# Engenharia de Sistemas de Informação

## Introdução ao E**X**treme **P**rogramming

Adenilso da Silva Simão

22/06/06

# Processos Ágeis

- Características
  - Elaboram pouca documentação;
  - Facilitam a incorporação de mudanças nos requisitos;
  - Exigem que a equipe de desenvolvimento seja pequena;
  - Preocupam-se com soluções simples;
  - Fornecem versões do software em intervalos freqüentes (a cada hora, a cada dia, ou, mais usualmente, a cada mês ou a cada bimestre);
  - Proporcionam constante interação e cooperação dos usuários;
  - Exigem que desenvolvedores e representantes dos usuários sejam bem informados, competentes e autorizados para tomar decisões;
  - Realizam testes no software constantemente.

# Processos Ágeis (II)

- Exemplos
  - Adaptive Software Development (ASD),
  - SCRUM,
  - The Crystal Methods,
  - Extreme Programming (XP),
  - Dynamic Systems Development Method (DSDM),
  - Pragmatic Programming,
  - Feature-Driven Development (FDD)

# E**X**treme **P**rogramming

- Incremental:
  - versões pequenas do software, com ciclos rápidos,
- Cooperativo:
  - clientes e desenvolvedores trabalham constantemente juntos,
- Direto:
  - o próprio método é fácil de aprender e modificar, bem documentado, e
- Adaptativo:
  - capaz de realizar mudanças a qualquer momento do desenvolvimento do software.-

# E**X**treme **P**rogramming (II)

- Comunicação
- Simplicidade
- *Feedback*
- Coragem

# EXtreme Programming (III)

- Jogo do planejamento
- Versões pequenas
- Metáfora
- Projeto simples
- Testes constantes
- Refatoração constante
- Programação em pares
- Propriedade coletiva do código
- Integração contínua
- Semanas de 40 horas
- Cliente presente
- Padrões de codificação

# Jogo do planejamento

- Um *tradeoff* entre o desejável e o possível
  - Escopo
  - Qualidade
  - Prazo
  - Custo
- Definição de Prioridades
- Estimativas
- Processo
  - Como organizar a equipe?

# Jogo do planejamento (II)

- Baseadas em *Estórias de Uso*
  - Uma descrição (informal) de **uma** utilização do sistema
  - Um conjunto de estórias de uso formaram um *release*
  - A implementação é voltada para as estórias de uso

# Releases pequenos

- Cada *release* de ser o menor possível
  - Foco nas partes mais importantes do sistema
  - Uma característica por vez
  - Subprodutos

# Metáfora

- Um jargão único para diálogo com o cliente
  - Define-se uma metáfora que facilite o diálogo
  - Os termos da aplicação devem vir da metáfora

# Projeto simples

- O projeto do sistema deve ser o mais simples possível
  - Passa em todos os testes
  - Não duplica a lógica do sistema
  - Tem o menor número de classes e métodos
- Cada parte do projeto de justificar sua existência
  - Projete para as necessidades atuais
  - Projete de forma a facilitar mudanças

# Testes constantes

- Para cada característica, um conjunto de teste deve ser criado
  - Se não há expectativas, não há surpresas!
- A execução dos teste deve ser automatizada
  - Executar os teste sempre que possível
  - A confiança no sistema deve ser construída junto com ele
- O ideal é criar os testes antes de desenvolver uma determinada parte

# Reescrita constante

- Reescrever (*refactoring*) trechos de código
  - Refazer trechos de código para simplificá-los
    - Sempre que novas funcionalidades são incluídas, corre-se o risco de se obter um código não estruturado
  - Os teste devem ser executados para garantir que não se *quebrou* nada
    - Novos testes podem ser criados antes da reescrita para assegurar isso

# Programação em pares

- Duas pessoas escrevendo o código simultaneamente
  - 2 programadores, 1 teclado e 1 mouse
- Revisão constante
  - Um programador (responsável pelo teclado e pelo mouse) busca uma solução para a implementação do método em questão
  - O outro programador é responsável por pensar mais estrategicamente:
    - Será que essa implementação vai funcionar?
    - Quais casos de teste podem ser feitos a mais? Quais não funcionarão?
    - É a melhor abordagem?
    - Existe alguma parte que poderia ser reescrita?

# Propriedade coletiva do código

- Todo mundo deve ser dono de todo o código
  - Todos são responsáveis pelo sistema todo
  - Se é preciso alterar alguma parte do código e alguém é capaz de fazê-lo, deve fazê-lo

# Integração contínua

- O código deve ser integrado e testado o mais brevemente possível
  - Pelo menos uma vez por dia
  - Os testes são executados freqüentemente
    - Todos os testes devem passar
- Integrar aos poucos é importante
  - Os problemas aparecem mais próximos de suas causas
- Controle de versão

# Semanas de 40 horas

- O trabalho deve ser feito da segunda a sexta
  - 8 horas por dia
- Duas semanas seguidas de hora extra é um sintoma de problemas

# Cliente presente

- Alguém que realmente entende o domínio deve estar presente na equipe de desenvolvimento
  - Para elucidar dúvidas rápidas
  - Participar das decisões

# Padrões de codificação

- Se todos devem ser capazes de alterar o código
- Se os pares devem ser trocados
  - O código deve ser padronizado

# Relacionamentos entre as Práticas