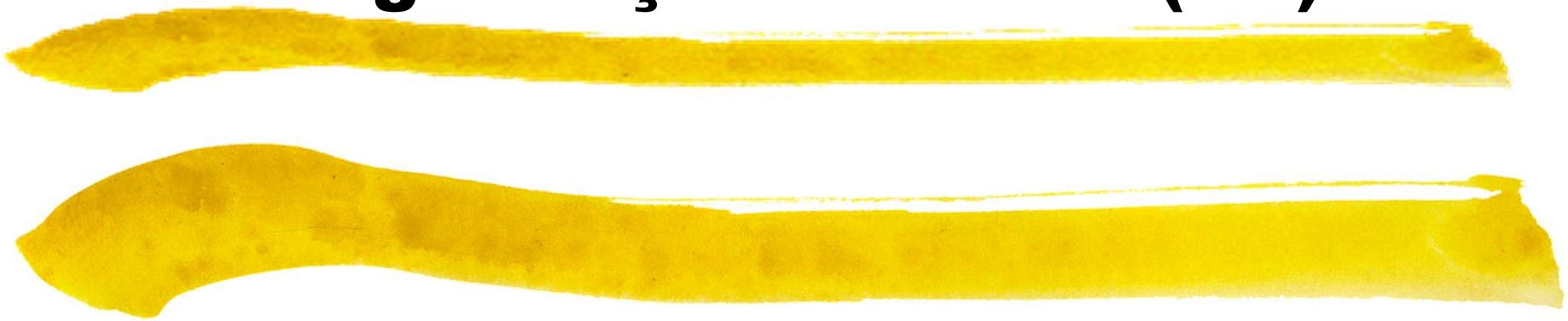
Desenvolvimento Ágil de Software com Programação eXtrema (XP)



Ricardo Argenton Ramos

Engenharia de Software I – 2012.2

Novos ventos 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.

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

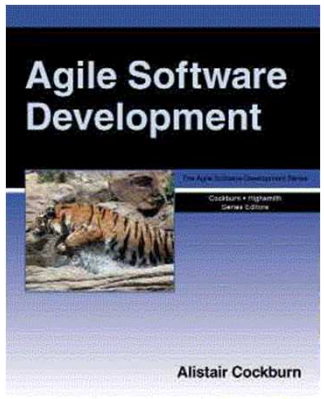
- Objetivo: satisfazer o cliente entregando, rapidamente e com freqüência, sistemas com algum valor.
  - Entregar versões funcionais em prazos curtos.
  - Estar preparado para requisitos mutantes.
  - Pessoal de negócios e desenvolvedores juntos.
  - Troca de informações através de conversas diretas.

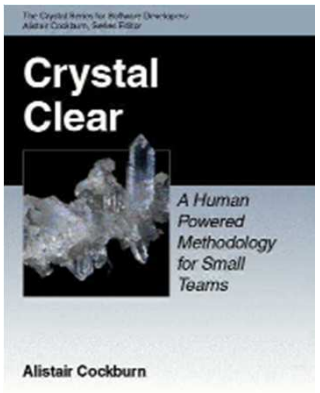
Principais Métodos Ágeis

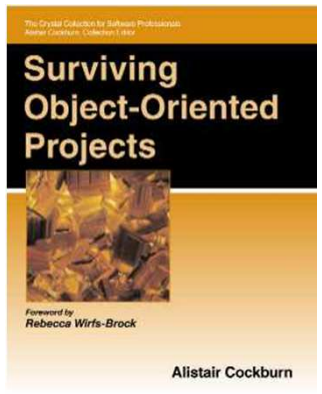
- Nesta palestra nos concentraremos em :
  - Programação eXtrema (XP)
- Outros métodos ágeis interessantes:
  - Crystal (uma família de métodos)
  - Scrum
  - Adaptive Software Development
  - Feature Driven Development
  - etc.

A família *Crystal* de Métodos

- Criada por Alistair Cockburn
- http://alistair.cockburn.us/crystal
- Editor da série *Agile Software Development* da Addison-Wesley.







Principais Características da Família Crystal

- Filosofia básica:
  - ênfase em comunicação
  - leve nos produtos gerados (evitar "peso morto")
- Princípios:
  - Precisamos de menos produtos intermediários se possuímos:
    1. **canais de comunicação informal** ricos e rápidos
    2. **entrega freqüente de código funcionando**
    3. **uso dos pontos fortes das pessoas** (conversas, olhar a sua volta, interagir com outros)
    4. **estar ciente dos pontos fracos das pessoas** (baixa disciplina, falta de cuidado)

Adaptação da Metodologia

- Em cada caso, escolha a metodologia mais leve possível que pode fazer o que você precisa.
- Quanto maior o projeto (número de pessoas), maior burocracia será necessária e pior será a produtividade.
- *Reflection Workshops* ao final de cada fase.

Oficinas de Reflexão (*Reflection Workshops*)

- Perguntas:
  - O que aprendemos na última fase (p.ex. mês)?
  - O que podemos fazer de uma forma melhor?
- Resultado:
  - pôster

Manter	*Tentar*
reuniões com cliente *programação pareada*	*testes automatizados* *multas para interrupções* *escrita pareada de testes*
Problemas	
muitas interrupções *erros no código entregue*	

Mais perguntas na reflexão

- O que fizemos de bom e de ruim?
- Quais são as nossas prioridades?
- O que mantivemos de mais importante?
- O que podemos mudar para a próxima vez?
- O que podemos adicionar/tirar?

- Após 2 ou 3 versões incrementais, a metodologia deve começar a convergir para uma metodologia tolerável para o projeto.

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 Suttherland
  - 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.

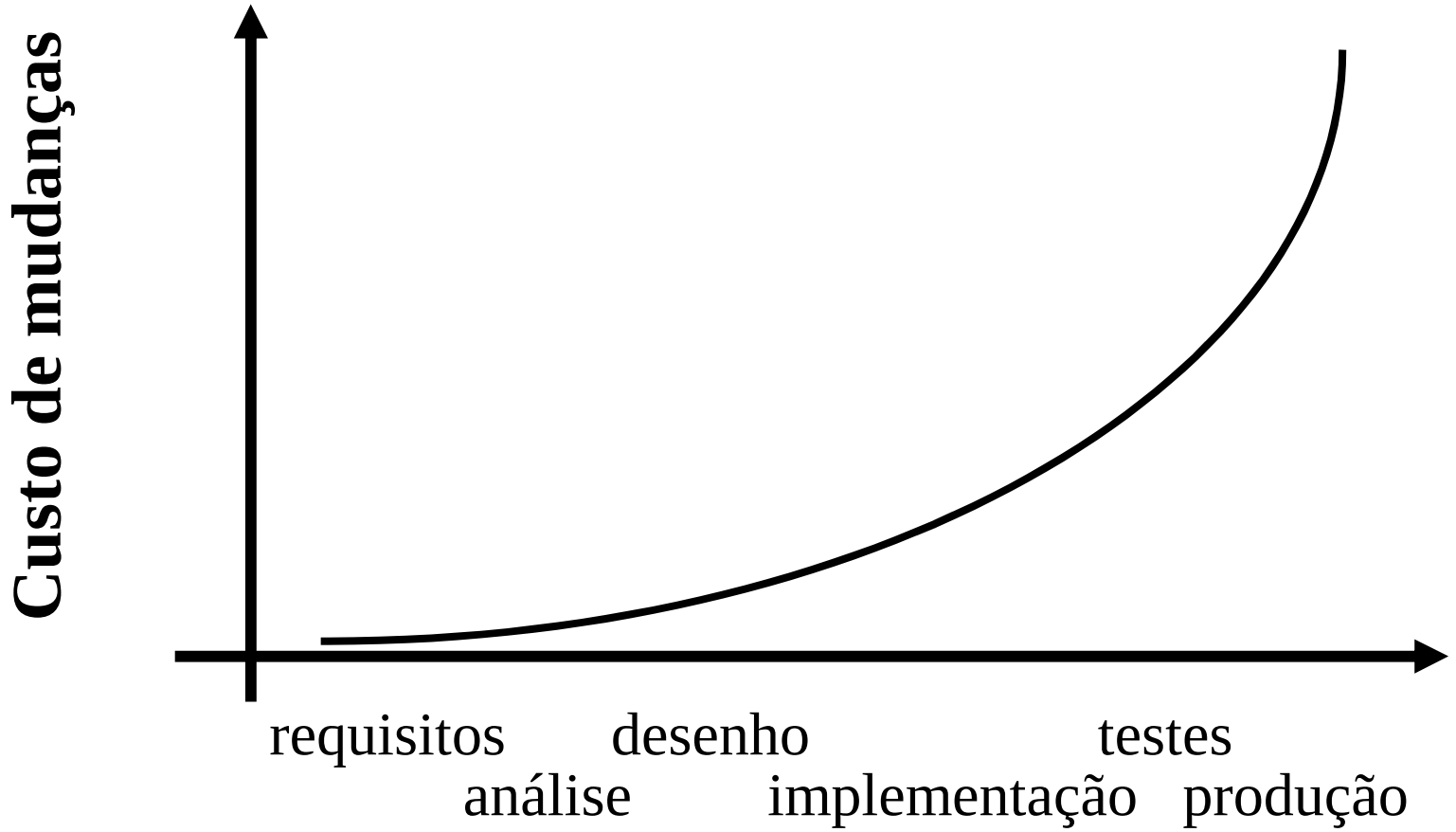
Modelo Tradicional de Desenvolvimento de Software

0. Levantamento de Requisitos
    1. Análise de Requisitos
        2. Desenho da Arquitetura
            3. Implementação
                4. Testes
5. Produção / Manutenção

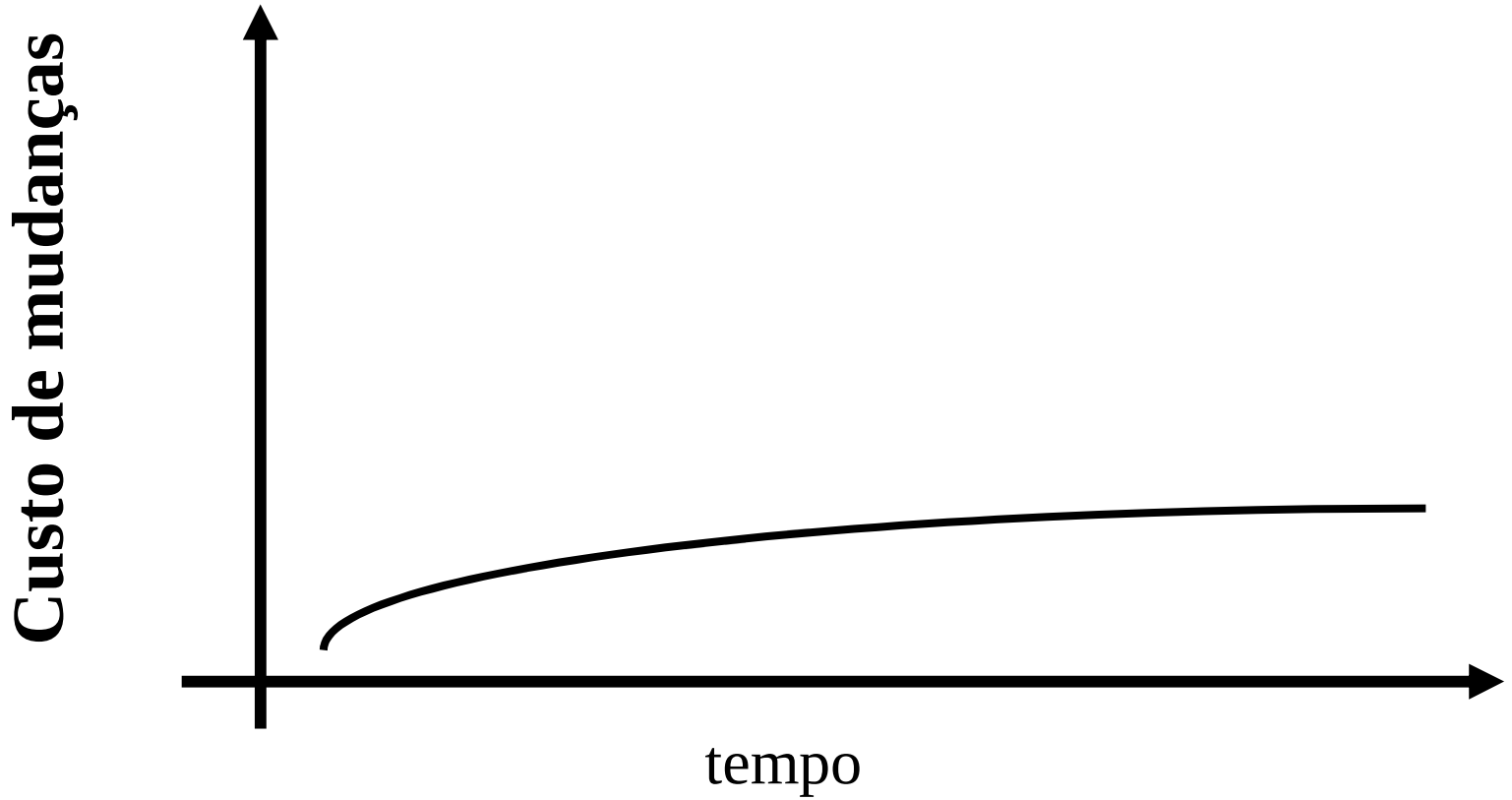
Premissas Básicas do Modelo Tradicional

- 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.

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 (*gambiarras*) 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.

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.