Processo de Desenvolvimento de Software – Desenvolvimento Ágil

Prof. Ingrid Nunes

INF01127 - Engenharia de Software N

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Prosoft

.inf
INSTITUTO DE INFORMÁTICA
UFRGS

Introdução

- Objetivos da Engenharia de Software
  - Como desenvolver software
    - Com funcionalidades esperadas
    - Dentro do prazo e custo
    - De qualidade
- Tendência até os anos 90
  - Processos Prescritivos e maior Formalidade
    - CMM, CMM-I
    - Processo unificado
  - “Heavy-weight process”
- Tendência atual
  - Processos Descritivos e Comportamentais
  - “Light-weight process”
  - **Métodos ágeis**

Por que ser ágil?

- Cerca de **30%** do projetos de desenvolvimento de software são bem sucedidos

Fonte: Standish Caos Report 2009

- Cerca de **55 %** do projetos de desenvolvimento de software são bem sucedidos*

Fonte: Agile Surveys 2010

*Obs: pela percepção ágil de sucesso, este número pode chegar a **80%**

A Crise de Software em Números

Standish project benchmarks over the years

	Challenged (%)	Failed (%)	
	53	31	
	33	40	IID
	46	28	
	49	23	
	53	18	
	46	19	Ágil
	44	24	

CHAOS Success Factors

1. User Involvement
2. Executive Support
3. Clear Business Objectives
4. Emotional Maturity
5. Optimization
6. Agile Process
7. Project Management Expertise
8. Skilled Resources
9. Execution
10. Tools and infrastructure

IID: Incremental Iterative Development

Caos Report Summary 2009

- “somente 20% dos custos estão relacionados ao desenvolvimento de software: o restante é necessário para apoiar a burocracia do negócio”
- “neste sentido, fizemos mudanças em nossos fatores de sucesso. [...] A grande mudança é que **removemos processos formais** [...] processos formais são um fator importante, mas não mais que uma coisa boa ...”
- “... estamos em um **círculo vicioso**: taxas de sucesso são baixas, adiciona-se controle; se elas permanecem baixas, mais controle ... “
- “**Precisamos quebrar este círculo, parar de adicionar controles, avaliar nosso ambiente, adicionar valores aos nossos esforços, acordar e refinar nossa execução**”

*Fonte: www.portal.state.pa.us/portal/server.pt/.../**chaos**_summary_**2009**_pdf*

Mundo Globalizado

- Empresas operam em um ambiente global, com mudanças rápidas
- Precisam responder a novas oportunidades e novos mercados, a mudanças nas condições econômicas e ai surgimento produtos e serviços concorrentes
- Softwares fazem parte de quase todas as operações de negócios

Desenvolvimento e entrega rápidos são fatores críticos para o desenvolvimento de software

Mundo Globalizado

- Empresas que operam em um ambiente de mudanças rápidas dificultam a identificação de um conjunto completo de requisitos de software
  - **Requisitos iniciais serão alterados**
- Processos de desenvolvimento de software que planejam especificar completamente os requisitos, e após, **projetar, construir e testar o sistema não estão adaptados ao desenvolvimento rápido de software**
  - Em 1980 a IBM desenvolveu o **processo incremental**, apoiado pelas linguagens de quarta geração
  - Em 1990 surge o desenvolvimento da noção de abordagens ágeis, como **Metodologia de Desenvolvimento de Sistemas Dinâmicos (DSDM)**

Métodos Ágeis

- Processos de desenvolvimento rápido de software são concebidos para produzir, rapidamente, softwares úteis
- Software não é produzido como uma única unidade, mas como uma série de incrementos
- Cada incremento inclui uma nova funcionalidade do sistema

Sommerville, 2011

Manifesto Ágil

- ***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*

http://www.agilemanifesto.org/

Método Ágil

1. É uma **atitude**, não um processo prescritivo
2. É um **suplemento** aos métodos existentes, e não uma metodologia completa
3. É uma forma efetiva de se **trabalhar em conjunto** para atingir as necessidades das partes interessadas no projeto.
4. É uma coisa que funciona na **prática**, não é teoria acadêmica
5. É para o desenvolvedor **médio**, mas não é um substituto de pessoas competentes
6. Não é um **ataque** à **documentação**, pelo contrário aconselha a criação de documentos que têm valor
7. Não é um **ataque** às **ferramentas** CASE

Metodologias Ágeis Enfatizam

- Comunicação face a face
- Colaboração entre cliente e desenvolvedores
- Software operacional como a principal demonstração de progresso
- Demonstração frequente de progresso
- Técnicas de engenharia que agreguem valor
  - Test Driven Development, Continuous Integration, Refactoring, Design Patterns, etc.
- Retrospectivas e melhoria contínua

Princípios Ágeis (1/4)

- Maior prioridade
  - **Satisfazer** o **cliente**
    - Através da entrega rápida e contínua de software com valor agregado
- **Acolher mudanças** nos requisitos
  - Até mesmo em estágios avançados do processo
  - Mudanças são em benefício da competitividade do negócio do cliente
- Entregar **software operacional frequentemente**
  - Poucas semanas a poucos meses
    - Quanto menor o período, melhor

Princípios Ágeis (2/4)

- Pessoas do **negócio** e **desenvolvedores**
  - Devem trabalhar juntas no projeto diariamente
- Construir projetos com **pessoas motivadas**
  - Dar o ambiente e apoio que elas precisam, é um voto de confiança de que o trabalho será feito
- Forma mais eficiente e efetiva de passar informação para e dentro do time de desenvolvimento é **conversar face a face**

Princípios Ágeis (3/4)

- **Software** pronto é a principal medida de **progresso**
  - "DONE WHEN IT'S DONE"
- Processos ágeis mantêm o **desenvolvimento sustentável**
  - Patrocinadores, desenvolvedores e usuários devem ser capazes de manter um passo constante indefinidamente
- Atenção contínua à **excelência técnica** e bom projeto aumentam a agilidade

Princípios Ágeis (4/4)

- **Simplicidade** é **essencial**
  - Arte de maximizar a quantidade de trabalho NÃO FEITO
- Melhores arquiteturas, requisitos e projetos são resultados de **times auto-gerenciáveis**
- Regularmente, time **reflete** sobre como se tornar mais eficiente, e ajusta seu comportamento de acordo

Princípios Ágeis: Dificuldades

- **Envolvimento** do **cliente** no desenvolvimento de software
  - Depende de um cliente disposto e capaz de passar o tempo com a equipe de desenvolvimento
- Membros individuais da equipe podem não ter **personalidade** adequada para interagirem bem com outros membros da equipe
- **Priorizar** mudanças pode ser **difícil**
  - Principalmente quando existem muitos stakeholders
- Manter **simplicidade**, exige **trabalho extra**

Práticas Básicas (Fonte: Ambler)

- Práticas para a **Modelagem Iterativa e Incremental**
  - Aplique o(s) **artefato(s) correto(s)**
    - "Use a ferramenta certa para o trabalho certo"
      - Ex.: diagrama de atividades vs. código para demonstrar comportamento
  - Crie **diversos modelos** em **paralelo**
    - Cada tipo de modelo possui aspectos positivos e fraquezas, nenhum é suficiente para as suas necessidades de modelagem
      - Ex.: caso de uso, um protótipo, cartões CRC (Class Responsibility Collaborator)
  - **Itere** em outro **artefato**
    - Cada artefato é bom para um determinado tipo de trabalho
  - **Modele incrementalmente**
    - Organize um trabalho mais abrangente em partes menores

Práticas Básicas (Fonte: Ambler)

- Práticas para um **Trabalho de Equipe Eficaz**
  - Modele com **outras pessoas**
    - Não há problema em desenhar um esboço simples para pensar em algo sozinho mas, após ter terminado discuta suas idéias
  - Organize uma **participação** ativa dos **clientes**
    - Possuem autoridade e habilidade para fornecer informações e para tomar decisões
  - Promova a **posse coletiva**
  - Mostre os **modelos publicamente**

Práticas Básicas (Fonte: Ambler)

- Práticas que **Permitem Simplicidade**
  - Crie **conteúdo simples**
    - Não inclua aspectos extras , a não ser que justificáveis (ex.: visibilidade de atributos e métodos)
  - **Mostre** os modelos de **forma simples**
    - Criar conteúdo simples foca o conteúdo do modelo
    - Mostre seus modelos de Modo Simples enfatiza a forma (ex.: evitar cruzamento de linhas, detalhes desnecessários)
  - Use as **ferramentas** mais **simples**

Alguns Critérios de Comparação

	Tradicional	Ágil
Abordagem	**Previsibilidade**	**Adaptabilidade**
Mudança	Controlar	Planejar
Medida de sucesso	Aderência ao plano	Valor para o negócio
Valor	Documentação	Comunicação
Planejamento antecipado	Amplo	Mínimo
Iterações	Poucas, pré-definidas	Muitas
Ênfase	**Orientado a processos**	**Orientado a pessoas**
Estilo de Gerência	Gerência centralizada	Gerência descentralizada
Cultura	Comando/Controle	Liderança/colaboração
Retorno do Investimento	Fim do projeto	Fases iniciais projeto

Tradicional ou Ágil?

- **É importante ter uma especificação e um projeto muito detalhados antes de passar para a implementação?**
  - Se sim, uma abordagem trandicional é necessária
- **É realista uma estratégia de entrega incremental?**
  - Em caso afirmativo, considere o uso de métodos ágeis
- **Quão grande é o sistema que está em desenvolvimento?**
  - Métodos Ágeis são mais eficazes quando a equipe é pequena colocalizada e capaz de se comunicar informalmente
- **Que tipo de sistema está sendo desenvolvido?**
  - Sistemas que exigem uma análise profunda antes da implementação demandam um projeto bem detalhado para atender a esssa análise. Uma abordagem tradicional pode ser mais indicada

Tradicional ou Ágil?

- **Qual é o tempo de vida esperado do sistema?**
  - Sistemas de vida longa podem exigir mais documentação de projeto, a fim de comunicar para a equipe de apoio as intenções originais dos desenvolvedores do sistema
- **Que tecnologias estão disponíveis para apoiar o desenvolvimento do sistema?**
  - Métodos ágeis contam com boas ferramentas para manter o controle de um projeto em desenvolvimento. Em um ambiente IDE (Integrated Development Environment) pode haver necessidade de mais documentação de projeto
- **Como é organizada a equipe de desenvolvimento (centralizada, distribuida)?**
  - Se está distribuída pode ser necessário o desenvolvimento de documentos de projeto para a comunicação entre as equipes de desenvolvimento

Métodos Ágeis e vertentes

- **Extreme Programming (XP)**
- **Scrum**
- Crystal
- Lean software Development
- Feature – Driven Design (FDD)
- Agile Modeling (e seus desdobramentos ...)
- Dynamic Systems Software Development (DSDS)
- Adaptive Software Development (ASD)
- Etc.

Princípios Ágeis – Resumo

Princípios	Descrição
Envolvimento do Cliente	Os clientes devem estar intimamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos e avaliar suas interações.
Entrega incremental	O software é desenvolvido em incrementos com o cliente, especificando os requisitos de em cada um.
Pessoas, não processos	As habilidades da equipe de desenvolvimento devem ser reconhecidas. Membros da equipe devem desenvolver suas próprias maneiras de trabalhar, sem processos prescritivos
Aceitar as mudanças	Deve-se considerar que os requisitos vão mudar. Projete o sistema para acomodar mudanças
Manter a simplicidade	Focalize a simplicidade . Trabalhe para eliminar a complexidade do sistema.

Sommerville, 2011

Agile is not a set of practices, but a core set of beliefs and principles

Jim Highsmith

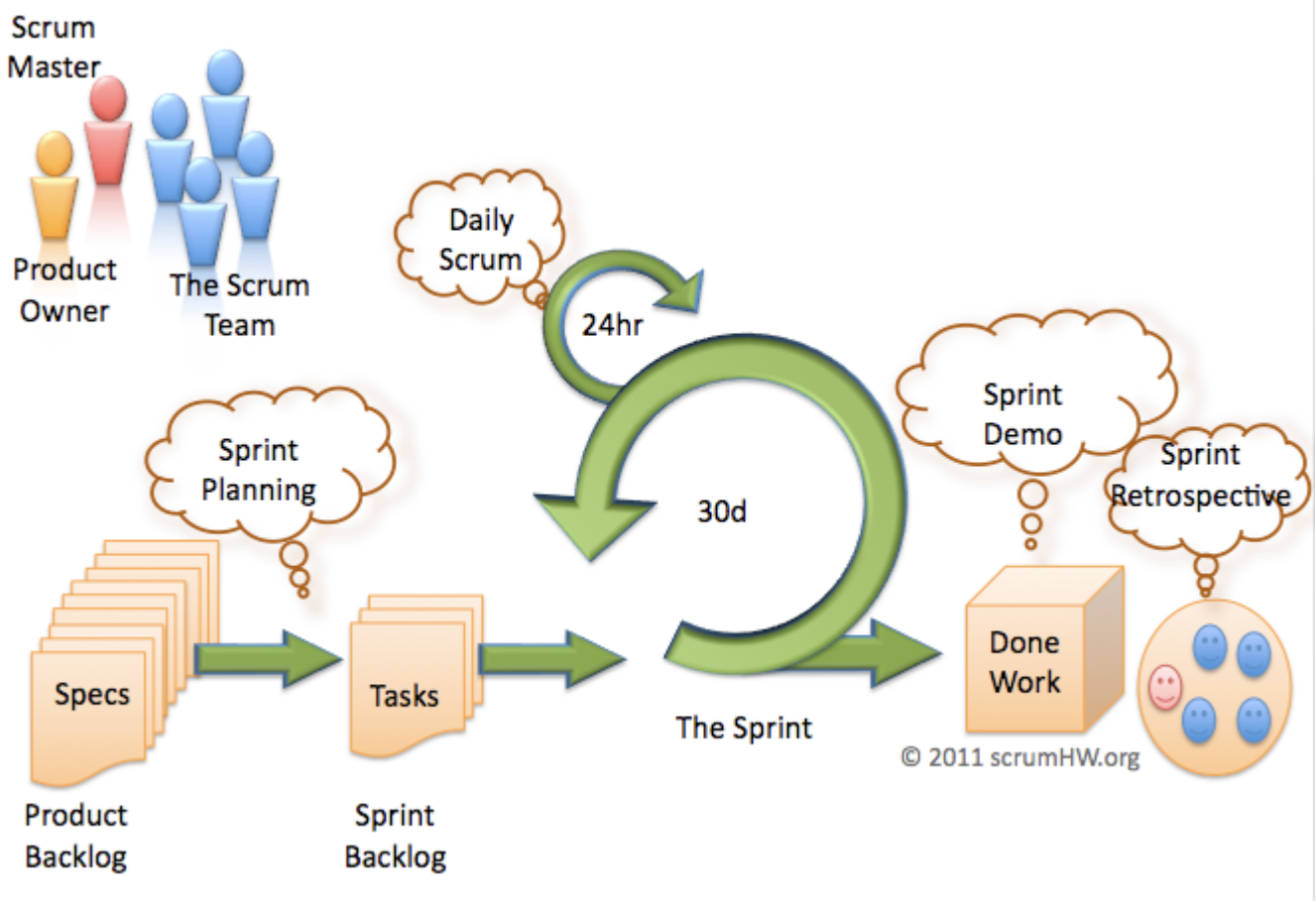
Desenvolvimento de Software Ágil

SCRUM

Scrum

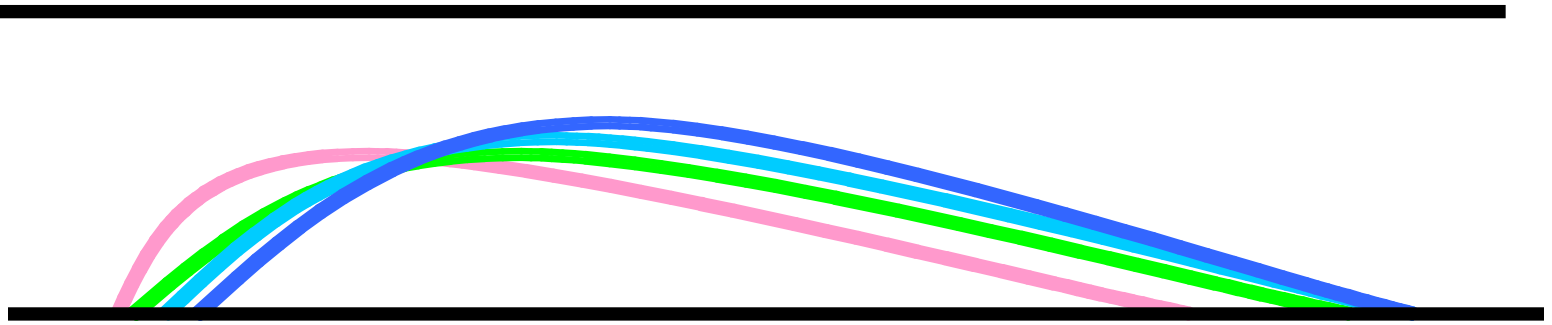
- Método iterativo e incremental
- Projeto é dividido em *sprints*
- *Sprint*
  - Unidade de trabalho para atingir um requisito definido no *backlog*
  - Entregue em um intervalo de 1 semana a um mês
  - Sprint Planning Meeting (prioridades + lista de tarefas)
- Daily Scrum
  - Sincroniza o trabalho da equipe
- Sprint Review
    - Demonstra a funcionalidade acrescentada pelo Scrum

Scrum


Scrum Master
Product Owner
The Scrum Team
Daily Scrum
24hr
30d
Sprint Planning
Sprint Demo
Sprint Retrospective
Specs
Tasks
Done Work
The Sprint
© 2011 scrumHW.org
Product Backlog
Sprint Backlog

Scrum: Sprint

Requirements | Design | Code | Test



DONE !!!

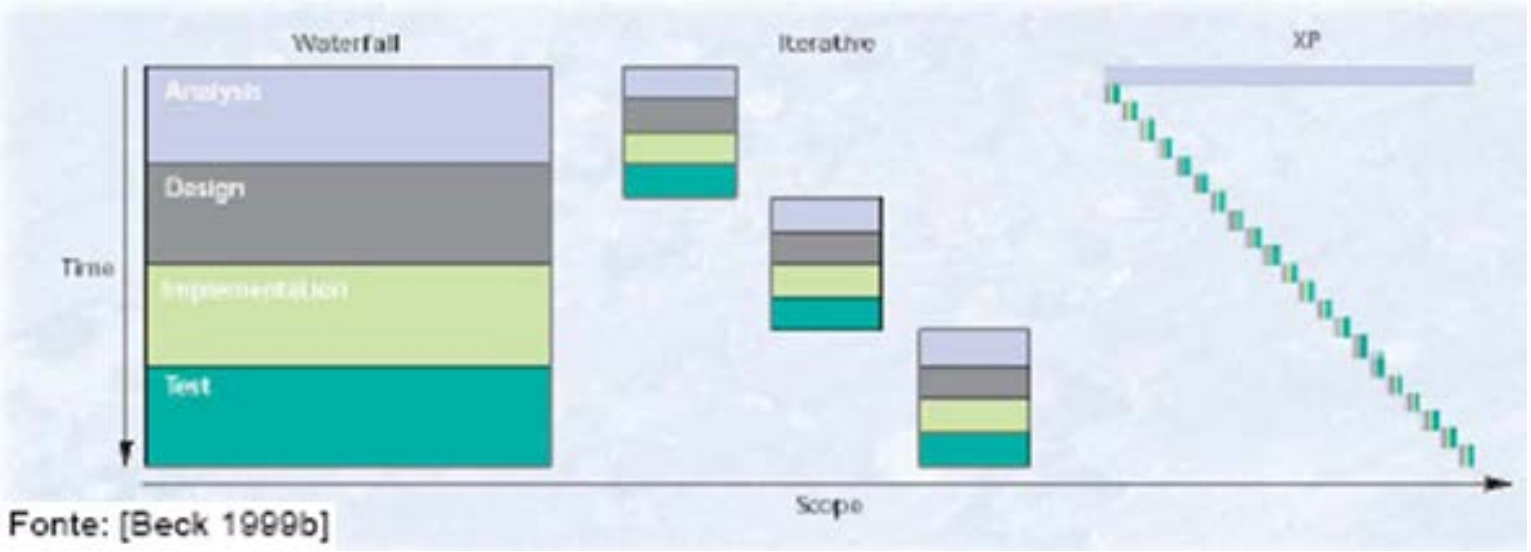
Fonte: “The New New Product Development Game” by Takeuchi and Nonaka. *Harvard Business Review,* January 1986.

Desenvolvimento de Software Ágil

EXTREME PROGRAMMING

Extreme Programming (XP)

- Desenvolvimento e na entrega de incrementos de funcionalidade muito pequenos



Fonte: [Beck 1999b]

- Baseia-se no
  - Aprimoramento constante do código
  - Envolvimento do usuário na equipe
  - Desenvolvimento e programação em pares
- “light-weight”

XP não é...

O método

O time

Extreme Programming

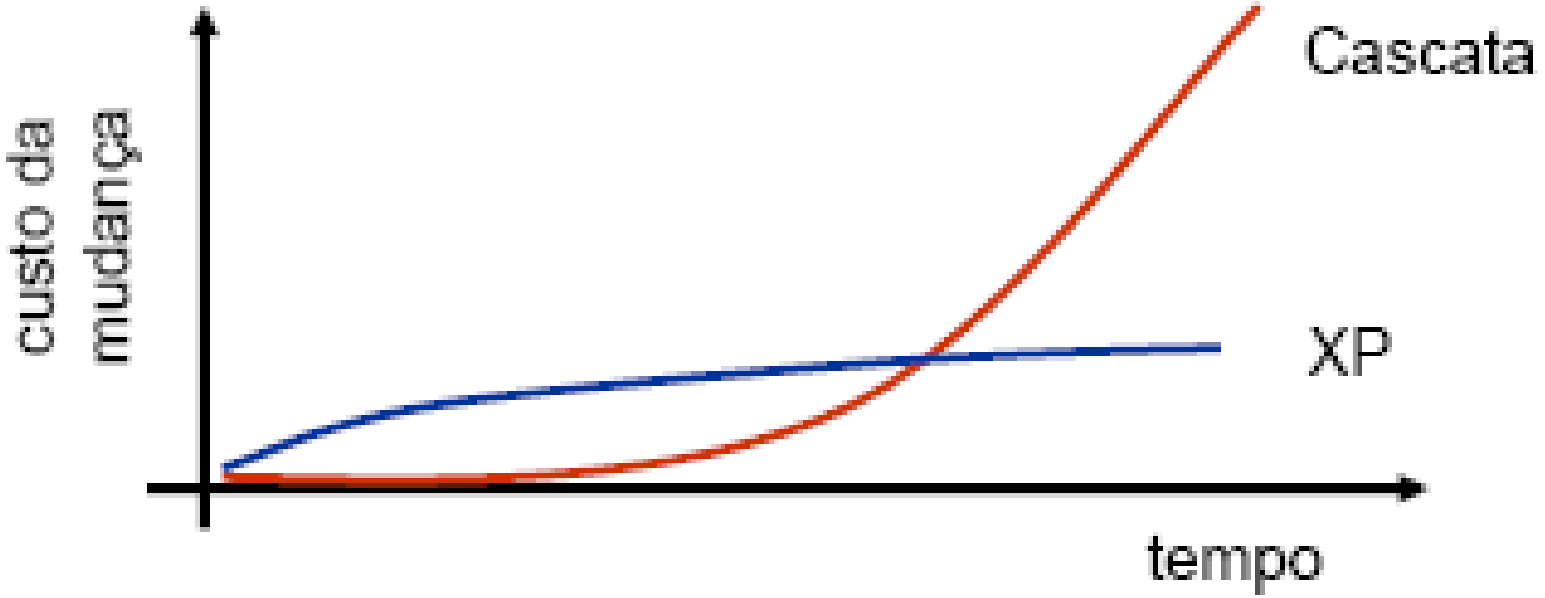
- Um dos métodos ágeis mais conhecidos e utilizados
- Criado por Beck em 2000 com o objetivo de impulsionar o desenvolvimento iterativo

Em XP os requisitos são expressos como cenários, que são implementados como uma série de tarefas
Os programadores trabalham em pares e desenvolvem testes para cada tarefa antes de escreverem o código
Quando o novo código é integrado ao sitema, todos os testes devem ser executados com sucesso

Sommerville

XP e Mudanças

- Desafiar a curva do custo da mudança



XP e Mudanças

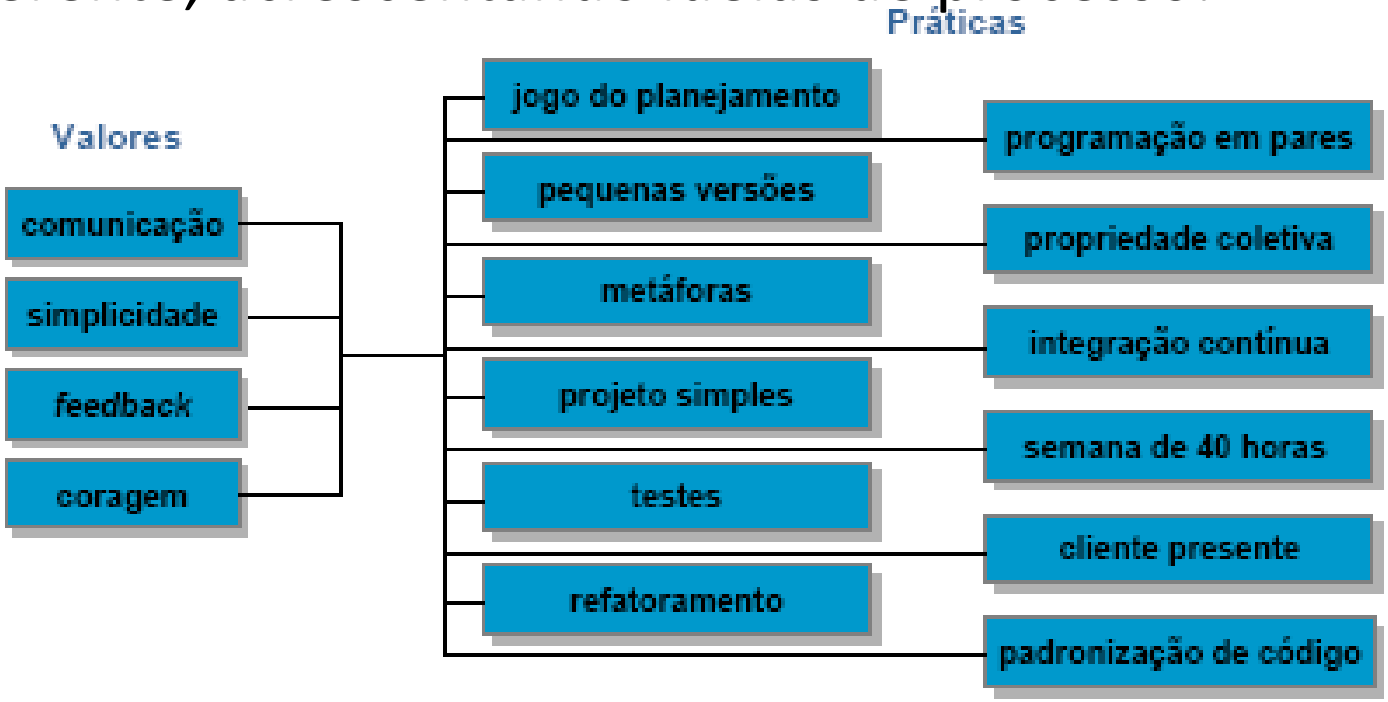
- Tradicionalmente, a engenharia de software recomenda projetar para mudança
  - Vale despender tempo e esforço antecipando mudanças quando isso reduz custos posteriores no ciclo de vida
- XP assume que este esforço não vale a pena quando as mudanças não podem ser confiavelmente previstas
- XP preconiza
  - **Ciclos curtos**
    - Previsibilidade e redução de incertezas/riscos
  - **Simplicidade e melhorias constantes de código (*refactoring*)**
    - Facilitar a mudança
  - **Testes automatizados e integração contínua**
    - Aumentar a confiança

O Mantra do Desenvolvedor XP

- **Escute**, para que saiba qual é o problema a resolver
- **Planeje**, para que você sempre faça a coisa mais importante ainda a fazer
- **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

XP

- Equipes pequenas e médias (2 a 10 pessoas)
- Iterações curtas (2 a 4 semanas)
- Reúne práticas de implementação em um conjunto coerente, acrescentando ideias de processo.



XP: Algumas Práticas

Tabela 17.2 Práticas da *extreme programming*

Princípio ou prática	Descrição
Planejamento incremental	Os requisitos são registrados em cartões de histórias e as histórias a serem incluídas em um release são determinadas pelo tempo disponível e sua prioridade relativa. Os desenvolvedores dividem essas histórias em 'tarefas'. Veja as figuras 17.4 e 17.5.
Pequenos releases	O conjunto mínimo útil de funcionalidade que agrega valor ao negócio é desenvolvido primeiro. Releases do sistema são freqüentes e adicionam funcionalidade incrementalmente ao primeiro release.
Projeto simples	É realizado um projeto suficiente para atender aos requisitos atuais e nada mais.
Desenvolvimento *test-first*	Um framework automatizado de teste unitário é usado para escrever os testes para uma nova parte da funcionalidade antes que esta seja implementada.
Refactoring	Espera-se que todos os desenvolvedores recriem o código continuamente tão logo os aprimoramentos do código forem encontrados. Isso torna o código simples e fácil de manter.
Programação em pares	Os desenvolvedores trabalham em pares, um verificando o trabalho do outro e fornecendo apoio para realizar sempre um bom trabalho.
Propriedade coletiva	Os pares de desenvolvedores trabalham em todas as áreas do sistema, de tal maneira que não se formem ilhas de conhecimento, com todos os desenvolvedores de posse de todo o código. Qualquer um pode mudar qualquer coisa.
Integração contínua	Tão logo o trabalho em uma tarefa seja concluído, este é integrado ao sistema como um todo. Depois de qualquer integração, todos os testes unitários do sistema devem ser realizados.
Ritmo sustentável	Grandes quantidades de horas extras não são consideradas aceitáveis, pois, no médio prazo, há uma redução na qualidade do código e na produtividade.
Cliente *on-site*	Um representante do usuário final do sistema (o cliente) deve estar disponível em tempo integral para apoiar a equipe de XP. No processo da *extreme programming*, o cliente é um membro da equipe de desenvolvimento e é responsável por trazer os requisitos do sistema à equipe para implementação.

Sommerville

Documento = Código

- Codificação é a atividade central do projeto
- Testes (que também são código) servem de especificação
- Comunicação **oral** entre desenvolvedores, baseada no código (testes e funcionalidade) que descreve o sistema
- Isto não quer dizer que equipe XP não se valham de modelos
  - Documento não é incremento
  - Modelos servem para entender e se comunicar

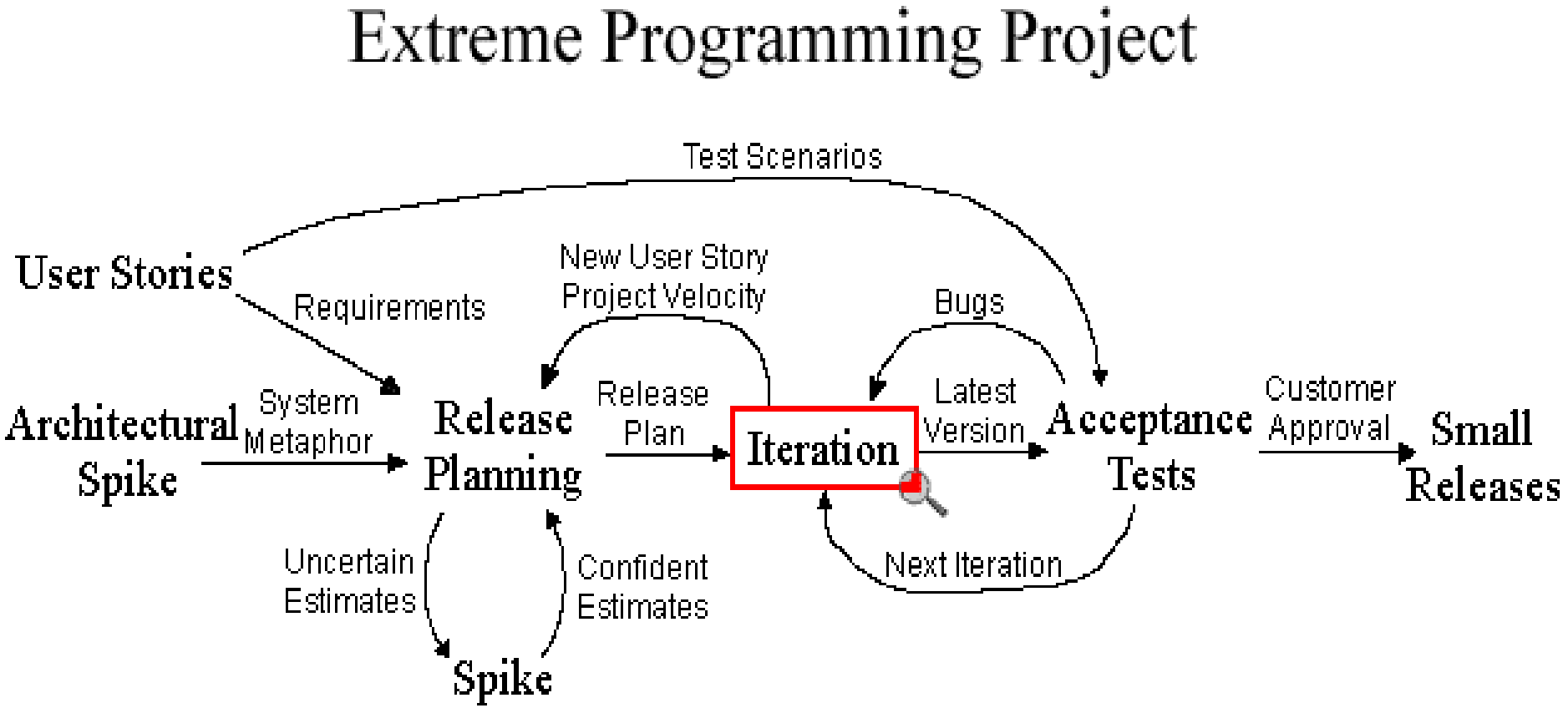
Unidades de Trabalho

- **Releases**
  - Pequenas e freqüentes (a cada 2-3 meses)
- **Iterações**
  - Iteração alcança algum objetivo
    - Tipicamente a adição de nova funcionalidade
  - Nada é feito que não seja imediatamente útil e necessário para não afetar os prazos de desenvolvimento
  - Divididas em tarefas
- **Tarefas**
  - Tarefas são a menor quantidade de trabalho que pode ser feita até que todos os testes voltem a funcionar
  - Tarefas não devem tomar mais que um dia
  - Uma vez concluídas, o resultado das tarefas são integrados imediatamente ao código

XP: Ciclo de desenvolvimento

- Desenvolvimento incremental apoiado em releases de sistema pequenas e frequentes
  - As mudanças são apoiadas em releases de sistema regulares
- **Alto planejamento**
  - Release em iterações
  - Iterações em tarefas
  - Replanejamento a cada iteração/release
- Envolvimento do cliente significa o seu engajamento em tempo integral com a equipe
- Pessoas, e não processos
  - Empowerment
  - Programação em pares, propriedade coletiva, teste automatizado, passo sustentável, etc.
- Manutenção da simplicidade por de meio de refactoring constante do código

XP : Ciclo de Desenvolvimento



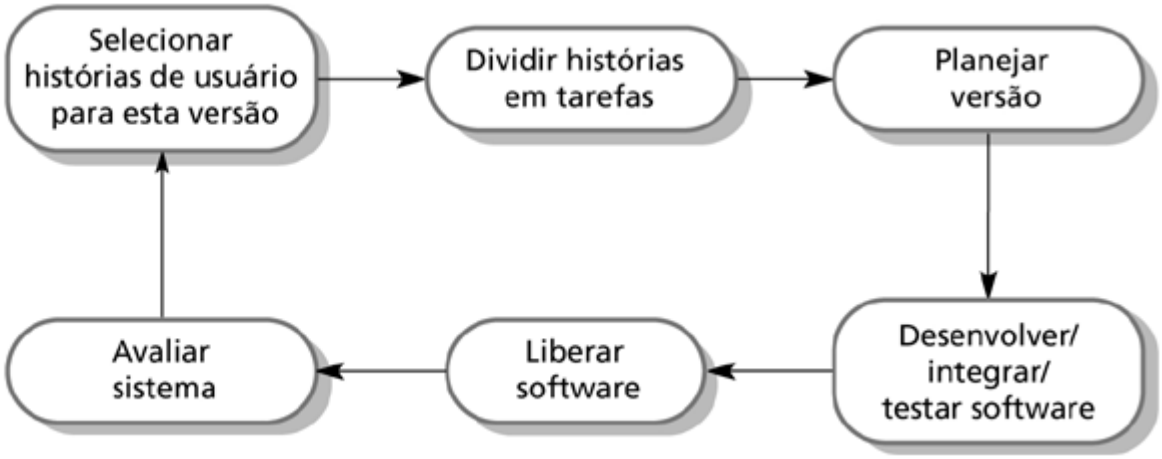
Fonte: www.extremeprogramming.org

Ciclo de Release/Iteração

- Release
  - Conjunto de histórias que são disponibilizados simultaneamente
  - Histórias mais importantes e/ou críticas têm prioridade

Figura 17.3

Ciclo de um release em *extreme programming*



Cenário de Requisitos

- Requisitos de usuários são expressos como cenários ou estórias de usuários
  - Estória, cenário, épico
- Sem formato definido
  - "Como usuário, desejo baixar e imprimir um artigo"

Figura 17.4

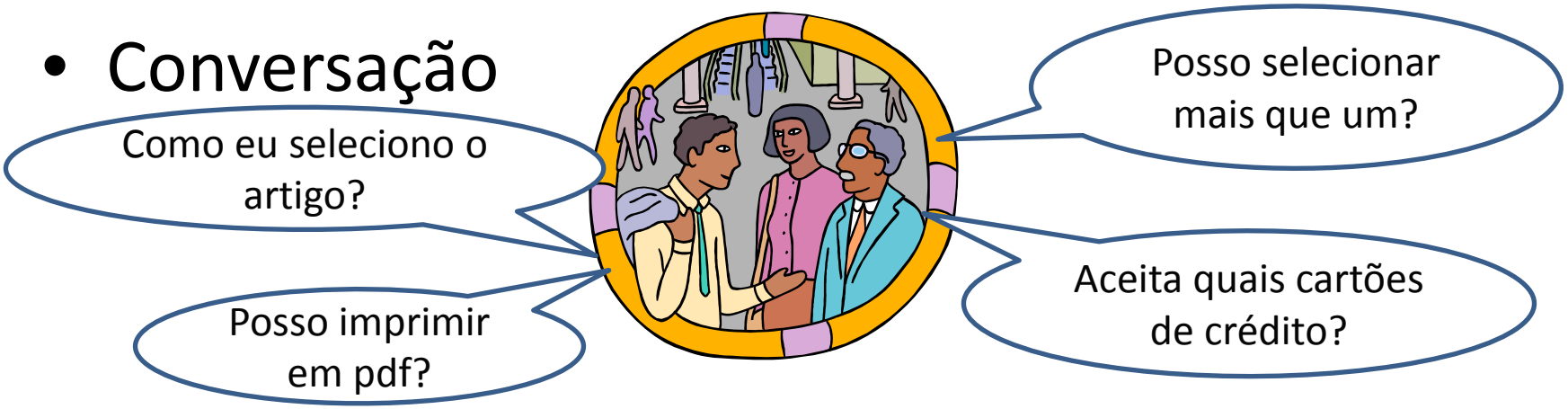
Cartão de histórias para baixar documentos.

Baixando e imprimindo um artigo
Primeiro, você seleciona o artigo que deseja em uma lista. Depois você precisa informar ao sistema como quer pagar pelo artigo — isso pode ser feito por meio de uma assinatura, de uma conta empresarial ou por cartão de crédito. Após, você obtém do sistema um formulário de direitos autorais para preenchimento. Após enviar o formulário, o artigo desejado é baixado para seu computador. Em seguida, você escolhe uma impressora para imprimir uma cópia do artigo. Você informa ao sistema que a impressão foi bem-sucedida. Se o artigo for somente para impressão, você não poderá manter uma versão em PDF, de modo que o artigo será excluído automaticamente de seu computador.

- CCC – Card, Conversation, Confirmation
- **Cartões de Estórias**
  - Principais entradas para o processo XP

Cenário de Requisitos

- Conversação



- Confirmação

Figura 17.6

Descrição de caso de teste para validação de cartão de crédito.

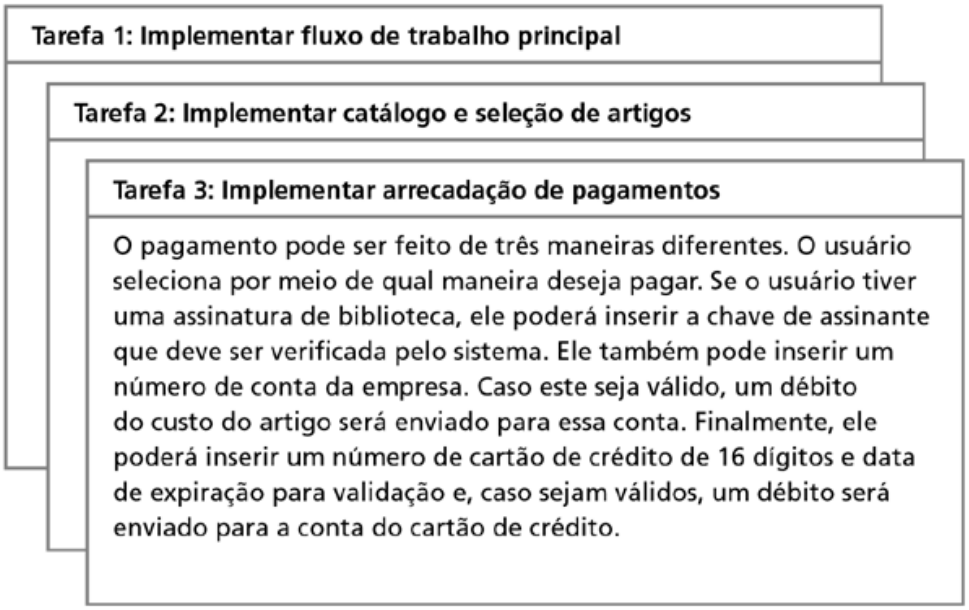
Baixando e imprimindo um artigo
Entrada: Uma string que representa o número de cartão de crédito e dois inteiros que representam o mês e o ano de expiração do cartão.
Testes: Verificar se todos os bytes na string são dígitos. Verificar se o mês varia entre 1 e 12 e o ano é posterior ou igual ao ano atual. Usando os primeiros 4 dígitos do número de cartão de crédito, verificar se o emissor do cartão é válido consultando a tabela de emissores de cartões. Verificar a validade do cartão de crédito enviando o número do cartão e a data de expiração para o emissor de cartões.
Saída: OK ou mensagem de erro indicando que o cartão é inválido.

Cenário de Requisitos

- Equipe de desenvolvimento detalha tarefas de implementação

Figura 17.5

Cartões de tarefa para baixar documentos.



sommerville

- Equipe usa as tarefas para estimativas
  - Todos são responsáveis pelas estimativas

Prática: Teste

- Principais características dos testes em XP
  - Desenvolvimento **test-first**
  - Desenvolvimento de **teste incremental** a partir de cenários
  - **Envolvimento** dos **usuários** no desenvolvimento de testes e validação
    - Teste de aceitação
  - Uso de **framework** de testes **automatizados**
- **Testes** se tornam as **especificações**
  - Escrevendo testes antes da funcionalidade, esclarecem-se as dúvidas sobre o que o software deve fazer
- Testes
  - Impõem **confiança** ao sistema
  - Dão coragem para **alterar** o sistema
  - **Refactoring**
  - Versões **pequenas**

Prática: Teste

- Desenvolvimento em XP é incremental
  - Cliente que faz parte da equipe escreve os testes
    - Enquanto o desenvolvimento avança
  - **Dificuldade: contar com o apoio do cliente**
    - Clientes têm pouco tempo e podem não conseguir trabalhar com a equipe de desenvolvimento em tempo integral
- Automação de testes é essencial para o desenvolvimento *test-first*
  - Testes escritos como componentes executáveis
    - Antes que a tarefa seja implemetada
  - E.g. JUnit é um framework de testes automatizados
  - Ferramentas usadas para executar todos os testes de componentes cada vez que uma nova release é construída

Prática: Teste – Test-first

- Testes são escritos antes do código
  - Teste pode ser executado enquanto o código está sendo escrito
    - Pode-se encontrar problemas durante o desenvolvimento
  - Problemas de requisitos e mal-entendidos de interface são reduzidos
  - Cartões de estória são divididos em tarefa
    - Sendo estas a principal unidade de implementação
  - Cada tarefa gera um ou mais testes de unidade
- Testar um pouco, codificar um pouco
  - “Test-first programming”
  - Se você não automatizou o teste, sua programação não está concluída

Prática: Teste – Test-first

- Descrição resumida de um caso de teste desenvolvido para verificar se a dose prescrita de uma medicação não fica for a dos limites de segurança conhecidos

Teste 4: Verificação de dose

Entrada:
1. Um número em mg representando uma única dose da medicação.
2. Um número que representa o número de doses únicas por dia.

Testes:
1. Teste para entradas em que a dose única é correta, mas a frequência é muito alta.
2. Teste para entradas em que a única dose é muito alta e muito baixa.
3. Teste para entradas em que a dose única x frequência é muito alta e muito baixa.
4. Teste para entradas em que a dose única x frequência é permitida.

Saída:
Mensagem de OK ou erro indicando que a dose está fora da faixa de segurança.

Prática: Projeto Simples

- Projetos flexíveis
  - Defesa contra mudanças imprevistas no software
  - Porém, também têm custos
    - Tempo para desenvolvimento e manutenção
    - Código fica mais complexo
    - Muita vezes a flexibilidade não é utilizada
- Como mudança é barata em XP
  - Mantém-se o **projeto** o mais **simples** possível
  - **Modificado** quando for **necessário** suportar mais funcionalidade
- Melhor projeto é aquele que
  - Passa em todos os **testes**
  - Não contém **duplicação** de funcionalidade
  - Salienta as **decisões** de projeto importantes
  - Tem o menor **número** possível de **classes** e **métodos**

Prática: Refatoração

- Refatorar
  - Melhorar o código sem alterar sua funcionalidade
- Uso desta técnica
  - Aprimora a concepção (*design*) de um *software*
  - Evita a deterioração durante o ciclo de vida de um código
- Exemplos de refatoração
  - Reorganização da hierarquia de classes, eliminando código duplicado
  - Arrumação e renomeação de atributos e métodos
  - Substituição do código com as chamadas para métodos definidos em uma biblioteca de programas

EVITE AO MÁXIMO ADIAR A REFATORAÇÃO!!!

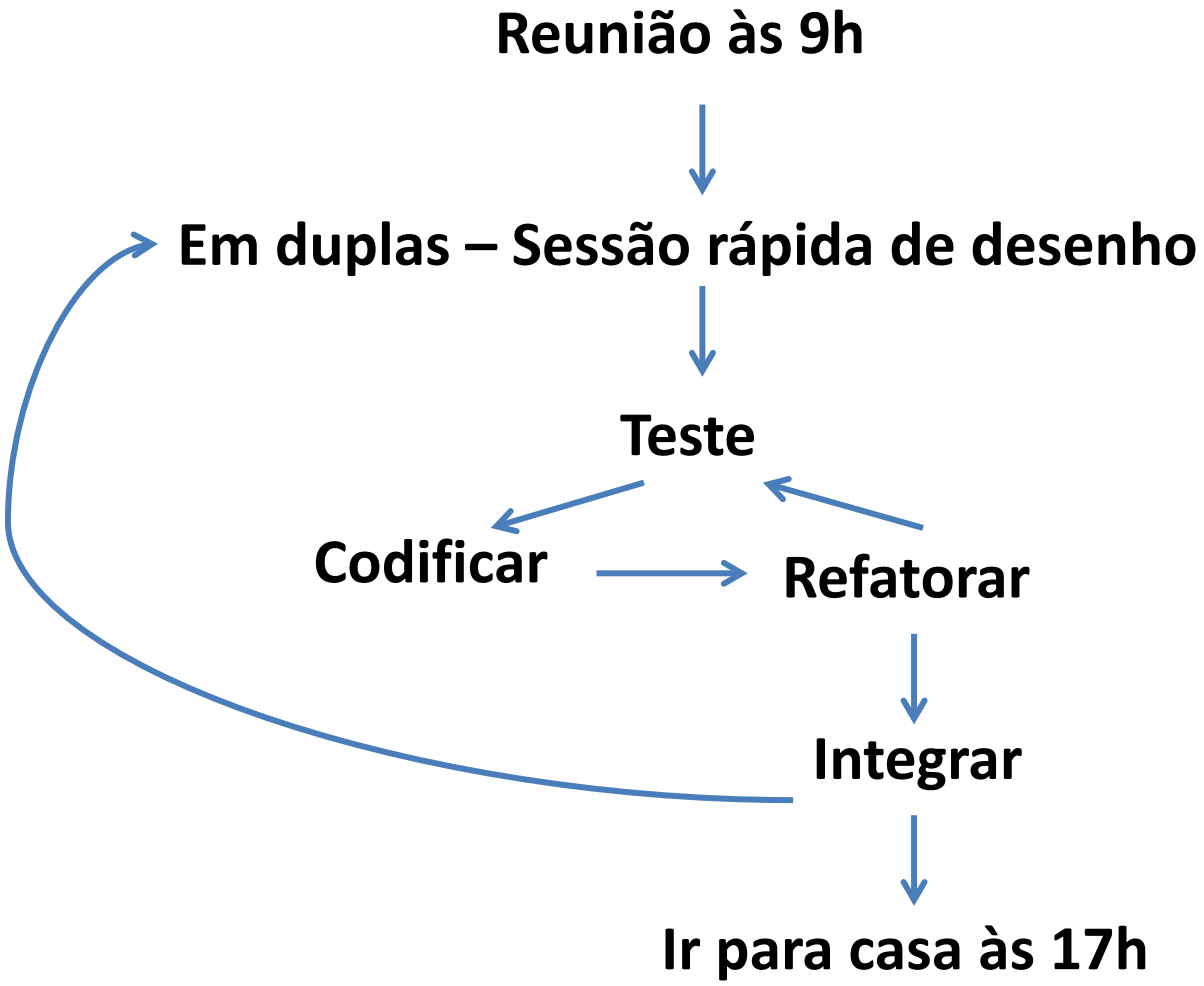
Prática: Refatorar

- Antes de uma mudança
  - Você refatora o código para facilitar a realização de mudanças
- Refatoração contínua
  - Possibilita manter um bom projeto, apesar das mudanças freqüentes
- Projeto é uma atividade diária
  - De responsabilidade de todos
- Aumento contínuo de qualidade do código
- Teste automatizado e Pair Programming dão a coragem de mudar

Prática: Programação em Pares

- Desenvolvedores trabalham em pares
- Pares são criados de maneira dinâmica
  - De modo que todos os membros da equipe trabalhem uns com os outros
- Vantagens
  - Suporte à ideia de propriedade e responsabilidade coletiva para o sistema
  - Processo de revisão informal, pois cada linha de código é observada por, pelo menos, duas pessoas
  - Dá suporte à refatoração
- Limitações
  - Demoradas para organizar e costumam apresentar atrasos no processo de desenvolvimento

Dia típico de um desenvolvedor XP



Formulação de uma estratégia inicial de projeto – duração de 10 a 30 min

Ambler, 2002

Considerações Finais

- Metodologias Ágeis
  - Desenvolvimento incremental
  - Se concentram em
    - Desenvolvimento rápido
    - Releases frequentes do software
    - Redução de overheads de processos
    - Produção de códigos de alta qualidade
- Cliente fortemente envolvido no desenvolvimento
- Decisão por uma abordagem ágil ou tradicional depende do tipo de software a ser desenvolvido
  - Habilidades da equipe de desenvolvimento
  - Cultura da empresa que desenvolve o sistema

Considerações Finais

- XP
  - Integra um conjunto de boas práticas de programação
    - Releases frequentes do software
    - Melhorias contínuas
    - Participação do cliente na equipe de desenvolvimento
- Ponto forte de XP
  - Testes automatizados antes da criação de um recurso de programa

Para saber mais

- Martin Fowler. The new methodology. (resume a corrente de pensamento que levou ao manifesto ágil) www.martinfowler.com/articles/newMethodology.html (se procurar, vai achar traduzido)
- Don Wells. Agile Software Development: A gentle introduction ("tutorial pró-XP")

  www.agile-process.org/
- Acompanhe alguns links/blogs
  - Organizações
    - www.agilealliance.org/
    - *www.**scrumalliance**.org*
  - Mike Cohn (Mountain Goat): blog.mountaingoatsoftware.com
  - Martin Fowler (Thoughtworks): www.martinfowler.com
  - Scott Ambler: www.agilemodeling.com

Para saber mais

- A Gentle Introduction to Extreme Programming.
  - Um tutorial bem acessível sobre XP.
  - www.extremeprogramming.org/
- Alguns blogs
  - Jeffries: xprogramming.com/index.php
  - Fowler: http://martinfowler.com/
  - Wells : www.extremeprogramming.org/
- Quer conhecer Kent Beck?
  - http://www.youtube.com/watch?v=4Awqwtyll8I

Referências

- **Leitura Obrigatória**
  - **Sommerville, I. Engenharia de software, 9a edição. Pearson, 2011.**
    - **Capítulo 3**
  - **Pressman, Roger. Engenharia de Software: Uma Abordagem Profissional, 7ª edição. McGraw-Hill, 2011.**
    - **Capítulo 3**
- Leitura Complementar
  - The Agile Manifesto

Referências

- Kent Beck. Extreme Programming Explained: Embrace Change* (2nd Edition). Addisson –wesley.
- Beck, K. Test-Driven Design. Addison-Wesley.
- Ambler, S. Agile modeling: effective practices for eXtreme programming and the unified proces. John Wiley&Sons.*
- Anderson, A., Beattie, Ralph, Beck, Kent et al. Chrysler goes to “extremes”, Distributed Computing (October 1998), 24-28.
- Cohn, M. User Stories Applied: For Agile Software Development. Pearson.
- Cohn, M. Agile Estimating and Planning. Pearson.
- Cockburn, A. Agile Software Development. Addison-Wesley, 2002.
- Crispin, L. & Gregory, J. Agile Testing. Pearson.
- Schwaber, K.; Beedle, M. Agile Software Development with SCRUM. Prentice Hall, 2001.
- Williams, L. and Kessler, R., All I need to know about pair programming I learned in the kindergarten”. Comm. ACM, 43(5), May 2002

* Existem traduções em português.

Perguntas?

- Este material tem contribuições de
  - Ingrid Nunes
  - Karin Becker
  - Lucinéia Thom
  - Marcelo Pimenta

Prosoft

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

.inf
INSTITUTO DE INFORMÁTICA
UFRGS