EXTREME PROGRAMMING (XP)

Ian Sommerville, 8º edição – Capítulo 17
A arte do Desenvolvimento Ágil

Aula de Luiz Eduardo Guarino de Vasconcelos

Metodologia XP

"XP é sobre mudança social"

Kent Beck

- Não basta apenas excelência técnica
- XP valoriza a construção de interações sociais boas e confiáveis

Metodologia XP

- Beck K., Andres C., "*Extreme Programming Explained: Embrance Change*", 2nd Edition, Addison-Wesley, 2004
- Refatoração da 1ª Edição
- 1ª Edição → Sugeria práticas claras e específicas para a programação
- 2ª Edição → Muda a abordagem, de forma mais positiva e inclusiva

Definições

- No início do projeto, normalmente não se sabe precisamente o que se quer
- Software evolui para atender ao business
  - Software nunca fica “pronto”
- Obviamente isso só é possível porque software é uma entidade abstrata
- Portanto...
  - Precisamos parar de tentar evitar mudanças
    - Mudanças são um aspecto intrínseco da vida do software
- Precisamos de uma metodologia de desenvolvimento que nos permita alterar **constantemente** o código sem comprometer sua qualidade

Definições

- Abandonar velhos hábitos técnicos e sociais
- Dedicar esforço máximo no trabalho do dia
- Buscar melhoria constante
- Avaliar seu desempenho em relação à sua contribuição ao grupo
- Atender algumas necessidades humanas no desenvolvimento de software
- Viva a mudança!!!
- Desenvolvimento de software é …
  - um aprendizado
  - como dirigir um carro

Definições

- Antiga:

  "XP é uma metodologia leve para times médios ou pequenos desenvolvendo software em face a requisitos vagos e que mudam rapidamente"

- Nova:
  - XP é leve
  - XP é focado no desenvolvimento de software
  - XP funciona em times de qualquer tamanho
  - XP se adapta à requisitos vagos e que mudam rapidamente
- Proposta
  - Queremos Poder Alterar Software
  - Cliente satisfeito significa "software eficiente"

Definições

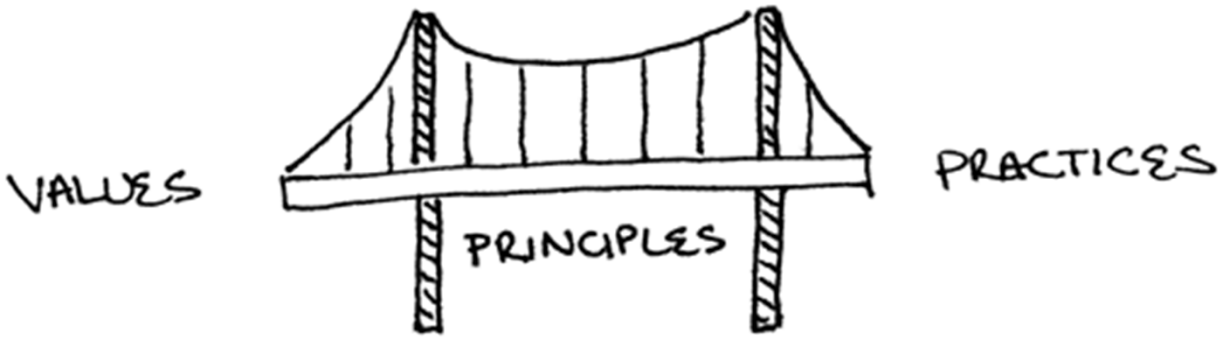
- Codificação é a atividade central do projeto
- Testes (que também é código) servem de especificação
- Comunicação entre desenvolvedores se baseia em código

O Mantra

- **Codifique**, senão o software não sai
- **Teste**, senão você não sabe se está funcionando
- **Refatore**, senão o código vai ficar tão ruim que será impossível dar manutenção (melhorar código sem alterar funcionalidade)
- **Escute**, para que saiba qual é o problema a resolver
- **Planeje**, para que você sempre faça a coisa mais importante ainda a fazer

Definições

- Valores dão razão às práticas
- Práticas evidenciam valores
- Princípios:
  - Preenchem a distância entre valores e práticas
  - Técnicas intelectuais para traduzir valores em práticas



Valores

Versão 1999	Versão 2004
• Comunicação • Simplicidade • Feedback • Coragem	+ Respeito

- Os valores se complementam

Princípios

- **Humanidade**
  - Balancear as necessidades pessoais com as necessidades do time
- **Economia**
  - Evite o risco do “Sucesso Técnico”. Tenha certeza que o sistema cria valor para o negócio
- **Benefício Mútuo**
  - Todas as atividades devem trazer benefício a todos os envolvidos
- **Auto-Semelhança**
  - Tente aplicar a estrutura de uma solução em outros contextos, até em diferentes escalas

Princípios

- **Melhoria**
  - Valorize atividades que começam agora e se refinam ao longo do tempo
- **Diversidade**
  - Times devem ser formados por uma variedade de habilidades, atitudes e perspectivas
- **Reflexão**
  - Reflexão vem após a ação. O aprendizado é o resultado da reflexão sobre a ação
- **Fluxo**
  - Entregue um fluxo contínuo de software que agregue valor

Princípios

- **Oportunidade**
  - Enxergue os problemas como uma oportunidade para mudança
- **Redundância**
  - Resolva os problemas difíceis de várias formas diferentes
- **Qualidade**
  - Sacrificar a qualidade nunca é um meio efetivo de controle

Princípios

- **Passos Pequenos**
  - A execução em passos pequenos diminui o risco de uma grande mudança
- **Aceitação da Responsabilidade**
  - Responsabilidade não pode ser imposta, deve ser aceita

Práticas

- Divididas em:
  - Práticas Primárias
    - Podem ser aplicadas separadamente e trarão melhoria imediata
  - Práticas Corolário
    - Mais difíceis de implementar. Exigem domínio e experiência com as práticas primárias
- Abordagem mais amigável:

  1999 – Utilize todas as 12 práticas

  2004 – Você não pode impor as práticas aos outros. Comece mudando por você

Práticas Primárias

- **Sentar Junto**
  - Desenvolva num ambiente grande o suficiente para o time todo ficar junto
- **Time Completo**
  - Monte um time multi-disciplinar, com todas as habilidades necessárias para o sucesso do projeto
- **Área de Trabalho Informativa**
  - Um observador deve ser capaz de ter uma idéia do andamento do projeto andando pela área de trabalho
- **Trabalho Energizado**
  - Trabalhe apenas enquanto puder ser produtivo e o número de horas que puder aguentar

Práticas Primárias

- **Programação Pareada**
- **Histórias**
  - Unidades de funcionalidade visíveis ao cliente
  - Cartões na parede têm mais valor
  - Estimativa é feita o mais cedo possível
- **Ciclo Semanal**
  - Representa uma iteração
  - Planeje o trabalho uma semana de cada vez
  - Reunião no início da semana para discutir o progresso, escolher histórias e dividir em tarefas

Práticas Primárias

- **Ciclo Quadrimestral**
  - Identificação de gargalos
  - Iniciar reparos
  - Planejamento do tema do quadrimestre
  - Foco no todo: onde o projeto se encaixa na organização
  - Temas x Histórias
- **Folga**
  - Inclua no plano algumas tarefas menores que podem ser removidas caso ocorra um atraso
  - Comprometimento x Estimativa
- **Build em 10 minutos**
  - Faça o build AUTOMÁTICO do sistema INTEIRO e rode TODOS os testes em 10 minutos

Práticas Primárias

- **Integração Contínua**
  - Integre e teste mudanças num intervalo de, no máximo, algumas horas
  - Síncrona ou Assíncrona
- **Desenvolvimento Orientado por Testes**
- **Design Incremental**
  - Invista no design do sistema todos os dias
  - O conselho não é minimizar o investimento em design no curto prazo, mas manter o investimento proporcional às necessidades do sistema
  - Quando? Como? Onde?

Práticas Corolário

- **Envolvimento Real com o Cliente**
  - Faça com que as pessoas cujas vidas e negócios serão afetados pelo sistema façam parte do time
- **Implantação Incremental**
  - Ao substituir um sistema legado, troque partes da funcionalidade gradualmente
- **Continuidade do Time**
  - Mantenha times eficientes trabalhando juntos
- **Diminuição do Time**
  - Mantenha a carga de trabalho constante e distribua as tarefas de modo a deixar alguém ocioso
  - Com o tempo, essa pessoa pode ser liberada para formar novos times

Práticas Corolário

- **Análise de Causa Inicial**
  - Sempre que encontrar um defeito, elimine o defeito e sua causa
  - “Os 5 Porquês” – **5W2H**
- **Código Compartilhado**
  - Qualquer um do time pode melhorar qualquer parte do sistema a qualquer momento
- **Código e Testes**
  - Os únicos artefatos a serem mantidos
- **Repositório de Código Único**
  - Desenvolva num repositório único. Branches podem existir, mas devem ser incorporados logo

Práticas Corolário

- **Implantação Diária**
  - Coloque software novo em produção toda noite
- **Contrato de Escopo Negociável**
  - Contratos devem fixar tempo, custo e qualidade, mas a negociação de escopo deve ser aberta
- **Pague-Pelo-Uso**
  - Pague por cada vez que o sistema é usado
  - O dinheiro é o feedback final

Práticas

Versão 1999	Versão 2004
Jogo do Planejamento	Histórias, Ciclo Semanal, Ciclo Quadrimestral, Folga
Versões Pequenas	Implantação Incremental, Implantação Diária
Metáfora	(Design Incremental)
Design Simples	Design Incremental
Refatoração	Design Incremental
Propriedade Coletiva do Código	Código Compartilhado, Repositório de Código Único
Ritmo Sustentável	Trabalho Energizado, Folga
Cliente com os Desenvolvedores	Time Completo e Envolvimento Real com o Cliente
Padrão de Código	(Código Compartilhado)

PLANEJAMENTO

Planejamento

- **Estratégia de XP**
  - Listar os items de trabalho
  - Estimar
  - Fixar o orçamento do projeto
  - Negociar o que deve ser entregue dentro daquele orçamento
- **Estimativas e orçamento não podem mudar**
- **Planejamento deve envolver todo o time**
- **Pode ser aplicado em diversas escalas:**
  - Par decidindo o que fazer nas próximas horas
  - Planejamento da semana ou do quadrimestre

Planejamento

- XP separa claramente o papel do cliente e da equipe
  - Cliente: tome as decisões do negócio
  - Equipe de desenvolvimento: tome as decisões técnicas
- Originou a declaração de direitos do cliente e do desenvolvedor

Direitos do cliente

- Ter um plano geral, para saber o que pode ser conseguido e a que custo
- Receber o melhor resultado possível a cada semana
- Ver o progresso no sistema, provando que o resultado do trabalho passa sucessivamente pelos testes especificados pelo cliente
- Mudar seu modo de pensar, substituir funcionalidades, e mudas as prioridades sem ter que pagar custos exorbitantes
- Ser informado das mudanças de planos a tempo de escolher como reduzir o escopo para garantir a data originalmente prevista de entrega do produto
- Cancelar o projeto, quando quiser, e ficar com um sistema funcionando, refletindo seu investimento até o momento.

Direitos do desenvolvedor

- Saber o que é necessário, com declarações explícitas de prioridades
- Produzir trabalhos que satisfaça tanto ao cliente quanto ao seu desenvolvimento profissional
- Pedir e receber ajuda de companheiros, gerentes e clientes
- Fazer e atualizar as estimativas
- Aceitar suas responsabilidades sem necessitar que elas lhe sejam impostas

Escrevendo estórias

- As funcionalidades do sistema são levantadas através de ESTÓRIAS que são registradas em pequenos cartões
- As estórias DEVEM ser escritas pelos clientes
- Porque:
  - Ao escrever a estória o cliente é forçado a pensar na melhor funcionalidade, pois ele formaliza o pensamento e analisa melhor o assunto sobre o qual irá tratar
  - O cliente tem uma responsabilidade sobre aquilo que está sendo solicitado
  - O cartão é fundamental para que o cliente compreenda que tudo tem um custo
  - É uma forma de documentar

Tarefas

- Caso a implementação de uma estória gaste uma ou mais semanas, a equipe divide um cartão em tarefas com duração máxima de alguns dias

Exemplos de estórias

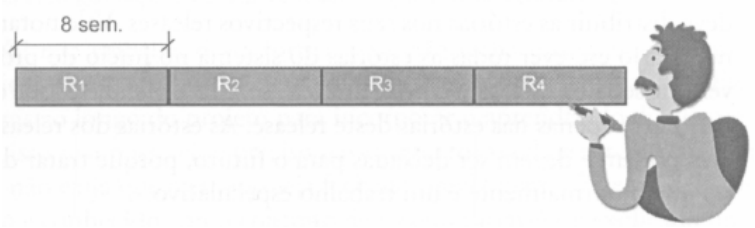
- A tela de login deve permitir que o usuário pule o login. Neste caso, o usuário entrará no sistema como guest
- O usuário deve poder alterar seu perfil (email, senha, primeiro nome, último nome, filiação). Deve ter dois campos de senha para confirmação
- Para cada conta, computar o saldo fazendo a adição de todos os depósitos e a subtração de todas as deduções

Estimando estórias

- Em XP, o tempo para desenvolver uma estória é medida em *Pontos*
- Um ponto representa um dia ideal de trabalho do desenvolvedor
- Um dia ideal é aquele onde apenas se implementa e não realiza outras atividades (atender telefone, reuniões, atender clientes, etc)
- No caso de projetos grandes, um ponto pode ser considerado uma semana ideal

Planejando os releases

- XP o software é entregue de forma incremental
- Cada entrega = release
- Projetos são divididos em releases que não devem passar de dois meses
- Cada release implementa funcionalidades que possui valor bem definido para o cliente
- Exemplo, projeto de oito meses dividido em quatro releases
  - R1: consulta dos produtos em estoc
  - R2: processamento de compras on-
  - R3: acompanhamento de pedidos
  - R4: campanha de marketing de relacionamento

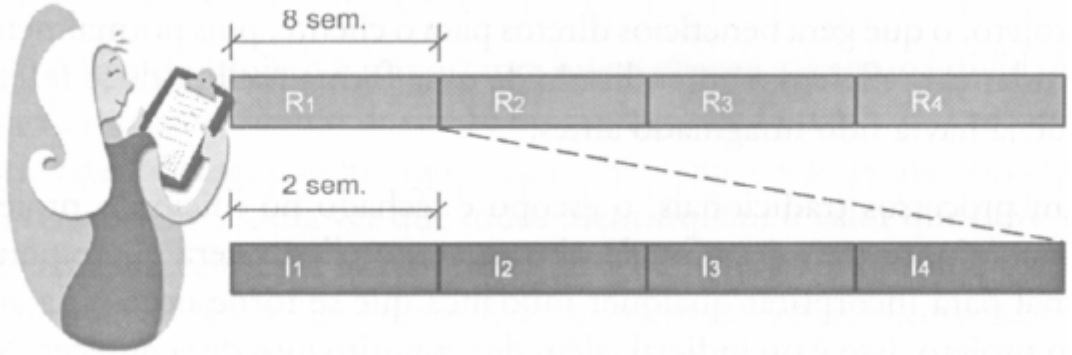


Priorizando estórias dos releases

- Após definir os objetivos de cada release, o cliente deve distribuir as estórias nos releases
- No início de cada release, a equipe deve informar quantos *pontos* serão necessários para implementar

Planejando as iterações

- Cada release é dividido em iterações
- Uma iteração é um pequeno espaço de tempo dedicado para a implementação de um conjunto de estórias
- Uma iteração pode variar de 01 a 03 semanas
- No início de cada iteração acontece a reunião de planejamento onde clientes e desenvolvedores definem as estórias que serão implementadas na iteração que se inicia



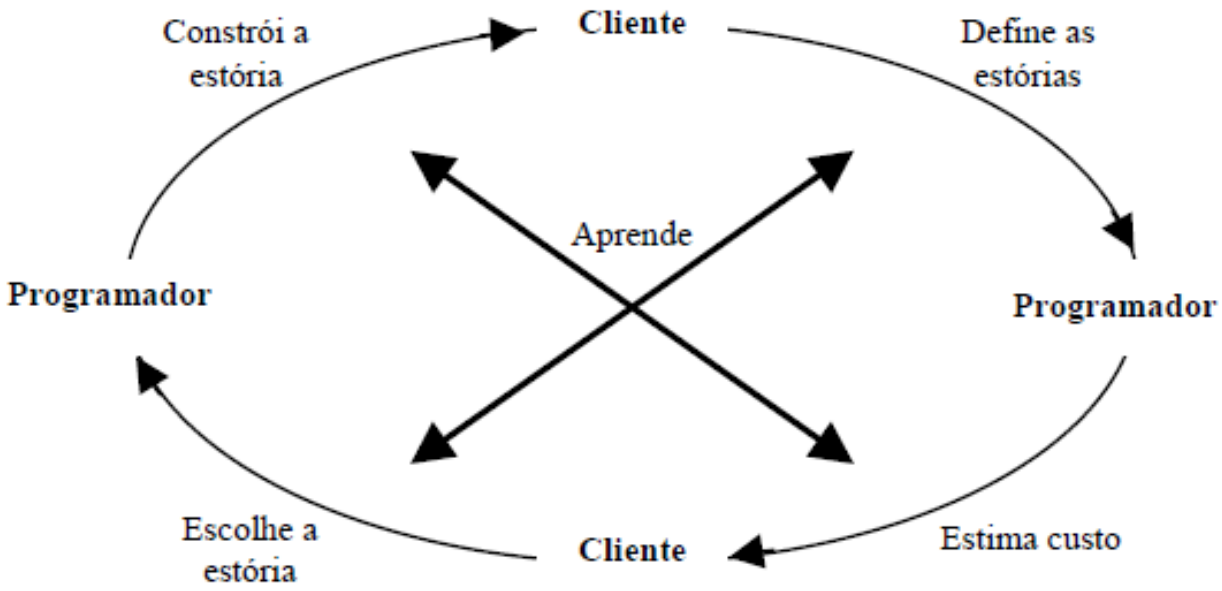
Exemplo

- Descobrir o número de pontos ideais que podem ser implementados em uma iteração
- Tamanho da iteração = 2 semanas = 10 dias úteis
- Deve-se descontar:
  - 1 dia útil para a reunião de planejamento da iteração (com a prática este tempo é reduzido)
  - 1 dia útil para a reunião de encerramento da iteração (com a prática este tempo é reduzido)
- Dias úteis disponíveis para desenvolvimento = 10 -2 = 8
- Número de desenvolvedores = 6 = 3 x 2 = 3 pares
- 1 par/dia = 1 ponto
- 1 par em 8 dias = 8 pontos
- 3 pares em 8 dias = 3 x 8 = 24 pontos

Encerramento de iterações e releases

- Encerrando uma iteração
  - O cliente executa todos os testes de aceitação que foram escritos para as estórias da iteração
- Encerrando um release
  - A equipe coloca o sistema em produção para que todos os usuários possam acessá-lo

Resumo do Planejamento



Processo de aprendizagem no desenvolvimento

STAND UP MEETING

Stand up meeting

- Um dia de trabalho do XP sempre começa com um stand up meeting (encontro em pé)
- Recomenda-se que essa reunião seja rápida e não passe de 10 minutos
- A idéia de manter as pessoas em pé serve para incentivá-las a fazer a reunião rapidamente
- Serve para que todos comentem o que fizeram no dia anterior. Manter o grupo informado.
- Serve para decidir o que será feito no dia que se inicia (priorizar atividades dos membros e decidir quais cartões de estórias serão desenvolvidos e quem cuidará de qual)

PROGRAMAÇÃO EM PAR

Programação em par

- Dois programadores trabalham em um mesmo problema, ao mesmo tempo e em um mesmo computador
- Um desenvolve e outro trabalha como estrategista
- Objetivos
  - Revisão: enquanto um digita, ou outro revisa o código e tenta evitar eventuais erros despercebidos
  - Criatividade: duas pessoas modelando soluções, usarão habilidades e experiências diferentes
  - Idéias mais eficazes e simples: se uma idéia é complicada, o outro pode sugerir simplificações

Os efeitos sobre a produtividade da equipe

- Por que colocar duas pessoas para fazer o trabalho que apenas uma poderia fazer sozinha?
- Não estamos desperdiçando uma pessoa?
- E como desperdiçar uma pessoa com os prazos apertados que são freqüentes nos projetos de software?
- No curto prazo, a produtividade não é afetada. Nem se pode dizer que ela será menor nem maior.
- A longo prazo, o código gerado por um par é quase livre de defeitos.
- Além disso como as soluções geradas pelos pares são mais simples e eficazes, isso melhora a manutenibilidade.

A pressão do par

- O ato de programar demanda grande concentração e produz um gasto de energia considerável.
- Entretanto no dia-a-dia, o programador se depara com fontes de distração: email, mensagens instantâneas, sites na internet, bate-papo com colegas, cansaço, desânimo etc.
- Diversas vezes, quando um problema exige uma solução mais elaborada, o programador acaba adotando uma solução insuficiente, porém mais rápida.
- Pressão do par: quando o programador está acompanhado de outra pessoa ele deixa de ter um compromisso apenas consigo mesmo e passa a ter compromisso com o outro.

Revezamento

- É fundamental que os programadores revezem o trabalho de programação diversas vezes ao longo de um dia de trabalho.
- Não é necessário estipular regras indicando o momento em que a troca deve ser efetuada.
- A troca de par também é importante para a disseminação do conhecimento.

Desafios da programação em par

- Organização do escritório
- A visão gerencial
- Relacionamento humano
- Competição

PADRÕES DE CODIFICAÇÃO

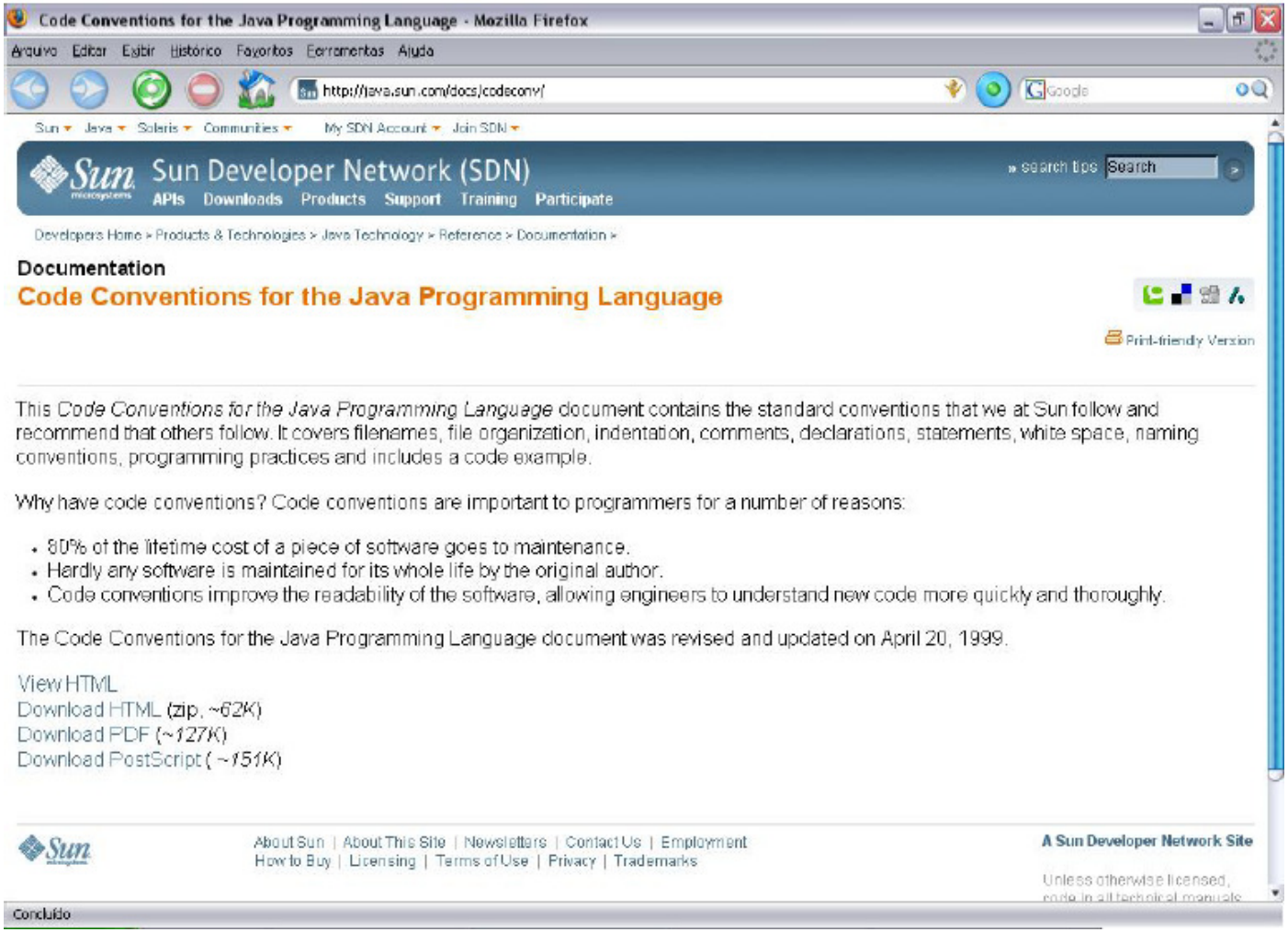
Padrões de codificação

- Auxiliam a equipe a se comunicar de forma clara através do código do sistema
- Se analisarmos o código de diferentes programadores, iremos notar diferenças de estilo
- Exemplo

```
public int soma (int a, int b) {
  return a + b;
}
```

```
public int soma (int a, int b)
{
  return a + b;
}
```

Padrão de codificação da SUN



http://java.sun.com/docs/codeconv/

Características do Padrão

- Indentação: procure usar um estilo de indentação consistente
  - Mesma largura na tabulação
  - Mesmo posicionamento de chaves
- Letras maiúsculas e minúsculas: seja consistente com o tamanho da letra.
- Comentários: procure evitá-los tanto quanto possível.
  - Manter o foco em um código que seja claro, simples e que transmita a sua intenção.
  - Escreva um comentário para explicar uma otimização ou algo que não pode ser simplificado.
  - O comentário deve evoluir juntamente com o código.
- Nomes: usar nomes que comuniquem e não nomes que sejam convenientes para digitar. Normalmente, nomes longos são melhores.

DESIGN SIMPLES

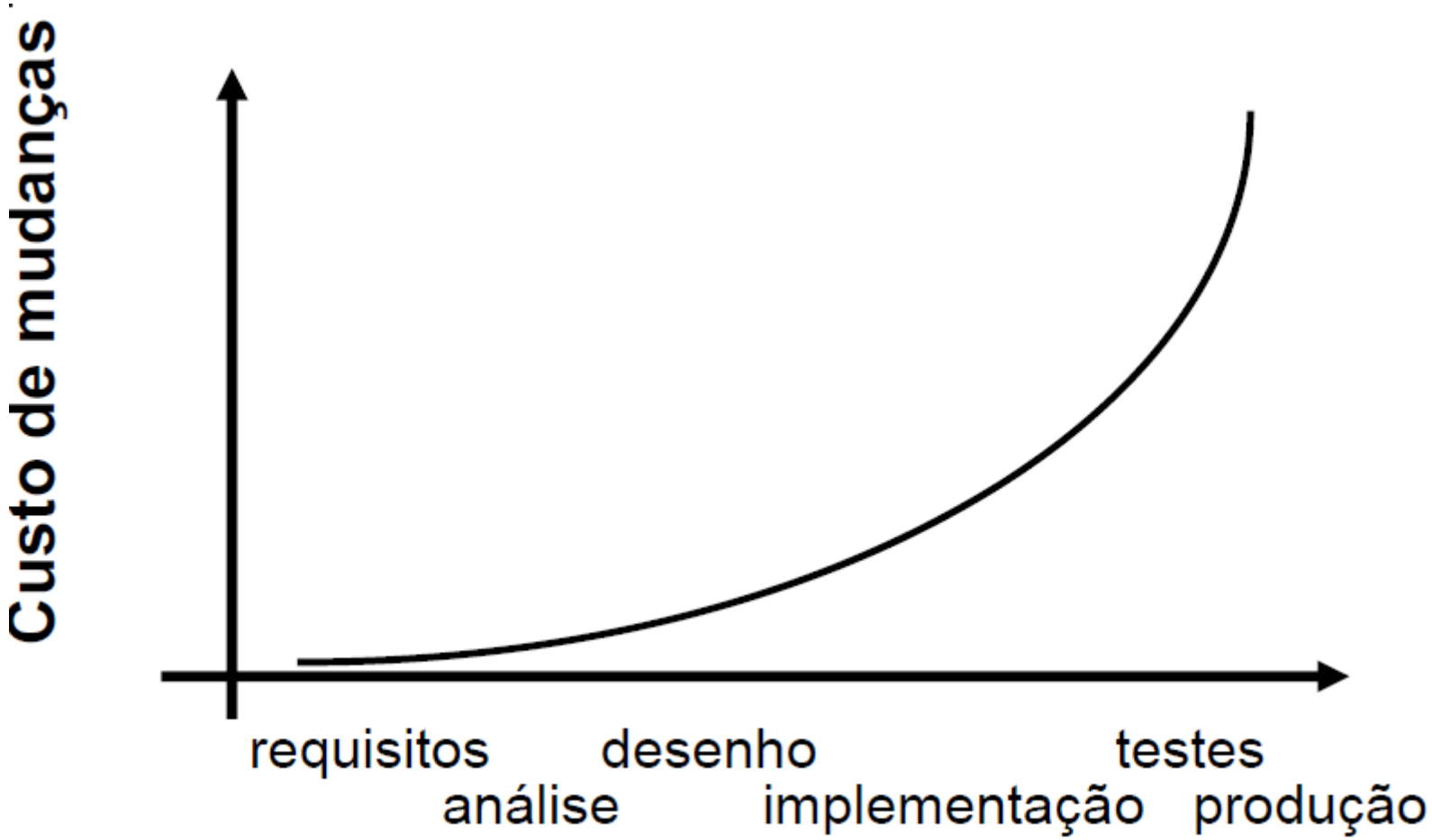
Design Simples

- XP determina que deve assumir sempre o design mais simples e funcional possível que seja capaz de atender as estórias e de passar nos testes

Design tradicional

- Uma das premissas universais da engenharia de software: o custo de fazer uma alteração em um programa cresce exponencialmente ao longo do tempo.
- Na tentativa de prever tudo os programadores criam "soluções genéricas", de modo que o sistema não sofra grandes impactos caso uma mudança aconteça.
- Isso pode afetar a manutenibilidade do código.
- Infelizmente, a única constante em um projeto é a mudança:
  - Requisitos mudam
  - O design muda.
  - A tecnologia muda.
  - A equipe muda.
  - Os membros da equipe mudam.

Custo de alteração no desenvolvimento tradicional



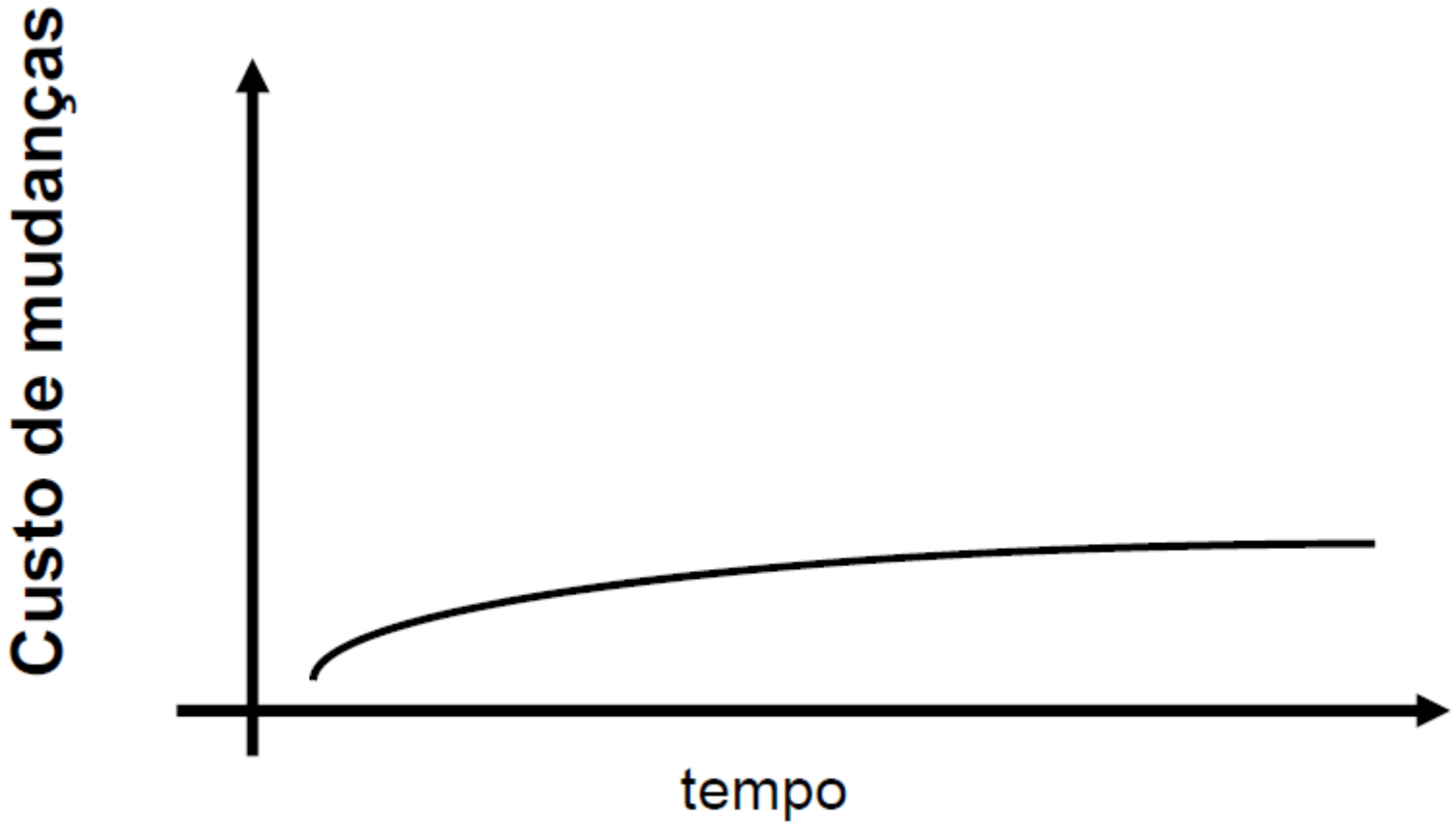
Custos de uma alteração no XP

- A comunidade de desenvolvimento investiu uma enorme quantidade de recursos no sentido de reduzir os custos de uma alteração em um software.
- Este investimento gerou diversos avanços:
  - Melhores linguagens de programação
  - Avanços na tecnologia de banco de dados
  - Melhores práticas de programação
  - Novos ambientes e ferramentas de desenvolvimento
  - Novas notações
  - Orientação a objetos

Design simples

- Uma das premissas técnica do XP: o custo de uma alteração cresce mais lentamente e tende a se estabilizar ao longo do tempo de modo a tornar-se praticamente linear.
- Dessa forma, não é necessário tentar prever todas as possíveis mudanças.
- Deve-se implementar a solução mais simples possível , se preocupando apenas com os problemas de hoje, deixando os problemas do futuro para o futuro.

Custos de alteração no XP



Design Simples e Valores do XP

- A prática de trabalhar com um design simples respeita os valores do XP:
  - Comunicação: um design simples comunica a sua intenção de forma mas eficaz que um design complicado, pois é mais fácil compreendê-lo.
  - Simplicidade: um design simples torna o software mais leve e mais fácil de ser alterado a qualquer momento, pois é mais fácil de ser compreendido.
  - Feedback: criando um design simples, a equipe será capaz de avançar de forma mais ágil e obter feedback do cliente mais rapidamente.
  - Coragem: o design é trabalhado para resolver problemas de hoje. É necessário coragem para assumir que a equipe será capaz de lidar com problemas no futuro.

Design

- **“Anti-Metáfora” comum: Construção Civil**
  - No mundo físico não podemos mudar o projeto
  - Disparidade: Extremamente difícil voltar atrás
  - Assimetria de Custos
- **Quando fazer design?**
- **Estratégia de XP**
  - “Faça design sempre”
  - “*Once and Only Once*”: Evite código duplicado
  - Decisões são transferidas para o momento em que a mudança seja baseada na experiência

Design

- **Critérios para avaliar a simplicidade do design**
  - Apropriado para o público alvo
  - Comunicativo
  - Fatorado
  - Mínimo

RITMO SUSTENTÁVEL

Trabalho em excesso

Segundo Frederick Brooks (1995) o problema mais recorrente nos projetos de software é a falta de tempo para finalizar o projeto.

As restrições de tempo sobre o projeto acabam levando muitas equipes a trabalhar em excesso.

Essa solução pode, ao invés de resolver o problema, piorá-lo.

As conseqüências vão, desde uma maior lentidão na resolução de problemas, até a introdução de erros no software.

Ritmo sustentável é mais produtivo

O XP recomenda que as pessoas trabalhem oito horas por dia (40h semanais).

Isso demonstra respeito pela individualidade e a integridade de cada membro da equipe.

Aumenta a agilidade do projeto.

No modelo industrial, o aumento do trabalho e da operação das máquinas normalmente aumenta a produção.

No desenvolvimento de software este princípio não pode ser aplicado, uma vez que se trata de um trabalho que exige muita concentração e criatividade.

INTEGRAÇÃO CONTÍNUA

Integração

Um software desenvolvido em equipe é dividido em partes.

Cada programador fica responsável por uma parte, e pelos seus testes.

Para que o sistema funcione é necessário integrar as várias partes desenvolvidas.

Para isso é necessário convencionar as interfaces.

Em teoria essa estratégia funciona bem, na prática podem ocorrer os seguintes problemas:

Causas dos problemas de integração

As interfaces convencionadas não são respeitadas. Por exemplo, assinatura de métodos não conferem ou objetos não se comportam como deveriam.

Não houve compreensão por parte de um desenvolvedor da interface do outro.

O sistema como um todo é pouco coeso.

Quanto mais tempo a equipe demora para integrar, maior é o risco de problemas na integração (por exemplo as interfaces podem ser alteradas e não serem publicadas corretamente para os desenvolvedores.

Integração contínua

O XP propõe que a equipe integre e teste o sistema inteiro diversas vezes ao dia.

Uma das dificuldades para efetuar essa prática é a compilação e ligação do sistema, que deve ser mantido baixo ao longo de todo o projeto. Existem diversas formas de fazer isso:

- Fazer com que a interdependência entre os arquivos seja baixa.
- Fazer com que o compilador só compile os arquivos que tiverem sido editados e utilize diretamente o código objeto para os demais.
- Trabalhar com a carga dinâmica de bibliotecas quando isso for possível.

Fases por onde o código passa

1. Local – Primeira fase do desenvolvimento. O par acabou de começar a trabalhar no código e as mudanças que executaram ainda não estão disponíveis para os outros desenvolvedores.
2. Candidato à liberação – O par terminou a sua tarefa e está pronto para disponibilizar o novo código contendo suas alterações para os demais desenvolvedores.
3. Disponibilizado no repositório – Esta é a versão oficial corrente do código. Os teste de unidade executam com 100% de sucesso. As alterações do par se tornam disponíveis para todos os desenvolvedores.

Liberação de mudanças

Enquanto um par programa, outros também estão fazendo alterações e sempre há uma chance de que mais de um par tenha editado e alterado o mesmo código.

Por isso é importante que exista um processo disciplinado para que se possa fazer a liberação de mudanças de forma rápida e com o mínimo de risco.

Passos para Liberar Mudanças

1. Sempre comece a trabalhar com o código mais atual que se encontra no repositório.
2. Escreva os testes de unidade que são necessários para a sua tarefa. Os testes de unidade devem ser escritos antes do código.
3. Rode todos os testes de unidade.
4. Conserte qualquer teste que esteja quebrado.
5. Quando todos os testes de unidade estiverem rodando com 100% de sucesso as alterações locais se tornam candidatas a liberação, podendo ser colocadas no repositório.

Passos para Liberar Mudanças

6. O código que é candidato a liberação é integrado com o código atual do repositório.
7. Se o código do repositório tiver sido alterado enquanto você fazia a mudança, compare as diferenças entre as suas mudanças e o código do repositório. Identifique as mudanças que tiverem sido adicionadas pelo outro par e integre estas mudanças com as suas.
8. Uma vez que você tenha integrado as suas mudanças , execute os testes de unidade na máquina de integração.
9. Quando os testes de unidade estiverem executando com 100% de sucesso, coloque todo o código da máquina de integração no repositório.

Máquina separada para a integração

Integrar em uma máquina separada isola o código no qual você está trabalhando daquele que está no repositório.

Ritual de passagem: o código deixa de pertencer ao par que o criou e passa a ser coletivo, estando disponível para todos os desenvolvedores.

- Apenas um par pode integrar de cada vez. Ter uma máquina separada ajuda a disciplinar a equipe, pois deixa muito evidente quando existe um par integrando.

Ferramentas

Ferramenta de build: utilizada para compilar todo o sistema e executar outras atividades que se façam necessárias para que ele possa ser executado.

Sistema de controle de versão (repositório): armazena todos os códigos fontes do sistema mantendo a versão deles.

Atividades suportadas pelo repositório

Identificar mudanças locais.

Mostrar a diferença entre alterações locais e o código que está no repositório.

Identificar quem colocou uma determinada mudança no repositório e quando isso aconteceu.

Fundir mudanças locais com o código que está no repositório.

Voltar a versão de um código que já existia no repositório antes da integração.

RELEASES CURTOS

Releases

Cada release representa um conjunto de funcionalidades implementadas que representam um valor bem definido para o cliente e que é colocado em produção para que todos os usuários possam ser beneficiar dele.

Retorno do investimento

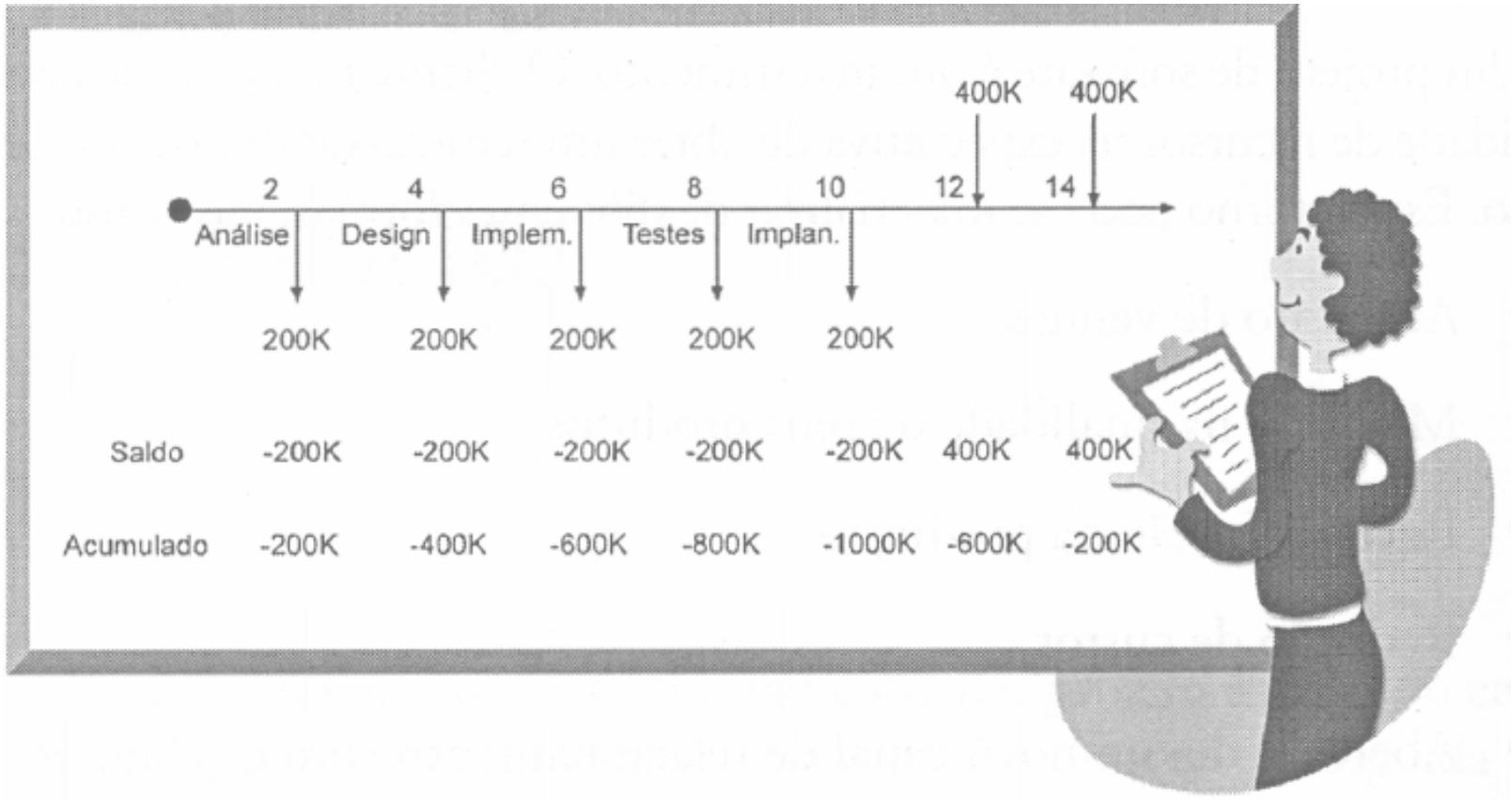
Um projeto de software é um investimento.

O cliente investe uma certa quantidade de recursos na expectativa de obter um retorno dentro de um certo prazo.

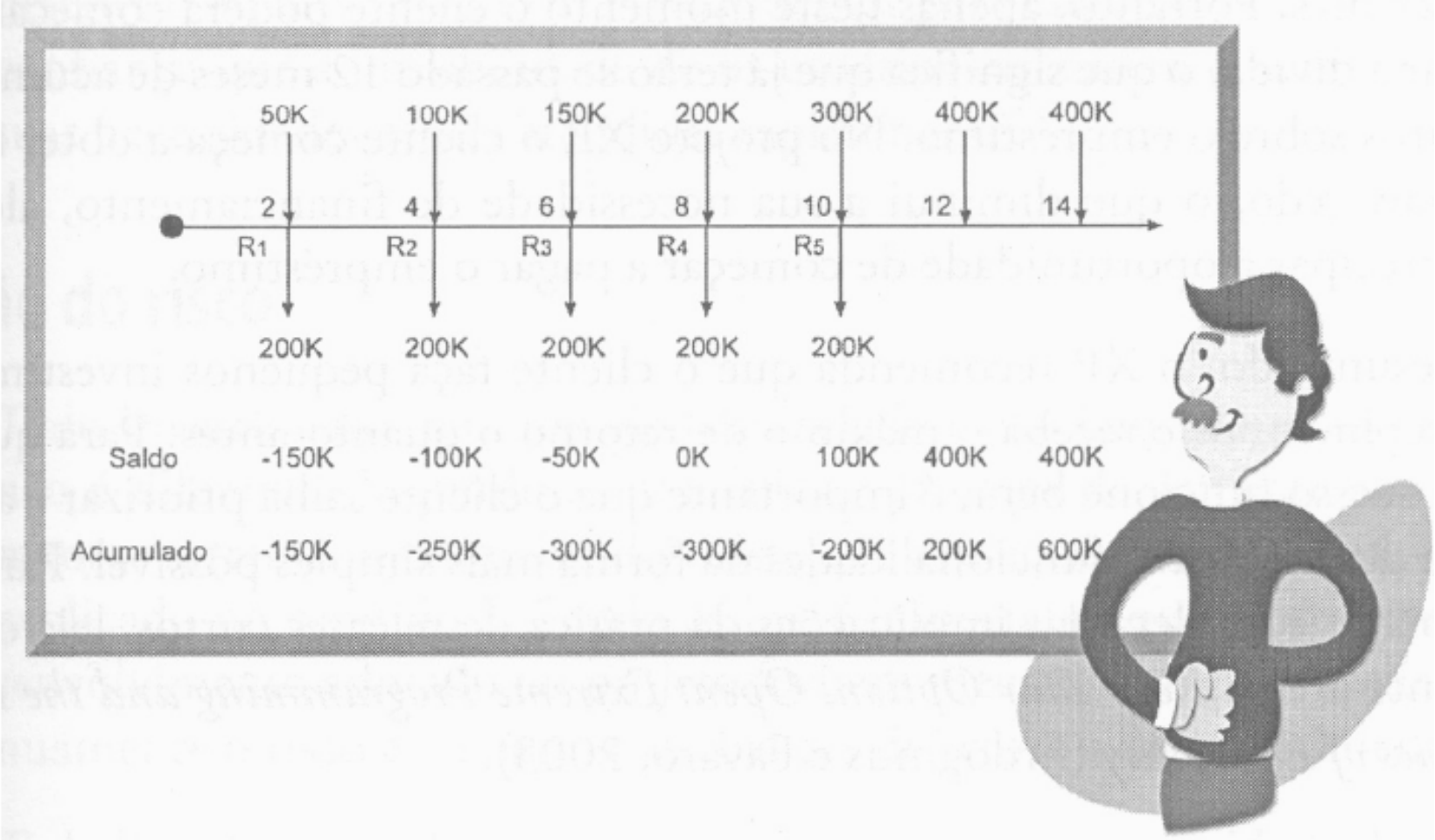
Este retorno pode se materializar de diferentes formas, tais como:

- Aumento de vendas
- Melhoria na qualidade de seus produtos
- Racionalização da produção
- Redução de custos
- Abertura de um novo canal de relacionamento com os clientes

Fluxo de caixa projeto tradicional



Fluxo de caixa projeto XP



ORGANIZAÇÃO DO AMBIENTE DE TRABALHO

Ambiente de trabalho

A forma como a equipe se organiza influencia diretamente a sua capacidade de adotar as práticas do XP.

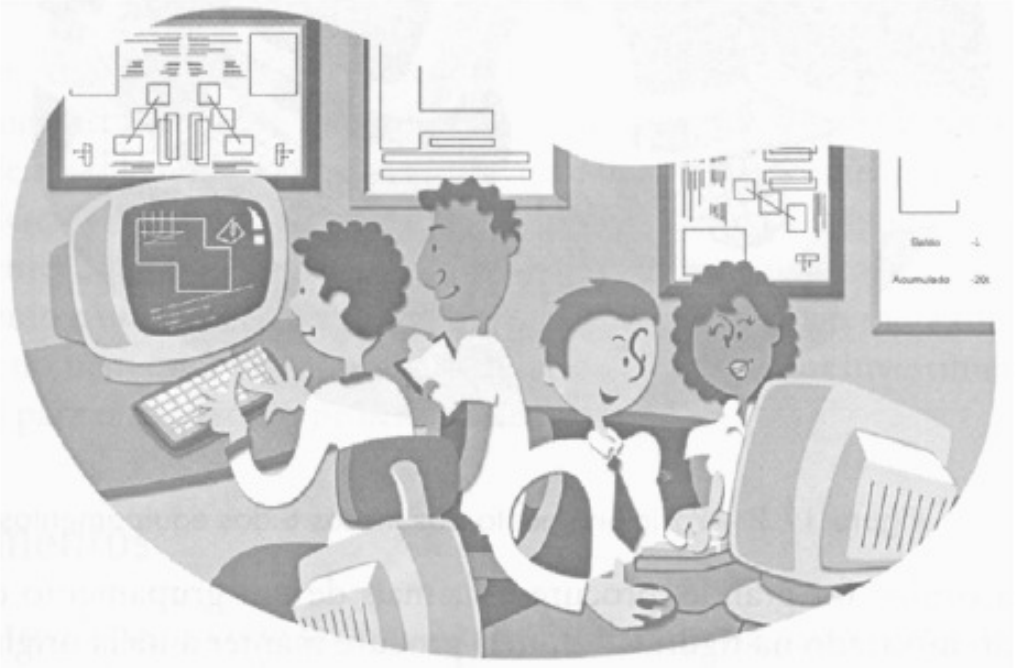
O XP recomenda que o ambiente de trabalho seja aberto e público de modo que:

- Todos os membros da equipe possam trabalhar próximos uns dos outros;
- Cada pessoa possa escutar as eventuais perguntas de seus colegas;
- Seja possível escutar uma conversa "acidentalmente" em relação à qual um membro da equipe tenha uma contribuição vital.

Ergonomia

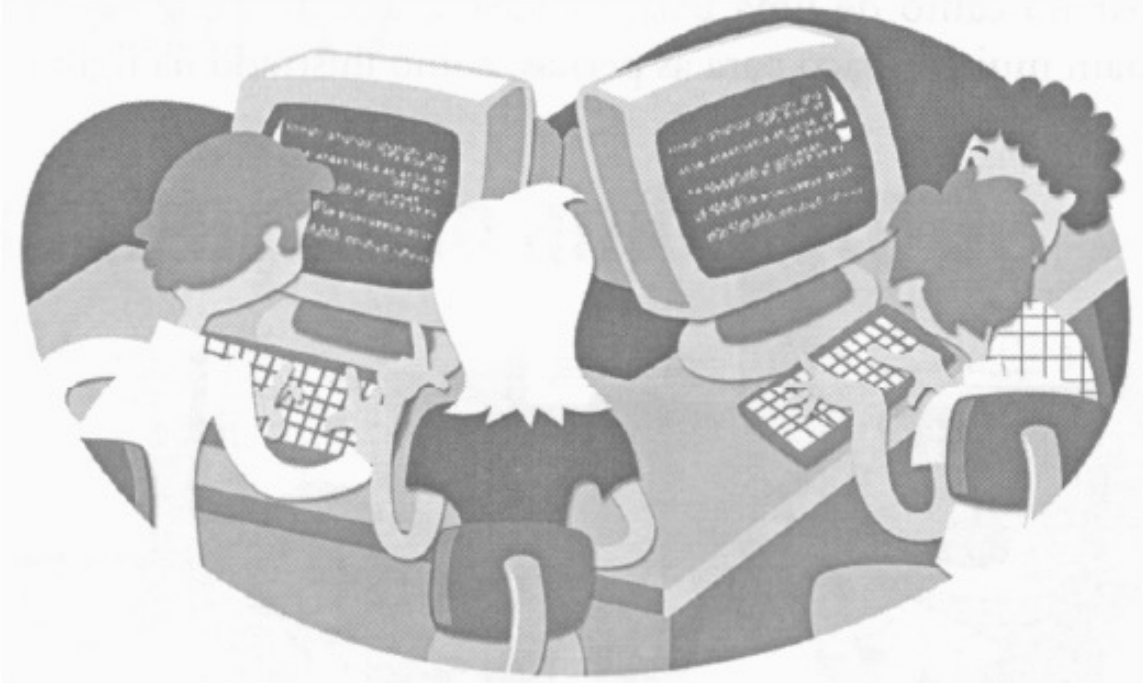
As mesas usadas pelos desenvolvedores devem permitir que duas pessoas trabalhem juntas.

As cadeiras devem ter braços ajustáveis e sejam ergonômicas.



Ergonomia

As mesas devem ser arranjadas de modo que ela consiga aproximar ao máximo os desenvolvedores. O ideal é arrumá-las de modo a formar uma área central ao redor da qual todos os desenvolvedores se posicionem.



Equipamentos

Os computadores devem ser velozes, com muita memória, discos rápidos e com grande capacidade de armazenamento.

A rede deve ser rápida.

Os monitores devem ser grandes e com alta qualidade de imagem.

Deve haver uma máquina separada para integração.

É recomendável que exista um servidor, acessível também pela Web, para armazenar documentações do projeto, manuais etc.

Mural

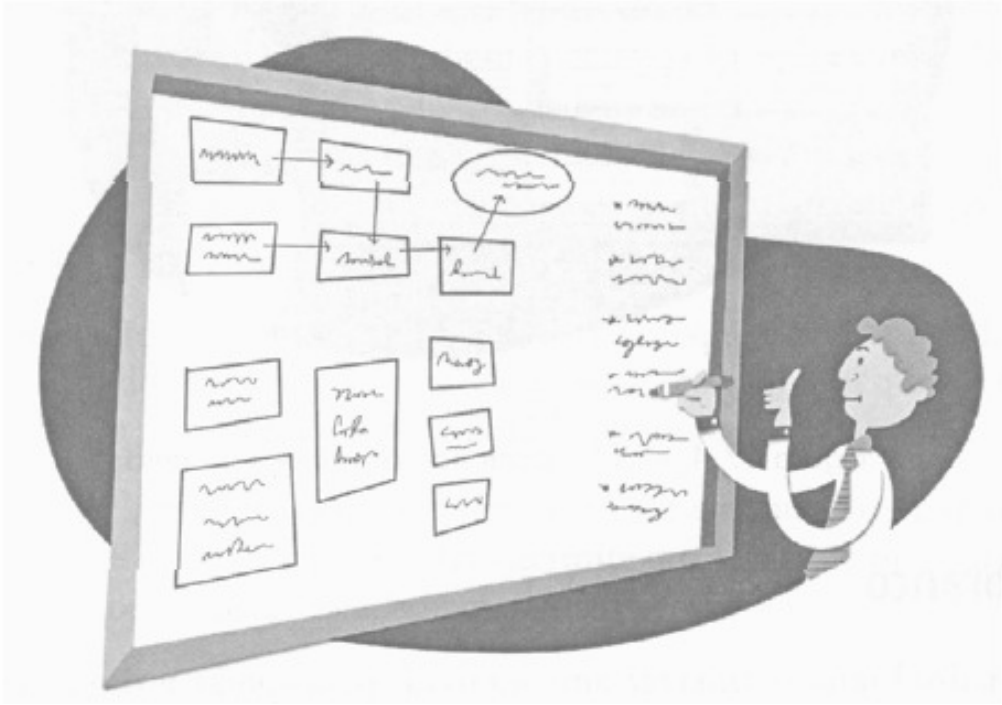
Serve para a colocação dos cartões da iteração.

Neste, deve-se criar três colunas com as seguintes informações: cartões não-iniciados, cartões em andamento, cartões finalizados.

Quadro branco

Usado para apoiar as discussões e debates da equipe.

Fornece um local no qual a equipe pode fazer desenhos e anotações e compartilhá-los facilmente com todos os membros.



Mandamentos

Representam regras que devem ser seguidas no projeto.
Costuma ser fixados na parede com letras grandes.

DOCUMENTAÇÃO

Importância da documentação

Um sistema que possua um certo porte costuma ser desenvolvido por um conjunto de pessoas durante meses ou anos de trabalho.

Portanto, são softwares complexos que dificilmente podem ser mantidos por uma única pessoa ao longo do tempo.

A documentação é importante para instruir o desenvolvedor, quando este for efetuar manutenção no código.

Limite da documentação

O XP considera que o código é o principal elemento de um projeto de software.

Um projeto de software não produz apenas código, mas também testes de unidade, testes de aceitação e outras formas de documentação.

Documentar, embora seja uma atividade necessária e importante, não é o objetivo principal do projeto.

A equipe precisa documentar apenas o suficiente para que os desenvolvedores, no futuro possam dar manutenção no código.

O esforço de documentação pode ser reduzido pela utilização de práticas como refactoring, código coletivo e programação em par, que levam a um código mais simples e mais limpo.

Quando documentar?

Tradicionalmente as equipes começam a implementar bem antes da documentação.

Durante a implementação, novas considerações surgem como fruto da codificação.

Isso leva a mudanças na forma de implementar a funcionalidade e, conseqüentemente, mudanças nos documentos que a descrevem.

Caso a documentação não seja atualizada, ela não estará de acordo com a implementação.

O XP documenta apenas o mínimo antes de se chegar ao código. A comunicação face-a-face é utilizada como fonte de informações para a codificação.

A documentação mais abrangente só ocorre quando o código já está pronto.

Documentos na XP

Estória
Testes de Aceitação
Testes de Unidade
Javadoc
Modelo de Classes
Modelo de Dados
Processos de Negócio
Manual do Usuário
Acompanhamento Diário
Acompanhamento do Projeto
Fotos

TESTES

Testes

- **Defeitos são caros**
- **Corrigir defeitos também é caro**
  1. Dupla Verificação
     - Teste X Implementação
  2. DCI (*Defect Cost Increase*)
     - Quanto mais cedo encontrar o defeito, mais barato é a correção
     - Testes devem ser feitos próximos à implementação
     - Antes ou depois? Tanto faz, contanto que o resultado seja verificado

Testes

- **Estratégia de XP:**
  - Testes em dois níveis
    - Perspectiva do programador (Teste Unitário)
    - Perspectiva do cliente (Teste de Sistema)
  - Testes geralmente são escritos antes da implementação
  - Testes automatizados
  - Testes Freqüentes

CONSIDERAÇÕES FINAIS

Escalabilidade

- **Valores e Princípios valem em qualquer escala**
- **Práticas podem ser adaptadas**
- **Número de pessoas não é a única medida de escalabilidade**
- **Medidas:**
  - Número de Pessoas
    - Transforme o problema em problemas menores
    - Aplique soluções simples
    - Aplique soluções complexas somente se sobrar algum problema

Escalabilidade

- Investimento
  - O problema é como contabilizar projetos XP
  - Fora do escopo de XP
- Tamanho da Organização
  - O objetivo não é esconder o trabalho do time nem forçar uma mudança no resto da empresa
  - Não mude o modo de comunicação com o resto da organização
- Tempo
  - Projetos longos funcionam bem com XP
  - Testes e Design Incremental são o “histórico”
- Complexidade do Problema
  - XP funciona melhor com a cooperação entre uma equipe de especialistas

Escalabilidade

- Complexidade da Solução
  - Desafio: parar de tornar o problema mais complicado
  - Estratégia de XP: elimine a complexidade gradualmente, sem parar de entregar
- Conseqüência das Falhas
  - XP não é suficiente para projetos onde a segurança e a confiabilidade são críticos
  - O princípio de fluxo sugere que processos de auditoria sejam feitos mais cedo e com freqüência
  - Rastreamento pode ser implementado em XP:
    - Código → Teste unitário → Teste de Sistema → História

Filosofia de XP

- **Taylorismo**
  - **Frederick Taylor: pioneiro da engenharia industrial**
  - **Princípios:**
    - As coisas funcionam de acordo com um plano
    - Micro-Otimização leva à Macro-Otimização
    - As pessoas são substituíveis e precisam ser instruídas sobre o que fazer
  - **Ecos no desenvolvimento de software:**
    - Separação de planejamento e execução
    - Criação de um departamento de qualidade separado

Filosofia de XP

- **Sistema de Produção da Toyota**
  - **Estrutura social alternativa de trabalho**
    - Todo operário é responsável pela linha de produção
    - Quando alguém vê algo errado, puxa uma corda e a linha de produção pára
    - Todos são responsáveis pela qualidade
  - **O maior desperdício é a super-produção**
  - Poppendieck M., Poppendieck T., "*Lean Software Development*", Addison-Wesley, 2003

Resumo

Valores	Princípios	Práticas Primárias
= Comunicação	Humanidade	Sentar Junto
= Simplicidade	Economia	Time Completo
= Feedback	Benefício Mútuo	Área de Trabalho Informativa
= Coragem	Auto-Semelhança	Trabalho Energizado
+ Respeito	Melhoria	Programação Pareada
	Diversidade	Histórias
	Reflexão	Ciclo Semanal
	Fluxo	Ciclo Quadrimestral
	Oportunidade	Folga
	Redundância	Build em 10 minutos
	Falha	Integração Contínua
	Qualidade	Desenvolvimento Orientado por Testes
	Passos Pequenos	Design Incremental
	Aceitação da Responsabilidade	

Resumo

Práticas Corolário	O Time de XP
Envolvimento Real com o Cliente	Testadores
Implantação Incremental	Projetistas de Interação (*Interaction Designer*)
Continuidade do Time	Arquitetos
Diminuição do Time	Gerentes de Projeto
Análise de Causa Inicial	Gerentes de Produto
Código Compartilhado	Executivos
Código e Testes	Escritores (*Technical Writers*)
Repositório de Código Único	Usuários
Implantação Diária	Programadores
Contrato de Escopo Negociável	Recursos Humanos
Pague-Pelo-Uso	

Conclusão

- **Novas Práticas**
  - Divididas em práticas primárias e práticas corolário
  - Facilitam uma “implantação incremental” de XP
- **Planejamento**
  - Baseado em negociação de escopo
  - Estimativas Reais
- **Testes**
  - Não importa se são feitos antes ou depois
  - Devem ser feitos: cedo, frequentemente e automaticamente

Conclusão

- **Design**
  - Uma atividade que deve ser feita constantemente
  - “Conquistar e Dividir”
- **Escalabilidade**
  - Práticas podem ser adaptadas
  - Valores e Princípios valem em qualquer escala
- **Principal mudança**
  - 1ª Edição focava muito mais em “Como” XP funciona
  - 2ª Edição foca muito mais no “Por quê”