

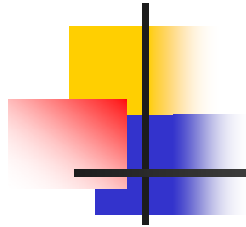


Engenharia de Software

Alessandreia Marta de Oliveira
alessandreia@gmail.com



Princípios Centrais



- “Não tenho idéia de que horas são.
- Não há janelas nesse escritório nem relógio.
- Joel e eu temos programado durante dias.
- Temos um erro, um teimoso de um erro...
- Mais tarde quando sairmos dessa sala vou me importar muito com quem, para quem e com que finalidade eu estou escrevendo software...
- Sou um engenheiro de software”



Princípios Centrais

- Princípios
 - “Uma importante lei ou pressuposto subjacente exigido em um sistema de raciocínio”
- David Hooker propôs sete princípios Centrais da prática e da ES com um todo



7 princípios de Hooker

- Tem que existir uma razão para se fazer sw
 - Se não for possível identificar essa razão, é melhor não fazer
 - Fazer software, em última instância, consiste em “agregar valor para o usuário”
 - É importante enxergar os reais requisitos do software!
- Keep it simple, sir! (KISS)
 - “um projeto deve ser o mais simples possível, mas não mais simples que isso”
 - As soluções mais elegantes normalmente são simples
 - Fazer algo simples usualmente demanda mais tempo do que fazer de forma complexa



7 princípios de Hooker

- Mantenha o estilo
 - O projeto de um sw deve seguir um único estilo
 - A combinação de diferentes estilos corretos pode levar a um software incorreto
 - Padrões e estilos devem ser estabelecidos no início e seguidos por todos
- O que é produzido por você é consumido por outros
 - Sempre especifique, projete e codifique algo pensando que outros vão ler
 - Sempre exija qualidade nos produtos que você consome e forneça qualidade nos produtos que você produz



7 princípios de Hooker

- Esteja pronto para o futuro
 - Sistemas de boa qualidade têm vida longa
 - Projete desde o início pensando na manutenção
- Planeje para reutilização
 - Pense no problema geral, e não só no problema específico
 - Busque por soluções já existentes
- Pense!
 - “plano é desnecessário, mas planejar é indispensável” – D. Eisenhower
 - Avalie alternativas
 - Mitigue os riscos



Exercício

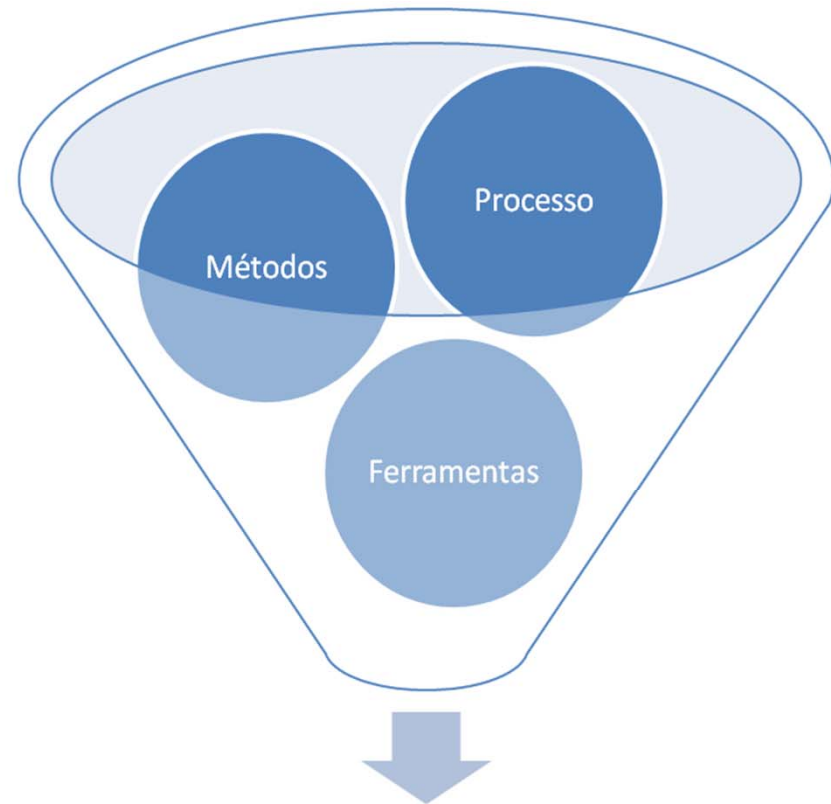
- O que acharam dos princípios de Hooker?
 - Apresentem uma avaliação separada de cada um deles.
- Se todo engenheiro de software e toda equipe simplesmente seguissem os setes princípios de Hooker, muitas das dificuldades pelas quais passamos seriam eliminadas.
 - Concordam? Por que?



Elementos da ES

Elementos da ES

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



Engenharia 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”



Elementos da ES

1. Coloque em uma panela funda o leite condensado, a margarina e o chocolate em pó.
2. Cozinhe [no fogão] em fogo médio e mexa sem parar com uma colher de pau.
3. Cozinhe até que o brigadeiro comece a desgrudar da panela.
4. Deixe esfriar bem, então unte as mãos com margarina, faça as bolinhas e envolva-as em chocolate granulado.

**O que é
processo,
método ou
ferramenta?**

<http://tudogostoso.uol.com.br/receita/114-brigadeiro.html>



Elementos da ES

1. **Coloque** em uma **panela** funda o leite condensado, a margarina e o chocolate em pó.
2. **Cozinhe** [no **fogão**] em fogo médio e **mexa** sem parar com uma **colher de pau**.
3. **Cozinhe** até que o brigadeiro comece a desgrudar da **panela**.
4. Deixe esfriar bem, então **unte** as mãos com margarina, **faça as bolinhas** e **envolva-as** em chocolate granulado.



método



ferramenta

Processo

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



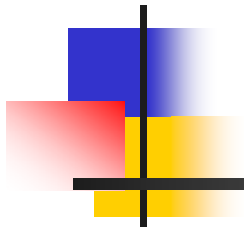
- Atividades Guarda Chuva: transversais às demais etapas
 - Acompanhamento e controle do projeto de software
 - Revisões técnicas formais
 - Garantia de qualidade de software
 - Gestão de configuração de software
 - Preparação e produção de documentos
 - Gestão de reutilização
 - Medição
 - Gestão de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

Modelos de Processo Software





Modelos de ciclo de vida

- Existem alguns processos pré-fabricados
 - Esses processos são conhecidos como modelos de ciclo de vida
 - Esses processos apresentam características predefinidas
- Devem ser adaptados ao contexto real de uso
 - Características do projeto
 - Características da equipe
 - Características do cliente

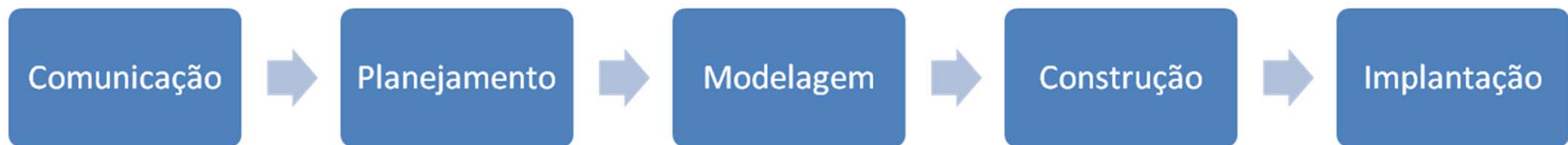


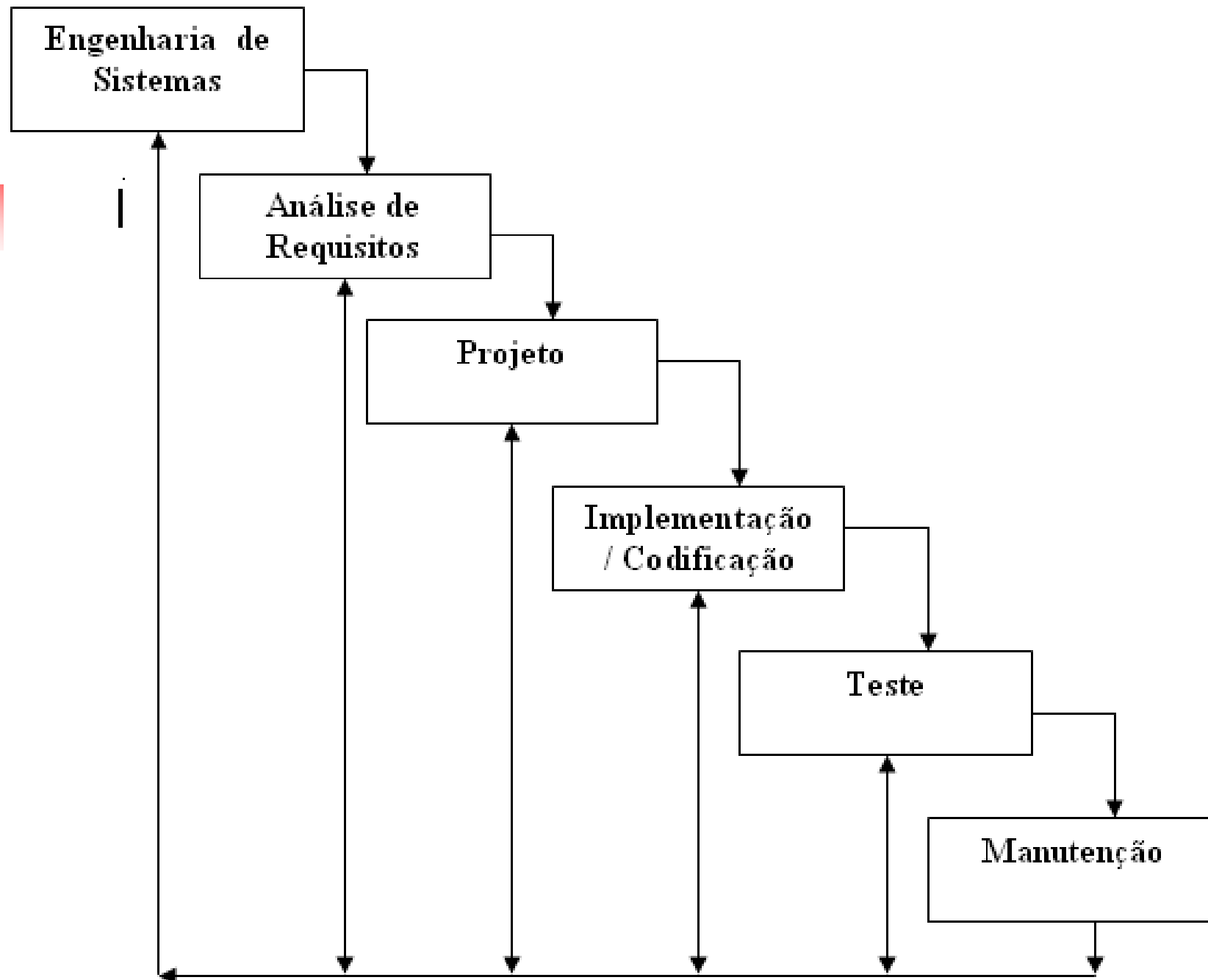
Ciclo de Vida Clássico



Ciclo de Vida Clássico (CVC)

- Às vezes chamado modelo 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







Engenharia de Sistemas

- Requisitos amplos, a nível de sistema, que podem envolver hw, bd, ...
 - Como o sw sempre faz parte de um sistema mais amplo, o trabalho inicia-se com o estabelecimento dos requisitos para todos os elementos do sistema e prossegue com a atribuição de certo subconjunto desses requisitos ao sw
 - Esta visão do sistema é essencial quando o sw deve fazer interface com outros elementos, tais como hw, pessoas e bd



Análise de requisitos de Sw

- Domínio da informação, função, desempenho, interfaces
 - Processo da coleta dos requisitos é intensificado e concentrado especificamente no sw
 - Para entender o sistema a ser construído, o analista deve compreender o domínio da informação para o sw, bem como a função, desempenho e interface exigidos
 - Os requisitos tanto para o sistema como para o sw, são documentados e revistos com o cliente



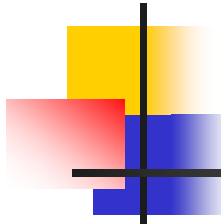
Projeto

- Estruturas de dados, arquitetura de software, especificação de programas, detalhamento de interfaces
 - O projeto é de fato, um processo de múltiplos passos que se concentra nos quatro atributos distintos do programa citados
 - O processo de construção do projeto traduz as exigências numa representação do sw que pode ser avaliada quanto à qualidade antes que a codificação se inicie
 - Como os requisitos, o projeto é documentado e torna-se parte da configuração do sw



Implementação/Codificação

- O projeto deve ser traduzido em uma linguagem de programação
- Se o projeto for bem detalhado, a codificação pode ser executada mecanicamente



Teste

- Assim que a codificação termina iniciam-se os testes de programa
 - Aspectos lógicos: Garantir que todas as instruções foram testadas
 - Aspectos Funcionais: Realizam-se testes para descobrir erros e garantir que todas as entradas definidas produzam resultados reais e esperados



Manutenção

- Mudanças no software após ser entregue ao cliente. Estas mudanças podem ocorrer devido:
 - Erros
 - Alterações no ambiente externo (troca de sistema operacional; novo periférico; etc ...)
 - Cliente solicita acréscimos/alterações funcionais ou de desempenho
- A manutenção reaplica cada uma das etapas precedentes do ciclo de vida a um programa existente, e não a um novo



Conclusão

- 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



Conclusão

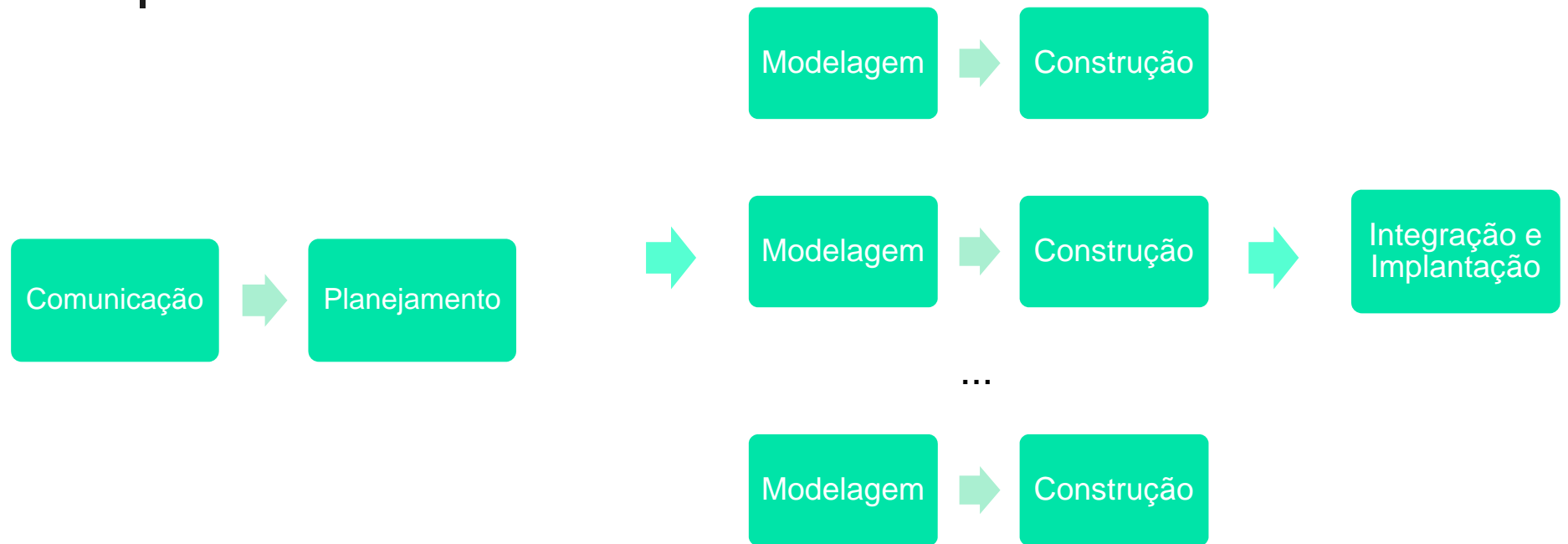
- Pontos Positivos
 - Tem lugar definido e importante na ES
 - É significativamente melhor do que uma abordagem casual ao desenvolvimento de sw
 - Produz um padrão onde métodos para análise, projeto, codificação, testes, etc podem ser usados
 - Etapas do CVC são muito semelhantes as etapas genéricas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



Rapid Application Development



Ciclo de vida RAD



tempo





Ciclo de vida RAD

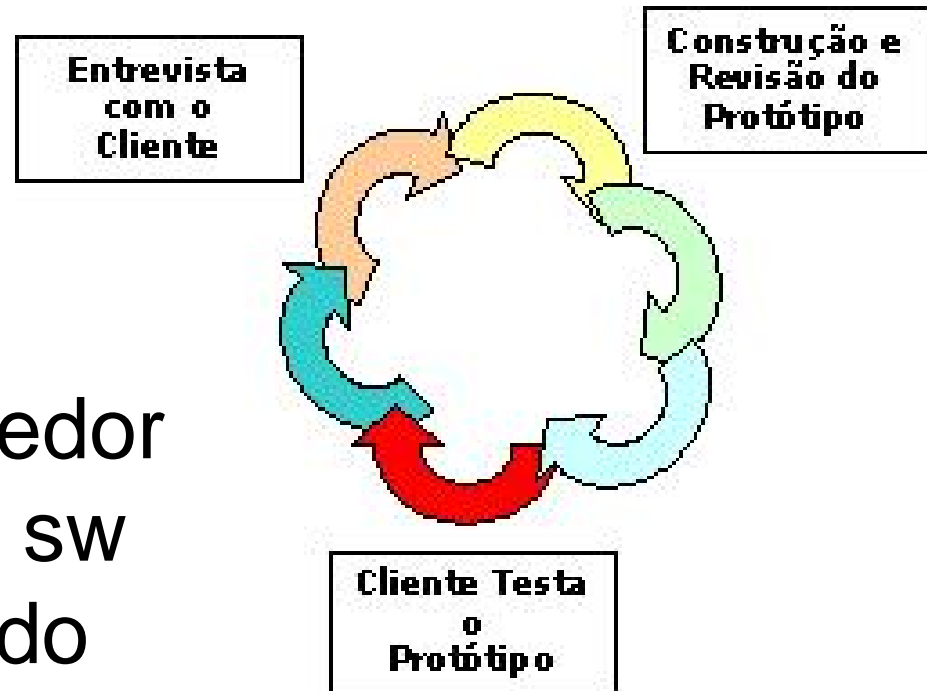
- Funcionamento equivalente ao cascata
- Principais diferenças
 - Visa entregar o sistema completo em 60 a 90 dias
 - Múltiplas equipes trabalham em paralelo na modelagem e construção
 - Assume a existência de componentes reutilizáveis e geração de código
- Difícil de ser utilizado em domínios novos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



Prototipação

Prototipação

- É um 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 da forma que a interação Homem-Máquina deve assumir



Prototipação

- Ou seja...
 - Em situações em que a incerteza está presente na definição de requisitos, objetivos e procedimentos a prototipagem pode representar uma abordagem interessante



Prototipação

- O paradigma (modelo, padrão) da prototipação pode assumir uma das três seguintes formas:
 - Um protótipo em papel ou modelo baseado em PC que retrata a interação Homem-Máquina de maneira a capacitar o cliente (usuário) a entender quanta interação ocorrerá



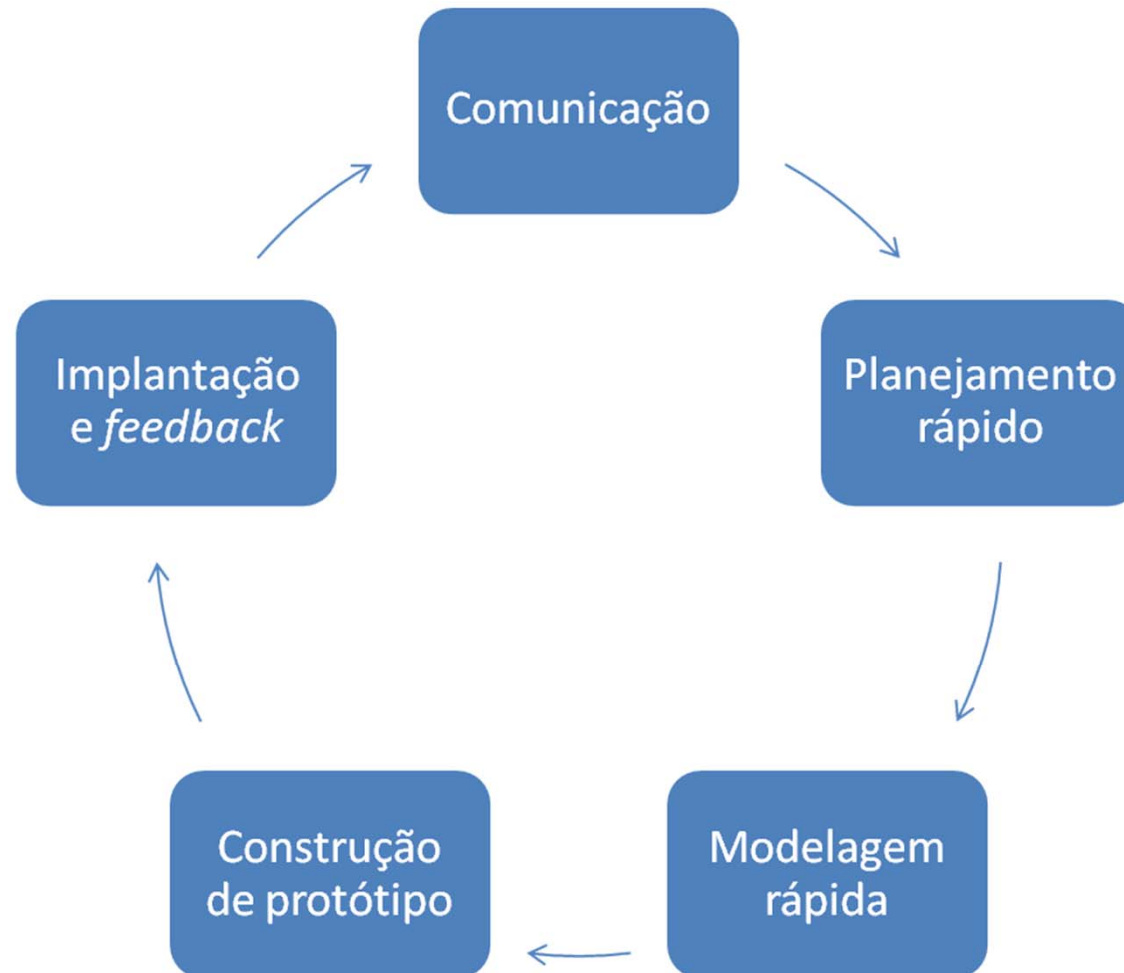
Prototipação

- (Cont.)

- Um protótipo de trabalho que implementa algum subconjunto da função exigida do software
- Um programa que executa parte ou toda a função desejada, mas que tem outras características que serão melhoradas em um novo esforço de desenvolvimento



Prototipação





Prototipação

- O desenvolvedor e o cliente se reúnem e definem os objetivos globais para o sw, identificam as exigências conhecida e esboçam as áreas em que uma definição adicional é obrigatória
- Ocorre a elaboração de um “projeto rápido” que se concentra na representação daqueles aspectos do sw que serão visíveis ao usuário (abordagens de entrada e formatos de saída)



Prototipação

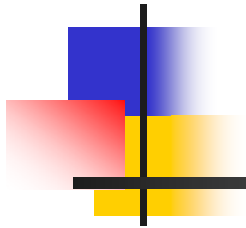
- O projeto rápido leva à construção de um protótipo que é avaliado pelo cliente/usuário e é usado para refinar os requisitos para o sw a ser desenvolvido
- Um processo de iteração ocorre quando é feita uma “sintonia fina” do protótipo para satisfazer as necessidades do cliente, capacitando, ao mesmo tempo, o desenvolvedor a compreender melhor aquilo que precisa ser feito



Conclusão

- 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

Modelos de Desenvolvimento Evolucionários



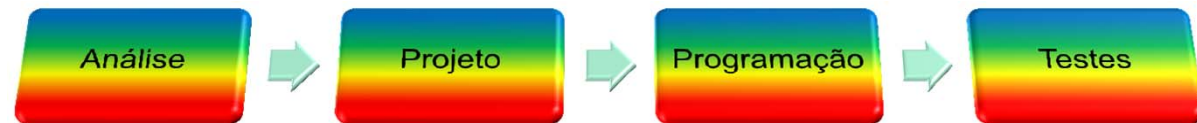
Desenvolvimento Evolucionário



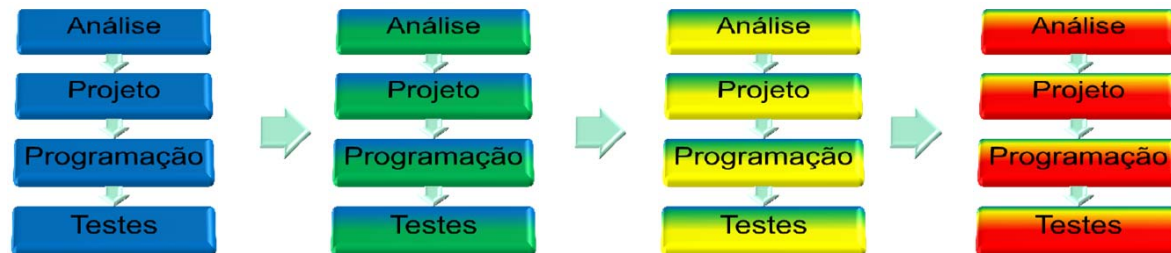
- O software evolui durante um período de tempo
- Requisitos do negócio e do produto mudam à medida que o desenvolvimento prossegue
- Prazos reduzidos de mercado exigem versão reduzida
- Os modelos evolucionários são iterativos e permitem o desenvolvimento de versões cada vez mais completas do software
- Para a maioria dos grandes sistemas, existe a necessidade de utilizar diferentes abordagens para diferentes partes do sistema
 - Abordagem híbrida

Cascata x Evolutivo

Ciclo de vida cascata



Ciclo de vida evolutivo





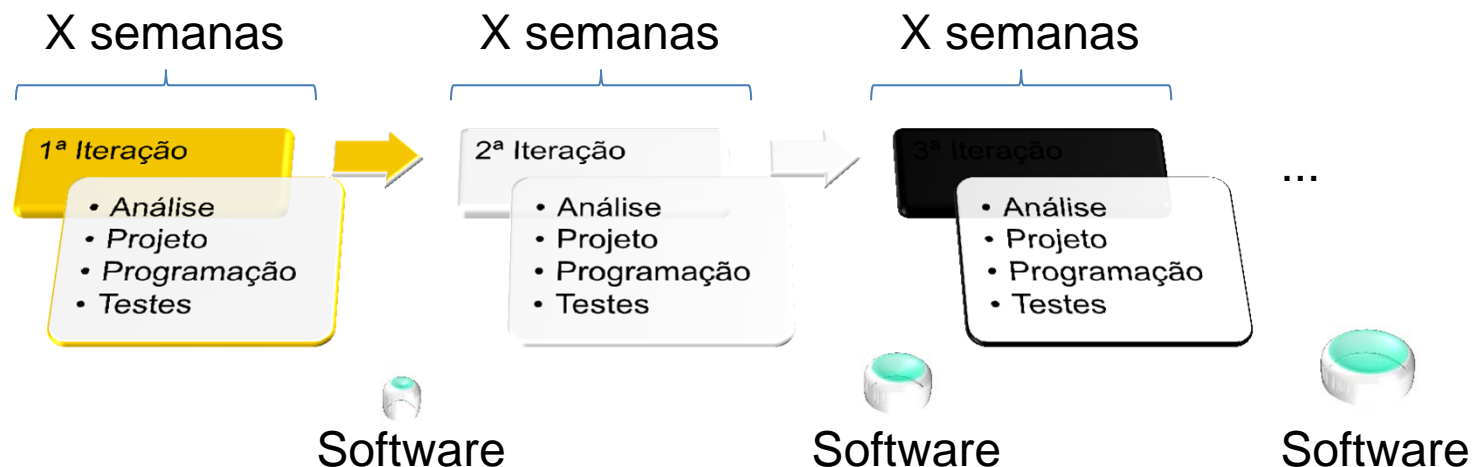
Mas o que é iterativo?

- Iteração

- Repetir partes do processo à medida que os requisitos do sistema evoluem
- Por exemplo, deve-se refazer (ou complementar) o projeto do sistema e sua implementação para incluir novos requisitos
- Cada ciclo desenvolve uma versão mais completa

Desenvolvimento Iterativo

- O desenvolvimento é organizado em “mini-projetos”
- Cada “mini-projeto” é uma iteração
 - Cada iteração tem duração curta e fixa (de 2 a 6 semanas)
 - Cada iteração tem atividades de análise, projeto, programação e testes
 - O produto de uma iteração é um software parcial



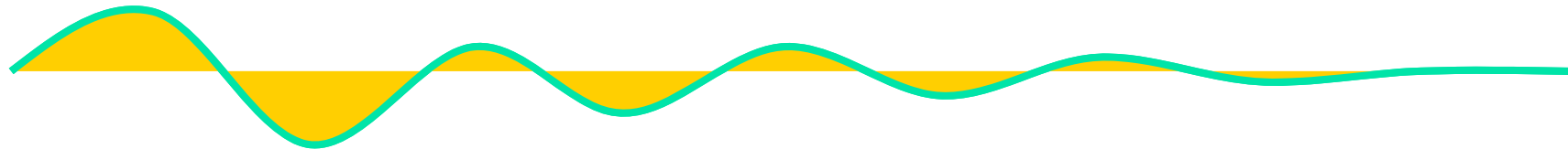


Desenvolvimento Iterativo

- A iteração deve ser fixa
 - Tarefas podem ser removidas ou incluídas
 - A iteração nunca deve passar da duração estipulada
- O resultado de cada iteração é um software...
 - Incompleto
 - Em desenvolvimento (não pode ser colocado em produção)
 - Mas não é um protótipo!!!
- Esse sw pode ser verificado e validado parcialmente
 - Testes
 - Usuários
- Podem ser necessárias várias iterações (10 a 15) para ter uma versão do sistema pronta para entrar em produção

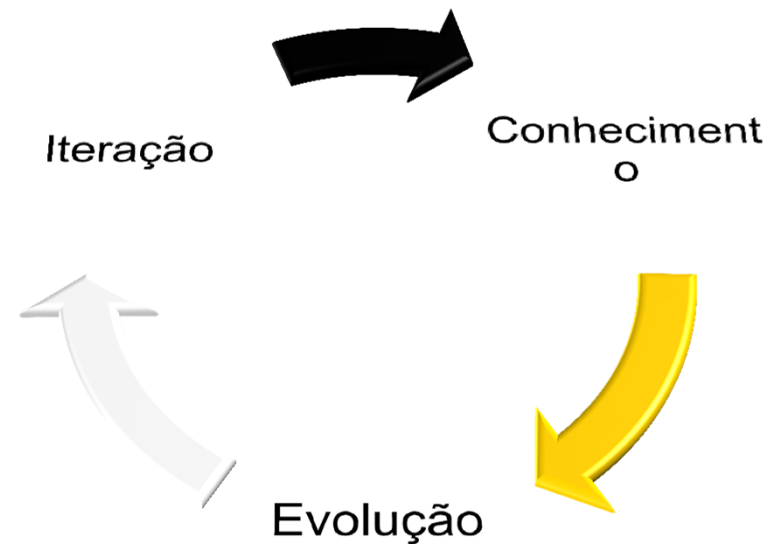
Desenvolvimento Iterativo

- Iterações curtas privilegiam a propagação de conhecimento
 - Aumento do conhecimento sobre o software
 - Diminuição das incertezas, que levam às mudanças



Desenvolvimento Evolutivo

- As especificações evoluem a cada iteração
 - A cada iteração, uma parte do software fica pronta
 - O conhecimento sobre o software aumenta
 - As especificações são evoluídas para retratar esse aumento de conhecimento sobre o que é o software



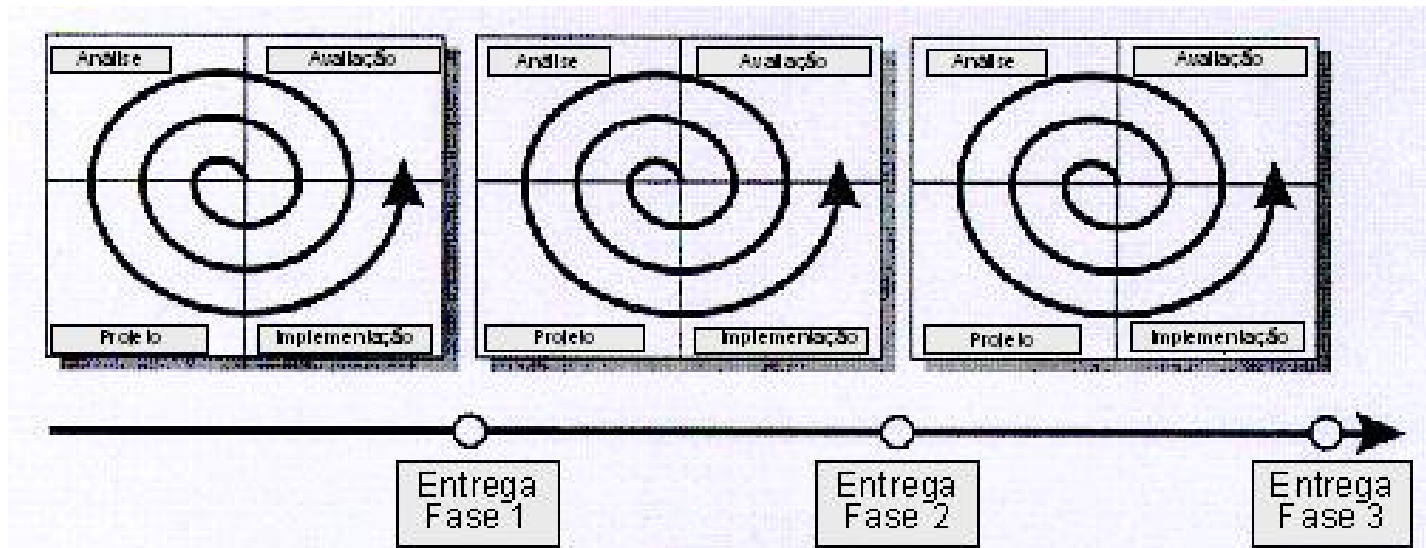


Desenvolvimento Evolutivo

- Mudanças sempre acontecem em projetos de software
 - Requisitos mudam
 - O ambiente em que o software está inserido muda
 - As pessoas que operam o software mudam

- Estratégias para lidar com mudanças
 - Evitar as mudanças (corretivas) fazendo uso de boas técnicas de engenharia de software
 - Acolher mudanças por meio de um processo evolutivo

Modelo Evolucionário Espiral





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

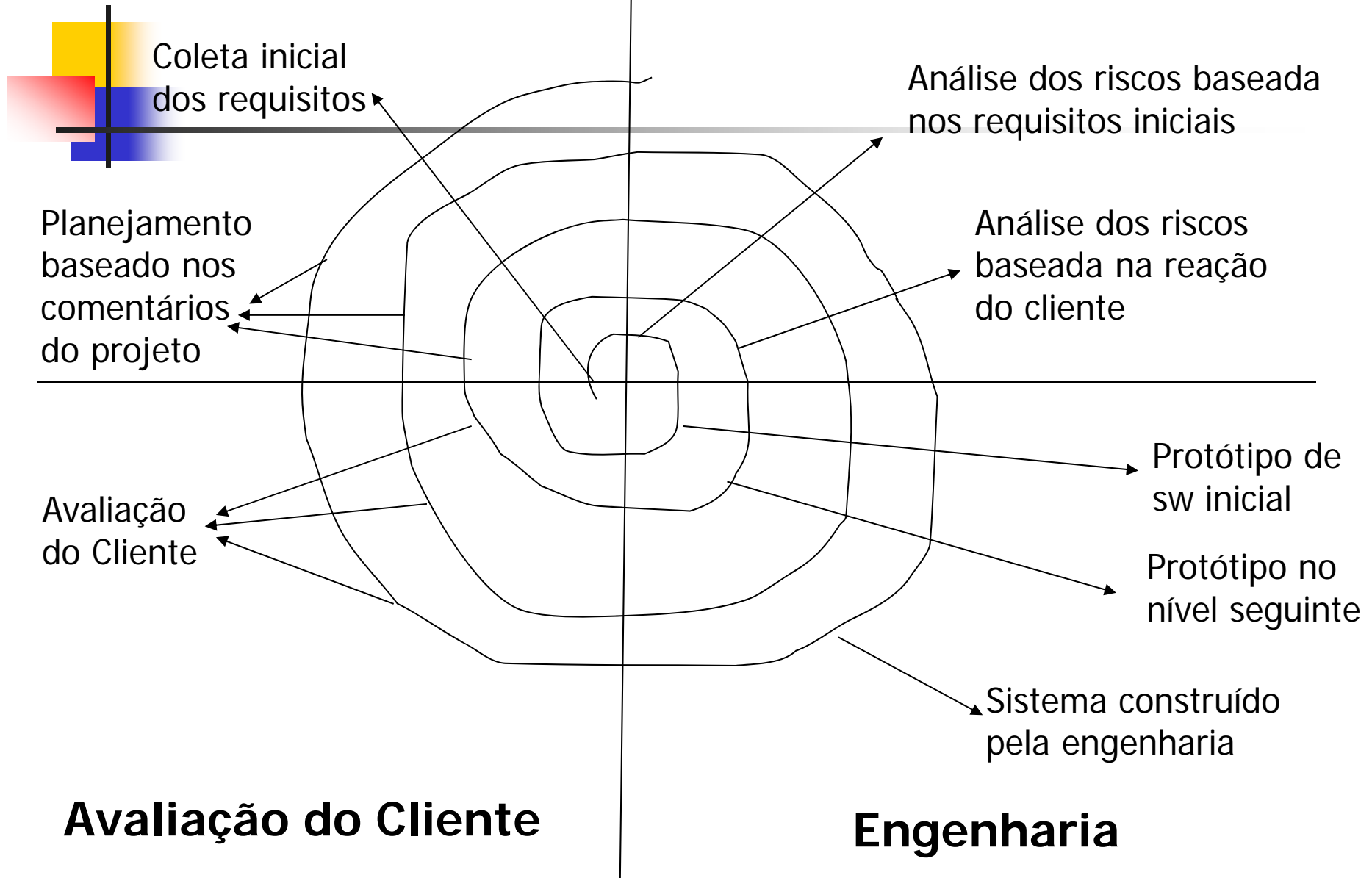


Modelo Espiral

- Define quatro importantes atividades:
 - Planejamento: determinação dos objetivos, alternativas e restrições
 - Análise dos riscos: Análise de alternativas e identificação/resolução dos riscos
 - Engenharia: Desenvolvimento do produto no nível seguinte
 - Avaliação feita pelo cliente: Avaliação dos resultados da engenharia

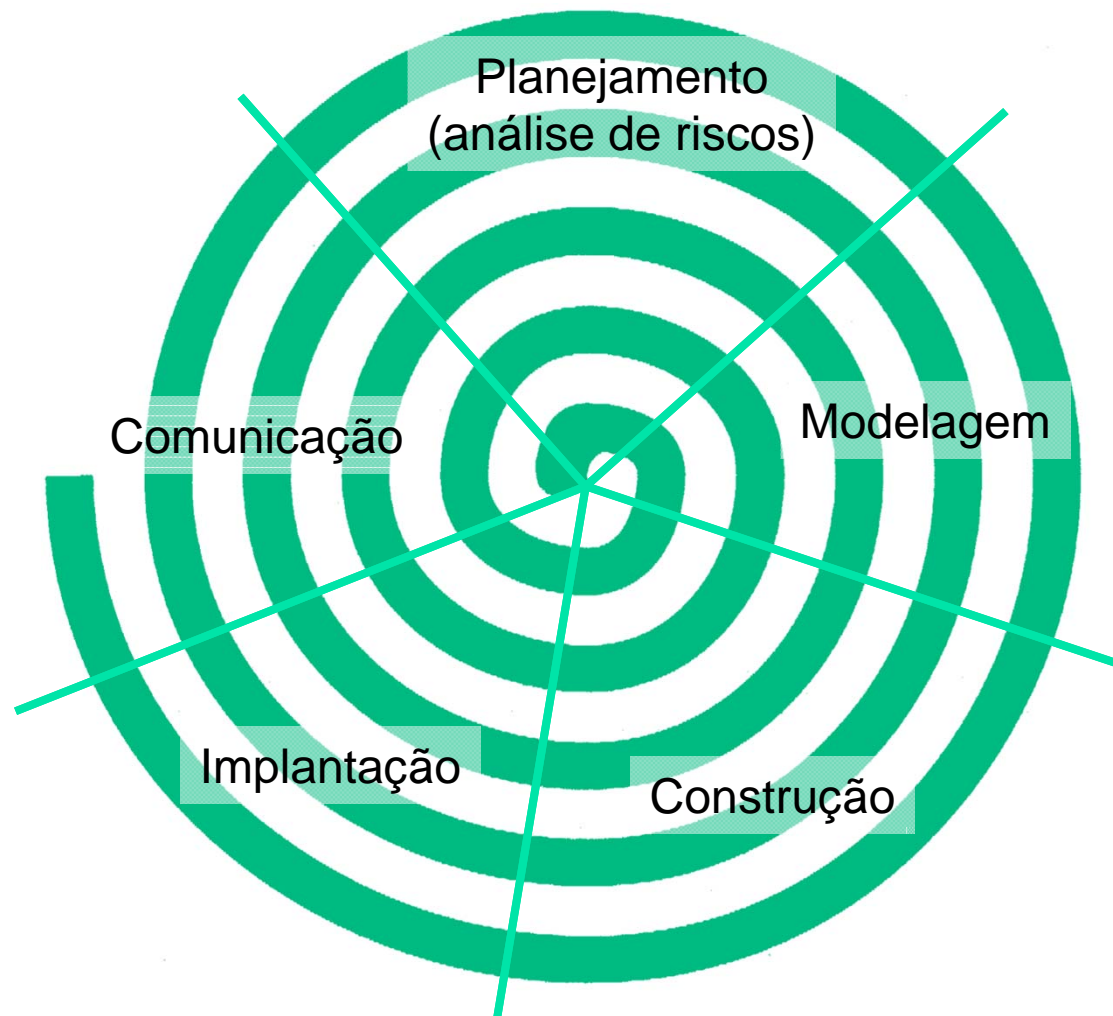
Planejamento

Análise de riscos





Modelo Espiral





Modelo Espiral

- A cada iteração ao redor da espiral, versões mais completas do sw são construídas
- Durante o primeiro giro ao redor da espiral, os objetivos, alternativas e restrições são definidos e os riscos são identificados e analisados
- Se a análise dos riscos indicar que há incertezas nos requisitos, a prototipação pode ser usada no parte da engenharia para ajudar o desenvolvedor e o cliente
- Simulações e outros modelos podem ser usados para definir ainda mais o problema e refinar os requisitos



Conclusão

- O modelo espiral para a ES 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



Conclusão

- 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



Conclusão

- 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



Resumindo

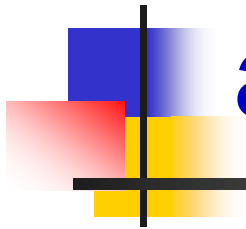
- Foco principal no gerenciamento de riscos
- A cada ciclo
 - O conhecimento aumenta
 - O planejamento é refinado
 - Produto gerado no ciclo anterior é evoluído (não é jogado fora)
- Cada ciclo evolui o sistema, mas não necessariamente entrega um software operacional
 - Modelo em papel
 - Protótipo
 - Versões do produto
 - Etc.



Exercício

- Cenário
 - Você deseja abrir uma empresa e lançar no mercado um produto inovador
- Detalhe um pouco mais este cenário.
- Qual ciclo de vida utilizar como base?
- Quais outras atividades de ES você incorporaria nesse processo?
- Quais são os maiores riscos que você está se expondo?

Desenvolvimento Orientado a Reuso





Introdução

- 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



Introdução

- Os sistemas atuais devem estar aptos:
 - Adaptar-se facilmente a mudanças:
tecnológicas, organizacionais e de domínio
- Necessidade de melhoria no processo de desenvolvimento de software: maior produtividade e menor custo



Introdução

- Requisitos das novas aplicações:
 - Distribuição entre vários sites, acesso a dados de múltiplas fontes, interface para Web
- Solução:
 - Uso de Técnicas de Reutilização de Software para criação e reuso de componentes interoperáveis



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



Tópicos

- Benefícios e problemas do reuso de software
- Desenvolvimento baseado em componentes



Reutilização de software

- Na maioria das disciplinas de engenharia, sistemas são projetados com base na composição de componentes existentes que foram utilizados em outros sistemas
- A engenharia de software, até então, tinha como base o desenvolvimento tradicional
- Para alcançar software com mais qualidade, de forma mais rápida e com baixo custo, é necessário adotar um processo de desenvolvimento na reutilização generalizada e sistemática de software



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

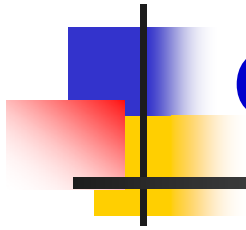
- Conformidade com padrões
 - Os padrões são embutidos ao reutilizar componentes. Ex: padrões de interfaces com o usuário
- Desenvolvimento acelerado
 - Evita o desenvolvimento e validação, acelerando a produção
- Maior flexibilidade na manutenção
- Maior qualidade dos produtos



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

Desenvolvimento Baseado em Componentes (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



- **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



- 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 no processo de desenvolvimento da aplicação



- Componente é um provedor de serviços
 - Não é necessário saber onde está sendo executado
 - Não é necessário conhecer seu estado interno
- Um componente é uma entidade executável independente, o código fonte pode não estar disponível
- Os componentes publicam sua interface e todas as interações são feitas por meio dessa interface
- Componentes podem ser algo simples como uma função matemática ou um sistema maior



Importante

- O DBC pode ser incorporado em um processo de software padrão, incorporando atividades de reuso no processo
- Engenharia de Software Baseada em Componentes usualmente envolve um processo de prototipação ou um processo de desenvolvimento incremental e uma linguagem para integrar componentes reutilizáveis



- **Dificuldade**

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



- 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



-
- Redução no tempo de entrega:
possibilidade de reuso
 - Melhora a consistência: reuso de partes
previamente testadas e padronizadas em
outros projetos
 - Uso de “best practices”: soluções testadas e
desenvolvidas por “experts”



- Aumento de produtividade: reuso e possibilidade de trocas
- Aumento de qualidade: componentes certificados
- Suporte ao desenvolvimento paralelo e distribuído
- Redução de custos de manutenção: facilidade de mudanças



Repositórios de Componentes

- Constitui em uma base preparada para o armazenamento, seleção e obtenção de componentes reutilizáveis
- Mecanismo de busca e seleção de componentes
- www.componentsource.com



Desenvolvimento Ágil



Introdução

- Processo de desenvolvimento de sw é caótico
- Imprevisibilidade e incertezas fazem parte desse processo
- Requisitos de usuário, pressões de tempo, competição, qualidade e recursos (base inicial do projeto) podem sofrer alterações
- Problema de modelos lineares (cascata, ...): assumir o processo como não sujeito a mudanças e incertezas



Agilidade

- 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



Desenvolvimento Ágil

- Na Economia da Informação as organizações necessitam ser mais competitivas, ágeis e oferecer diferenciais em seus produtos
- A Tecnologia da Informação está altamente presente nestes momentos





Desenvolvimento Ágil

- As solicitações de desenvolvimento de software são freqüentes. E demandam rapidez na entrega do sistema
- Rápidas mudanças no negócio implicam em alterações tão rápidas quanto nos sistemas





Desenvolvimento Ágil

- Problemas como:
 - Entrega do sistema fora do prazo estabelecido
 - Custo acima do orçado
 - Requisitos dos usuários não atendidos
- Geram impactos sérios para organizações



Desenvolvimento Ágil

- Desenvolvimento Ágil de Software é uma necessidade nestes ambientes altamente competitivos em que as organizações se encontram





Características

- Procuram minimizar riscos desenvolvendo software em pequenos espaços de tempo (iterações)
- Cada iteração é como um pequeno projeto
 - Planejamento, requisitos, projeto, codificação, testes...



Características

- Objetivo de cada iteração
 - Produzir componentes de software
 - Arquitetura vai sendo desenhada a partir da refatoração dos componentes
- Enfatizam comunicação “cara a cara” em relação à documentação



Extreme Programming – XP

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

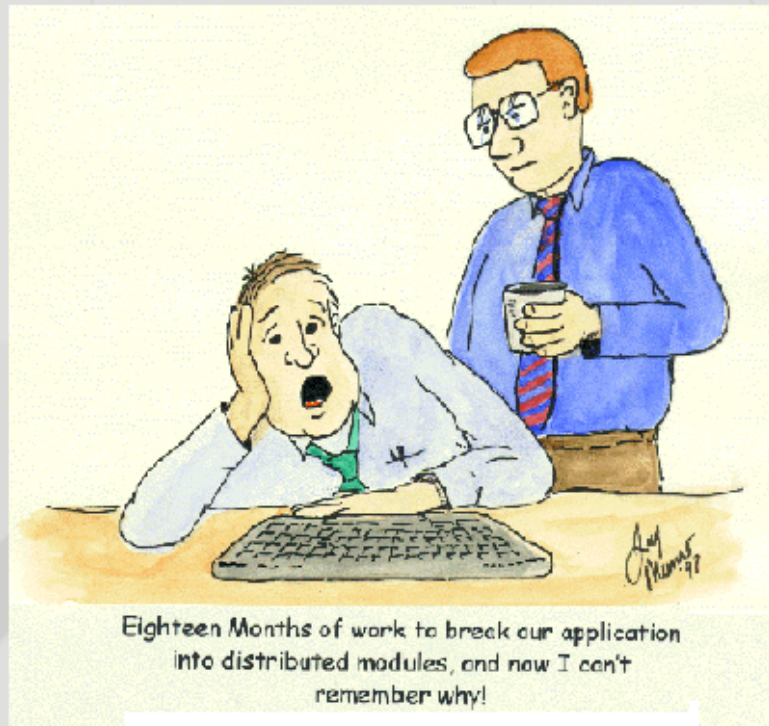


Extreme Programming – XP

- Um dos métodos de desenvolvimento ágeis de sw
- Processo de desenvolvimento de sw inovador
- Disciplina de desenvolvimento de sw que valoriza a simplicidade, comunicação, feedback, respeito e coragem
- Busca assegurar máxima geração de valor para o cliente com: Alta qualidade, Agilidade, Flexibilidade e Custo reduzido

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 frequência
 - 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 - Coragem

- XP não tem uma solução mágica para eliminar esse risco
 - Ele existe em um projeto XP , como em qualquer outro
 - O que muda é a forma de lidar com ele
 - Equipes XP acreditam que errar é natural e quebrar o que vinha funcionando pode acontecer eventualmente
 - É necessário ter coragem para lidar com esse risco, o que em XP se traduz em confiança nos seus mecanismos de proteção



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

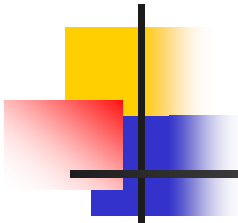
Valores – Respeito





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





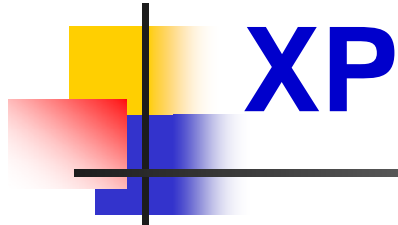
Valores – Comunicação

- 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



Valores – Comunicação

- Existem inúmeras formas de se comunicar idéias, algumas são mais eficazes que outras. Por exemplo:
 - Quando duas pessoas estabelecem um diálogo presencial, inúmeros elementos colaboram para a compreensão do assunto: gestos, expressões faciais, postura, palavras verbalizadas, tom de voz, emoções, entre outros.
 - Quanto maior a capacidade de compreensão, maiores as chances de evitar problemas
 - Diálogos são mais eficazes que videoconferências, que são melhores que telefonemas, que são mais expressivos que emails e assim sucessivamente.



- É baseado nas técnicas de Revisão e Teste
 - **Atividades de Revisão Conjunta**
 - Pair Programming
 - **Atividades de Teste**
 - Teste de Unidade
 - Teste de Aceitação
- O foco é desenvolver habilidades nas pessoas





Testes

- Fundamento mais importante de XP
- É o que dá segurança e coragem ao grupo
- Testes de Unidades (Unit tests)
 - escritos pelos programadores para testar cada elemento do sistema
- Testes Funcionais (Functional tests)
 - escritos pelos clientes para testar a funcionalidade do sistema



XP – Alguns Princípios

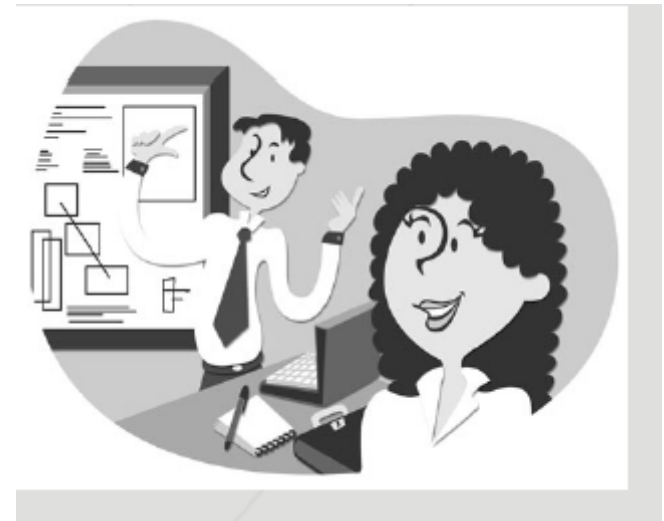


Trabalhe com seus clientes

- 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

Trabalhe com seus clientes

- 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

Stand-up meeting

- 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 filosofia da XP. 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 exhibe 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

Programação em pares

- Um digita, enquanto o outro revisa, corrige e sugere
 - Redução drástica de bugs
 - Disseminação de conhecimento
 - Pressão do par
 - Simplicidade
 - 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



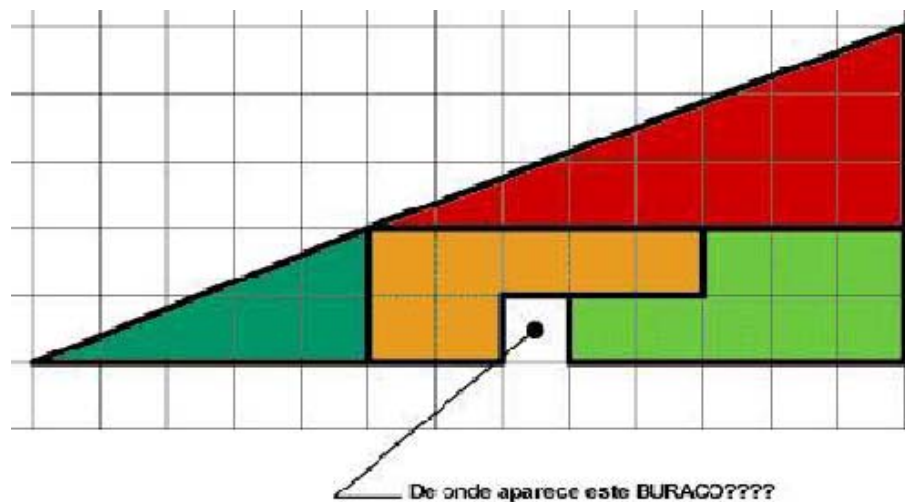


Codifique dentro dos padrões

- 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

- É o processo de alterar um sistema de software de tal forma que ele não altere o comportamento externo do código e melhore a sua estrutura interna
 - Essa é uma forma disciplinada de limpar o código que minimiza as chances de introdução de bugs



Refactoring

- Uma [pequena] modificação no sistema que não altera o seu comportamento funcional
- Mas que melhora alguma qualidade não-funcional:
 - simplicidade
 - flexibilidade
 - clareza
 - desempenho



Exemplos de Refactoring

- Mudança do nome de variáveis
- Mudanças nas interfaces dos objetos
- Pequenas mudanças arquiteturais
- Encapsular código repetido em um novo método
- Generalização de métodos
 - `raizQuadrada(float x) ⇒ raiz(float x, int n)`

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



Questionamentos



A Quem se Destina XP?

- Grupos de 2 a 10 programadores
- Projetos de 1 a 36 meses (calendário)
- De 1000 a 250 000 linhas de código
- Papéis:
 - Programadores (foco central)(sem hierarquia)
 - “Treinador” (coach)
 - “Acompanhador” (tracker)
 - Cliente



Coaches Corner

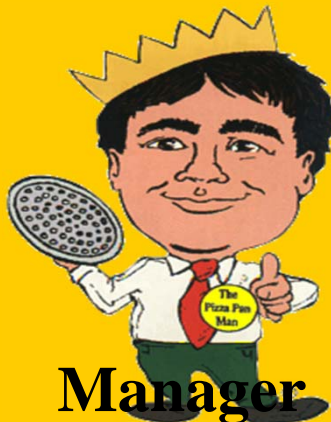
Coach



Tracker



Goal Donnor



Manager



Programador



Gold Owner



Cliente

- Responsável por escrever “histórias”
- Muitas vezes é um programador ou é representado por um programador do grupo
- Trabalha no mesmo espaço físico do grupo
- Novas versões são enviadas para produção todo mês (ou toda semana)
- Feedback do cliente é essencial
- Requisitos mudam (e isso não é mau)



Coach (Treinador)

- Em geral, o mais experiente do grupo
- Identifica quem é bom no que
- Lembra a todos as regras do jogo (XP)
- Eventualmente faz programação pareada
- Não desenha arquitetura, apenas chama a atenção para oportunidades de melhorias
- Seu papel diminui à medida em que o time fica mais maduro



Tracker (Acompanhador)

- A “consciência” do time
- Coleta estatísticas sobre o andamento do projeto. Alguns exemplos:
 - Número de histórias definidas e implementadas
 - Número de unit tests
 - Número de testes funcionais definidos e funcionando
 - Número de classes, métodos, linhas de código
- Mantém histórico do progresso
- Faz estimativas para o futuro



Todos gostam de XP?

- Alguns odeiam, outros amam
- Quem gosta de programar ama!
- Deixa o bom programador livre para fazer o que ele faria se não houvesse regras
- Força o [mau] programador a se comportar de uma forma similar ao bom programador



Como é um Dia na Vida de um Programador XP?

- Escolhe uma história do cliente
- Procura um par livre
- Escolhe um computador para programação em par
- Seleciona uma tarefa claramente relacionada a uma característica desejada pelo cliente



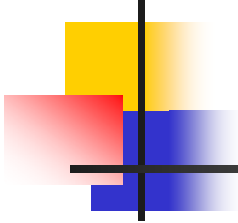
Como é um Dia na Vida de um Programador XP?

- Discute modificações recentes no sistema
- Discute história do cliente
- Testes
- Implementação
- Desenho
- Integração



Como é um Dia na Vida de um Programador XP?

- Atos constantes no desenvolvimento:
 - Executa testes antigos
 - Busca oportunidades para simplificação
 - Modifica desenho e implementação incrementalmente baseado na funcionalidade exigida no momento
 - Escreve novos testes
 - Enquanto todos os testes não rodam a 100%, o trabalho não está terminado
 - Integra novo código ao repositório



Programação em par é uma boa técnica? Sim...

- Erro de um detectado imediatamente pelo outro (grande economia de tempo)
- Maior diversidade de idéias, técnicas, algoritmos
- Enquanto um escreve, o outro pensa em contra-exemplos, problemas de eficiência, etc
- Vergonha de escrever código feio na frente do seu par
- Pareamento de acordo com especialidades



Propriedade Coletiva do Código... O que é isso?

- Modelo tradicional: só autor de uma função pode modificá-la
- XP: o código pertence a todos
 - Se alguém identifica uma oportunidade para simplificar, consertar ou melhorar código escrito por outra pessoa, que o faça
 - Mas rode os testes!



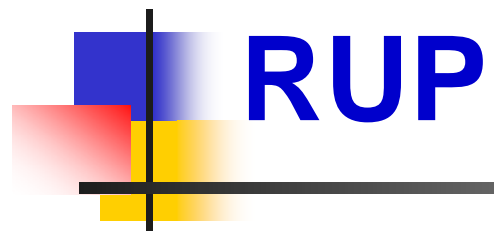
Quando XP Não Deve Ser Experimentada?

- Quando o cliente não aceita as regras do jogo
- Quando o cliente quer uma especificação detalhada do sistema antes de começar
- Quando os programadores não estão dispostos a seguir (todas) as regras
- Se (quase) todos os programadores do time são “mediócras”



Quando XP Não Deve Ser Experimentada?

- Grupos grandes (>10 programadores)
- Quando feedback rápido não é possível:
 - sistema demora 6h para compilar
 - testes demoram 12h para rodar
 - exigência de certificação que demora meses
- Quando o custo de mudanças é essencialmente exponencial
- Quando não é possível realizar testes





Processo

- Um processo é um conjunto de passos ordenados com a intenção de atingir uma meta
- Em ES, a meta é criar um software ou aperfeiçoar um existente
 - Em engenharia de processos, a meta é desenvolver ou aperfeiçoar um processo



Processo

- Em termos de modelagem de negócios, o processo de desenvolvimento de software é uma metodologia de negócios onde se descreve uma família de processos de ES relacionados, que compartilham uma estrutura comum, uma arquitetura de processos comum



Processo

- O mesmo proporciona uma abordagem disciplinada para a atribuição de tarefas e de responsabilidades dentro de uma organização de desenvolvimento
- Sua meta é garantir a produção de software de alta qualidade que atenda às necessidades dos usuários, dentro de uma programação e um orçamento previsíveis



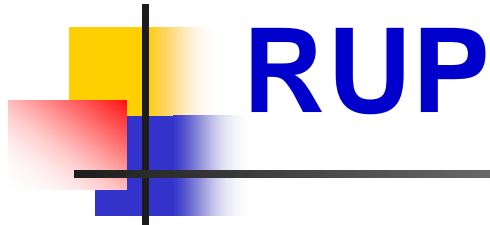
Processo

- Quando um sistema de software é desenvolvido começando do zero, o desenvolvimento é o processo de criação de um sistema a partir dos requisitos

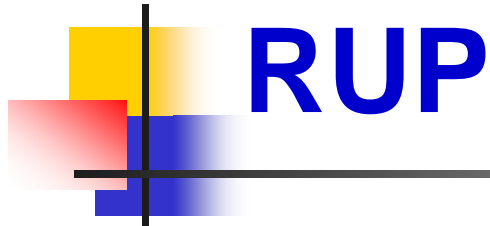


Processo

- Porém, depois que os sistemas tiverem tomado forma, ou seja, tiverem passado pelo ciclo de desenvolvimento inicial, os desenvolvimentos subseqüentes serão o processo de adaptação do sistema aos requisitos novos ou modificados
 - Isso se aplica durante todo o ciclo de vida do sistema



- O Processo Unificado (*Rational Unified Process – RUP*) foi criado para apoiar o desenvolvimento orientado a objetos, fornecendo uma forma sistemática para se obter reais vantagens no uso da Linguagem de Modelagem Unificada (*Unified Modeling Language – UML*)



- 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



- Suas atividades tendem a ser bem definidas, com responsáveis, artefatos de entrada e saída, dependências e ordem de execução, modelo de ciclo de vida e com uma descrição sistemática de como executar essas atividades



- Ele é baseado em componentes e usa a UML (*Unified Modeling Language*), para denotar a modelagem



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



Princípios do RUP

- Dirigido a use-case
 - O processo de desenvolvimento segue um fluxo de ações para a realização das use-case, o que procede através da execução dos workflows
 - Use-cases podem ser definidos como sendo “uma descrição de um conjunto de seqüência de ações, incluindo variantes, que um sistema utiliza para gerar um resultado observável de relevância de um ator”
 - Desta maneira os use-cases são especificados, projetados e no fim, são a fonte a partir da qual os engenheiros de teste constroem os casos de teste
 - Eles dirigem o processo de desenvolvimento



Princípios do RUP

- Centrado na arquitetura
 - A arquitetura é tanto percebida pelos usuários como refletida nos use-cases
 - Contudo, ela também é influenciada por muitos outros fatores, tais como a plataforma de software, blocos de construção reusáveis disponíveis, considerações de implantação, sistemas legados e requisitos não-funcionais



Princípios do RUP

- A arquitetura é uma visão do projeto como um todo, que acaba tornando visível as características mais importantes do mesmo, ou seja proporciona uma perspectiva mais clara do sistema desenvolvido



Princípios do RUP

- Iterativo e incremental
 - O desenvolvimento de um produto de software pode continuar por vários anos
 - Por isso é prático que o trabalho seja dividido em pequenos ciclos ou mini-projetos
 - Cada mini-projeto é uma iteração que resulta em um incremento
 - Assim iterações referem-se a passos no workflow, e incrementam a evolução do produto



Ciclo de vida do RUP

- O ciclo de vida adotado no RUP é tipicamente evolutivo
 - Contudo, uma forma de organização em fases é adotada para comportar os ciclos de desenvolvimento, permitindo uma gerência mais efetiva de projetos complexos



Ciclo de vida do RUP

- Ao contrário do tradicionalmente definido como fases na maioria dos modelos de ciclo de vida – planejamento, levantamento de requisitos, análise, projeto, implementação e testes, são definidas fases ortogonais a estas



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

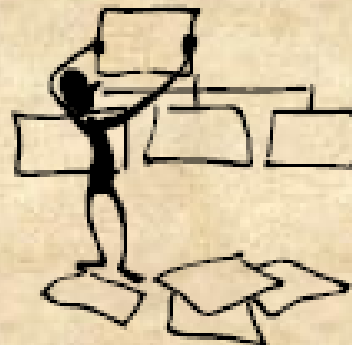
Concepção

Estabelecer o escopo e viabilidade econômica do projeto



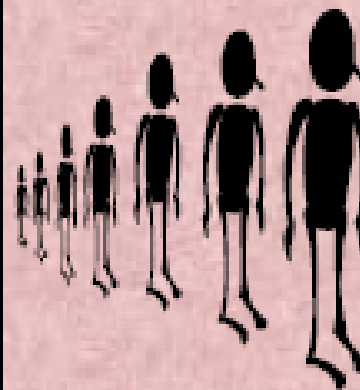
Elaboração

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

- Iniciação

- Durante a fase de concepção uma idéia do sistema a ser produzido é desenvolvida para uma visão do produto final e os casos de negócio do produto, isto é, a forma como o sistema poderá ser aplicado é apresentada
- Nesta fase a maioria dos riscos é identificada e priorizada



Fases do RUP

- **Elaboração**

- Durante a fase de elaboração o objetivo é eliminar os principais riscos e estabelecer uma arquitetura a partir da qual o sistema poderá evoluir
- O resultado desta fase é uma linha base para a arquitetura



Fases do RUP

- Construção

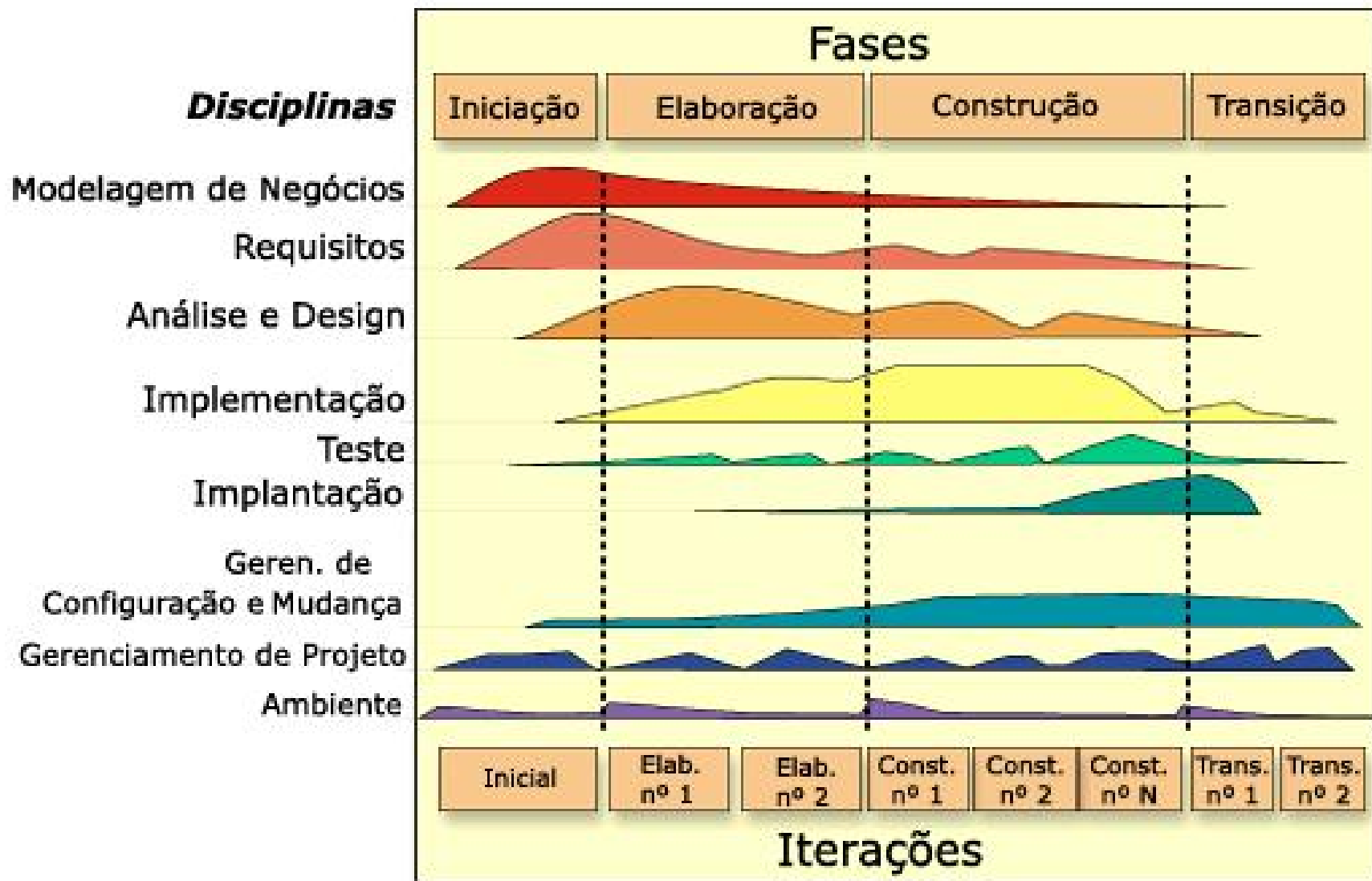
- Durante esta fase o produto é construído, ou seja, o músculo do software é adicionado ao seu esqueleto, que é composto pela arquitetura
- Nesta fase a linha base da arquitetura cresce e torna-se o sistema



Fases do RUP

- Transição

- Esta fase cobre o período pelo qual o produto se move em beta versões
- Nas beta versões, um pequeno número de usuários experientes testa o produto e indica defeitos e deficiências
- Esta fase envolve atividades tais como: treinamento de pessoal do cliente, assistência e correção de defeitos encontrados antes da entrega do produto





Fases do RUP

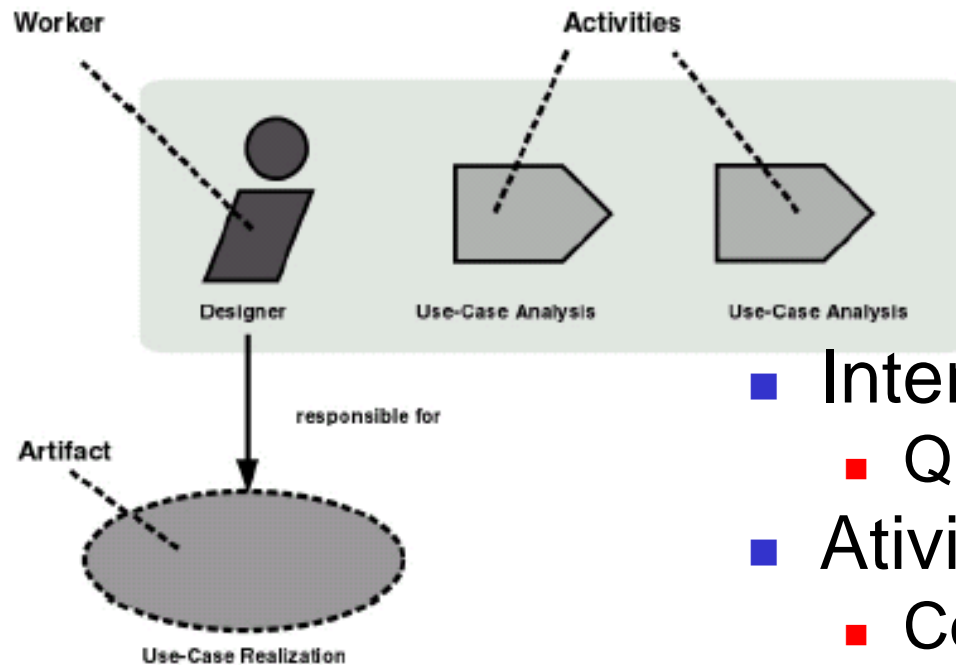
- Cada fase é adicionalmente dividida em iterações e finalizada com um milestone (marco de referência) que verifica se seus objetivos foram alcançados



Fases do RUP

- Toda iteração é organizada em termos de *workflows* (conjunto de atividades realizadas por responsáveis que produzem artefatos)
- Cada um dos workflows possui tarefas, profissionais e artefatos bem definidos, sendo divididos em duas categorias: workflows de processo e workflows de suporte

Workflow de processo



- Intervenientes (*Workers*)
 - Quem? (*who*)
- Atividades (*Activities*)
 - Como? (*how*)
- Artefatos (*Artifacts*)
 - O Que? (*what*)
- Fluxo de Trabalho (*Workflows*)
 - Quando? (*when*)



Workflows de processo

- Modelagem de negócios
- Requisitos
- Análise e projeto
- Implementação
- Teste
- Implantação



Workflows de processo

- Modelagem de negócios
 - Permite o entendimento da estrutura e dinâmica da organização, garantindo que clientes, usuários finais e desenvolvedores tenham um entendimento comum da mesma e possam derivar os requisitos do sistema que a suportam



Workflows de processo

- Requisitos

- Direciona o desenvolvimento rumo ao sistema certo, que é alcançado através da descrição dos requisitos do sistema, de forma que um acordo sobre o que o sistema deve ou não fazer possa ser alcançado entre o cliente (incluindo usuários) e desenvolvedores do sistema
- O maior desafio é representar a captura dos requisitos do sistema de forma que o cliente, que se assume que não seja um especialista em computadores, possa entender
- O resultado ajuda ao gerente do projeto a planejar as iterações e versões para o cliente



Workflows de processo

- Análise e projeto
 - possibilita a transformação dos requisitos em um projeto do sistema, sugerindo uma arquitetura robusta e adaptando o projeto para o ambiente de implementação



Workflows de processo

- Implementação

- Define a organização do código, em termos de implantação de subsistemas dispostos em camadas, implementa as classes e objetos em termos de componentes, testa os componentes desenvolvidos como unidades e integra os resultados obtidos por implementadores individuais ou equipes em um sistema executável



Workflows de processo

- Teste

- Verifica a integração entre objetos, a integração formal de todos os componentes de software, se todos os requisitos foram corretamente implementados, e identifica e garante que todos os defeitos serão resolvidos antes da implantação do software



Workflows de processo

- Implantação
 - Produz versões do produto e disponibiliza essas versões para o usuário
 - O workflow cobre uma grande variedade de atividades, incluindo
 - produzir versões, empacotar, distribuir e instalar o software, produzir assistência/ajuda ao usuário e em muitos casos, planejar e conduzir os beta testes e realizar a migração de software ou dados existentes



Workflows de suporte

- Configuração e gerenciamento de mudanças
- Gerenciamento de projeto
- Ambiente



Workflows de suporte

- Configuração e gerenciamento de mudanças
 - É essencial para controlar os diversos artefatos produzidos por pessoas que trabalham em um mesmo projeto
 - O controle ajuda a evitar confusões e garante que os artefatos resultantes não são conflitantes



Workflows de suporte

- Gerenciamento de projeto
 - Fornece um framework para gerenciamento de projetos de softwares, planejamento, contratação de pessoal, execução e monitoração de projetos e um *framework* para gerenciamento de riscos



Workflows de suporte

- Ambiente

- Provê a organização com o ambiente de desenvolvimento de software (processos e ferramentas)
- O workflow enfatiza as atividades para configurar o processo no contexto de um projeto



Exercício

- Discuta com o seu grupo como as idéias apresentadas pelo Processo Unificado poderiam ser adotadas no cenário apresentado anteriormente.
- Qual?
 - Cenário
 - Você deseja abrir uma empresa e lançar no mercado um produto inovador



E para terminar...



Como o cliente explicou



Como o lider de projeto entendeu



Como o analista planejou



Como o programador codificou



O que os beta testers receberam



Como o consultor de negocios descreveu



Valor que o cliente pagou



Como o projeto foi documentado



O que a assistencia tecnica instalou



Como foi suportado



Quando foi entregue



O que o cliente realmente necessitava