Conhece algum desses simpáticos senhores?

Origem dos Métodos Ágeis

Porque criar software é uma atividade complexa?

- **Alto índice de incerteza**
  - Ferramenta volátil (tecnologia)
  - Objetivo incerto (escopo)
  - Tarefa envolvendo criatividade
  - Dificuldade inerente
- **Totalmente dependente de pessoas**
  - Capacidade individual
  - Capacidade de trabalho em grupo

Origem dos Métodos Ágeis

Porque criar software é uma atividade complexa?

- **Do nada (anos 50 e 60) ...**
  - Desenvolvimento era, originalmente, uma atividade solitária e "simples" (**gerenciável por uma única pessoa**)
- **... Para o monumental (anos 70, 80 e 90) ...**
  - **Crise do software** → **Aplicação de métodos, técnicas, ferramentas e processos de engenharia** para o desenvolvimento de software
- **... Para o ágil (hoje)**
  - Necessidade de **entrega rápida e flexibilidade**
  - **Economia Moderna:** difícil ou impossível prever como um software (ex.: aplicação Web) evoluirá com o passar do tempo. Condições de mercado, necessidades dos usuários, competição.

Origem dos Métodos Ágeis

Tipos de Processo

- Existem dois tipos básicos de processos [Schwaber]
  - **Processos Definidos**
    - Dado um conjunto de entradas, pode-se definir as saídas com razoável precisão
    - Controle feito com base no acompanhamento de um planejamento inicial
  - **Processos Empíricos**
    - Considera a **mudança** e a incerteza como inevitável
    - **Controle** é feito por acompanhamento freqüente e pequenos ajustes.

- Métodos atuais usam processos do tipo **Definido**, que não se aplicam "bem" ao desenvolvimento de todo tipo de software.

Origem dos Métodos Ágeis

Manifesto Ágil (2001)

- ***Por meio deste trabalho passa-se a valorizar:***
  - ***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.*

Kent Beck et al

http://www.agilemanifesto.org

Origem dos Métodos Ágeis

Definição de Métodos Ágeis [Pressman, 2006]

- A engenharia de software ágil combina:
  - **Filosofia:**
    - encoraja a satisfação do cliente e a entrega incremental do software logo de início;
    - equipes pequenas, altamente motivadas;
    - métodos informais;
    - produtos de trabalho de ES mínimos e simplicidade global de desenvolvimento.
  - **Diretrizes de desenvolvimento:**
    - Enfatizam a entrega em contraposição à análise e ao projeto (apesar dessas atividades não serem desencorajadas) e
    - Comunicação ativa e contínua entre desenvolvedores e clientes.

Métodos Ágeis

O que é Agilidade?

- Resposta a **mudanças** efetiva (rápida e adaptativa).
- **Comunicação efetiva** entre todos os *stakeholders*
- **Atrair o cliente** para o time de desenvolvimento
- **Organizando** o time para que o trabalho executado seja **controlado**.

Com o objetivo de...

Entrega de Software **rápida** e **incrementa**l.

Métodos Ágeis

Processos Ágeis

- É **guiado** pelas descrições do que o cliente necessita (**cenários**)
- Reconhece que **planos tem vida curta**
- **Desenvolvimento de software iterativo** com forte ênfase em atividades de construção (modelos de projeto, componentes de SW / codificação, testes, documentação).
- **Entrega freqüente**, múltiplos "incrementos de software"
- **Adapta** assim que a **mudança** ocorre (validação através do *feedback* do cliente).

Métodos Ágeis

Time de Desenvolvimento Ágil

- **Competência:** talento inato, habilidades específicas relacionadas a software e conhecimento global do processo.
- **Foco comum:** todos devem estar **focados em uma meta** (entregar um incremento de sw em funcionamento ao cliente dentro do prazo prometido).
- **Colaboração:** avaliar, analisar e usar informações que são comunicadas a equipe de sw, **informação para ajudar o cliente** e outros a **entender o trabalho** e construir informações que **forneçam valor ao negócio do cliente**.

Métodos Ágeis

Principais métodos, processos ou abordagens

- ***Extreme Programming* (XP)** – Mais utilizado (Princípios, valores e **Práticas**)
- ***Adaptive Software Development* (ASD)** – Projetos que mudam bastante
- ***Dynamic System Development Method* (DSDM)** – Prática de processo sólido, originou o XP.
- **Scrum** - fornece um **processo** conveniente para projetos e desenvolvimento orientado a objetos – sprints, backlogs.
- ***Crystal Methods*** - diferentes métodos, dos quais se deve selecionar o mais adequado para cada projeto.
- ***Feature Driven Development* (FDD)** – conceitos e práticas.
- ***Agile Modeling* (AM)** – princípios para modelagens ágeis
- **Outros Métodos Ágeis**

Extreme Programming (XP)

- O processo ágil mais utilizado, originalmente proposto por Kent Beck.
- **Valores:** base para aplicação das práticas e princípios (5)
  - Comunicação, Simplicidade, *Feedback, Coragem e Respeito*
- **Princípios:** ferramentas para a **tradução dos valores em práticas**. (11)
  - Passos pequenos, aprender com o erro...
- **Práticas** (Principais e Corolárias) – (24)
  - TDD, Pair-Programming...
- **Fases:** Jogo do Planejamento, Projeto, Codificação e Testes.

Extreme Programming (XP)

- **Valores:** base para aplicação das práticas e princípios. **(5 valores)**
  - **Comunicação:** Maioria dos **problemas** em um projeto de software (**dificuldade na comunicação**).
  - **Simplicidade:** Membros de uma equipe de XP estão freqüentemente **buscando a solução mais simples** para resolver seus problemas atuais. Valor mais intenso de XP.
  - **Feedback:** **Resposta rápida sobre ações realizadas** para se **adaptar as mudanças**. XP promove **ciclos curtos** e **constantes de *feedback***, nos mais variados aspectos do desenvolvimento.

Extreme Programming (XP)

- **Valores:** base para aplicação das práticas e princípios
  - **Coragem:** Valor primário que **deve ter influência e balanceamento com os outros valores**. **Práticas:**
    - coragem da equipe para **priorizar funcionalidades**,
    - incentivar os desenvolvedores para o ***pair-programming***,
    - investir tempo em **refatoração e testes automatizados**,
    - estimar **histórias na presença do cliente**, **compartilhar o código** com todos os membros da equipe,
    - **integração completa** do sistema **diversas vezes ao dia**,
    - adotar **ritmo sustentável**, **abrir mão da documentação**,
    - propor **contratos de escopo variável** e propor a **adoção de um processo novo**.

Extreme Programming (XP)

- **Valores:** base para aplicação das práticas e princípios
  - **Respeito:** A excelência no desenvolvimento de software **depende das pessoas**, e elas **devem se respeitar** para conseguir extrair o seu **máximo potencial**. **Pontos que podem ser influenciados pela falta de respeito:**
    - **Comunicação sem respeito** (conflitos internos)
    - **Coragem sem respeito** (atitudes que vão contra o bem estar da equipe)
    - **Programação Pareada** (exercício contínuo de respeito),
    - **Horas-extras excessivas** (impacto no ritmo sustentável da equipe).
    - **Colaboração entre equipe e cliente** (comunicação aberta e respeitosa).

Extreme Programming (XP)

- **Princípios:** ferramentas para a tradução dos valores em práticas. **(11 princípios)**
  - **Humanidade:** **Balancear necessidades pessoais e** da **equipe**.
  - **Economia:** evitar que o projeto seja apenas um "sucesso técnico". **Agregar valor ao sistema que estão desenvolvendo**. **Clientes** são responsáveis pela **priorização das histórias nas reuniões** de planejamento. Resolver os **problemas mais importantes primeiro**, **maximizando o valor do projeto**.

Extreme Programming (XP)

- **Princípios:** ferramentas para a tradução dos valores em práticas.
  - **Benefício Mútuo:** Todas as atividades devem trazer **benefício a todos os envolvidos** (difícil de aplicar)
  - **Auto-semelhança:** **solução em outros contextos, inclusive em diferentes escalas. Ex.: test-first** não somente no desenvolvimento de **testes unitários**, mas também especificação **teste de aceitação**.
  - **Melhoria:** Valorizar atividades que começam agora e se refinam ao longo do tempo.

Extreme Programming (XP)

- **Princípios:** ferramentas para a tradução dos valores em práticas.
  - **Oportunidade:** **Problemas → oportunidade para mudanças.**
  - **Redundância:** **Problemas difíceis e críticos → resolvido de várias maneiras diferentes.** Reduzir Defeitos e aumentar Qualidade do software produzido
  - **Falha:** **todo erro é um aprendizado.**
  - **Qualidade:** Aumentar Qualidade → aumento produtividade, eficiência e motivação.
  - **Passos pequenos**
  - **Aceitação de responsabilidade:** Não deve ser imposta e sim aceita.

Extreme Programming (XP)

- **Práticas**
  - **2ª edição: 24 práticas**, adaptar da maneira que achar mais apropriada.
    - **Não impor as práticas:** cada mudança deve começar pelos próprios membros da equipe.
    - **Práticas primárias (13):** podem ser aplicadas separadamente, trazendo melhoria imediata a equipe.
    - **Práticas corolárias (11):** mais difíceis de implementar, mostrando eficiência somente após domínio e experiência prévia com as práticas primárias

Extreme Programming (XP)

- **Práticas Primárias**
  - **Sentar Junto:** Espaço amplo e aberto, onde todos possam ficar juntos, fortalecendo a comunicação.
  - **Time completo:** Equipes XP multidisciplinares, com habilidades necessárias para o sucesso do projeto. Trabalhar com espírito de contribuição, visando o bom andamento do projeto.
  - **Área de trabalho informativa:** ambiente de trabalho num reflexo do projeto. Pode ter idéia do andamento do projeto apenas andando pela área de trabalho.
  - **Trabalho energizado:** Ritmo de trabalho não deve afetar a vida pessoal dos membros da equipe. Trabalhar enquanto estiverem produtivos.

Extreme Programming (XP)

- **Práticas Primárias**
  - **Programação pareada:** programadores trabalham em par para realizar suas tarefas. Trabalho coletivo e colaborativo. Pares devem ser trocados regularmente, inclusive várias vezes ao dia.
    - **Objetivo:** Espalhar o conhecimento pela equipe inteira
    - **Efeito colateral importante:** compartilhamento de técnicas e competências entre membros da equipe.

Extreme Programming (XP)

- **Práticas Primárias**
  - **Histórias:** histórias escritas em pequenos cartões. Escritos pelo cliente e devem descrever uma unidade de funcionalidade (geralmente representa um requisito funcional).
    - **Ex.:** "Como um <usuário/papel> Eu gostaria de <funcionalidade> Para que <valor do negócio> (*Mike Cohn*)
    - **Informações utilizadas no jogo do planejamento** (início dos ciclos semanais e trimestrais).
    - Apenas **lembrete do diálogo com o cliente**.

Extreme Programming (XP)

- **Práticas Primárias**
  - **Ciclo Semanal:** produzido de forma iterativa e incremental.
    - **Cada semana reunião para:** refletir sobre o progresso até o momento, planejar e priorizar histórias da semana, quebrar cada história em tarefas.
  - **Ciclo Trimestral:** *releases* planejadas a cada trimestre.
    - **Mais alto nível:** abrangem o todo, **identificando gargalos** (principalmente externos a equipe), **iniciar reparos** e **escolher histórias** mais alinhadas ao tema **que serão implementadas** durante o trimestre.
  - **Folga: caráter subjetivo das estimativas** (experiência do pessoal e desenvolvedor, sujeita a erros).
    - **Time deve se comprometer com a entrega para o cliente.**
    - **Folga incluída no plano.**

Extreme Programming (XP)

- **Práticas Primárias**
  - ***Buid* em 10 minutos:** *Build* **automático** do **sistema inteiro** e a bateria **completa de testes** deve rodar em **até 10 minutos**.
  - **Integração Contínua:** código-fonte armazenado num **repositório compartilhado** e cada par deve **integrar suas alterações ao final de cada tarefa**, após **garantir** que **tudo está funcionando**. **Diversas integrações pequenas e freqüentes** (várias vezes ao dia).
  - **Desenvolvimento Dirigido por Testes (TDD): Testes antes do código**, com ênfase na automatização.
    - ***Design* com problemas** (dificuldade para escrever o teste)
    - **Confiança** (verifica o comportamento agora e no futuro).
    - **Ritmo** (vermelho, verde e refatoração)

Extreme Programming (XP)

- **Práticas Primárias**
  - ***Design* Incremental:**
    - **Implementar o design mais simples (e não o mais simplista): mínimo de complexidade e flexibilidade** para atender às **necessidades do negócio** atuais.
    - Tomar **cuidado para não minimizar o investimento com o design** no curto prazo, mas sim **manter o investimento proporcional às necessidades do sistema** conforme ele **evolui.**
    - **Suporte:** garantir que a equipe seja capaz de solucionar os **problemas futuros com rapidez** – **Exs.:** Refatoração, testes automatizados (TDD)

Extreme Programming (XP)

- **Práticas Corolárias**
  - **Envolvimento Real com o Cliente**
  - **Implantação incremental**
  - **Continuidade da equipe (equipes eficientes trabalhando juntas)**
  - **Diminuição da equipe** (**melhora capacidade** → **reduz carga** sobre um desenvolvedor → **liberar desenvolvedor** para formar **novas equipes**)
  - **Código compartilhado**
  - **Código e Testes** (únicos artefatos mantidos) – **documentação deve ser evitada**, e caso seja estritamente necessária, deve ser gerada a partir de código e testes. **Artefatos obsoletos não agregam valor**.

Extreme Programming (XP)

- **Práticas Corolárias**
  - **Análise da Causa Inicial: defeito encontrado → conserte o problema e suas causas.**
    - Escrever **teste de aceitação** automatizado que **demonstre o problema** e **comportamento esperado**
    - **Teste unitário com menor escopo** que também **reproduz o defeito.**
    - **Corrigir o sistema fazendo todos os testes passarem**
    - **Tentar descobrir a causa inicial do defeito e realizar mudanças** para evitar que o erro **aconteça novamente.**

Extreme Programming (XP)

- **Práticas Corolárias**
  - **Repositório de Código Unificado** (**ramificações devem ser evitadas** p/ **evitar trabalho de sincronização** e **dificuldades no entendimento** da equipe).
  - **Implantação Diária:** colocar **novas versões** do sistema em produção **toda a noite**.
  - **Contrato de escopo negociável:** **fixam tempo, custo e qualidade**, deixando o **escopo preciso** aberto para **negociação**. **Garantir que a equipe está sempre trabalhando no que é mais importante para o cliente.**

Extreme Programming (XP)

- **Práticas Corolárias**
  - **Pague-Pelo-Uso:** utilização do dinheiro como *feedback* final.
    - **Conectar o fluxo econômico ao desenvolvimento informações precisas e atualizadas** para **direcionar melhorias no sistema.**
    - **Ex.: Cobrar a cada vez que o sistema é utilizado, a cada release (problemas com o cliente!)**

Extreme Programming (XP)