



Desenvolvimento Ágil

Prof. Ricardo Argenton Ramos
Aula 13 e 14

[baseada na apresentação dos professores Fabio Kon e Alfredo Goldman – DCC-IME-USP]

Novos rumos no mundo do Desenvolvimento de Software

- Sociedade demanda
 - grande quantidade de sistemas/aplicações
 - software complexo, sistemas distribuídos, heterogêneos
 - requisitos mutantes (todo ano, todo mês, todo dia)
- Mas, infelizmente,
 - não há gente suficiente para desenvolver tanto software com qualidade.

Problemas

- Com metodologias de desenvolvimento
 - Supõem que é possível prever o futuro
 - Pouca interação com os clientes
 - Ênfase em burocracias (documentos, formulários, processos, controles rígidos, etc.)
 - Avaliação do progresso baseado na evolução da burocracia e não do código
- Com software
 - Grande quantidade de erros
 - Falta de flexibilidade

Como resolver esse impasse?

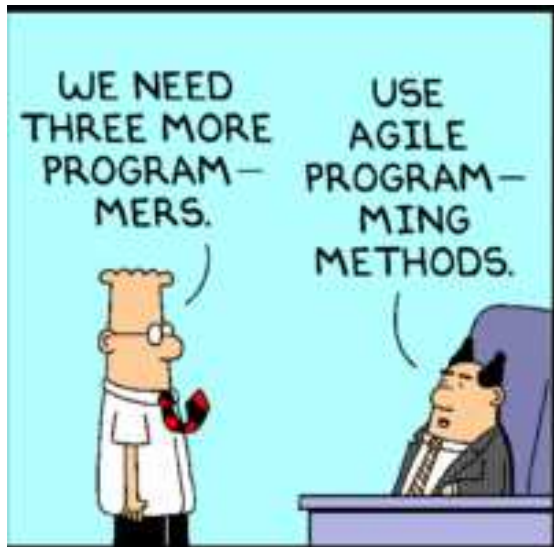
- Melhores Tecnologias
 - Padrões de Projeto (reutilização de idéias)
 - Componentes (reutilização de código)
 - Middleware (aumenta a abstração)
- Melhores Metodologias
 - Métodos Ágeis (o foco nesta palestra)
 - outras... (RUP, relacionadas a CMM, etc.)

Metodologias de Desenvolvimento de Software OO

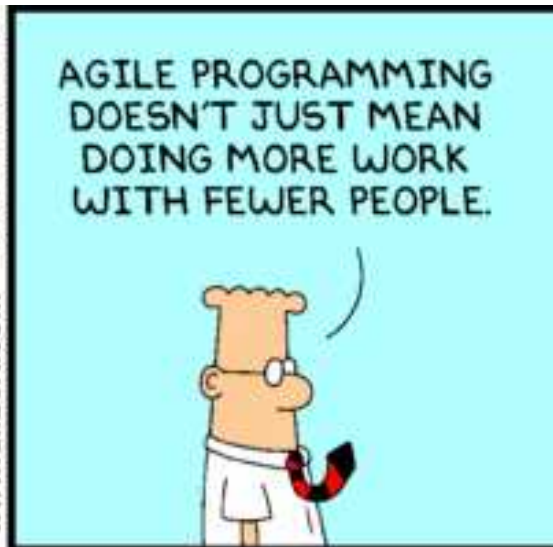
- “Tradicionais”
 - Comunidade de Engenharia de Software
 - IEEE/ACM ICSE
 - p.ex. Carnegie-Mellon SEI
 - RUP, CMM, etc.
- Ágeis
 - Comunidade de POO
 - ACM OOPSLA
 - p.ex. Johnson @ Illinois, Beck, Cockburn, Jeffries, Cunningham...
 - XP, Crystal, Scrum, etc.

Métodos Ágeis de Desenvolvimento de Software

- Movimento iniciado por programadores experientes e consultores em desenvolvimento de software.
- Questionam e se opõe a uma série de mitos/práticas adotadas em abordagens tradicionais de Engenharia de Software e Gerência de Projetos.
- Manifesto Ágil:
 - Assinado por 17 desenvolvedores em Utah em fevereiro/2001.



www.dilbert.com
scottadams@aol.com



1/14/05 © 2005 Scott Adams, Inc./Dist. by UFS, Inc.



O Manifesto do *Desenvolvimento Ágil de Software*

1. **Indivíduos e interações** são mais importantes que *processos e ferramentas*.
2. **Software funcionando** é mais importante do que *documentação completa e detalhada*.
3. **Colaboração com o cliente** é mais importante do que *negociação de contratos*.
4. **Adaptação a mudanças** é mais importante do que *seguir o plano inicial*.

Princípios do Manifesto Ágil

- Objetivo: satisfazer o cliente entregando, rapidamente e com frequência, sistemas com algum valor.
 - Entregar versões funcionais em prazos curtos.
 - Estar preparado para requisitos mutantes.
 - Pessoal de negócios e desenvolvedores juntos.
 - Troca de informações através de conversas diretas.

Principais Métodos Ágeis

- Nesta palestra nos concentraremos em :
 - Programação eXtrema (XP)
- Outros métodos ágeis interessantes:
 - Crystal (uma família de métodos)
 - Scrum
 - Adaptive Software Development
 - Feature Driven Development
 - etc.

Scrum



Definição informal:
Estratégia em um jogo de rugby onde jogadores colocam uma bola quase perdida novamente em jogo através de trabalho em equipe.

Scrum

- Jeff Sutherland
 - <http://jeffsutherland.com>
- Ken Schwaber
 - <http://www.controlchaos.com>
- Conferências
 - OOPSLA 96, PLoP 98
- Inspiração
 - Desenvolvimento Iterativo e Incremental em empresas (DuPont, Honda, etc) nos anos 80

Programação eXtrema XP

- Metodologia de desenvolvimento de software aperfeiçoada nos últimos 5 anos.
- Ganhou notoriedade a partir da OOPSLA'2000.
- Nome principal: Kent Beck
- Também importante: Ward Cunningham

Reações a 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.

Modelo Tradicional de Desenvolvimento de Software

0. Levantamento de Requisitos

1. Análise de Requisitos

2. Desenho da Arquitetura

3. Implementação

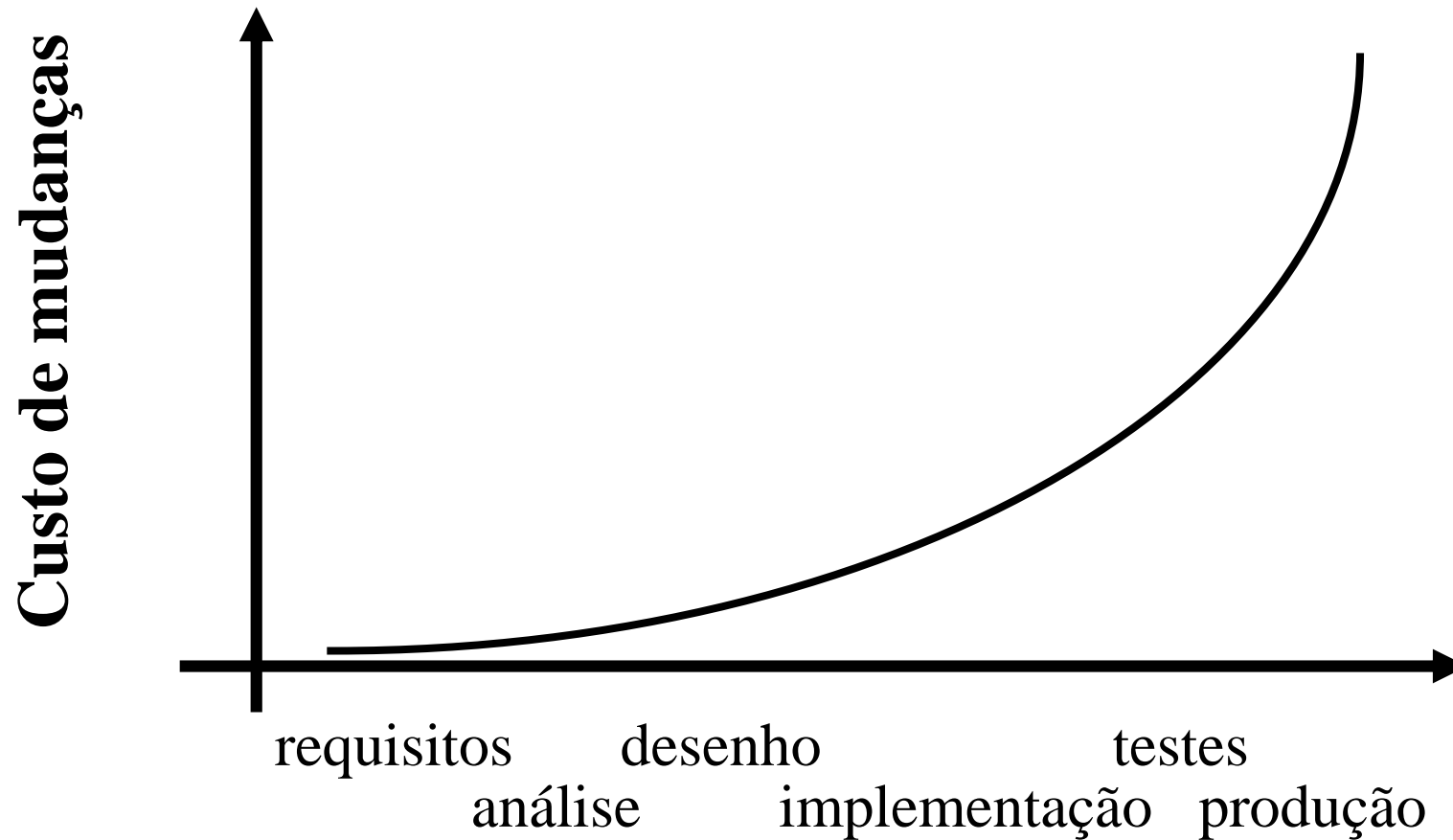
4. Testes

5. Produção / Manutenção

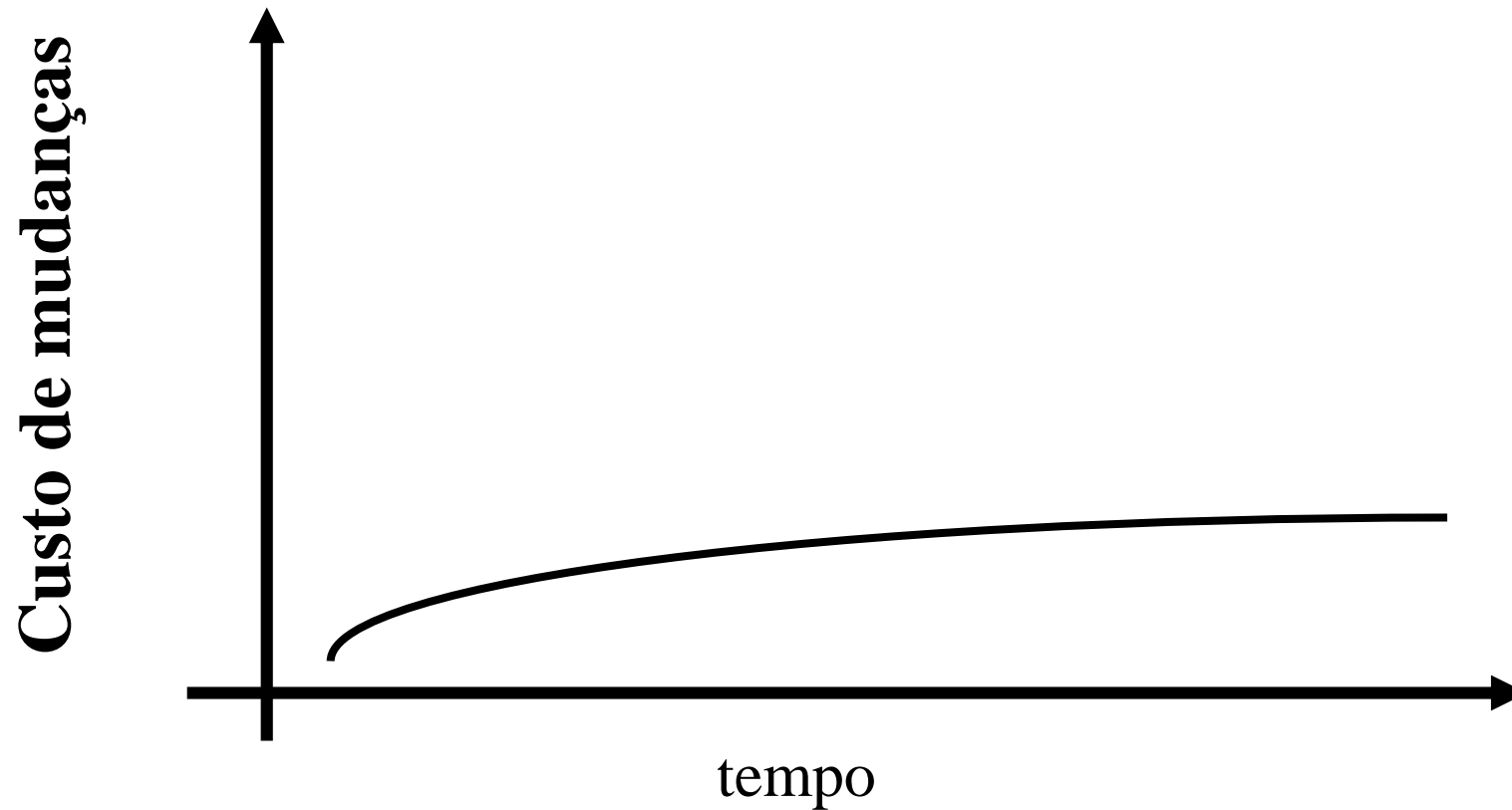
Premissas Básicas do Modelo Tradicional

- É necessário fazer uma análise de requisitos profunda e detalhada antes de projetar a arquitetura do sistema.
- É necessário fazer um estudo minucioso e elaborar uma descrição detalhada da arquitetura antes de começar a implementá-la.
- É necessário testar o sistema completamente antes de mandar a versão final para o cliente.

O que está por trás deste modelo?



E se a realidade hoje em dia
fosse outra?



E se essa fosse a realidade?

- A atitude dos desenvolvedores de software seria completamente diferente:
 - Tomaríamos as grandes decisões o mais tarde possível.
 - Implementaríamos agora somente o que precisamos *agora*.
 - Não implementaríamos flexibilidade desnecessária (não anteciparíamos necessidades).

E essa é a nova realidade !
(pelo menos em muitos casos)

- **Orientação a Objetos:** facilita e cria oportunidades para mudanças.
- **Técnicas de Refatoração.**
- **Testes automatizados:** nos dão segurança quando fazemos mudanças.
- **Prática / cultura de mudanças:** aprendemos técnicas e adquirimos experiência em lidar com código mutante.

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” ou “Técnico” (*coach*)
 - “Acompanhador” (*tracker*)
 - Cliente

E Se Eu Não Me Encaixo Nesses Casos?

- Você ainda pode aprender muito sobre desenvolvimento de software.
- Terá elementos para repensar as suas práticas.
- No início se dizia:
 - “Ou você é 100% eXtremo ou não é eXtremo. Não dá prá ser 80% XP.”
 - *XP is like teenage sex.*
- Hoje não é mais necessariamente assim.

As 12 Práticas de XP

(versão 2000)

1. Planejamento
2. Fases Pequenas
3. Metáfora
4. Design Simples
5. Testes
6. Refatoração
7. Programação Pareada
8. Propriedade Coletiva
9. Integração Contínua
10. Semana de 40 horas
11. Cliente junto aos desenvolvedores
12. Padronização do código

Princípios Básicos de XP

- *Feedback* rápido
- Simplicidade é o melhor negócio
- Mudanças incrementais
- Carregue a bandeira das mudanças / não valorize o medo (*Embrace change*)
- Alta qualidade do código

As 4 Variáveis do Desenvolvimento de Software

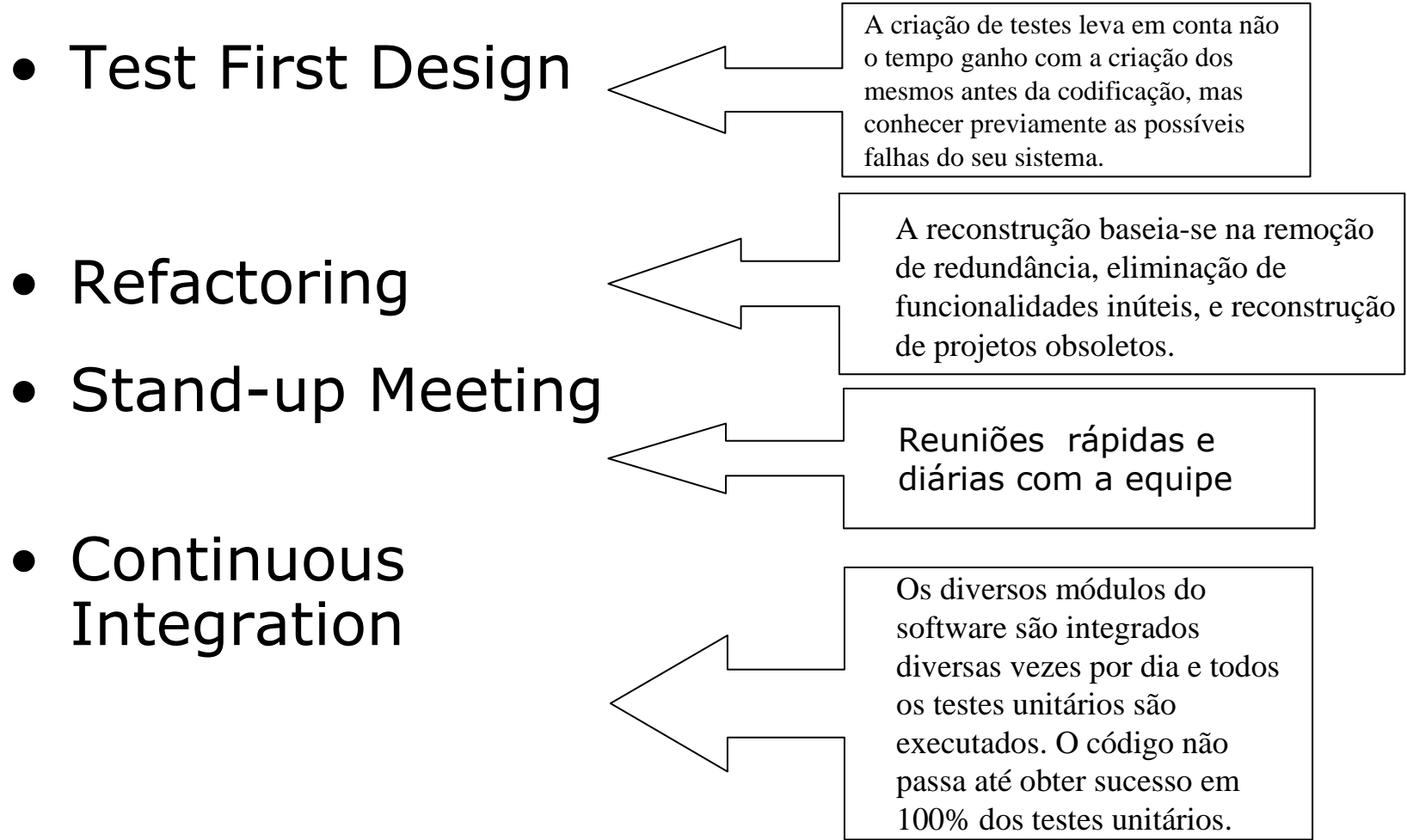
- **Tempo**
- **Custo**
- **Qualidade**
- **Escopo (foco principal de XP)**

Um Projeto XP

- Fase de Exploração
 - duração: 2 a 6 meses.
 - termina quando a primeira versão do software é enviada ao cliente.
 - clientes escrevem “historias” (*story cards*).
 - programadores interagem com clientes e discutem tecnologias.
 - Não só discutem, **experimentam** diferentes tecnologias e arquiteturas para o sistema.
 - Planejamento: 1 a 2 dias.

Boas Práticas

- Test First Design



A criação de testes leva em conta não o tempo ganho com a criação dos mesmos antes da codificação, mas conhecer previamente as possíveis falhas do seu sistema.

- Refactoring

A reconstrução baseia-se na remoção de redundância, eliminação de funcionalidades inúteis, e reconstrução de projetos obsoletos.

- Stand-up Meeting

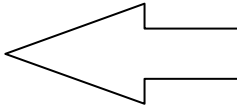
Reuniões rápidas e diárias com a equipe

- Continuous Integration

Os diversos módulos do software são integrados diversas vezes por dia e todos os testes unitários são executados. O código não passa até obter sucesso em 100% dos testes unitários.

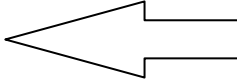
Boas Práticas

- Pair Programming



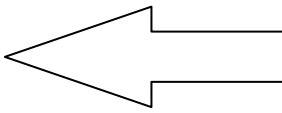
Todo código de produção é desenvolvido por duas pessoas trabalhando com o mesmo teclado, o mesmo mouse e o mesmo monitor.

- Move People Around



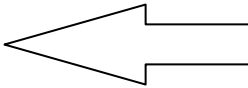
As duplas de programação são revezadas em média a cada 2h.

- Collective Code Ownership



E equipe como um todo é responsável por cada arquivo de código. Não é preciso pedir autorização para alterar qualquer arquivo.

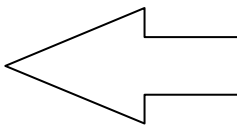
- Coding Standards



Todo código é desenvolvido seguindo um padrão.

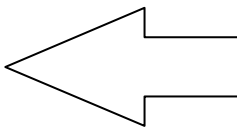
Boas Práticas

- The Customer is Always Available



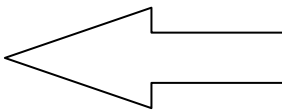
O cliente está sempre disponível para resolver dúvidas, alterar o escopo de uma iteração e definir prioridades.

- Small Release



O software é entregue em pequenas versões para que o cliente possa obter o seu ganho o mais cedo possível e para minimizar riscos.

- Simple Design



O código está, a qualquer momento, na forma mais simples que passe todos os testes.

Boas Práticas

- 40-Hour Week



Cada programador trabalha 40 horas por semana. Se o horário for flexível, deve-se respeitar o horário do par naquele período, senão enquanto um trabalha o outro vai pra casa .

- On-Site Customer

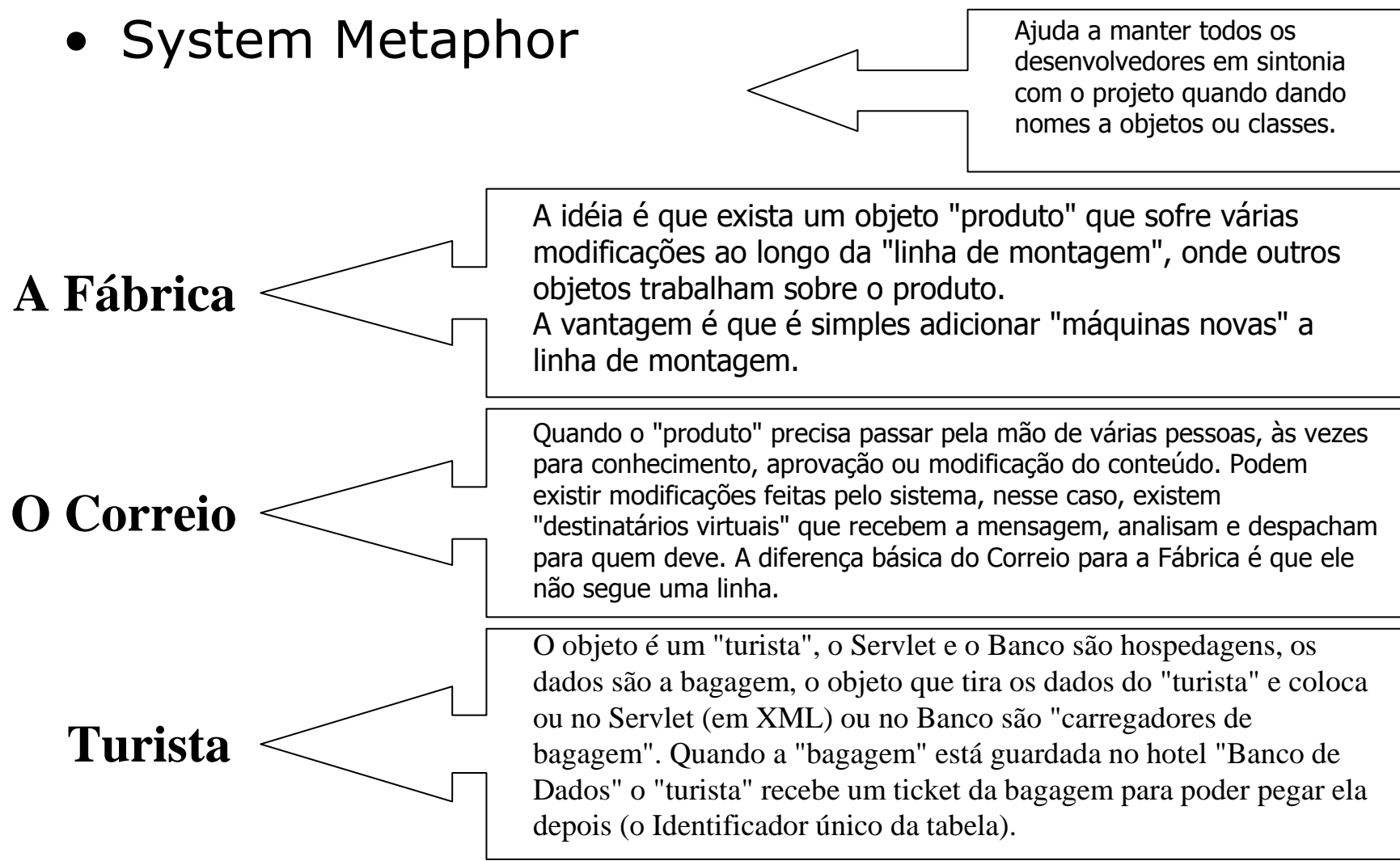
Ter um papel de cliente na equipe XP em tempo integral para responder as perguntas

- Acceptance Tests

São definidos pelo usuário e são os critérios de aceitação do software.

Boas Práticas

- System Metaphor



Ajuda a manter todos os desenvolvedores em sintonia com o projeto quando dando nomes a objetos ou classes.

A Fábrica

A idéia é que exista um objeto "produto" que sofre várias modificações ao longo da "linha de montagem", onde outros objetos trabalham sobre o produto.
A vantagem é que é simples adicionar "máquinas novas" a linha de montagem.

O Correio

Quando o "produto" precisa passar pela mão de várias pessoas, às vezes para conhecimento, aprovação ou modificação do conteúdo. Podem existir modificações feitas pelo sistema, nesse caso, existem "destinatários virtuais" que recebem a mensagem, analisam e despacham para quem deve. A diferença básica do Correio para a Fábrica é que ele não segue uma linha.

Turista

O objeto é um "turista", o Servlet e o Banco são hospedagens, os dados são a bagagem, o objeto que tira os dados do "turista" e coloca ou no Servlet (em XML) ou no Banco são "carregadores de bagagem". Quando a "bagagem" está guardada no hotel "Banco de Dados" o "turista" recebe um ticket da bagagem para poder pegar ela depois (o Identificador único da tabela).

Papéis no XP

Big Boss / XpManager

Deve fazer:

- Agendar reuniões;
- Escrever atas;
- Manter o XP Tracker informado dos acontecimentos das reuniões

Não deve fazer:

- Deixar que problemas externos interfiram no desenvolvimento
- Dizer quando as coisas devem acontecer
- Dizer às pessoas o que fazer
- Cobrar das pessoas

Papéis no XP

Xp Gold Owner (Cliente - O proprietário do ouro)

É quem paga pelo sistema, geralmente o dono da empresa.

Xp Goal Donor

Deve fazer:

- Escrever User Stories
- Definir prioridades
- Definir testes de aceitação
- Validar testes de aceitação
- Esclarecer dúvidas

Não deve fazer:

- Implementar código
- Definir quanto tempo uma tarefa leva para ser feita

Papéis no XP

Coordenadores

Xp Coach

Responsável pela negociação com o cliente quanto ao escopo e pela coordenação do *Planning Game*.

Xp Tracker

Deve fazer:

- Coletar métricas
- Manter todos informados do que está acontecendo
- Definir testes de aceitação
- Tomar atitudes diante de problemas
- Sugerir sessões de CRC (Class, Responsibilities, Collaboration)



métricas

Papéis no XP

Programador (Driver/Partner)

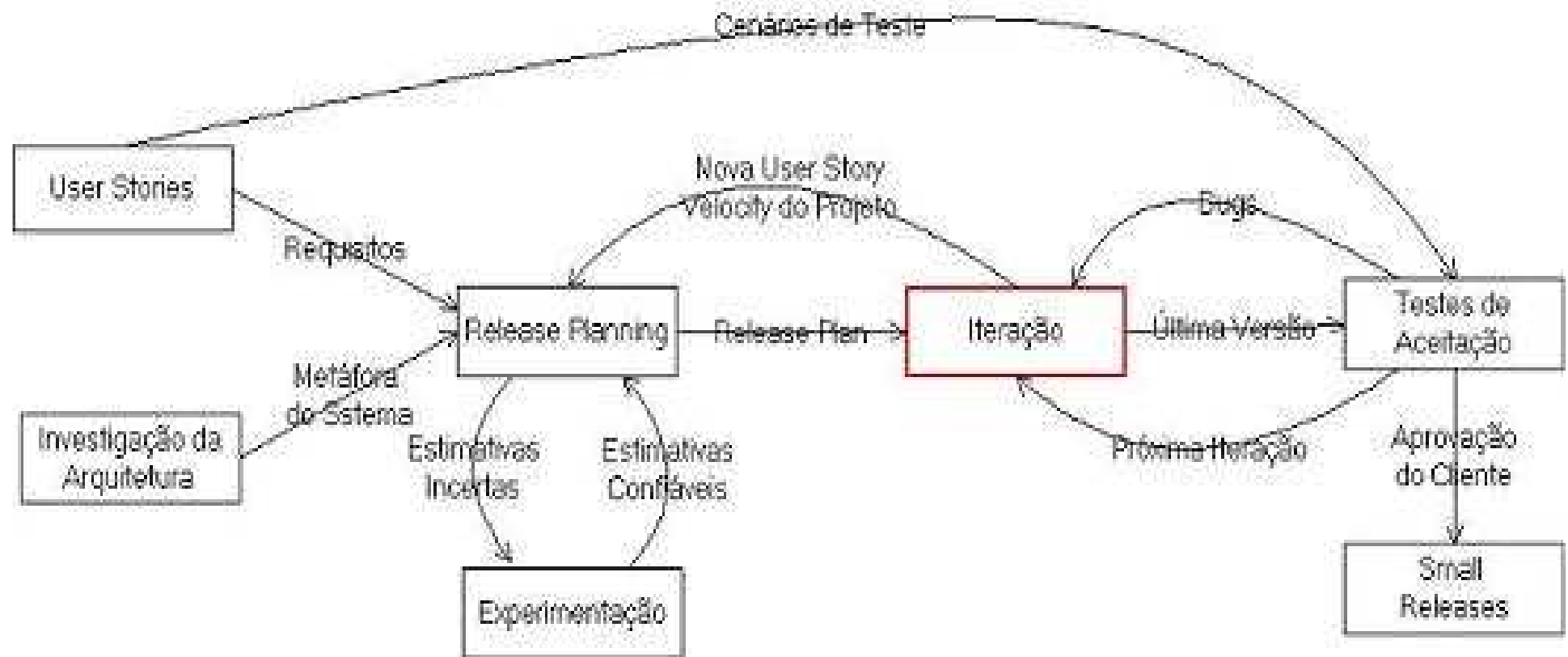
Deve fazer:

- Estimar prazos de User Stories
- Implementar código de produção
- Trabalhar em par
- Fazer refactoring
- Corrigir bugs

Não deve fazer:

- Criar/Alterar novas funcionalidades
- Escrever testes de aceitação

Um Projeto XP



CARTÃO DE USER STORY

Projeto: *Everest*

Iteração: *02*

Prioridade: *08*

Título: *Cadastro de usuário*

Descrição:

Acessando-se o menu principal, administradores do sistema têm privilégio de acesso ao módulo de cadastro de usuário.

O módulo consiste num formulário simples que permite o cadastramento de informações como: nome, sobrenome, endereço, e-mail, telefone, estado e cidade.

Após o cadastramento, é exibida uma página com a listagem de usuários.

O evento de cadastramento é registrado em log de atividades.

Analisado em *01 / 08 / 2005*

Por:

Luiz e Bruno

Planejado em *05 / 08 / 2005*

Por:

Leandro

Estimado em *02* story points

Iniciado em *07 / 08 / 2005*

Por:

Leandro

Terminado em *09 / 08 / 2005*

Realizado em *03* story points

Um quadro de “histórias”



Um Dia na Vida de um Programador XP

- Escolhe uma história do cliente.
- Procura um par livre.
- Escolhe um computador para programação pareada.
- Seleciona um “cartão de história” contendo uma tarefa claramente relacionada a uma característica (*feature*) desejada pelo cliente.

Um Dia na Vida de um Programador XP

- Discute modificações recentes no sistema
- Discute história do cliente
- Testes
- Implementação
- Projeto (*design*)
- Integração

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.

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 (e.g., cada método em cada caso).
- Testes de funcionalidades (*Functional tests*)
 - escritos pelos clientes para testar a funcionalidade do sistema.

Testes

- Testes das unidades do sistema
 - Tem que estar sempre funcionando a 100%.
 - São executados várias vezes por dia.
 - Executados à noite automaticamente.
- Testes das funcionalidades
 - Vão crescendo de 0% a 100%.
 - Quando chegam a 100% acabou o projeto.

O Código

- Padrões de estilo adotados pelo grupo inteiro.
- O mais claro possível.
 - XP não se baseia em documentações detalhadas e extensas (perde-se sincronia).
- Comentários sempre que necessários.
- Comentários padronizados.
- Programação Pareada ajuda muito!

Programação Pareada

- 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 (*gambiarrras*) na frente do seu par.
- Pareamento de acordo com especialidades.
 - Ex.: Sistema Multimídia Orientado a Objetos

Propriedade Coletiva do Código

- 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!

Refatoração (*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 Refatoração

- 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)`

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.

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 medíocres.

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 (muito raro).

Conclusão

Vencendo os Medos

- Escrever código.
- Mudar de idéia.
- Ir em frente sem saber tudo sobre o futuro.
- Confiar em outras pessoas.
- Mudar a arquitetura de um sistema em funcionamento.
- Escrever testes.

As 12 Práticas de XP

(versão 2000)

1. Planejamento
2. Fases Pequenas
3. Metáfora
4. Design Simples
5. Testes
6. Refatoramento
7. Programação Pareada
8. Propriedade Coletiva
9. Integração Contínua
10. Semana de 40 horas
11. Cliente junto aos desenvolvedores
12. Padronização do código

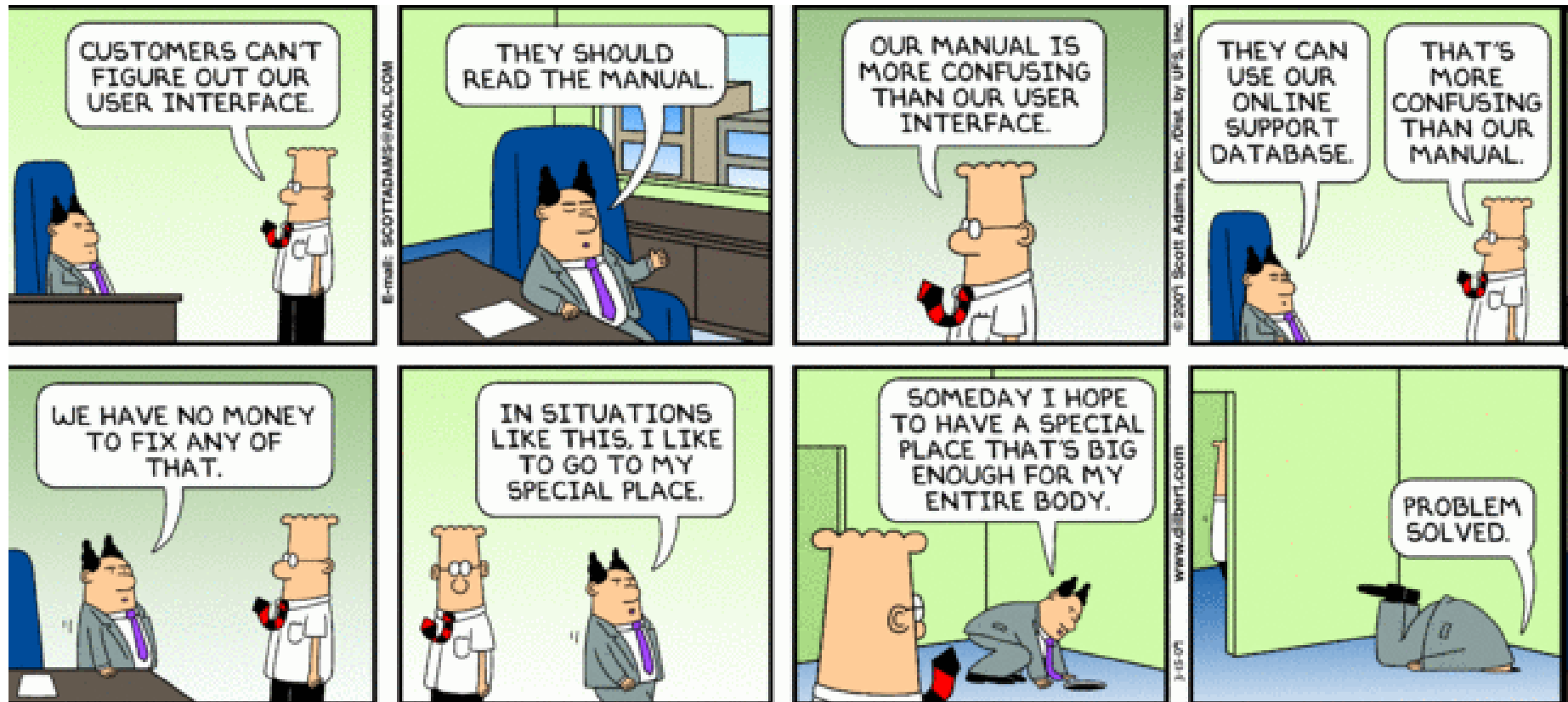
Práticas de XP

- As práticas foram refatoradas
(veja www.extremeprogramming.org/rules.html)
- Mas a idéia é exatamente a mesma
- Novas práticas interessantes:
 - Conserte XP quando a metodologia quebrar.
 - Mova as pessoas.
 - Client Proxy (by Klaus)

Portanto

- XP não é para todo mundo.
- Mas todo mundo pode aprender com ela.

Para que Engenharia de Software?



Características Comuns dos Métodos Ágeis

- Coloca o foco
 - Na entrega freqüente de sub-versões do software [funcionando] para o cliente.
 - Nos seres humanos que desenvolvem o software.
- Retira o foco de
 - Processos rígidos e burocratizados.
 - Documentações e contratos detalhados.
 - Ferramentas que são usadas pelos seres humanos.