



Metodologias Ágeis

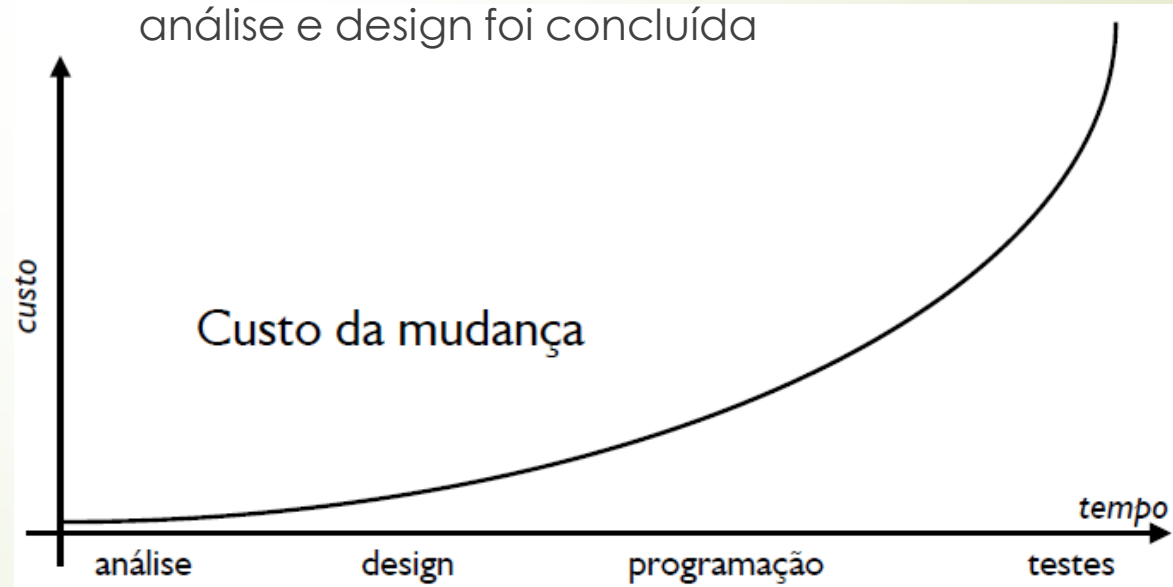
Prof. Adriano Nakamura

Baseado no material do prof. André Luis Belini – UFSP (Instituto Federal de Educação, Ciência e Tecnologia – São Paulo)

Metodologias Ágeis

Motivação

- Engenharia de software tradicional
 - Analisar, projetar, e só depois começar a construir
- Era preciso prever o futuro
 - Como ter certeza que se sabe hoje exatamente o que se quer amanhã?
- Temores
 - Mudanças nos requerimentos depois que a fase de análise e design foi concluída

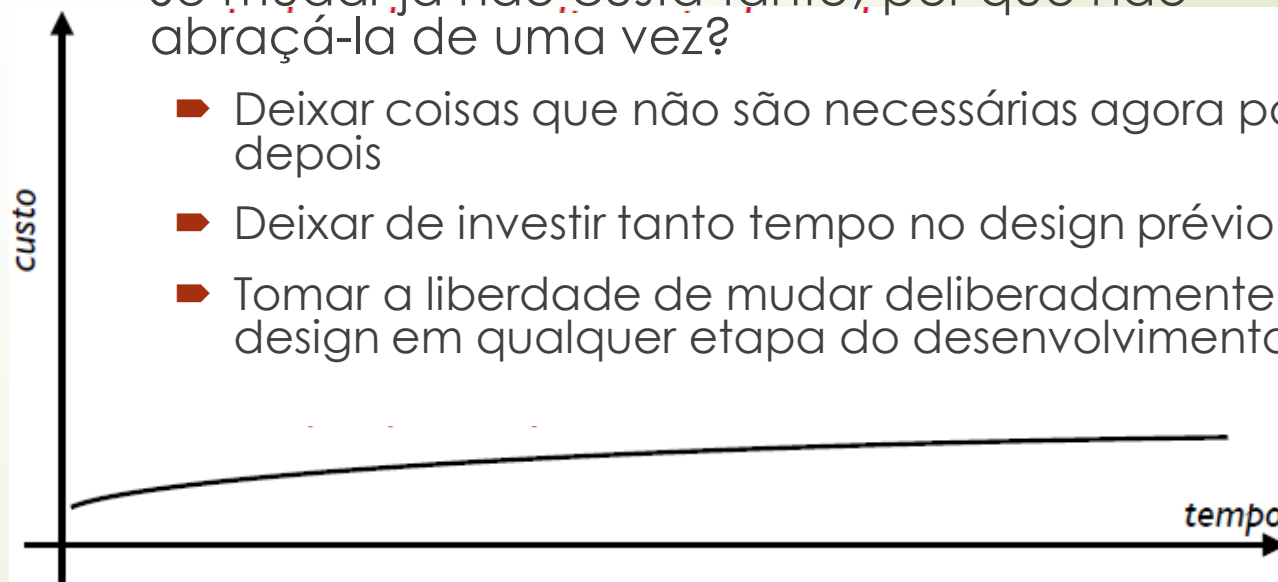


Metodologias Ágeis

Motivação


- Metodologias modernas incentivam iterações curtas
 - Mudanças em etapas avançadas do desenvolvimento seriam mais baratas
- Mudança não é mais um bicho tão feio
 - Podemos conviver com a mudança
 - Patterns, componentes, bancos de dados já absorvem ou podem absorver alto custo da mudança

- Se mudar já não custa tanto, por que não abraçá-la de uma vez?
 - Deixar coisas que não são necessárias agora para depois
 - Deixar de investir tanto tempo no design prévio
 - Tomar a liberdade de mudar deliberadamente o design em qualquer etapa do desenvolvimento



Metodologias Ágeis

- Métodos ágeis
- Desenvolvimento ágil e dirigido a planos
- Gerenciamento ágil de projetos



Desenvolvimento rápido de software

A especificação, o projeto e a implementação são intercaladas.

O sistema desenvolvido como uma série de versões, com os stakeholders envolvidos na avaliação das versões.

Geralmente as interfaces de usuário são desenvolvidas usando uma IDE e um conjunto de ferramentas gráficas.



Métodos Ágeis

Surgimento

- A insatisfação com o overhead que envolve os métodos de projeto de software dos anos de 1980 e 1990 levou a criação de métodos ágeis. Esses métodos:
 - Têm foco no código ao invés de no projeto.
 - São baseados em uma abordagem iterativa de desenvolvimento de software.
 - São planejados para entregar rapidamente o software em funcionamento e evoluí-lo rapidamente para alcançar os requisitos em constante mudança.
- O objetivo dos métodos ágeis é reduzir o overhead nos processos de software (ex. limitando a documentação) e permitir uma resposta rápida aos requisitos em constante mudança sem retrabalho excessivo.



Manifesto Ágil

Princípios	Descrição
Envolvimento do cliente	Os clientes devem estar intimamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar suas iterações.
Entrega incremental	O software é desenvolvido em incrementos com o cliente, especificando os requisitos para serem incluídos em cada um.
Pessoas, não processos	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Membros da equipe devem desenvolver suas próprias maneiras de trabalhar, sem processos prescritivos.
Aceitar as mudanças	Deve-se ter em mente que os requisitos do sistema vão mudar. Por isso, projete o sistema de maneira a acomodar essas mudanças.
Manter a simplicidade	Focalize a simplicidade, tanto do software a ser desenvolvido quanto do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema.



Métodos Ágeis

Aplicabilidade

- Desenvolvimento de produto, quando a empresa de software está desenvolvendo um produto pequeno ou médio para venda.
- Desenvolvimento de sistema personalizado dentro de uma organização, quando existe um compromisso claro do cliente em se envolver no processo de desenvolvimento e quando não existem muitas regras e regulamentos externos que afetam o software.
- Devido ao foco em equipes pequenas e fortemente integradas, existem problemas na escalabilidade de métodos ágeis em sistemas grandes.



Métodos Ágeis

Problemas

- Pode ser difícil manter o interesse dos clientes que estão envolvidos no processo.
- Membros da equipe podem não ser adequados ao envolvimento intenso que caracteriza os métodos ágeis.
- Priorizar mudanças pode ser difícil onde existem múltiplos stakeholders.
- Manter a simplicidade requer trabalho extra.
- Os contratos podem ser um problema assim como em outras abordagens que usam o desenvolvimento iterativo.



Métodos Ágeis

Manutenção de Software

- A maioria das organizações gasta mais na manutenção de softwares existentes do que no desenvolvimento de softwares novos. Devido a isso, para que os métodos ágeis obtenham sucesso, os softwares devem receber tanta manutenção quanto o desenvolvimento original.
- Duas questões muito importantes:
 - É possível dar suporte aos sistemas que são desenvolvidos usando uma abordagem ágil, tendo em vista a ênfase no processo de minimização da documentação formal?
 - Os métodos ágeis podem ser usados efetivamente, para evoluir um sistema em resposta a mudanças nos requisitos do cliente?
- Podem ocorrer problemas no caso do tempo original de desenvolvimento não poder ser mantido.



Desenvolvimento
Ágil

X

Desenvolvimento
Dirigido a Planos

Desenvolvimento dirigido a planos

- Para a engenharia de software, uma abordagem dirigida a planos, é baseada em estágios de desenvolvimento separados, com os produtos a serem produzidos em cada um desses estágios planejados antecipadamente.
- O desenvolvimento incremental é possível no modelo cascata - dirigido a planos.
- Iterações ocorrem dentro das atividades.

Desenvolvimento ágil

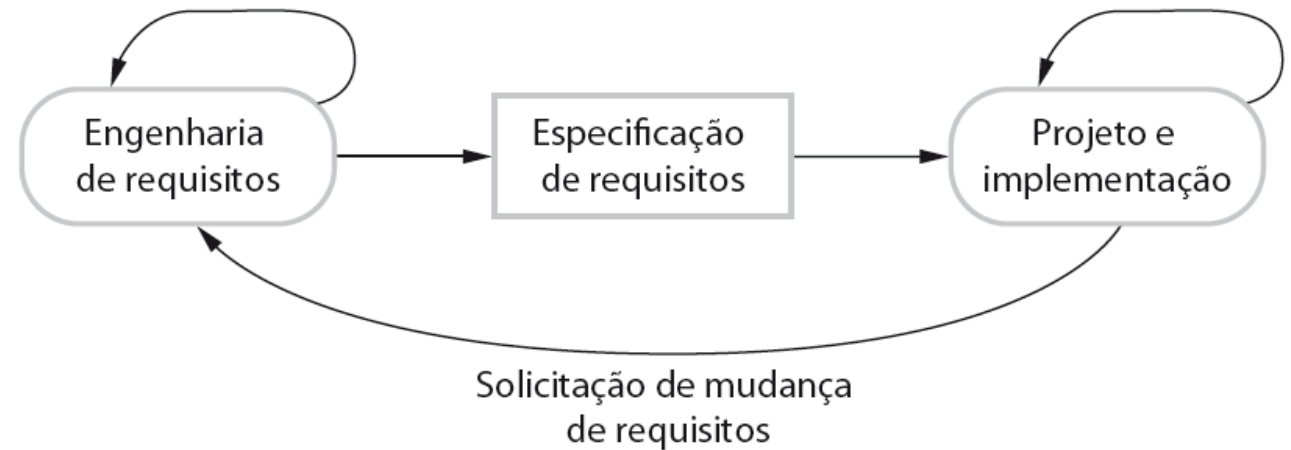
- Especificação, projeto, implementação e teste são intercalados e os produtos do processo de desenvolvimento são decididos através de um processo de negociação, durante o processo de desenvolvimento do software.

Desenvolvimento Ágil

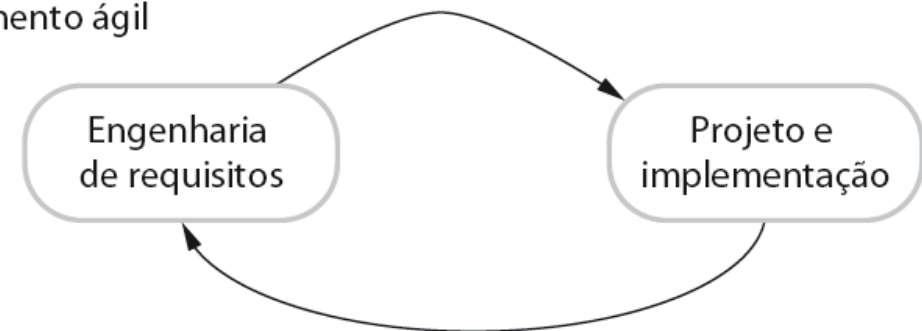
X

Desenvolvimento Dirigido a Planos

Desenvolvimento baseado em planos



Desenvolvimento ágil



Desenvolvimento
Ágil

X

Desenvolvimento
Dirigido a Planos

Questões técnicas,
Humanas e
Organizacionais

- A maioria dos projetos incluem elementos de processos dirigidos a planos e ágeis. Decidir no equilíbrio depende de:
 1. É importante ter uma especificação e projeto bem detalhados antes de passar para a implementação? Caso seja, provavelmente você precisa usar uma abordagem dirigida a planos.
 2. Uma estratégia de entrega incremental onde você entrega o software para os clientes e recebe feedback rápido deles é possível? Caso seja, considere usar métodos ágeis.
 3. Qual o tamanho do sistema a ser desenvolvido? Os métodos ágeis são mais efetivos quando o sistema pode ser desenvolvido com uma equipe pequena que pode se comunicar informalmente. O que pode não ser possível para sistemas grandes que requerem grandes equipes de desenvolvimento, nesses casos, deve ser usada uma abordagem dirigida a planos.

Desenvolvimento
Ágil

X

Desenvolvimento
Dirigido a Planos

Questões técnicas,
Humanas e
Organizacionais

4. Que tipo de sistema está sendo desenvolvido?
Abordagens dirigidas a planos podem ser necessárias para sistemas que requerem muita análise antes da implementação (ex. sistema que opere em tempo real com requisitos de temporização complexos).
5. Qual é o tempo de vida esperado para o sistema?
Sistemas com longo tempo de vida podem precisar de mais documentação de projeto para comunicar as intenções originais dos desenvolvedores do sistema para equipe de suporte.
6. Quais tecnologias estão disponíveis para manter o desenvolvimento do sistema? Métodos ágeis dependem de boas ferramentas para acompanhar um sistema em evolução.
7. Como está organizada a equipe de desenvolvimento?
Se a equipe de desenvolvimento está distribuída ou se parte do desenvolvimento está sendo terceirizado você pode precisar desenvolver documentos de projeto para que haja comunicação entre as equipes de desenvolvimento.

Desenvolvimento
Ágil

X

Desenvolvimento
Dirigido a Planos

Questões técnicas,
Humanas e
Organizacionais

8. Existem questões culturais ou organizacionais que podem afetar o desenvolvimento do sistema? As organizações tradicionais de engenharia têm uma cultura de desenvolvimento dirigido a planos, o que é padrão em engenharia.
9. O quão bons são os projetistas e os programadores da equipe de desenvolvimento? É dito que os métodos ágeis requerem um nível de habilidade mais alto do que as abordagens dirigidas a planos, nas quais os programadores simplesmente traduzem um projeto detalhado em código.
10. O sistema está sujeito a regulamentação externa? Se o sistema precisa ser aprovado por um regulador externo então provavelmente requisitaram a você a produção de documentação detalhada como parte da documentação de segurança do sistema.

Desenvolvimento
Ágil

X

Desenvolvimento
Clássico

- A maioria dos projetos incluem elementos de processos dirigidos a planos e ágeis. Decidir no equilíbrio depende de:

Desenvolvimento Ágil

X

Desenvolvimento
Clássico

Desenvolvedor

Cliente

Requisitos

Retrabalho

Planejamento

Foco

Objetivo

Clássica	Ágil
hábil	ágil
pouco envolvido	comprometido
conhecidos, estáveis	emergentes, mutáveis
caro	barato
direciona resultados	resultados o direcionam
grandes projetos	projetos de natureza exploratória e inovadores
controlar, em busca de alcançar o planejado	simplificar processo de desenvolvimento

Gerenciamento Ágil de Projetos



Um conjunto de **valores, princípios e práticas** que auxiliam a equipe de projeto a entregar produtos ou serviços de valor em um ambiente **complexo, instável e desafiador**



É o exercício de princípios e práticas ágeis aliados aos conhecimentos, habilidades e técnicas na elaboração das atividades de projeto, de forma a diminuir o *time-to-market*, e se adequar às mudanças durante o projeto.



Objetivo

Garantir que exista um equilíbrio entre demandas de qualidade, escopo, tempo e custos



Gerenciamento Ágil de Projetos

- Valores centrais
 - As **respostas às mudanças** são mais importantes que o segmento de um plano
 - A **entrega de produtos** está acima da entrega de documentação
 - **Priorização da colaboração** do cliente sobre a negociação de contratos
 - Os **indivíduos e suas interações** são mais importantes que os processos e ferramentas



Gerenciamento Ágil de Projetos

➤ Principais objetivos

- **Inovação contínua:** a idéia de inovação é associada a um ambiente cuja cultura estimule o auto-gerenciamento e a autodisciplina
- **Adaptabilidade do produto:** os produtos adaptáveis às novas necessidades do futuro
- **Tempos de entrega reduzidos:** direcionamento preciso e capacidade técnica da equipe
- **Capacidade de adaptação do processo e das pessoas:** equipe confortável com mudanças, processo leve
- **Resultados confiáveis:** entrega de produtos que garantam operação, crescimento e lucratividade da empresa



XP (eXtreme Programming)

Prof. Adriano Nakamura

Baseado no material do prof. Helder Rocha (Argonavis – www.argonavis.com.br)



XP (eXtreme Programming)

- É uma metodologia ágil para equipes pequenas e médias e que irão desenvolver software com requisitos vagos e em constante mudança.
- A codificação é a atividade principal.
- Ênfase:
 - Menor em processos formais de desenvolvimento.
 - Maior na disciplina rigorosa.
- Baseia-se na revisão permanente do código, testes freqüentes, participação do usuário final, refatoramento contínuo, integração contínua, planejamento, design e redesign a qualquer hora.



XP (eXtreme Programming)

Valores

Comunicação eficiente entre os membros da equipe

Simplicidade: práticas que reduzem a complexidade do sistema

Feedback: práticas que garantem um rápido feedback sobre várias etapas do processo

Coragem: melhorar um projeto que está funcionando, investir tempo em testes, pedir ajuda aos que sabem, dizer “não” ao cliente quando necessário, abandonar processos formais...



XP (eXtreme Programming)

Valores

- **Comunicação**
- “Os problemas nos projetos invariavelmente recaem sobre alguém não falando com alguém sobre algo importante” (Kent Beck)
- Comunicação efetiva entre clientes e desenvolvedores; XP é organizado em práticas que não podem ocorrer se não houver comunicação.
- A programação em pares onde os desenvolvedores programam num mesmo computador; nesse formato de programação ambos estão constantemente se comunicando e trocando ideias.
- O Jogo do planejamento (planning poker) também é outra forma de comunicação visto que a equipe de desenvolvimento está dando sua visão técnica, o cliente por sua vez está dando requisitos em pró do negocio e dando as prioridades.



XP (eXtreme Programming)

Valores

- **Simplicidade**
- XP incentiva ao extremo práticas que reduzam a
- complexidade do sistema
- A solução adotada deve ser sempre a mais simples que alcance os objetivos esperados
 - Use as tecnologias, design, algoritmos e técnicas mais simples que permitirão atender aos requerimentos do usuário-final
 - Design, processo e código podem ser simplificados a qualquer momento
 - Qualquer design, processo ou código criado pensando em iterações futuras deve ser descartado



XP (eXtreme Programming)

Valores

- **Feedback**
- Várias práticas do XP garantem um rápido feedback sobre várias etapas/partes do processo
 - Feedback sobre qualidade do código (testes de unidade, programação em pares, posse coletiva)
 - Feedback sobre estado do desenvolvimento (estórias do usuário-final, integração contínua, jogo do planejamento)
- Permite maior agilidade
 - Erros detectados e corrigidos imediatamente
 - Requisitos e prazos reavaliados mais cedo
 - Facilita a tomada de decisões
 - Permite estimativas mais precisas
 - Maior segurança e menos riscos para investidores



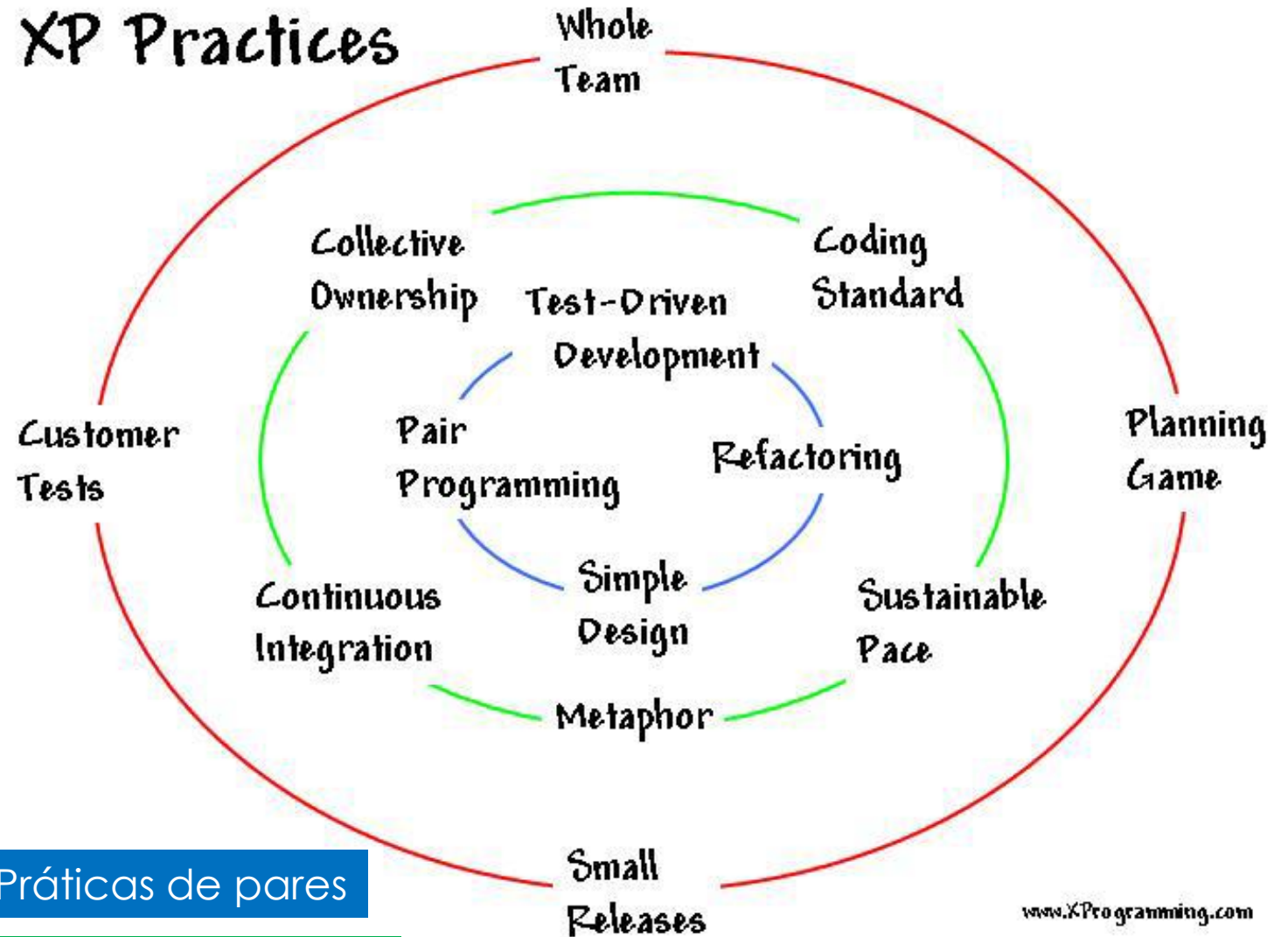
XP (eXtreme Programming)

Valores

- **Coragem**
- Testes, integração contínua, programação em pares e outras práticas do XP aumentam a confiança do programador e ajudam-no a ter **coragem** para
 - melhorar o design de código que está funcionando para torná-lo mais simples
 - jogar fora código desnecessário
 - investir tempo no desenvolvimento de testes
 - mexer no design em estágio avançado do projeto
 - pedir ajuda aos que sabem mais
 - dizer ao cliente que um requisito não vai ser implementado no prazo prometido
 - abandonar processos formais e fazer design e documentação em forma de código

XP (eXtreme Programming)

Práticas



Práticas de pares

Práticas de equipe


Práticas organizacionais



XP (eXtreme Programming)

Práticas: Equipe inteira


- Todos em um projeto XP são parte de uma equipe.
- Esta equipe deve incluir um representante do **cliente**, que
 - estabelece os requerimentos do projeto
 - define as prioridades
 - controla o rumo do projeto
- O representante (ou um de seus assessores) é **usuário final** que conhece o domínio do problema e suas necessidades
- Outros papéis assumidos pelos integrantes da equipe:
 - programadores
 - testadores (que ajudam o cliente com testes de aceitação)
 - analistas (que ajudam o cliente a definir requerimentos)
 - gerente (garante os recursos necessários)
 - coach (orienta a equipe, controla a aplicação do XP)
 - tracker (coleta métricas)



XP (eXtreme Programming)

Práticas: Jogo do Planejamento

- Prática XP na qual se define
 - Estimativas de prazo para cada tarefa
 - As prioridades: quais as tarefas mais importantes
- Dois passos chave:
 - Planejamento de um release
 - Cliente propõe funcionalidades desejadas (estórias)
 - Programadores avaliam a dificuldade de implementá-las
 - Planejamento de uma iteração (de duas semanas)
 - Cliente define as funcionalidades prioritárias para a iteração;
 - Programadores as quebram em tarefas e avaliam o seu custo (tempo de implementação)
- Cliente e equipe concordam com prioridades para iteração
- Ótimo feedback para que cliente possa dirigir o projeto
 - É possível ter uma idéia clara do avanço do projeto
 - Clareza reduz riscos, aumenta chance de sucesso



XP (eXtreme Programming)

Práticas: Testes de usuário

- No **Planning Game**, usuário-cliente elabora "estórias" que descrevem cada funcionalidade desejada. Programador as implementa
 - Cada estória deve ser entendida suficientemente bem para que programadores possam estimar sua dificuldade
 - Cada estória deve ser testável
- Testes de aceitação são elaborados pelo cliente
 - São testes automáticos
 - Quando rodarem com sucesso, funcionalidade foi implementada
 - Devem ser rodados novamente em cada iteração futura
 - Oferecem feedback: pode-se saber, a qualquer momento, quantos % do sistema já foi implementado e quanto falta.



XP (eXtreme Programming)

Práticas: Entregas curtas

- Disponibiliza, a cada iteração, software 100% funcional
 - **Benefícios** do desenvolvimento disponíveis imediatamente
 - **Menor risco** (se o projeto não terminar, parte existe e funciona)
 - Cliente pode medir com precisão **quanto já foi feito**
 - **Feedback** do cliente permitirá que problemas sejam detectados cedo e facilita a comunicação entre o cliente e o desenvolvimento
- Cada lançamento possui funcionalidades prioritárias
 - Valores de negócio implementados foram escolhidos pelo cliente
- Lançamento pode ser destinado a
 - usuário-cliente (que pode testá-lo, avaliá-lo, oferecer feedback)
 - usuário-final (sempre que possível)
- **Design simples** e **integração contínua** são práticas essenciais para viabilizar pequenos lançamentos frequentes
- Lançamentos incluem histórias prioritárias



XP (eXtreme Programming)

Práticas: Design simples

- Design está presente em todas as etapas de no XP
 - Projeto começa simples e se mantém simples através de testes e refinamento do design (reestruturação).
- Todos buscamos design simples e claro. Em XP, levamos isto a níveis extremos
 - Não permitimos que se implemente nenhuma função adicional que não será usada na atual iteração
- Implementação ideal é aquela que
 - Roda todos os testes
 - Expressa todas as idéias que você deseja expressar
 - Não contém código duplicado
 - Tem o mínimo de classes e métodos
- O que não é necessário AGORA não deve ser implementado
 - Prever o futuro é "anti-XP" e impossível (requisitos mudam!)



XP (eXtreme Programming)

Práticas: Programação em Pares


- Todo o desenvolvimento em XP é feito em pares
 - Um computador, um teclado, dois programadores
 - Um piloto, um co-piloto
 - Papéis são alternados frequentemente
 - Pares são trocados periodicamente
- Benefícios
 - Melhor qualidade do design, código e testes
 - Revisão constante do código
 - Nivelamento da equipe
 - Maior comunicação
- "Um" programando pelo preço de dois???
 - Pesquisas demonstram que duplas produzem código de melhor qualidade em aproximadamente o mesmo tempo que programadores trabalhando sozinho
 - 90% dos que aprendem programação em duplas a preferem



XP (eXtreme Programming)

Práticas:
Desenvolvimento
dirigido a teste

- Desenvolvimento que não é guiado por testes não é XP
 - Feedback é um valor fundamental do XP, mas ...
 - ... não há feedback sem testes!
- "Test first, then code"
 - Testes "puxam" o desenvolvimento
 - Programadores XP escrevem testes primeiro, escrevem código e rodam testes para validar o código escrito
 - Cada unidade de código só tem valor se seu teste funcionar 100%
 - Todos os testes são executados automaticamente, o tempo todo
 - Testes são a documentação executável do sistema
- Testes dão maior segurança: **coragem** para mudar
 - Que adianta a OO isolar a interface da implementação se programador tem medo de mudar a implementação?
 - Código testado é mais confiável
 - Código testado pode ser alterado sem medo



XP (eXtreme Programming)

Práticas: Reestruturação ou Refatoração

- Não existe uma etapa isolada de design em XP
 - O código é o design!
- Design é melhorado continuamente através de reestruturação
 - Mudança proposital de código que está funcionando
 - Objetivos: melhorar o design, simplificar o código, remover código duplicado, aumentar a coesão, reduzir o acoplamento
 - Realizado o tempo todo, durante o desenvolvimento
- Reestruturação é um processo formal realizado através de etapas reversíveis
 - Passos de reestruturação melhoram, incrementalmente, a estrutura do código sem alterar sua função
 - Existência prévia de testes é essencial (elimina o medo de que o sistema irá deixar de funcionar por causa da mudança)
- Documentação de design é menos importante uma vez que design pode ser mudado continuamente



XP (eXtreme Programming)

Práticas: Integração contínua

- Projetos XP mantêm o sistema integrado o tempo todo
 - Integração de todo o sistema pode ocorrer várias vezes ao dia (pelo menos uma vez ao dia)
 - Todos os testes (unidade e integração) devem ser executados
- Integração contínua "reduz o tempo passado no inferno da integração"
 - Quanto mais tempo durarem os bugs de integração, mais difíceis serão de eliminar
- Benefícios
 - Expõe o estado atual do desenvolvimento (viabiliza lançamentos pequenos e frequentes)
 - Estimula design simples, tarefas curtas, agilidade
 - Oferece feedback sobre todo o sistema
 - Permite encontrar problemas de design rapidamente



XP (eXtreme Programming)

Práticas:
Posse coletiva

- Em um projeto XP qualquer dupla de programadores pode melhorar o sistema a qualquer momento.
- Todo o código em XP pertence a um único dono: a equipe
 - Todo o código recebe a atenção de todos os participantes resultando em maior comunicação
 - Maior qualidade (menos duplicação, maior coesão)
 - Menos riscos e menos dependência de indivíduos
- Todos compartilham a responsabilidade pelas alterações
- Testes e integração contínua são essenciais e dão segurança aos desenvolvedores
- Programação em pares reduz o risco de danos



XP (eXtreme Programming)

Práticas: Padrões de codificação

- O código escrito em projetos XP segue um padrão de codificação, definido pela equipe
 - Padrão para nomes de métodos, classes, variáveis
 - Organização do código (chaves, etc.)
- Todo o código parece que foi escrito por um único indivíduo, competente e organizado
- Código com estrutura familiar facilita e estimula
 - Posse coletiva
 - Comunicação mais eficiente
 - Simplicidade
 - Programação em pares
 - Refinamento do design



XP (eXtreme Programming)

Práticas: Metáfora

- Equipes XP mantêm uma visão compartilhada da arquitetura do sistema
 - Pode ser uma analogia com algum outro sistema (computacional, natural, abstrato) que facilite a comunicação entre os membros da equipe e cliente
- Exemplos:
 - "Este sistema funciona como uma colméia de abelhas, buscando pólen e o trazendo para a colméia" (sistema de recuperação de dados baseados em agentes) [XPRO]
 - Este sistema funciona como uma agência de correios (sistema de mensagens) [Objective]
- Facilita a escolha dos nomes de métodos, classes, campos de dados, etc.
 - Serve de base para estabelecimento de padrões de codificação



XP (eXtreme Programming)

Práticas:
Ritmo sustentável

- ▶ Projetos XP estão na arena para ganhar
 - ▶ Entregar software da melhor qualidade
 - ▶ Obter a maior produtividade dos programadores
 - ▶ Obter a satisfação do cliente
- ▶ Projetos com cronogramas apertados que sugam todas as energias dos programadores não são projetos XP
 - ▶ "Semanas de 80 horas" levam à baixa produtividade
 - ▶ Produtividade baixa leva a código ruim, relaxamento da disciplina (testes, refatoramento, simplicidade), dificulta a comunicação, aumenta a irritação e o stress da equipe
 - ▶ Tempo "ganho" será perdido depois
- ▶ Projeto deve ter ritmo sustentável por prazos longos
 - ▶ Eventuais horas extras são aceitáveis quando produtividade é maximizada no longo prazo



XP (eXtreme Programming)

Quando não utilizar

- Equipes grandes e espalhadas geograficamente
 - Comunicação é um valor fundamental do XP
 - Não é fácil garantir o nível de comunicação requerido em projetos XP em grandes equipes
- Situações onde não se tem controle sobre o código
 - Código legado que não pode ser modificado
- Situações onde o feedback é demorado
 - Testes muito difíceis, arriscados e que levam tempo
 - Programadores espalhados em ambientes físicos distantes e sem comunicação eficiente



SCRUM

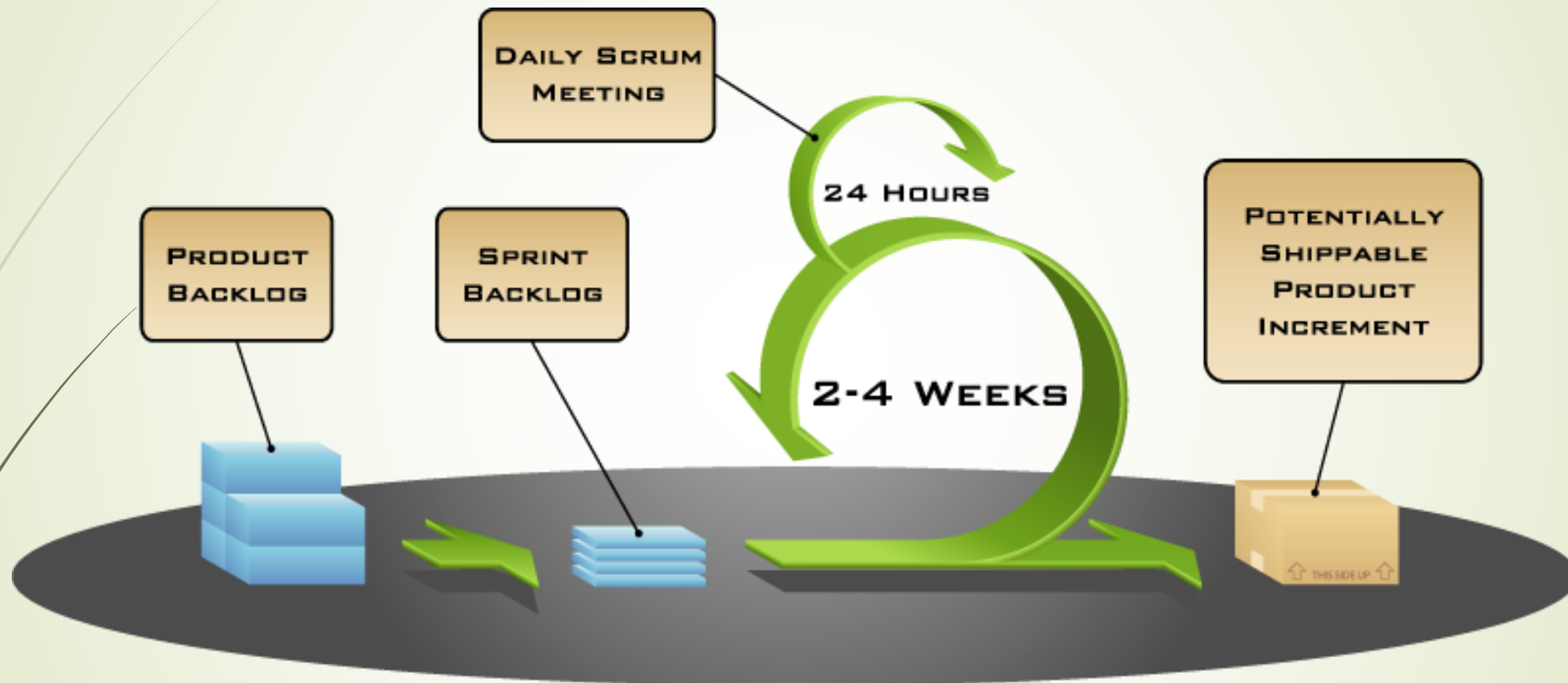
Prof. Adriano Nakamura



SCRUM

- A abordagem Scrum é um método ágil genérico mas seu foco é na gerência de desenvolvimento iterativo ao invés de práticas ágeis específicas.
- Existem três fases no Scrum:
 1. A fase inicial é uma fase de planejamento em que se estabelece os objetivos gerais do projeto e se projeta a arquitetura do software.
 2. Essa é seguida por uma série de ciclos de Sprint, em que cada ciclo desenvolve um incremento do sistema.
 3. A fase de encerramento do projeto finaliza o projeto, completa a documentação necessária como frames de ajuda do sistema e manuais de usuário e avalia as lições aprendidas no projeto.

Em resumo...



COPYRIGHT © 2005, MOUNTAIN GOAT SOFTWARE

Fonte: www.mountangoatsoftware.com/scrum

Sprints

- Projetos Scrum progridem em uma série de “sprints”
 - Similar às iterações do XP
- Ocorre em um período de duas a quatro semanas
- Um período constante leva a um melhor “ritmo”
- O produto é projetado, codificado e testado durante o sprint

Scrum framework

Papéis

- Dono do produto
- ScrumMaster
- Equipe

Cerimônia

- Planejamento
- Revisão
- Retrospectiva
- Reunião diária

Artefatos

- Product backlog
- Sprint backlog
- Burndown charts

Scrum framework

Papéis

- Dono do produto
- ScrumMaster
- Equipe

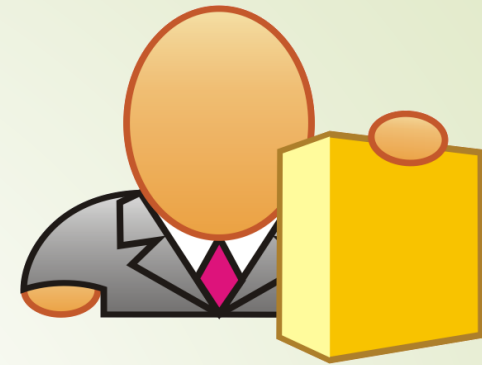
Cerimônia

- Planejamento
- Revisão
- Retrospectiva
- Reunião diária

Artefatos

- Product backlog
- Sprint backlog
- Burndown charts

Dono do produto



- Define as funcionalidades do produto
- Decide datas de lançamento e conteúdo
- Responsável pela rentabilidade (ROI)
- Prioriza funcionalidades de acordo com o valor de mercado
- Ajusta funcionalidades e prioridades
- Aceita ou rejeita o resultado dos trabalhos

ScrumMaster



- Representa a gerência para o projeto
- Responsável pela aplicação dos valores e práticas do Scrum
- Remove obstáculos
- Garante a plena funcionalidade e produtividade da equipe
- Garante a colaboração entre os diversos papéis e funções
- Escudo para interferências externas

Scrum framework

Papéis

- Dono do produto
- ScrumMaster
- Equipe

Cerimônia

- Planejamento
- Revisão
- Retrospectiva
- Reunião diária

Artefatos

- Product backlog
- Sprint backlog
- Burndown charts

Planejamento do Sprint

- A equipe seleciona itens do Product Backlog com os quais compromete-se a concluir
- O Sprint Backlog é criado
 - Tarefas identificadas e estimadas (1 a 16 horas)
 - De forma colaborativa, não apenas feito pelo ScrumMaster
- Planejamento de alto nível é considerado

Scrum diário

- Parâmetros
 - Diário
 - 15 minutos
- Todos em pé!
- Não é para a solução de problemas
 - Todo mundo é convidado
 - Apenas os membros da equipe, ScrumMaster, dono do produto podem falar
- Ajuda a evitar reuniões adicionais desnecessárias



Revisão do Sprint

- Equipe apresenta os resultados obtidos durante o Sprint
- Tipicamente, demonstração de novas funcionalidades ou sua arquitetura
- Informal
 - 2 horas de preparação
 - Sem slides
- Todo o time participa
- O mundo é convidado





Retrospectiva do Sprint

- Periodicamente, observe o que funciona e o que não funciona
- Tipicamente de 15 a 30 minutos
- Feita após cada Sprint
- Toda a equipe participa
 - ScrumMaster
 - Dono do produto
 - Membros da equipe
 - Clientes e outros

Retrospectiva do Sprint

- A equipe discute o que gostaria de:

Iniciar a fazer

Parar de fazer

Continuar
fazendo

Esta é uma das
várias maneiras
de se conduzir
uma
retrospectiva do
Sprint

Scrum framework

Papéis

- Dono do produto
- ScrumMaster
- Equipe

Cerimônia

- Planejamento
- Revisão
- Retrospectiva
- Reunião diária

Artefatos

- Product backlog
- Sprint backlog
- Burndown charts

Product Backlog



Este é o Product Backlog

- **Os requisitos**
- Uma lista de todo o trabalho desejado no projeto
- Idealmente, na forma em que cada item tenha seu peso de acordo com a vontade do cliente ou usuários
- Priorizado pelo dono do produto
- Repriorizado no início de cada Sprint

Exemplo de Product Backlog

Item do Backlog	Estimativa
Permitir que o usuário faça uma reserva	3
Permitir que o usuário cancele a reserva	5
Permitir a troca de datas da reserva	3
Permitir que empregados do hotel gerem relatórios de lucratividade	8
Melhorar manipulação de erros	8
...	30
...	50

O objetivo do Sprint

- Breve declaração que exemplifica o foco do trabalho durante o Sprint

Base de Dados

Fazer com que a aplicação rode no SAL Server além do PostgreSQL

Ciências da vida

Funcionalidades para estudos genéticos da população

Serviços financeiros

Criar suporte para indicadores de desempenho em tempo real

Gerenciando o Sprint Backlog

- Cada indivíduo escolhe o trabalho que fará
 - Trabalhos nunca são atribuídos
- Atualização diária da estimativa do trabalho restante
- Qualquer membro da equipe pode adicionar, apagar ou mudar tarefas
- O trabalho aparece a partir do Sprint
- Se uma tarefa não é clara, defina-a como um item com uma quantidade maior de tempo e subdivida-a depois
- Atualize as coisas a serem feitas na medida em que se tornam mais conhecidas

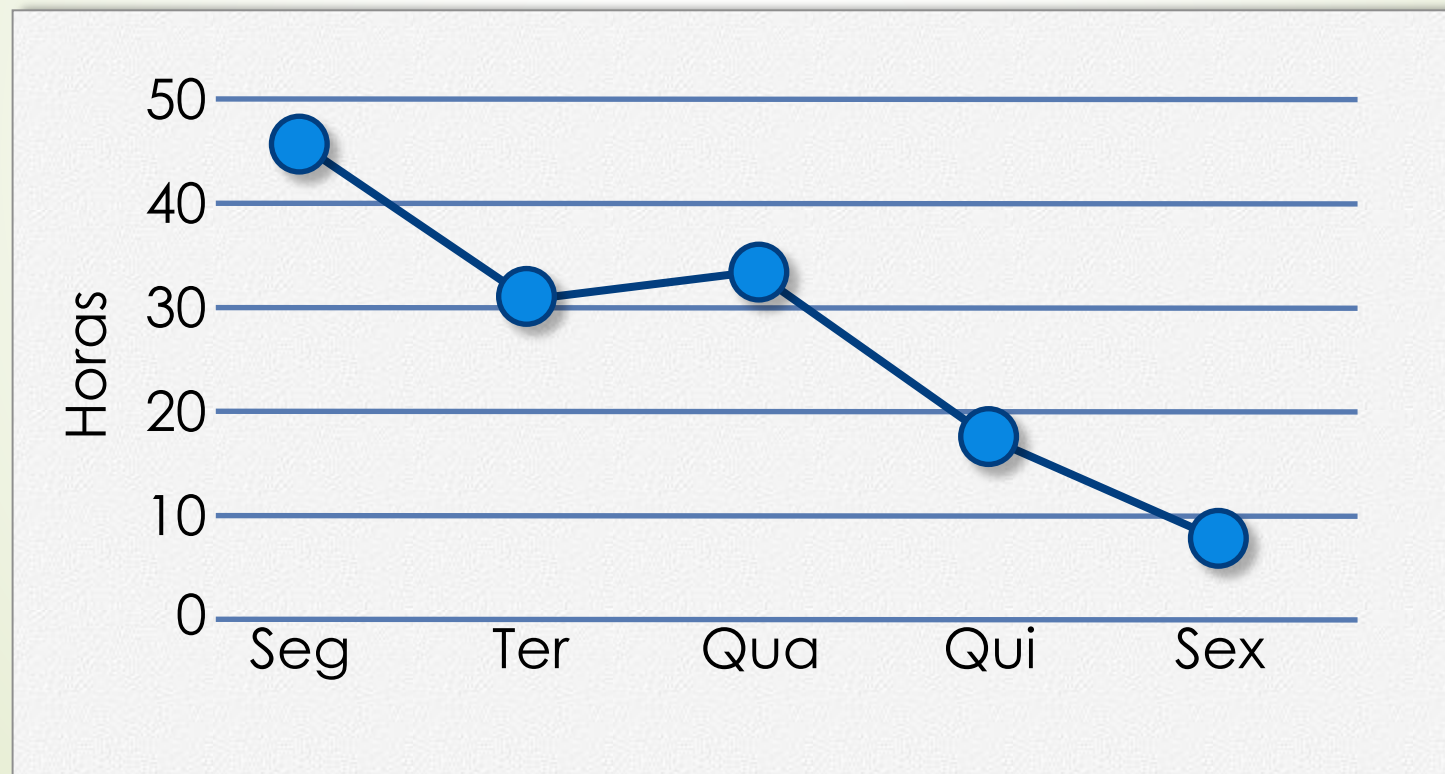
Sprint Backlog

Tarefas	Seg	Ter	Qua	Qui	Sex
Codificar interface de usuário	8	4	8		
Codificar regra de negócio	16	12	10	4	
Testar	8	16	16	11	8
Escrever help online	12				
Escrever a classe foo	8	8	8	8	8
Adicionar log de erros			8	4	

Burndown Chart



Tarefas	Seg	Ter	Qua	Qui	Sex
Codificar interface de usuário	8	4	8		
Codificar regra de negócio	16	12	10	4	
Testar	8	16	16	11	8
Escrever help online	12				
Escrever a classe foo	8	8	8	8	8
Adicionar log de erros			8	4	



Considerações finais

- Um ponto particularmente forte da programação extrema é o desenvolvimento de testes automatizados antes de se criar um atributo do programa.
- Todos os testes devem ser executados com sucesso quando um incremento é integrado ao sistema.
- O método Scrum é um método ágil que provê um framework de gerenciamento de projeto. É baseado em um conjunto de Sprints, que são períodos fixos de tempo em que um incremento de sistema é desenvolvido.
- Escalamento de métodos ágeis para sistemas de grande porte é difícil. Tais sistemas precisam de mais projeto inicial e alguma documentação.