

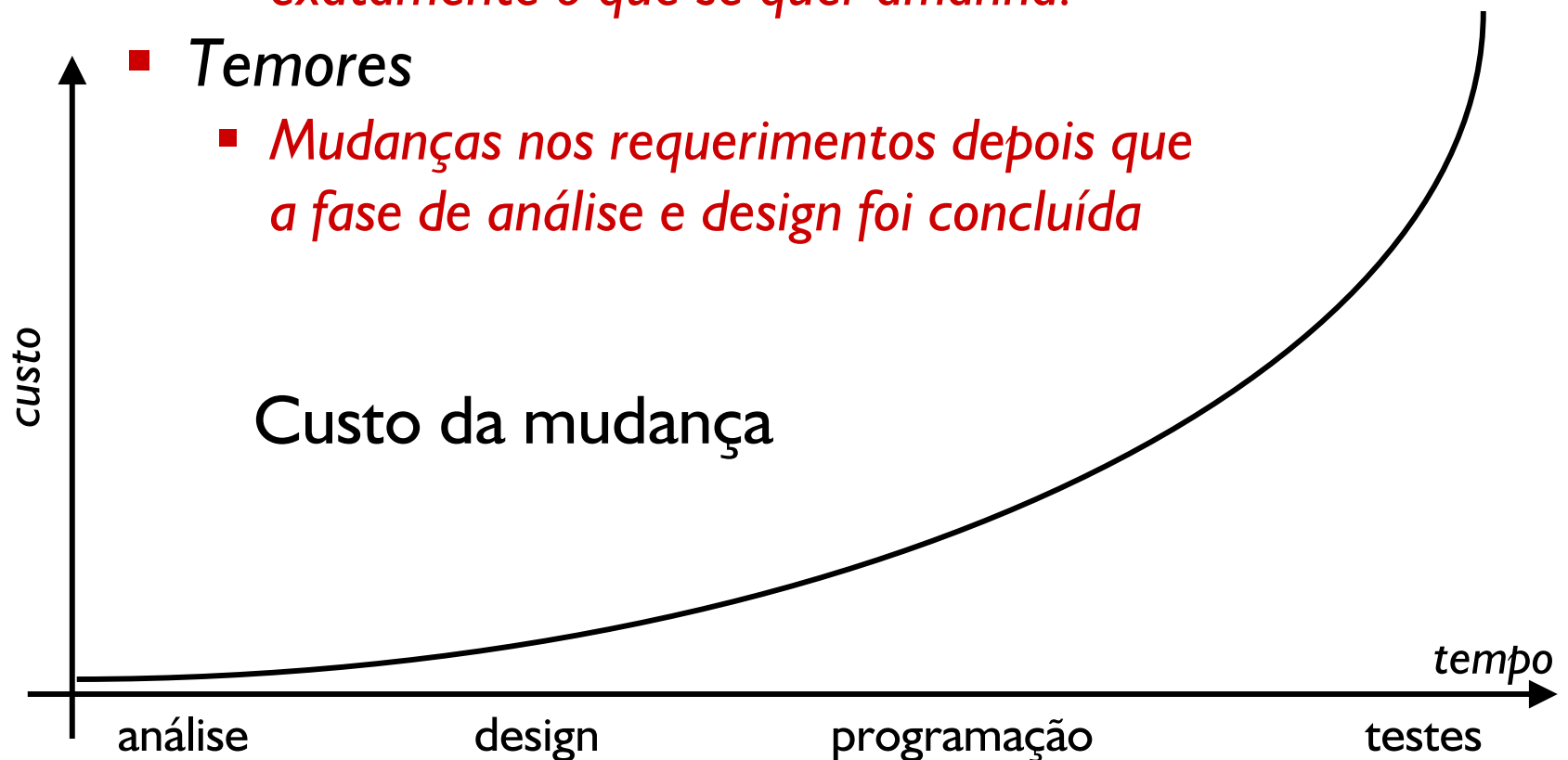
abraçando a mudança

~~extreme~~ Programming

Helder da Rocha
www.argonavis.com.br

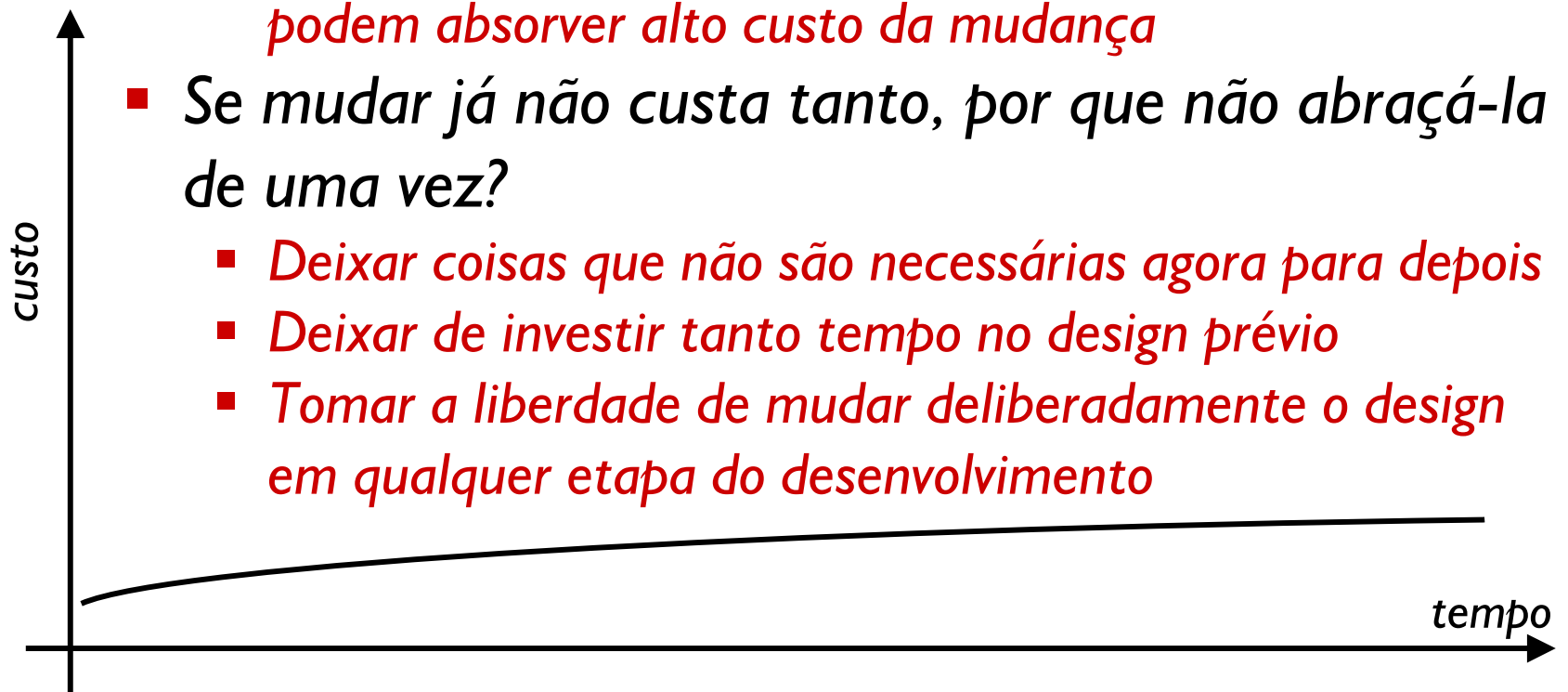
Desenvolvimento de software no passado

- Engenharia de software tradicional
 - *Analisar, projetar, e só depois começar a construir*
- Era preciso prever o futuro
 - *Como ter certeza que se sabe hoje exatamente o que se quer amanhã?*
- Temores
 - *Mudanças nos requerimentos depois que a fase de análise e design foi concluída*



Desenvolvimento hoje

- *Metodologias modernas incentivam iterações curtas*
 - *Mudanças em etapas avançadas do desenvolvimento seriam mais baratas*
- *Mudança não é mais um bicho tão feio*
 - *Podemos conviver com a mudança*
 - *Patterns, componentes, bancos de dados já absorvem ou podem absorver alto custo da mudança*
- *Se mudar já não custa tanto, por que não abraçá-la de uma vez?*
 - *Deixar coisas que não são necessárias agora para depois*
 - *Deixar de investir tanto tempo no design prévio*
 - *Tomar a liberdade de mudar deliberadamente o design em qualquer etapa do desenvolvimento*



eXtreme Programming

- **Metodologia ágil** para equipes pequenas a médias desenvolvendo software com requerimentos vagos ou que mudam frequentemente [Beck 2000]
- Em XP, **codificação** é principal tarefa
- Ênfase
 - menor em processos formais de desenvolvimento
 - maior em disciplina rigorosa
- Baseia-se em
 - revisão permanente do código, testes frequentes, participação do usuário final, refatoramento contínuo, refinação contínua da arquitetura, integração contínua, planejamento, design e redesign a qualquer hora

“Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através desse trabalho, passamos a valorizar:

- *Indivíduos e interações **mais que** processos e ferramentas;*
- *Software em funcionamento **mais que** documentação abrangente;*
- *Colaboração com o cliente **mais que** negociação de contratos;*
- *Responder a mudanças **mais que** seguir um plano.*

Ou seja, mesmo dando valor aos itens à direita, valorizamos mais os itens à esquerda.”

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

Quatro valores do XP

- *1. Comunicação*
- *2. Simplicidade*
- *3. Feedback*
- *4. Coragem*

I. Comunicação

- *Várias práticas do XP promovem uma maior **comunicação** entre os membros da equipe*
- *A comunicação não é limitada por procedimentos formais. Usa-se o melhor meio possível, que pode ser*
 - *Uma conversa ou reunião informal*
 - *Um e-mail, um bate-papo, um telefonema*
 - *Diagramas, se necessário (pode, mas não precisa, ser UML)*
 - *O próprio código*
 - *"Estórias" elaboradas pelo usuário-final*
 - *...*
- *Preferência à comunicação mais ágil*
 - *Telefonema melhor que e-mail*
 - *Presença física melhor que comunicação remota*
 - *Código auto-explicativo melhor que documentação escrita*

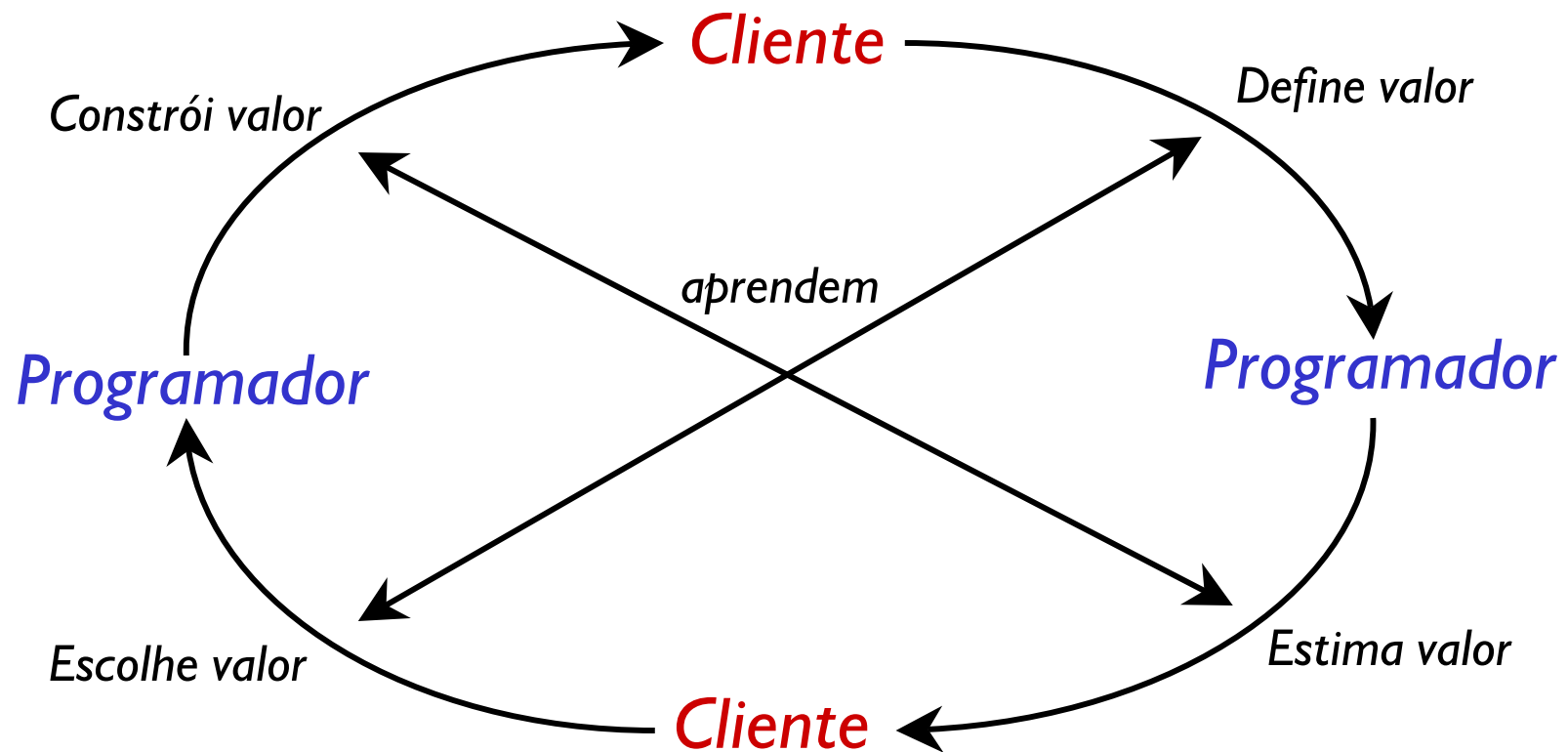
2. Simplicidade

- *XP incentiva ao extremo práticas que reduzam a complexidade do sistema*
- *A solução adotada deve ser sempre a mais simples que alcance os objetivos esperados*
 - *Use as tecnologias, design, algoritmos e técnicas mais simples que permitirão atender aos requerimentos do usuário-final*
 - *Design, processo e código podem ser simplificados a qualquer momento*
 - *Qualquer design, processo ou código criado pensando em iterações futuras deve ser descartado*

3. Feedback

- *Várias práticas do XP garantem um rápido feedback sobre várias etapas/partes do processo*
 - *Feedback sobre qualidade do código (testes de unidade, programação em pares, posse coletiva)*
 - *Feedback sobre estado do desenvolvimento (estórias do usuário-final, integração contínua, jogo do planejamento)*
- *Permite maior agilidade*
 - *Erros detectados e corrigidos imediatamente*
 - *Requisitos e prazos reavaliados mais cedo*
 - *Facilita a tomada de decisões*
 - *Permite estimativas mais precisas*
 - *Maior segurança e menos riscos para investidores*

Ciclo de vida e feedback

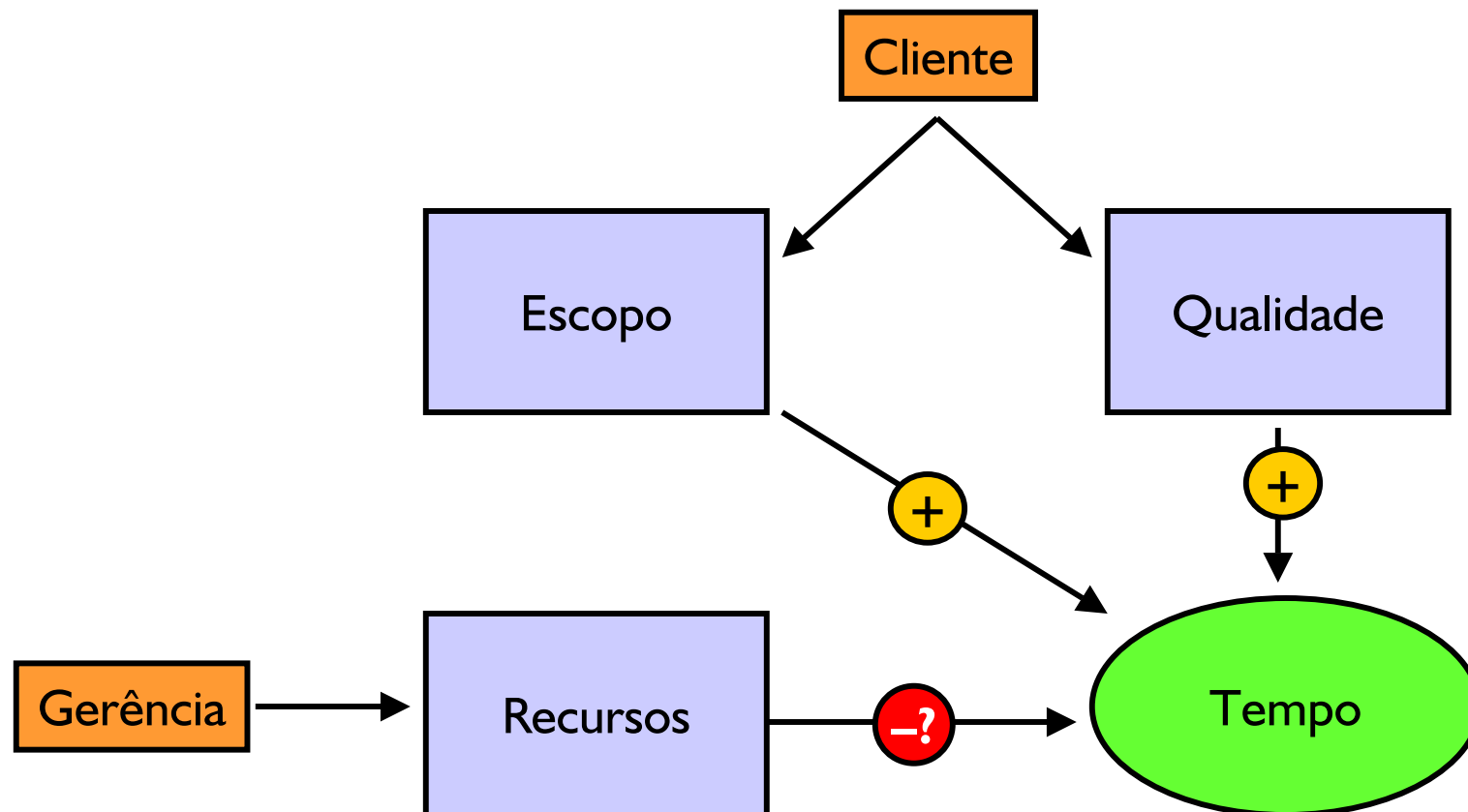


4. Coragem

- *Testes, integração contínua, programação em pares e outras práticas do XP aumentam a confiança do programador e ajudam-no a ter **coragem** para*
 - *melhorar o design de código que está funcionando para torná-lo mais simples*
 - *jogar fora código desnecessário*
 - *investir tempo no desenvolvimento de testes*
 - *mexer no design em estágio avançado do projeto*
 - *pedir ajuda aos que sabem mais*
 - *dizer ao cliente que um requisito não vai ser implementado no prazo prometido*
 - *abandonar processos formais e fazer design e documentação em forma de código*

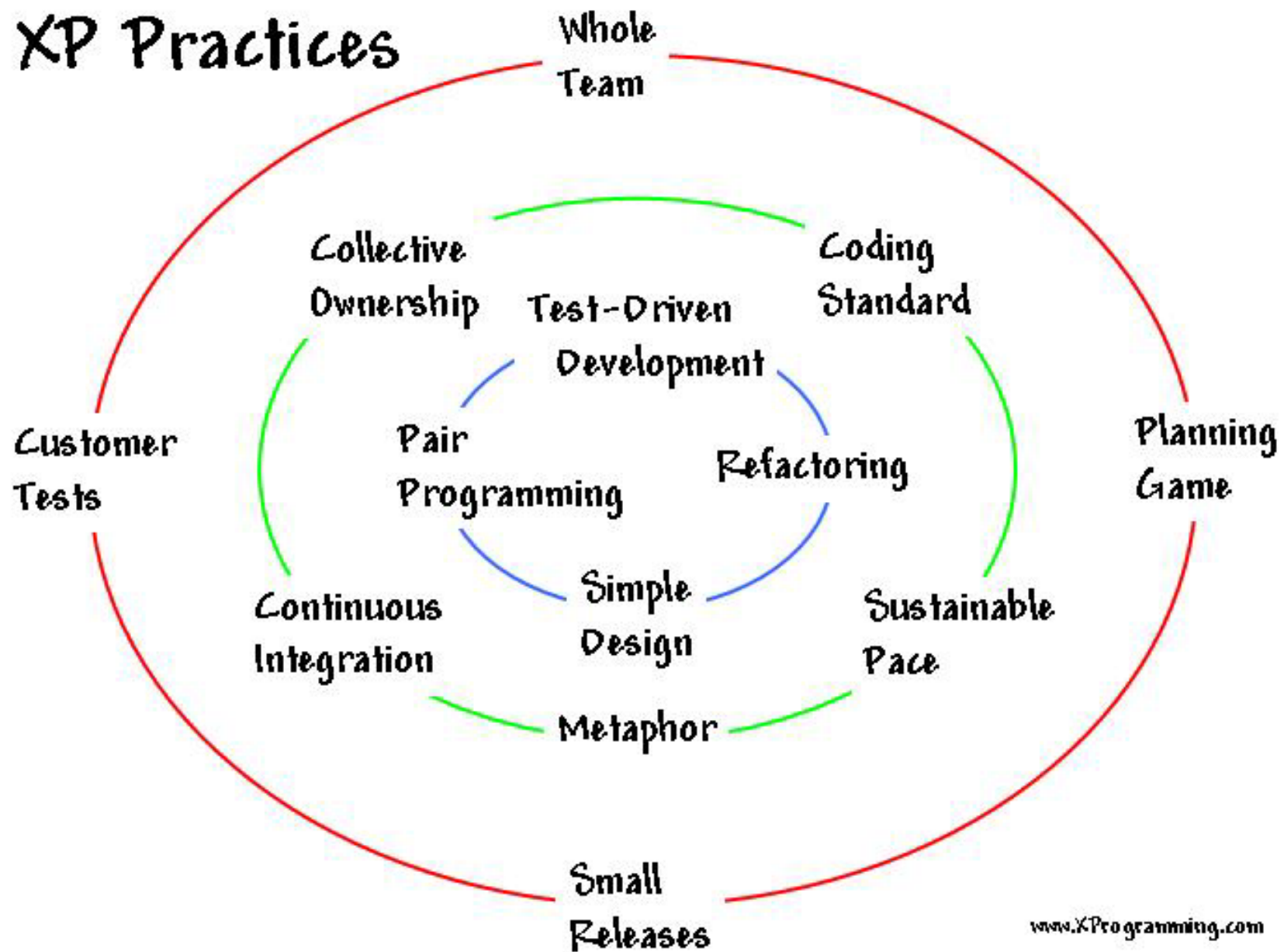
Como viabilizar?

- *Quatro variáveis*
 - *3 controláveis: escopo, qualidade, recursos*
 - *1 determinada pela dinâmica do desenvolvimento: tempo*



- *XP leva o senso comum aos extremos*
 - *Senso comum: revisão de código*
 - *XP: programar em pares*
 - *Senso comum: testes frequentes*
 - *XP: testes o tempo todo, até antes de escrever código*
 - *Senso comum: simplicidade*
 - *XP: simplicidade mandatória - recursos não prioritários são descartados*
 - *Senso comum: design*
 - *XP: design a qualquer hora*
 - ...

XP Practices



I. A Equipe (Whole Team)

- Todos em um projeto XP são parte de uma equipe.
- Esta equipe deve incluir um representante do **cliente**, que
 - *estabelece os requerimentos do projeto*
 - *define as prioridades*
 - *controla o rumo do projeto*
- O representante (ou um de seus assessores) é **usuário final** que conhece o domínio do problema e suas necessidades
- Outros **papéis** assumidos pelos integrantes da equipe:
 - *programadores*
 - *testadores (que ajudam o cliente com testes de aceitação)*
 - *analistas (que ajudam o cliente a definir requerimentos)*
 - *gerente (garante os recursos necessários)*
 - *coach (orienta a equipe, controla a aplicação do XP)*
 - *tracker (coleta métricas)*

2. Jogo do Planejamento (Planning Game)

- Prática XP na qual se define
 - **Estimativas** de prazo para cada tarefa
 - As **prioridades**: quais as tarefas mais importantes
- Dois passos chave:
 - Planejamento de um release
 - Cliente propõe funcionalidades desejadas (**estórias**)
 - Programadores avaliam a dificuldade de implementá-las
 - Planejamento de uma iteração (de duas semanas)
 - Cliente define as funcionalidades prioritárias para a iteração;
 - Programadores as quebram em tarefas e avaliam o seu custo (tempo de implementação)
- Cliente e equipe concordam com prioridades para iteração
- Ótimo feedback para que cliente possa **dirigir** o projeto
 - É possível ter uma idéia clara do avanço do projeto
 - Clareza reduz riscos, aumenta chance de sucesso

3. Testes de aceitação (Customer Tests)

- *No Planning Game, usuário-cliente elabora "estórias" que descrevem cada funcionalidade desejada. Programador as implementa*
 - *Cada estória deve ser entendida suficientemente bem para que programadores possam estimar sua dificuldade*
 - *Cada estória deve ser testável*
- **Testes de aceitação** são elaborados pelo cliente
 - *São testes automáticos*
 - *Quando rodarem com sucesso, funcionalidade foi implementada*
 - *Devem ser rodados novamente em cada iteração futura*
 - *Oferecem feedback: pode-se saber, a qualquer momento, quantos % do sistema já foi implementado e quanto falta.*

4. Pequenos lançamentos (Small Releases)

- Disponibiliza, a cada iteração, software 100% funcional
 - **Benefícios** do desenvolvimento disponíveis imediatamente
 - **Menor risco** (se o projeto não terminar, parte existe e funciona)
 - Cliente pode medir com precisão **quanto já foi feito**
 - **Feedback** do cliente permitirá que problemas sejam detectados cedo e facilita a comunicação entre o cliente e o desenvolvimento
- Cada lançamento possui funcionalidades prioritárias
 - Valores de negócio implementados foram escolhidos pelo cliente
- Lançamento pode ser destinado a
 - usuário-cliente (que pode testá-lo, avaliá-lo, oferecer feedback)
 - usuário-final (sempre que possível)
- **Design simples** e **integração contínua** são práticas essenciais para viabilizar pequenos lançamentos frequentes
- Lançamentos incluem histórias prioritárias

5. Design simples (Simple Design)



- *Design está presente em todas as etapas de no XP*
 - *Projeto começa simples e se mantém simples através de testes e refinamento do design (refatoramento).*
- *Todos buscamos design simples e claro. Em XP, levamos isto a níveis extremos*
 - *Não permitimos que se implemente **nenhuma** função adicional que não será usada na atual iteração*
- *Implementação ideal é aquela que*
 - *Roda todos os testes*
 - *Expressa todas as idéias que você deseja expressar*
 - *Não contém código duplicado*
 - *Tem o mínimo de classes e métodos*
- *O que não é necessário AGORA não deve ser implementado*
 - *Prever o futuro é "anti-XP" e impossível (requerimentos mudam!)*
 - *YAGNI (You Aint Gonna Need It)!*

6. Programação em duplas (Pair programming)

- *Todo o desenvolvimento em XP é feito **em pares***
 - *Um computador, um teclado, dois programadores*
 - *Um piloto, um co-piloto*
 - *Papéis são alternados frequentemente*
 - *Pares são trocados periodicamente*
- **Benefícios**
 - **Melhor qualidade** do design, código e testes
 - **Revisão constante** do código
 - **Nivelamento** da equipe
 - **Maior comunicação**
- *"Um" programando pelo preço de dois???*
 - *Pesquisas [Pair] demonstram que duplas produzem código de melhor qualidade em aproximadamente o mesmo tempo que programadores trabalhando sozinho*
 - *90% dos que aprendem programação em duplas a preferem*



7. Testes (Test-driven Development)

- *Desenvolvimento que não é guiado por testes não é XP*
 - *Feedback é um valor fundamental do XP, mas ...*
 - *... não há feedback sem testes!*
- **"Test first, then code"**
 - *Testes "puxam" o desenvolvimento*
 - *Programadores XP escrevem testes primeiro, escrevem código e rodam testes para validar o código escrito*
 - *Cada unidade de código só tem valor se seu teste funcionar 100%*
 - *Todos os testes são executados automaticamente, o tempo todo*
 - *Testes são a documentação executável do sistema*
- **Testes dão maior segurança: **coragem** para mudar**
 - *Que adianta a OO isolar a interface da implementação se programador tem medo de mudar a implementação?*
 - *Código testado é mais confiável*
 - *Código testado pode ser alterado sem medo*



8. Refinamento do design (Refactoring)

- *Não existe uma etapa isolada de design em XP*
 - *O código é o design!*
- *Design é melhorado continuamente através de refatoramento*
 - *Mudança proposital de código que está funcionando*
 - *Objetivos: melhorar o design, simplificar o código, remover código duplicado, aumentar a coesão, reduzir o acoplamento*
 - *Realizado o tempo todo, durante o desenvolvimento*
- *Refatoramento [Fowler 2000] é um processo formal realizado através de etapas reversíveis*
 - *Passos de refatoramento melhoram, incrementalmente, a estrutura do código sem alterar sua função*
 - *Existência prévia de testes é essencial (elimina o medo de que o sistema irá deixar de funcionar por causa da mudança)*
- *Documentação de design é menos importante uma vez que design pode ser mudado continuamente*

9. Integração contínua (Continuous Integration)

- **Projetos XP mantêm o sistema integrado o tempo todo**
 - *Integração de todo o sistema pode ocorrer várias vezes ao dia (pelo menos uma vez ao dia)*
 - *Todos os testes (unidade e integração) devem ser executados*
- **Integração contínua "reduz o tempo passado no inferno da integração" [Fowler]**
 - *Quanto mais tempo durarem os bugs de integração, mais difíceis serão de eliminar*
- **Benefícios**
 - *Expõe o estado atual do desenvolvimento (viabiliza lançamentos pequenos e frequentes)*
 - *Estimula design simples, tarefas curtas, agilidade*
 - *Oferece feedback sobre todo o sistema*
 - *Permite encontrar problemas de design rapidamente*



8. Posse coletiva (Collective Ownership)

- *Em um projeto XP qualquer dupla de programadores pode melhorar o sistema a qualquer momento.*
- *Todo o código em XP pertence a um único dono: a equipe*
 - *Todo o código recebe a atenção de todos os participantes resultando em maior comunicação*
 - *Maior qualidade (menos duplicação, maior coesão)*
 - *Menos riscos e menos dependência de indivíduos*
- *Todos compartilham a responsabilidade pelas alterações*
- *Testes e integração contínua são essenciais e dão segurança aos desenvolvedores*
- *Programação em pares reduz o risco de danos*

10. Padrões de codificação (Coding Standards)

- O código escrito em projetos XP segue um padrão de codificação, definido pela equipe
 - *Padrão para nomes de métodos, classes, variáveis*
 - *Organização do código (chaves, etc.)*
- *Todo o código parece que foi escrito por um único indivíduo, competente e organizado*
- *Código com estrutura familiar facilita e estimula*
 - *Posse coletiva*
 - *Comunicação mais eficiente*
 - *Simplicidade*
 - *Programação em pares*
 - *Refinamento do design*



11. Metáfora (Metaphor)

- *Equipes XP mantêm uma visão compartilhada da arquitetura do sistema*
 - *Pode ser uma analogia com algum outro sistema (computacional, natural, abstrato) que facilite a comunicação entre os membros da equipe e cliente*
- *Exemplos:*
 - *"Este sistema funciona como uma colméia de abelhas, buscando pólen e o trazendo para a colméia" (sistema de recuperação de dados baseados em agentes) [XPRO]*
 - *Este sistema funciona como uma agência de correios (sistema de mensagens) [Objective]*
- *Facilita a escolha dos nomes de métodos, classes, campos de dados, etc.*
 - *Serve de base para estabelecimento de padrões de codificação*



12. Ritmo saudável (Sustainable Pace)

- **Projetos XP estão na arena para ganhar**
 - *Entregar software da melhor qualidade*
 - *Obter a maior produtividade dos programadores*
 - *Obter a satisfação do cliente*
- **Projetos com cronogramas apertados que sugam todas as energias dos programadores não são projetos XP**
 - *"Semanas de 80 horas" levam à baixa produtividade*
 - *Produtividade baixa leva a código ruim, relaxamento da disciplina (testes, refatoramento, simplicidade), dificulta a comunicação, aumenta a irritação e o stress da equipe*
 - *Tempo "ganho" será perdido depois*
- **Projeto deve ter ritmo sustentável por prazos longos**
 - *Eventuais horas extras são aceitáveis quando produtividade é maximizada no longo prazo*



- *Uma prática de cada vez*
 - *Enfatize o problema mais importante*
- *Dificuldades culturais*
 - *Deixar alguém mexer no seu código*
 - *Trabalhar em pares*
- *Dificuldades devido a mudança de hábitos*
 - *Manter as coisas simples (e não tentar prever o futuro escrevendo "design flexível")*
 - *Jogar fora código desnecessário*
 - *Escrever testes antes de codificar*
 - *Refatorar com frequência (vencer o medo)*

Quando não usar XP

- **Equipes grandes e espalhadas geograficamente**
 - *Comunicação é um valor fundamental do XP*
 - *Não é fácil garantir o nível de comunicação requerido em projetos XP em grandes equipes*
- **Situações onde não se tem controle sobre o código**
 - *Código legado que não pode ser modificado*
- **Situações onde o feedback é demorado**
 - *compile-link-build-test que leva 24 horas*
 - *testes muito difíceis, arriscados e que levam tempo*
 - *Programadores espalhados em ambientes físicos distantes e sem comunicação eficiente*

XP versus CMM

- "CMM e XP podem ser considerados **complementares**. O Software CMM explica **o que** fazer em termos gerais, mas não diz **como** fazer. XP é um conjunto de boas práticas contendo informação específica de 'como fazer' para um tipo de ambiente particular"
- "Para aqueles interessados em melhoria de processos, as idéias em XP devem ser cuidadosamente consideradas para adoção em um ambiente de negócios de uma organização, assim como organizações usando XP devem considerar a gerência de recursos de infraestrutura descritas em CMM."

Mark C. Paulk, SEI/CMU

Como alcançar CMM com XP

- *XP lida com várias das práticas recomendadas em CMM níveis 2 e 3*
 - *Outras práticas, e níveis 4 e 5 devem ser resolvidas em projetos reais apesar do XP não as abordar diretamente*
- *O que XP implementa de CMM?*

Level 2 KPA's	Satis- faction	Level 3 KPA's	Satis- faction	High Maturity KPA's	Satis- faction
RM	√√	OPF	√	QPM	--
SPP	√√	OPD	√	SQM	--
SPTO	√√	TP	--		
SSM	--	ISM	--	DP	√
SQA	√	SPE	√√	TCM	--
SCM	√	IC	√√	PCM	--
		PR	√√		

√

partially addressed in XP

√√

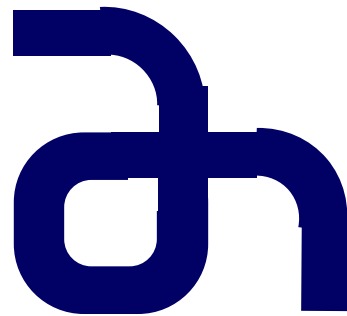
largely addressed in XP (perhaps by inference)
(in the appropriate environment)

Fonte: [Paulk]

- *Extreme Programming (XP) é uma metodologia de desenvolvimento ágil de software baseada nos valores simplicidade, comunicação, feedback e coragem.*
- *Para implementar XP não é preciso usar diagramas ou processos formais. É preciso fazer uma equipe se unir em torno de algumas práticas simples, obter feedback suficiente e ajustar as práticas para a sua situação particular.*
- *XP pode ser implementada aos poucos, porém a maior parte das práticas são essenciais.*
- *Nem todos os projetos são bons candidatos a usar uma metodologia ágil como XP. XP é mais adequado a equipes pequenas ou médias (até 10 pessoas).*
- *XP não concorre com CMM e pode ser complementar, ajudando a implementar recomendações dos níveis 2 e 3.*

- [Beck 2000] *Kent Beck. Extreme Programming Explained. Addison-Wesley, 2000*
- [Jeffries 2001] *Ron Jeffries, Ann Anderson, Chet Hendrickson. Extreme Programming Installed. Addison-Wesley, 2001*
- [Fowler 2000] *Martin Fowler. Refactoring: improving the design of existing code. Addison-Wesley, 2000*
- [Fowler] *Martin Fowler, Matthew Foemmel. Continuous Integration. <http://www.martinfowler.com/articles/continuousIntegration.html>*
- [Pair] www.pairprogramming.com
- [XPRO] www.xprogramming.com
- [Objective] <http://www.xispe.com.br/index.html>
- [Mentor] www.objectmentor.com
- [Xpers] <http://www.xpers.com.br>
- [C2] <http://www.c2.com/cgi/wiki?CategoryExtremeProgramming>
- [Paulk] *Mark C. Paulk. Extreme Programming from a CMM Perspective. www.cmu.edu*

helder@ibpinet.net



www.argonavis.com.br