Programação eXtrema (XP)

Cursos de Verão 2010 - IME/USP

Alfredo Goldman
Departamento de Ciência da Computação

www.agilcoop.org.br

AgilCoop

Agenda

- Primeira versão de XP
- Segunda versão de XP
- Perguntas durante a apresentação são bem vindas !

Programação eXtrema
XP

- Metodologia de desenvolvimento de software aperfeiçoada nos últimos 10 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 pequenos de 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
1. Propriedade Coletiva
2. Integração Contínua
3. Semana de 40 horas
4. Cliente junto aos desenvolvedores
5. 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

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 (*gambiarras*) 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 [>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. Refatoração
7. Programação Pareada
1. Propriedade Coletiva
2. Integração Contínua
3. Semana de 40 horas
4. Cliente junto aos desenvolvedores
5. 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

Por que mudou?

- 5 anos de retorno e experiência na comunidade
- Críticas ao 1º livro:
  - "Você quer nos forçar a programar da sua maneira!"
  - XP é muito radical, XP despreza o *design*
  - XP despreza a documentação
  - XP despreza conhecimentos especializados
  - XP não escala e não funciona para equipes dispersas geograficamente

Sobre a 2ª edição

- Reconhece a responsabilidade de cada indivíduo pelas suas próprias decisões
- Mantém o conteúdo, mas muda a forma, refraseando a mesma mensagem de maneira mais positiva e inclusiva
- 1ª Edição focava muito mais em “Como” XP funciona
- 2ª Edição foca muito mais no “Por quê” XP funciona
- Apresenta a filosofia por trás da criação de XP e paralelos com outras áreas

Definindo XP

- 1ª edição - 1999
  - *"XP é uma metodologia leve para times médios ou pequenos desenvolvendo software em face a requisitos vagos e que mudam rapidamente"*
- 2ª edição - 2004
  - *"XP é sobre mudança social"*

O que faz parte deste método?

- Uma filosofia de desenvolvimento de software baseada em alguns **valores**
- Um conjunto de **práticas** comprovadamente úteis que expressam esses valores, se complementam e se amplificam
- Um conjunto de **princípios** complementares que traduzem os valores em práticas
- Uma **comunidade** que compartilha tais valores e muitas das mesmas práticas

Nova definição

- 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

Adaptando tudo isso

- Outros valores da sua organização podem ser incorporados a XP
- Práticas podem ser adaptadas
- Princípios ajudam a entender as práticas e até improvisar práticas complementares
- Não existe XP puro, o importante é o comprometimento do time com os valores

Valores – o que mudou?

- Comunicação, simplicidade, *feedback*, coragem
- + Respeito
  - valor complementar aos 4 anteriores
  - respeito entre a equipe
  - respeito ao projeto

Princípios – o que mudou?

- 1ª edição tem princípios básicos e alguns menos importantes
- 2ª edição explica melhor o propósito dos princípios, que ganham mais importância
- Alguns foram mantidos, como *trabalho de qualidade*, *mudanças incrementais* e *aceitação de responsabilidade*
- Outros princípios eram implícitos e agora ganharam mais destaque

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

Princípios

- Auto-Semelhança
  - Tente aplicar a estrutura de uma solução em outros contextos e escalas
- 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

Princípios

- Reflexão
  - Reflexão vem após a ação. O aprendizado é o resultado da reflexão sobre a ação
- Fluxo
  - Desenvolva todas atividades em um fluxo contínuo, ao invés de fases discretas
- Oportunidade
  - Enxergue os problemas como uma oportunidade para mudança, aprendizado e melhoria

Princípios

- Redundância
  - Resolva os problemas difíceis de várias formas diferentes
- Falha
  - Quando você não sabe o que fazer, arriscar o fracasso pode ser o caminho mais curto para o sucesso
- 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 – o que mudou?

- Abordagem mais leve para adoção das práticas
  - as práticas não devem ser impostas, devem ser escolhidas
  - não precisam ser adotadas todas de uma vez
- Práticas primárias
  - independentes e seguras de adotar
  - proporcionam melhorias imediatas
- Práticas corolárias
  - difíceis de adotar sem domínio das primárias

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 interessado deve ser capaz de ter uma idéia do andamento do projeto andando pela área de trabalho

Práticas Primárias

- Trabalho Energizado
  - Trabalhe apenas enquanto puder ser produtivo e o número de horas que puder agüentar
- Programação Pareada
- Histórias
  - Unidades de funcionalidade visíveis ao cliente
  - Estimativa é feita o mais cedo possível para facilitar a interação entre o cliente e a equipe

Práticas Primárias

- 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 dividi-las em tarefas
- Ciclo de Estação
  - Identificar gargalos e iniciar reparos
  - Planejamento do tema da estação
  - Foco no todo: onde o projeto se encaixa na organização

Práticas Primárias

- Folga
  - Inclua no plano algumas tarefas menores que podem ser removidas caso ocorra um atraso
  - Não se comprometa além do que é possível ser realizado
- *Build* de 10 minutos
  - Faça o *build* AUTOMÁTICO do sistema INTEIRO e rode TODOS os testes em até 10 minutos
- Desenvolvimento Orientado por Testes

Práticas Primárias

- Integração Contínua
  - Integração pode ser síncrona ou assíncrona (no *build*)
- *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
  - *Design* feito mais próximo de quando é usado é mais eficiente

Como começar?

- Práticas dependem da situação e do contexto, são um guia para onde você pode chegar
- Adote XP em *passos pequenos* (mude uma coisa de cada vez)
- Crie consciência da necessidade de uma prática usando métricas
- Avalie a experiência de adoção

Práticas Corolárias

- 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

Práticas Corolárias

- Redução 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
- Análise de Causa Inicial
  - Sempre que encontrar um defeito, elimine o defeito e sua causa
  - “Os 5 Porquês”
- Código Compartilhado

Práticas Corolárias

- Código e Testes
  - Os únicos artefatos permanentes
- Repositório Único de Código
  - Desenvolva num repositório único. *Branches* podem existir, mas devem ser incorporados logo
- Implantação Diária
  - Coloque software novo em produção toda noite

Práticas Corolárias

- Contrato de Escopo Variável
  - Contratos devem fixar tempo, custo e qualidade, mas a negociação de escopo deve ser aberta
- Pague-Pelo-Uso
  - Cobre por cada vez que o sistema é usado
  - O dinheiro é o *feedback* máximo

O time de XP – o que mudou?

- Abordagem mais inclusiva, destaca outros papéis típicos do desenvolvimento de software, além do programador
- Diversidade de pessoas trás benefício ao *Time Completo*
- Cada parte do grupo deve comprender seu papel no todo, estamos todos no mesmo barco!
- Interação entre pessoas deve seguir princípios de *fluxo* e *benefício mútuo*

Sobre a equipe

- Diversos papéis
  - Testadores, Projetistas de Interface, Arquitetos, Gerentes de Projeto, etc.

Planejamento – o que mudou?

- Estimar usando tempo real de desenvolvimento e não sistema de pontos
- Definição mais detalhada do ciclo diário, semanal e de estação

Testes – o que mudou?

- Em vez de testar *antes*, testar *cedo*
- Dupla verificação: princípio da redundância
  - Testes e o código lidam com uma mesma funcionalidade
  - Não necessariamente a mesma dupla faz as duas coisas
- Aumento de Custo de Defeito (*DCI*)
  - Quanto mais cedo encontrar o defeito, mais barato é a correção

Escalabilidade

- 1ª versão -> times pequenos
- Valores e Princípios valem em qquer escala
- Práticas podem ser adaptadas

Equipes distribuídas

- Valores se aplicam para equipes distribuídas geograficamente
- É necessário nutrir comunicação e *feedback* devido à separação
- Adaptar as práticas, *base de código única* é a conexão entre as equipes
- Não abandonar práticas difíceis de manter à distância
  - eg.: programação pareada usando VNC

Quando não usar XP

- Quando os valores reais da sua organização não se alinham com os valores de XP

Comunidade

- O suporte de uma comunidade é muito importante
- Reconhecimento e crítica dos seus pares
- Oferece uma perspectiva de pessoas que compartilham os mesmo valores, mas estão outras organizações
- Melhoria através de grupos de estudo
- Entre na comunidade da AgilCoop!

Conclusão

- Depois de 5 anos de experiência, XP amadureceu
- Adotar XP deixou de ser tão extremo
- Novas práticas e princípios facilitam o aprendizado
- A comunidade cresceu, venha fazer parte
- Adote os valores, e comece você mesmo com uma prática que faça sentido na sua organização

Maiores Informações

AgilCoop:

www.agilcoop.org.br/portal/agilcast

Outros:

www.agilealliance.org

www.xispe.com.br

www.extremeprogramming.org

www.agilealliance.com.br