eXtreme Programming - XP

Engenharia de Software

UPF - Universidade de Passo Fundo

Prof. Jeangrei Veiga

Ementa

- Apresentação do XP - eXtreme Programming
- Práticas e Estrutura do XP
- Planejando com XP
- Análise dos processos quando usar RUP ou XP
- Tendências em Engenharia de Software

Modelo Tradicional

0. Levantamento de Requisitos
    1. Análise de Requisitos
        2. Desenho da Arquitetura
            3. Implementação
                4. Testes
                    5. Produção / Manutenção

Premissas Básicas do Modelo Tradicional

- É necessário fazer uma análise de requisitos profunda e detalhada antes de projetar a arquitetura do sistema.
- É necessário fazer um estudo minucioso e elaborar uma descrição detalhada da arquitetura antes de começar a implementá-la.
- É necessário testar o sistema completamente antes de mandar a versão final para o cliente.

eXtreme Programming -XP

- Metodologia de desenvolvimento de software.
- Ganhou notoriedade a partir da OOPSLA'2000
  - *Conference on Object-Oriented Programming, Systems, Languages, and Applications.*
- Desenvolvido por Kent Beck nos anos 90

eXtreme Programming -XP

- Desenvolvimento rápido focado em
  - cenários
  - liberação de versões da aplicação
  - teste rápido
- Baseado fortemente em estratégias de desenvolvimento acelerado.
- Principais diferenças das demais metodologia:
  - *Feedback* constante.
  - Abordagem incremental.
  - Comunicação entre as pessoas.

eXtreme Programming -XP

- Abordagem focada de modo a simplificar o projeto
- Começa com cenários (user stories)
  - Desenvolvedores documentam os cenários com modelos informais

    (metáfora do sistema)
  - Testes de aceitação validam as saídas
- Iterativo por natureza

A quem se destina o XP?

- Grupos de 2 a 10 programadores
- Projetos de 1 a 36 meses (calendário)
- De 1000 a 250 000 linhas de código
- Papéis:
  - Programadores (foco central)(sem hierarquia)
  - "Treinador" (*coach*)
  - "Acompanhador" (*tracker*)
  - Cliente

Princípios Básicos

- *Feedback* rápido
- Simplicidade é o melhor negócio
- Mudanças incrementais
- Carregue a bandeira das mudanças (*Embrace change*)
- Alta qualidade

4 Variáveis do Desenvolvimento de Software

- Tempo
- Custo
- Qualidade
- Escopo (foco principal de XP)

Práticas do XP

1. Planejamento
2. Fases Pequenas
3. Metáfora
4. Design Simples
5. Testes
6. Refatoramento
7. Programação Pareada
8. Propriedade Coletiva
9. Integração Contínua
10. Semana de 40 horas
11. Cliente junto aos desenvolvedores
12. Padronização do código

Principais Práticas

- Planejamento
- *Design* Simples
- Teste
- Codificação

XP– Planejamento

- Coleta de requisitos.
- Baseia-se nos requisitos atuais, não em requisitos futuros.
- Histórias do usuário
  - Tem o mesmo propósito dos casos de uso e tomam o lugar de documentos de requisitos extensos.
  - Essas histórias servem de base para a criação dos teste de aceitação.
- *Release Plan* e Plano de Iteração
  - Determinam quais “histórias” serão implementadas e em quantos ciclos isso irá acontecer.
- Pequenos releases
  - São liberados pequenos releases com as iterações.
- Rotatividade da equipe – Programação em Par
  - Prática que pede para não deixar um programador em uma só área com o intuito de:
    - democratizar o conhecimento do software;
    - aumentar a qualidade do código;
- Reuniões diárias – em pé
  - Prática que exige reuniões diárias sobre o andamento do projeto, mas deverá ser realizada em pé

XP– Planejamento

- User Stories são escritas:
  - *User Stories* têm o mesmo propósito dos Casos de Uso da UML.
  - Usualmente, cada *User Story*, é um conjunto de frases curtas, em média 3 sentenças, escritas pelo cliente para definir o que o sistema deve fazer para ele.
  - *Stories* são levadas em consideração quando se é estipulado um prazo para o desenvolvimento.
  - O mais importante ainda, é analisar se o que foi desenvolvido atende as necessidades especificadas na mesma

XP– Planejamento

Caso de uso: Autenticação caixa automático

Ator: Cliente

Descrição: Após inserir o cartão o sistema solicita a senha. O sistema verifica se o cartão é válido e se a senha confere. Se não o usuário tem mais três chances.

- Várias historinhas
  - O sistema mostra telas de boas vindas
  - O sistema verifica se o cartão é válido
  - Implementar sistema de leitura de senha
  - Verificação de senha
  - Releitura de senha se a mesma não confere
  - ...

XP– Planejamento

- O Cronograma é definido pelo *Release Planning:*
  - O *Release Planning* é uma reunião onde são apresentados os *User Stories*, para a definição de alguns fatores para a continuidade do projeto.
  - Esses fatores são:
    - Tempo ideal para o desenvolvimento de cada *User Story*
    - A prioridade de cada *Story*, feita em conjunto com o cliente, a *Project Velocity*, que pode ser definida por tempo ou escopo.
    - Quantas *Stories* podem ser agrupadas e desenvolvidas por iteração.

XP– Planejamento

- O Cronograma é definido pelo *Release Planning:*
  - Essas reuniões são regradas por 4 (quatro) variáveis, que são:
    - Escopo, o quanto pode ser desenvolvido;
    - Recurso, a quantidade de pessoas disponíveis;
    - Tempo,quando o software ou *release* vai ser entregue;
    - Qualidade, o quão bom é o produto e o quanto ele foi testado.

XP– Planejamento

- Fazer pequenos *releases* freqüentemente.
  - esta prática aumenta a qualidade final do produto, pois é possível que o cliente teste o sistema e veja o que está e o que não está de acordo com suas necessidades.

XP– Planejamento

- A Velocidade do Projeto é medida.
  - É a medida de o quanto de trabalho foi feito no projeto, a unidade de medida é a quantidade de *User Stories* que foram concluídas durante uma iteração.
  - Pode ser mensurada por:
    - Escopo, que vai basear quanto tempo durará a próxima iteração, de acordo com a quantidade de *User Stories* implementadas na iteração anterior.
    - Tempo, que determina a quantidade de *User Stories*, que serão desenvolvidas na próxima iteração.
      - Esse modelo por tempo é mais usado, pois assim as iterações têm o mesmo tempo de desenvolvimento, podendo lançar *releases* em um tempo pré-determinado.

XP– Planejamento

- O Projeto é dividido em iterações.
  - As iterações (*iterations*) determinam o passo do projeto.
  - Usualmente as iterações têm a mesma duração, ficam no intervalo de uma a três semanas.
  - Durante o desenvolvimento em iterações, apenas é criado aquilo que estava programado para a atual iteração, não sendo permitido desenvolver algo que esteja previsto para alguma próxima iteração.
  - Durante uma iteração, primeiro são desenvolvidas as *Stories* mais importantes, sendo esta importância definida pelo cliente, durante uma prévia reunião.

XP– Planejamento

- *Move People Around.*
  - Quando apenas um membro da equipe conhece determinada área do desenvolvimento,e esta pessoa está bastante atarefada, ou algum imprevisto acontece, o projeto pára.
  - Solução: todos os membros da equipe tenham o mesmo conhecimento do projeto, fazendo com que a perda de um desenvolvedor seja menos impactante ao projeto.

XP– Planejamento

- Reuniões *Stand-Up* diárias.
  - Reuniões formais demoram muito e fazem com que os desenvolvedores percam tempo que deveria ser gasto nas iterações.
  - XP diz que essas reuniões devem ser feitas informalmente, em pé, onde os colaboradores podem trocar experiências e compartilhar soluções ao projeto.

XP– *Design*

- Simplicidade é a chave
  - Um projeto simples sempre demora menos tempo para ser completado do que um complexo.
- Escolha uma metáfora para o sistema
  - Dessa forma a equipe terá em mente sempre os mesmos padrões para nomear classes, objetos e etc.
- Crie sempre soluções pontuais
  - Resolva o problema como se ele fosse único, isso reduz riscos.
- Nunca adicione funcionalidade “adiantadas”
  - Faça agora aquilo que precisa ser feito agora. Apenas 10% das funcionalidades extra são usadas.
- Refatoração
  - Reestruture o sistema durante todo o tempo, sem modificar o seu comportamento externo. Isso é feito para simplificar o sistema, adicionar flexibilidade ou melhorar o código.

XP– Codificação

- *Cliente sempre disponível*
  - O cliente deve ser tornar parte da equipe. Todas as fases do XP exigem a presença do cliente.
- *Criar testes antes da implementação*
  - Isso ajuda a criar a implementação depois, pois você já sabe o que será testado.
- *Fazer com que todos sejam donos do código*
  - Dessa forma todos se sentiram a vontade para sugerir novas idéias ou consertar *bugs*;
- *Semana de 40 horas*
  - XP defende um ritmo de trabalho que possa ser mantido sem prejudicar o bem estar da equipe

XP– Codificação

- Padronizar a codificação
  - Isso possibilita a comunicação pelo código;
  - Código mais claro possível.
  - XP não se baseia em documentações detalhadas e extensas (perde-se sincronia).
  - Comentários sempre que necessários.
  - Comentários padronizados.
  - Programação pareada ajuda muito.

XP– Codificação

- *Pair Programming*

  "Todo o código destinado ao projeto deve ser desenvolvido por duas pessoas trabalhando juntas em um mesmo computador, compartilhando teclado e monitor"

  - Erro 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.

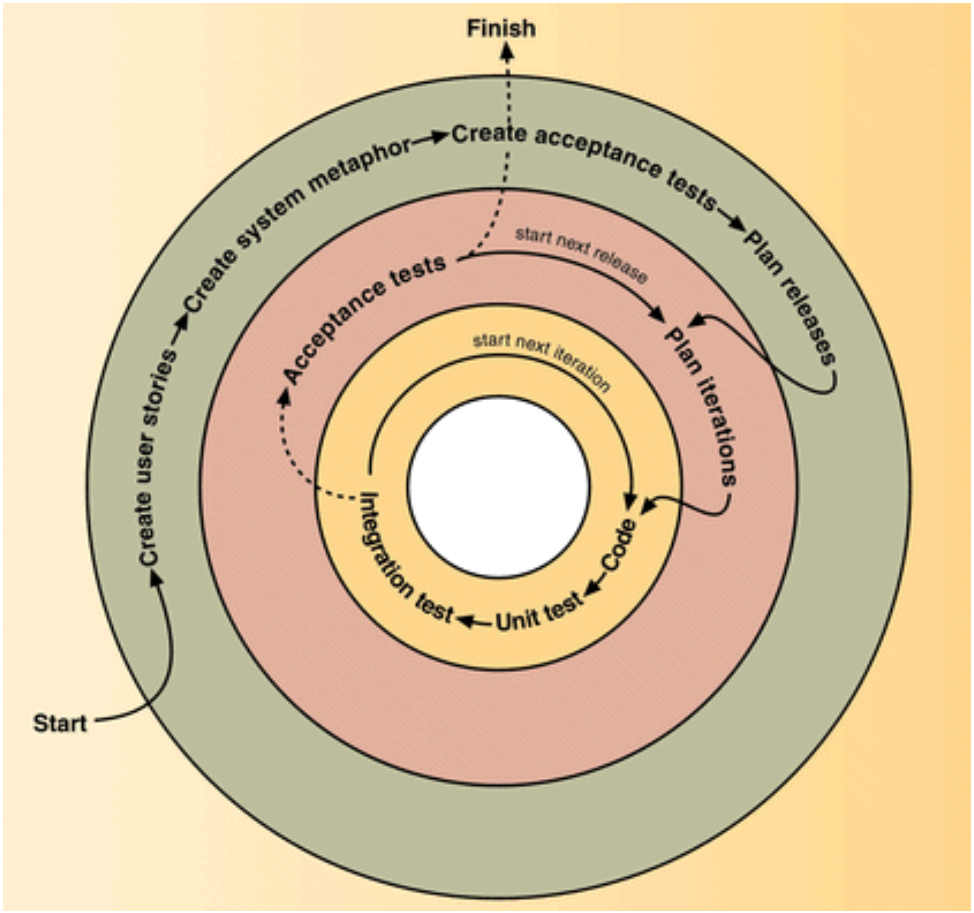
XP– Testes

- É o que dá segurança e coragem ao grupo.
- Os testes são elaborados a partir das especificações do cliente.
- Toda vez que um *bug* é encontrado um teste é criado
  - Isso serve para prevenir que este *bug* volte a acontecer.
- Aplicação dos testes de aceitação
  - Aplicar os testes de aceitação com cenários sugeridos pelo cliente.

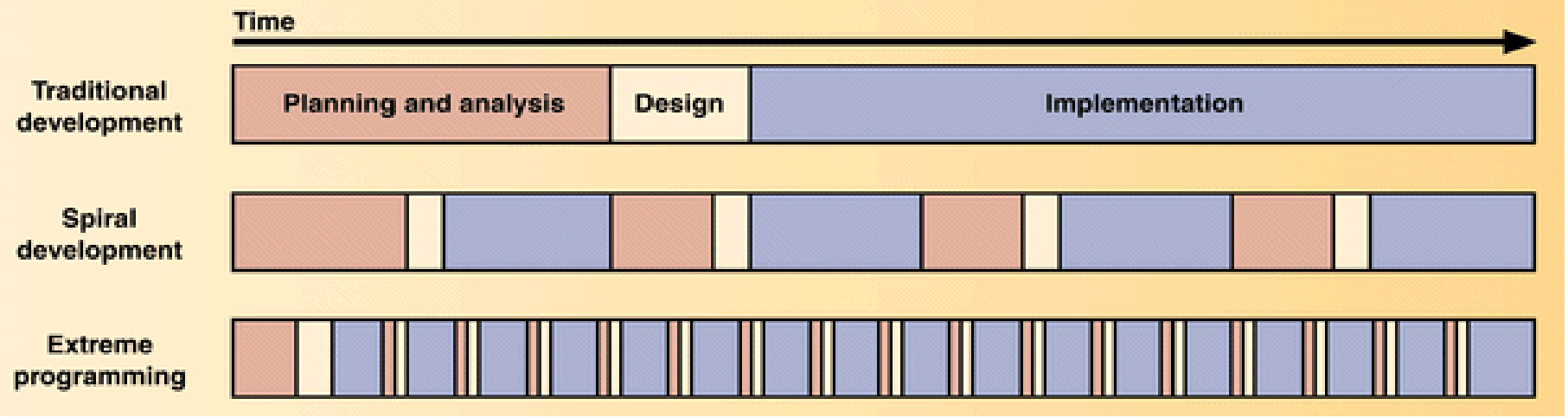
XP– Testes

- Testes de unidade (*Unit tests*)
  - Escritos pelos programadores para testar cada elemento do sistema (ex.: cada método em cada caso).
  - Tem que estar sempre funcionando a 100%.
  - São executados várias vezes por dia.
- Testes de funcionalidades *(Functional tests)*
  - Escritos pelos clientes para testar a funcionalidade do sistema.
  - Vão crescendo de 0% a 100%.
  - Quando chegam a 100% acabou o projeto.

Abordagem no Desenvolvimento com XP



Comparação de Ciclos de Vida



Um Dia na Vida de um Programador XP

- Escolhe uma história do cliente.
- Procura um par livre.
- Escolhe um computador para programação pareada (*pair programming*).
- Seleciona 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
- Desenho
- 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.

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.

Quando XP não deve ser experimentada?

- Grupos grandes (+ de 10 programadores);
- Quando *feedback* rápido não é possível:
  - sistema demora 6h para compilar.
  - testes demoram 12h para rodar.
  - exigência de certificação que demora meses.
- Quando o custo de mudanças é essencialmente exponencial;
- Quando não é possível realizar testes.

Metodologias Tradicionais versus Ágeis

- Alocação de Tempo e Esforço:
  - RUP
    - É divido em fases, essas fases são divididas em iterações e essas iterações contém vários ciclos.
    - Uma iteração do RUP leva entre 2 (duas) semanas e 6 (seis) meses.
  - XP
    - As iterações levam em média 2 (duas) semanas para serem concluídas.
    - Os *releases* são usualmente programados de 2 (dois) em 2 (dois) meses, visto que são determinados em reuniões com os clientes e a equipe.
    - Esses *releases* da XP equivalem as iterações do RUP

Metodologias Tradicionais versus Ágeis

- Análise
  - Na XP antes dos ciclos e *releases* começarem, é necessário uma análise prévia do escopo do projeto para definir se ele é viável ou não, determinar custo, esforço e tempo, esse momento na XP tem a mesma razão que a fase de Concepção no RUP.

Metodologias Tradicionais versus Ágeis

- Análise
  - XP prega que a análise deve ser feita de maneira rápida e objetiva.
  - RUP considera que esta fase levará o tempo que for prudente, pois é feita a análise de risco, fator determinante no processo.

Metodologias Tradicionais versus Ágeis

- Arquitetura e Infra-estrutura
  - XP não se preocupa com um planejamento de infra-estrutura a longo prazo:
    - considera que somente as funcionalidades previstas para a atual iteração devem ser levadas em consideração.
  - RUP planejamento forte na arquitetura e infra-estrutura.

Metodologias Tradicionais versus Ágeis

- Construção
  - A fase de Construção do RUP se equivale aos *releases* da XP, desde que ao final de cada iteração dentro da fase de Construção sejam entregues pacotes de funcionalidade ao cliente.

Metodologias Tradicionais versus Ágeis

- Transição
  - A fase de Transição do RUP não encontra um equivalente dentro da XP, pois ao final de cada *release* este já é entregue ao cliente.

Metodologias Tradicionais versus Ágeis

- Artefatos
  - RUP
    - excesso de burocracia e de documentos.
    - descreve mais de 100 documentos diferentes, sendo esse número um muito alto considerando a abrangência de pequenos projetos.
    - A idéia de especificar com tantos documentos o projeto é deixar o processo mais objetivo com metas bem definidas.
  - XP
    - documentos devem ser gerados quando necessários.

Metodologias Tradicionais versus Ágeis

- Conclusão:
  - Ambas as abordagens têm pontos fortes e fracos.
  - Principal foco do RUP são os grandes projetos, onde se necessitam de especializações e especificações maior do projeto.
  - XP tem seu foco nos pequenos projetos, onde o excesso de documentos e burocracia recomendada pelo RUP é considerado perda de tempo.

Metodologias Tradicionais versus Ágeis

- Tendências e desafios:
  - O desafio futuro das metodologias ágeis é encontrar maneiras de eliminar alguns de seus pontos fracos, como a falta de análise de riscos, sem torná-las metodologias pesadas.
  - Outro desafio é como usar essas metodologias ágeis em grandes empresas e equipes, uma vez que normalmente essas metodologias são baseadas em equipes pequenas.
  - Mescla de metodologias para melhor atender as empresas no âmbito de desenvolvimento de software.