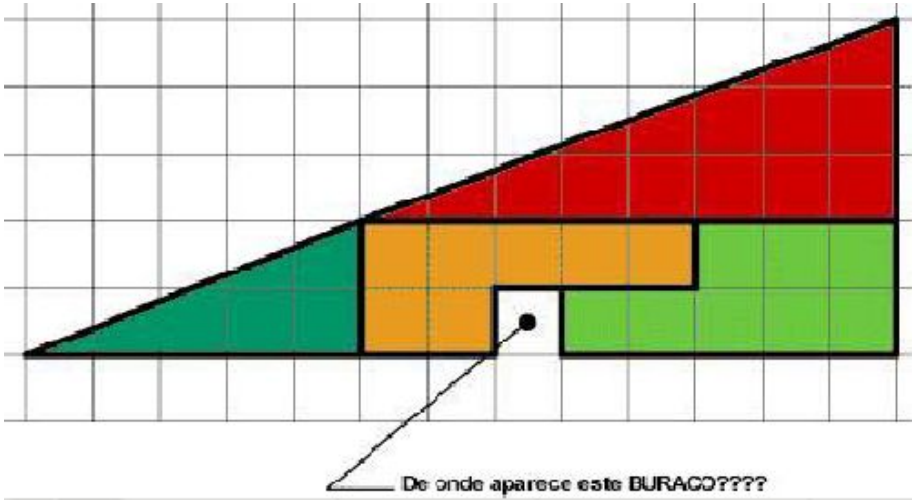
- Duas cabeças pensam melhor do que uma, mas na XP duas cabeças juntas são melhores do que duas cabeças separadas
  - Na XP, as pessoas desenvolvem aos pares
  - O piloto tem o controle do teclado e do mouse e o parceiro observa e ajuda
  - O piloto escreve ativamente os testes ou codifica
  - Um digita e o outro revisa, corrige e sugere
    - Redução drástica de bugs
    - Disseminação de conhecimento
    - Pressão do par
    - Simplicidade e Velocidade

Codifique dentro dos padrões

- Para dar apoio efetivo ao trabalho conjunto, você deve adotar um conjunto de padrões de código no nível da equipe
- Padrões de estilo adotados pelo grupo inteiro
- O mais claro possível
  - XP não se baseia em documentações detalhadas e extensas (perde-se sincronismo)
- Comentários padronizados sempre que necessários
  - “Programas bem escritos dispensam comentários”
- Programação Pareada ajuda muito!

Integre continuamente

- Para trabalhar agressivamente a todo vapor, você precisa integrar continuamente o seu trabalho ao código-base
  - Você deve integrar pelo menos diariamente



Refactoring

- Processo de alterar um sistema de tal forma que ele não altere o comportamento externo do código e melhore a sua estrutura interna
  - Forma disciplinada de limpar o código que minimiza as chances de introdução de bugs
  - Uma [pequena] modificação no sistema que não altera o seu comportamento funcional
  - Mas melhora alguma qualidade não-funcional:
    - simplicidade
    - flexibilidade
    - clareza e desempenho

Sem Refactoring

Com Refactoring

Não se desgaste

- Ninguém faz seu melhor trabalho quando está estressado, sob pressão e cansado
  - É curioso que essas sejam exatamente as condições que prevalecem em nosso setor
  - A idéia é não trabalhar mais do que aquilo que funciona para você

RUP

- Processo Unificado (*Rational Unified Process* – RUP)
  - Criado para apoiar o desenvolvimento orientado a objetos
  - Fornece uma forma sistemática para se obter reais vantagens no uso da Linguagem de Modelagem Unificada (UML)

RUP

- De fato, ele não é exatamente um processo, mas sim uma infraestrutura genérica de processo que pode ser especializada para uma ampla classe de sistemas de software, para diferentes áreas de aplicação, tipos de organização, níveis de competência e tamanhos de projetos

Melhores Práticas do RUP

- Desenvolvimento de software interativo
- Gerenciamento de Requisitos
- Arquitetura baseada em Componentes
- Modelo de Software Visual (UML)
- Verificação contínua da qualidade do Software
- Gerenciamento e controle de mudanças

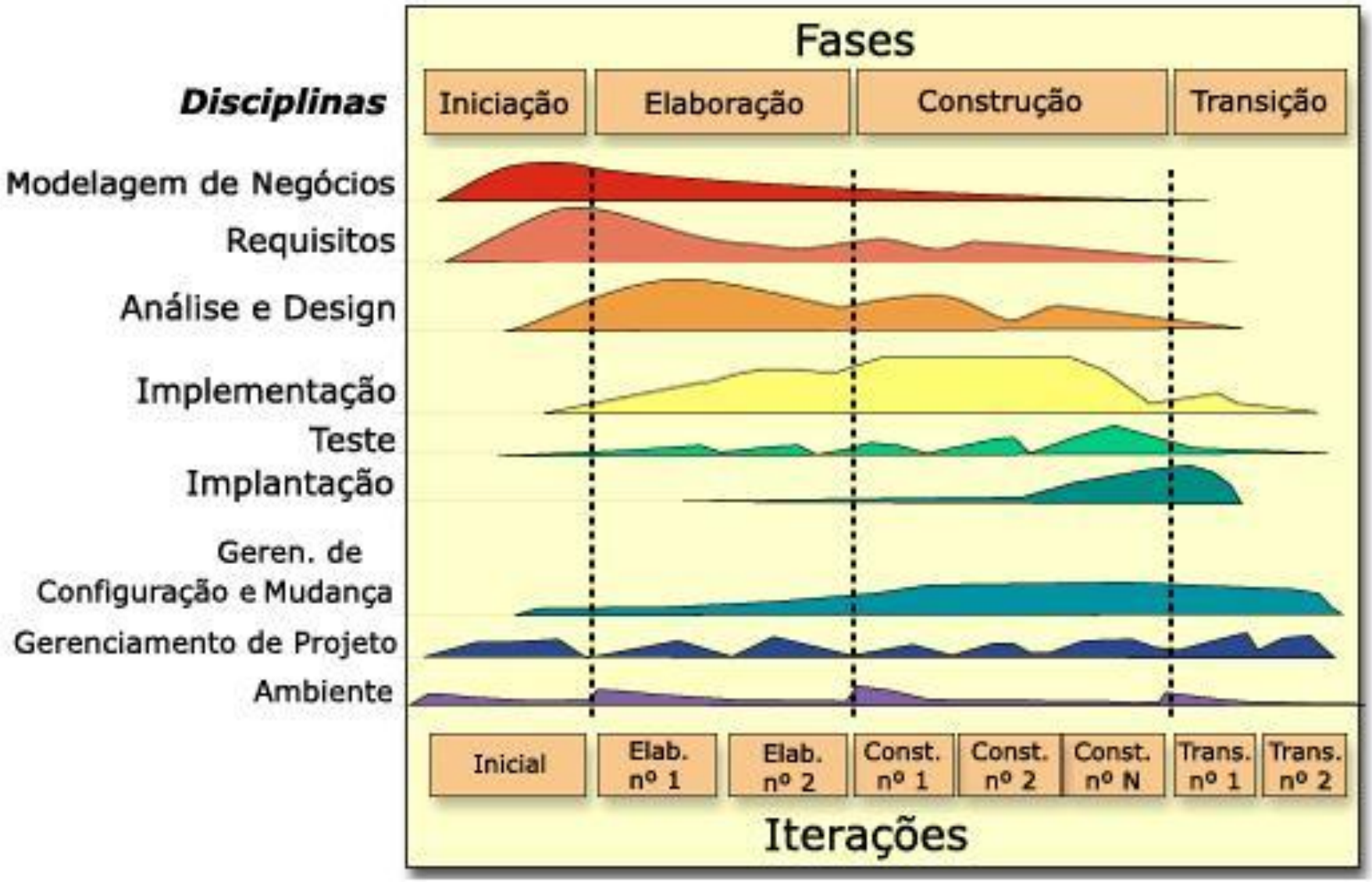
Fases do RUP

- O RUP repete uma série de ciclos durante a vida de um sistema
- Cada ciclo termina com uma versão do produto, e é composto por quatro fases, onde cada uma possui objetivos e conteúdo associados:
  - Iniciação
  - Elaboração
  - Construção
  - Transição

Fases do RUP


Concepção
Estabelecer o escopo e viabilidade econômica do projeto
Elaboração
Eliminar principais riscos e definir arquitetura estável
Construção
Desenvolver o produto até que ele esteja pronto para beta testes
Transição
Entrar no ambiente do usuário

Fases do RUP


Fases
Disciplinas
Iniciação
Elaboração
Construção
Transição
Modelagem de Negócios
Requisitos
Análise e Design
Implementação
Teste
Implantação
Geren. de Configuração e Mudança
Gerenciamento de Projeto
Ambiente
Inicial
Elab. nº 1
Elab. nº 2
Const. nº 1
Const. nº 2
Const. nº N
Trans. nº 1
Trans. nº 2
Iterações