

Métodos Ágeis de Desenvolvimento de Software e a Programação eXtrema (XP)



Cursos de Verão 2007 - IME/USP

Profs. Alfredo Goldman e Fabio Kon
Departamento de Ciência da Computação

www.agilcoop.org.br

Nossa experiência

- Cursos de graduação em XP desde 2001
- Apresentações
 - SBES, SUCESU, SEPAI, SEBRAE, etc.
 - Arquivos disponíveis: AgilCoop
- Assessorias em métodos ágeis
- Artigos científicos

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.

Roteiro

- Motivação
 - Problemas e possíveis direções
- Métodos ágeis
 - Princípios
 - O erro dos métodos tradicionais
 - Alguns métodos ágeis
 - Apresentação mais detalhada de XP

Problemas

- Com métodos tradicionais 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 desta apresentação)
 - outras... (abordadas em ES)

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, Lean, etc.

O que é desenvolvimento de software ?

- Por: Alistair Cockburn

Modelagem (Jacobson)

Engenharia (Meyer)

Disciplina (Humphreys)

Poesia (Cockburn)

Artesanato (Knuth)

Arte (Gabriel)

- Se você sabe o que é, pode aplicar soluções conhecidas

Desenvolver software é um processo reprodutível ?

- Para isto é necessário o controle das variáveis
 - Problema. \exists fase para definição de requisitos
 - Podem não ser bem definidos, ou mudar
 - Solução. Especificação completa da arquitetura
 - Além do problema acima, envolve processo criativo
 - Desenvolvedores.
 - São humanos (logo, diferentes)
 - Ambiente.
 - Pressão de cronograma, prioridades e clientes mudam

Métodos Ágeis de Desenvolvimento de Software

- Movimento iniciado por programadores experientes e consultores em desenvolvimento de software.
- Questionam e se opõem 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.
 - <http://agilemanifesto.org>

O Manifesto do *Desenvolvimento Ágil de Software*

- **Indivíduos e interações** são mais importantes que *processos e ferramentas*.
- **Software funcionando** é mais importante do que *documentação completa e detalhada*.
- **Colaboração com o cliente** é mais importante do que *negociação de contratos*.
- **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.

Mais princípios do Manifesto Ágil

- Medir o progresso através de software funcionando.
- Desenvolvimento sustentável.
- Construir projetos com indivíduos motivados.
- Simplicidade é essencial.
- De tempos em tempos, o time pensa em como se tornar mais eficiente e ajusta o seu comportamento de acordo.

Em um projeto ágil ideal.... (1/2)

- O gerente de projeto concorda em prosseguir sem que todos os requisitos estejam bem definidos.
- Os desenvolvedores concordam em prosseguir sem ter todos os requisitos documentados.
- Os membros da equipe sabem que alguém vai ajudar quando ocorrerem problemas.

Em um projeto ágil ideal....(2/2)

- Os gerentes percebem que não precisam dizer à equipe o que fazer, ou garantir o que vai ser feito.
- A equipe percebe que ninguém vai dizer o que fazer, isto faz parte do trabalho da equipe.
- Não existem mais a impressão de divisão (*testers and programmers*), todos são desenvolvedores.

Enquanto isso, no mundo tradicional...

0. Levantamento de Requisitos

1. Análise de Requisitos

2. Desenho da Arquitetura

3. Implementação

4. Testes

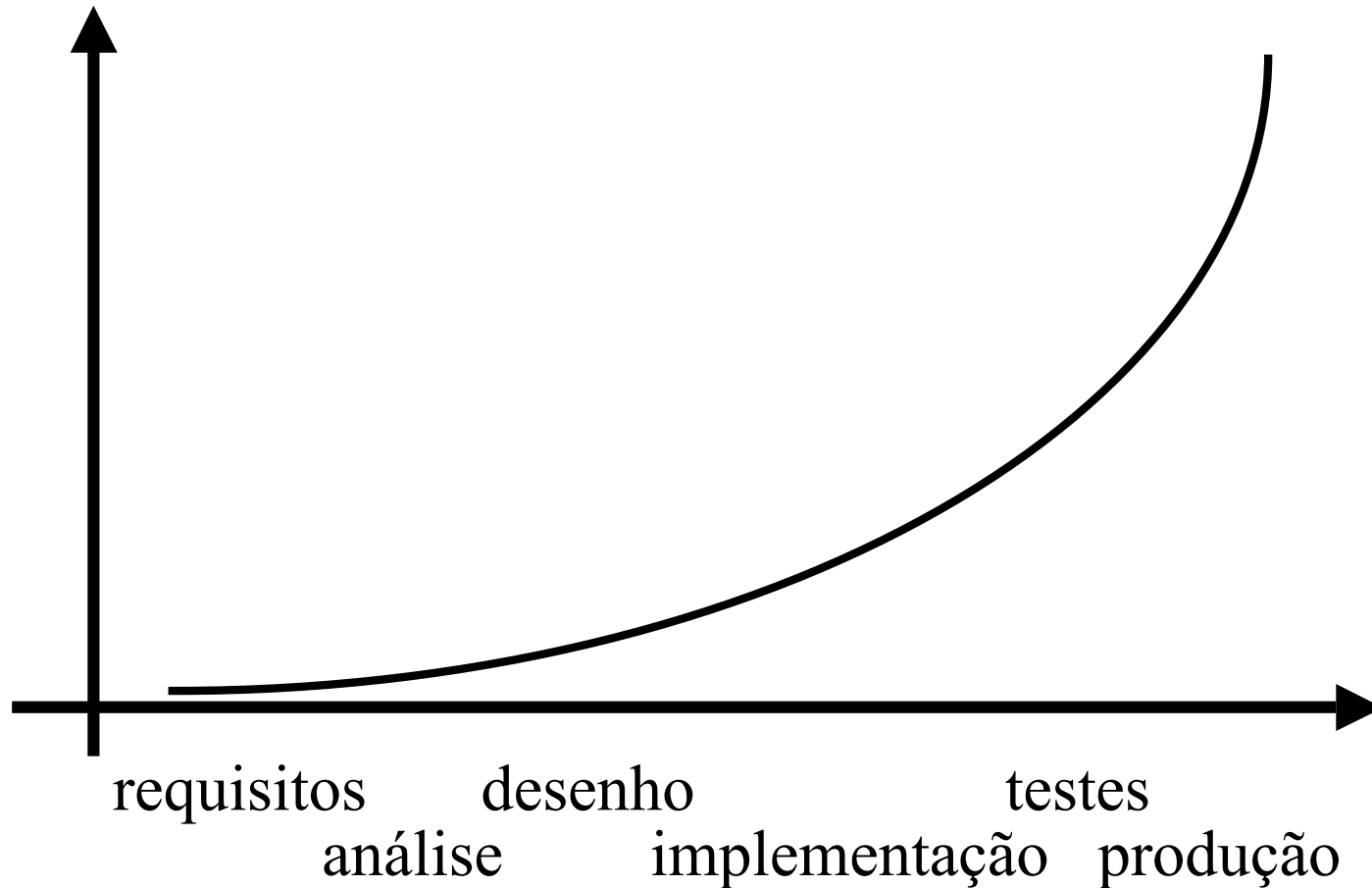
5. Produção / Manutenção

Premissas Básicas do Modelo Tradicional

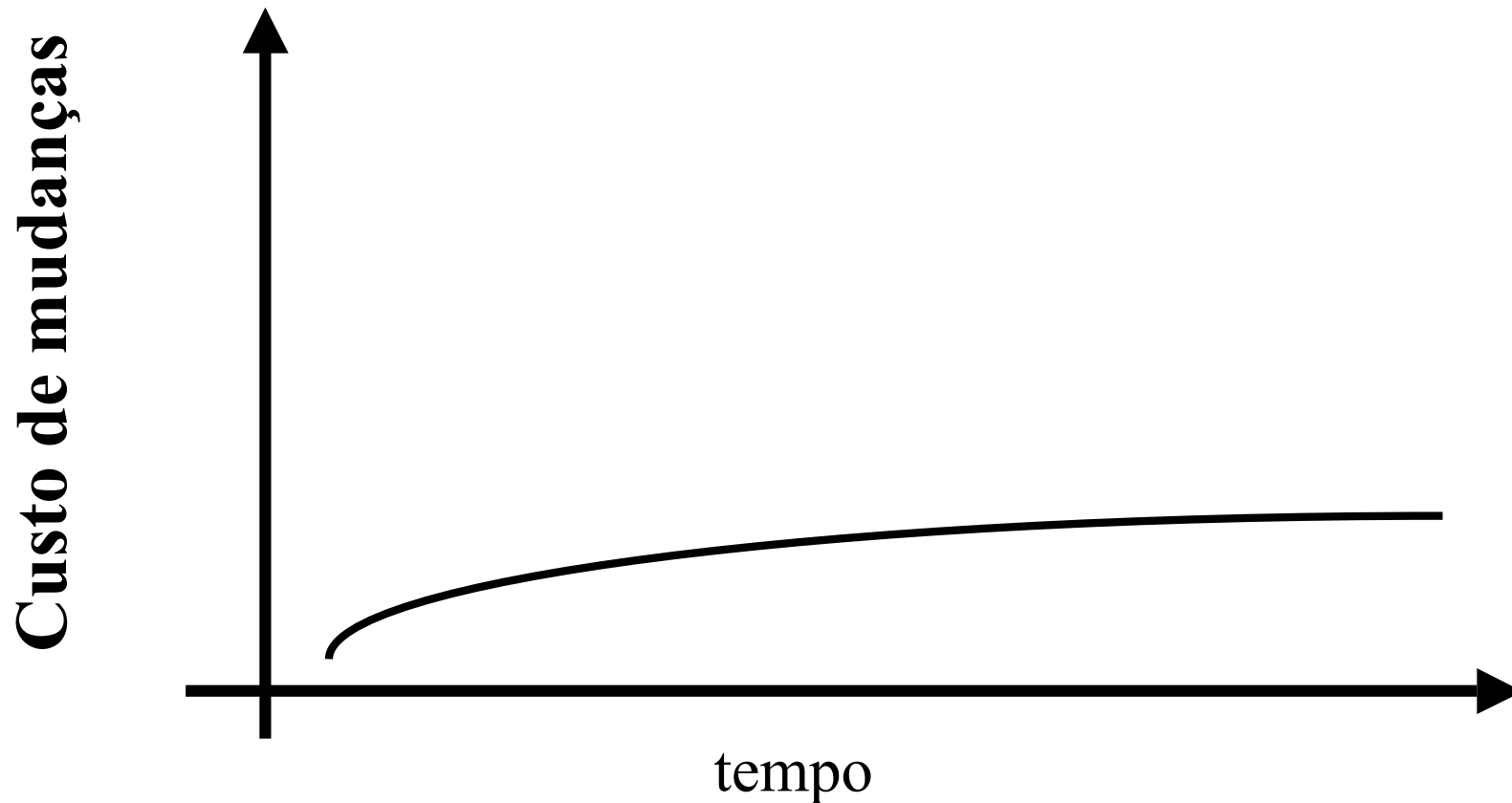
- É necessário fazer uma análise de requisitos profunda e detalhada antes de projetar a arquitetura do sistema.
- É necessário fazer um estudo minucioso e elaborar uma descrição detalhada da arquitetura antes de começar a implementá-la.
- É necessário testar o sistema completamente antes de mandar a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

O que está por trás deste modelo?

Custo de mudanças



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)

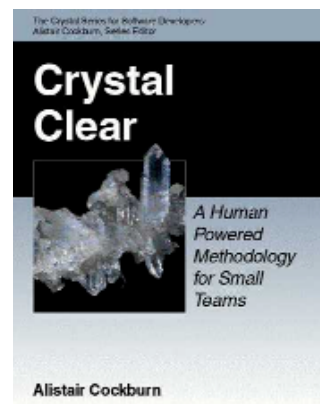
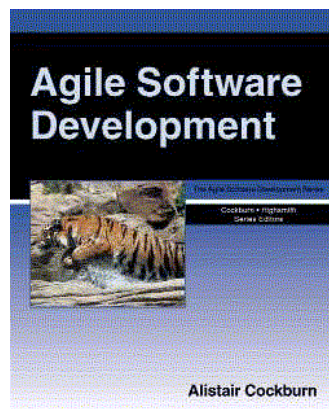
- **Orientação a Objetos:** facilita e cria oportunidades para mudanças.
- **Técnicas de Refatoração.**
- **Testes automatizados:** nos dão segurança quando fazemos mudanças.
- **Prática / cultura de mudanças:** aprendemos técnicas e adquirimos experiência em lidar com código mutante.

Principais Métodos Ágeis

- Hoje concentrar-nos-emos 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.



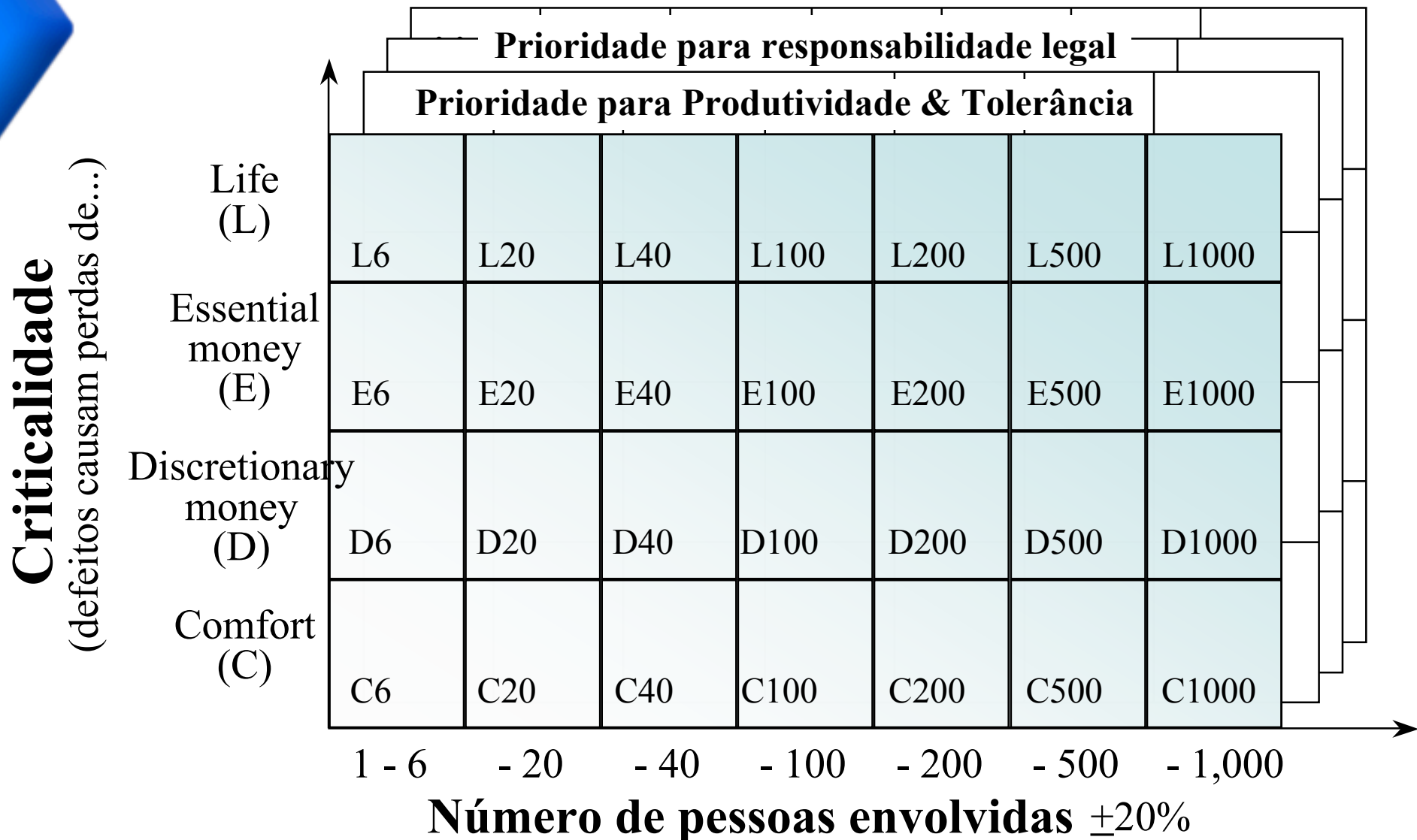
Pequeno histórico

- Foi contratado em 1990 pela IBM
 - documentar a metodologia de desenvolvimento
 - optou por entrevistar as diversas equipes
- Resultado:
 - Pedidos de desculpa por não utilizar ferramentas
 - e entregar software funcionando
 - Explicações sobre como metodologias foram seguidas à risca
 - e o projeto falhou

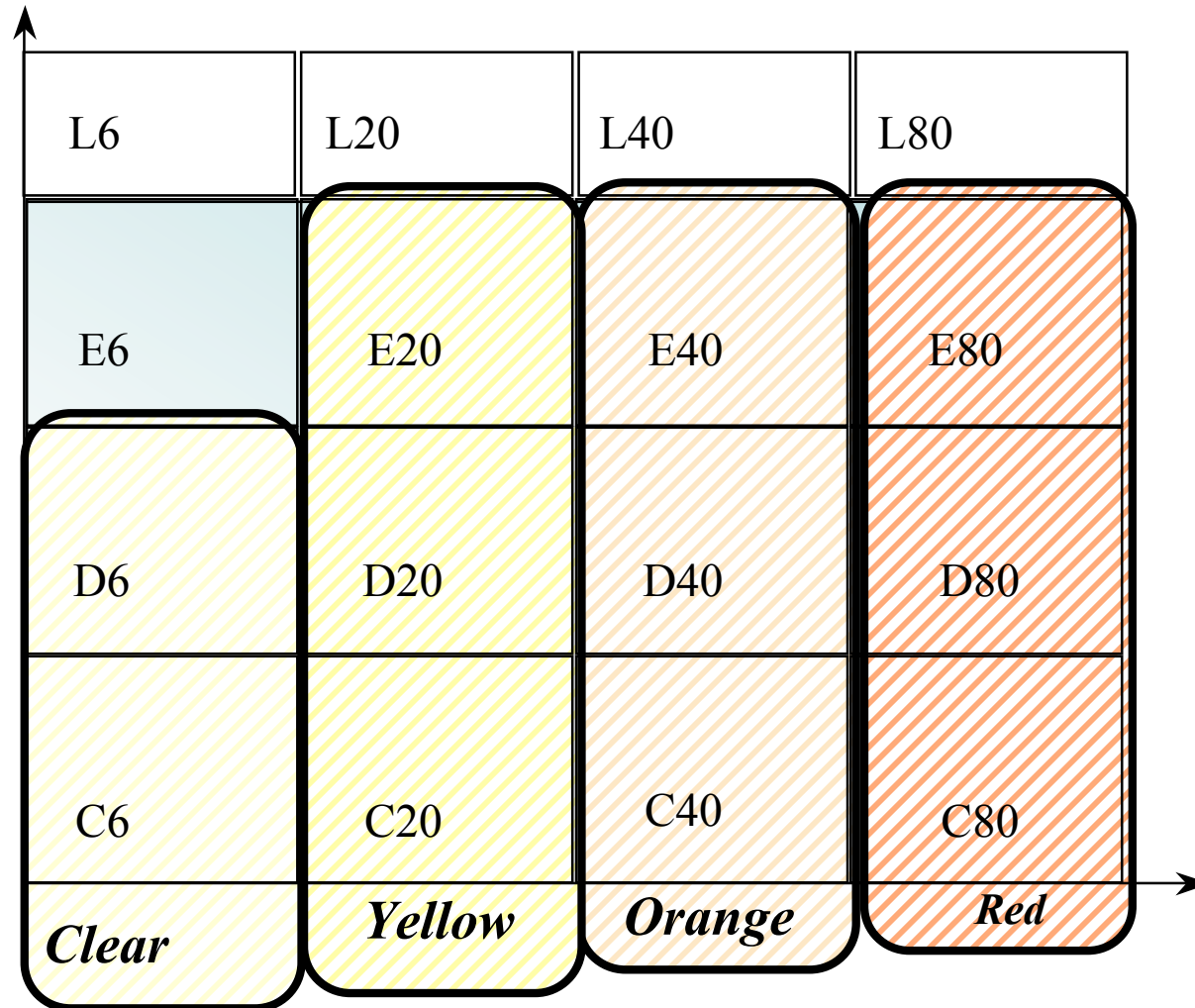
Conclusões de Cockburn

- Pode-se trocar (com vantagens) documentação escrita por conversas face a face.
- Quanto mais se entrega software testado e funcionando, menos se depende da documentação escrita (promissórias).

Classificação de Projetos de Desenvolvimento de Software



Escopo da Família Crystal



The diagram illustrates the scope of the Crystal family of agile frameworks. It features a 5x4 grid of cells. The columns are labeled L6, L20, L40, and L80 at the top. The rows are labeled E6, D6, C6, and Clear on the left. The cells are color-coded and patterned: E6 is light blue; E20, D6, and C20 are yellow with diagonal lines; E40, D40, and C40 are orange with diagonal lines; E80, D80, and C80 are red with diagonal lines. The bottom row contains the labels Clear, Yellow, Orange, and Red. A vertical arrow is on the left and a horizontal arrow is on the right of the grid.

	L6	L20	L40	L80
E6	E6	E20	E40	E80
D6	D6	D20	D40	D80
C6	C6	C20	C40	C80
	<i>Clear</i>	<i>Yellow</i>	<i>Orange</i>	<i>Red</i>

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:
 - **canais de comunicação informal** ricos e rápidos
 - **entrega freqüente de código funcionando**
 - **uso dos pontos fortes das pessoas** (conversas, olhar a sua volta, interagir com outros)
 - **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

*reuniões com cliente
programação pareada*

Tentar

*testes automatizados
muitas 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

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

Programação eXtrema XP

- Metodologia de desenvolvimento de software aperfeiçoada nos últimos 8 anos.
- Ganhou notoriedade a partir da OOPSLA'2000.
- Nome principal: Kent Beck.

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.

A Quem se Destina XP?

- Grupos de 2 a 10 [20] 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.

Os Valores de XP

- Comunicação
- *Feedback*
- Coragem
- Simplicidade
- Respeito
 - Novo valor

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: 1 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 (p.ex., cada método em cada caso).
- Testes de funcionalidades (*Functional tests*)
 - escritos pelos clientes para testar a funcionalidade do sistema (Testes de aceitação).

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.
 - nomes claros (*intention-revealing names*)
 - 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
- Experimentos controlados mostraram que a qualidade do código produzido aumenta sem perda de velocidade [Laurie Williams]

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.
 - (eXPeriente em metodologia, não na tecnologia)
- 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 testes de unidade.
 - 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 [>20] programadores).
- Quando *feedback* rápido não é possível:
 - sistema demora 6h para compilar.
 - testes demoram 12h para rodar.
 - exigência de certificação que demora meses.
- Quando não é possível realizar testes (muito raro).
- Quando o custo de mudanças é essencialmente exponencial.

Conclusão:

Ser Ágil = Vencer 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).

Conclusão

- XP não é para todo mundo
 - mas é para *quase* todo mundo
- Mas todos podemos aprender com ela
 - melhorando a qualidade do software
 - melhorando a produtividade

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.

Maiores Informações

AgilCoop:

`www.agilcoop.org.br/portal/agilcast`

Outros:

`www.agilealliance.org`

`www.xispe.com.br`

`www.extremeprogramming.org`

`www.agilealliance.com.br`