Métodos Ágeis em Engenharia de Software

Introdução

- Desenvolvimento de software é imprevisível e complicado;
- Empresas operam em ambiente global com mudanças rápidas;
- Reconhecer e se adaptar as mudanças é imprescindível;
- Construir o software pela metodologia clássica e perceber que as especificações mudaram durante o processo;

Introdução

- Métodos Ágeis surgem para suprir esta necessidade rápida de mudança;
- O software é desenvolvido como uma série de incrementos;
- Seguem uma filosofia diferente focando principalmente nos resultados (processo ainda é importante);
- Menos ênfase nas definições de atividades e mais na pragmática e nos fatores humanos.

Introdução

Ágil

é diferente de

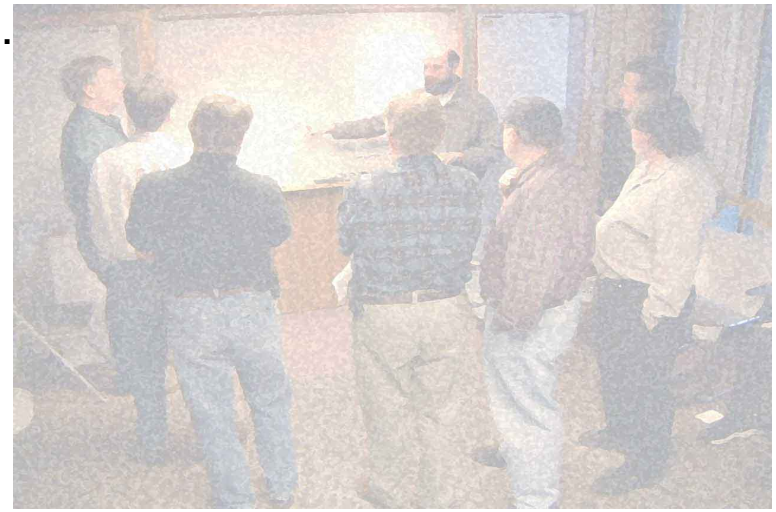
Rápido

Introdução

- Visão em 1980 até o início de 1990 era de que o melhor software tem:
  - Um planejamento cuidado do projeto;
  - Qualidade da segurança formalizada;
  - Uso de métodos de análise e projeto apoiados por ferramentas CASE;
  - Um processo de software rigoroso e controlado.
- Visão baseada nos engenheiros de software responsáveis pelo desenvolvimento de sistemas de software grandes e duradouros, como sistemas aeroespaciais e de governo.
- Por exemplo, sistema de uma aeronave moderna pode demorar 10 anos da especificação até a implantação;

Introdução

- O problema surge quando se aplica essa abordagem em sistemas corporativos de pequeno e médio porte;
- Gasta-se mais tempo em análises de como o sistema deve ser construído do que o programa e os testes em si;
- A insatisfação com essas abordagens pesadas levou em 1990 uma grande quantidade de desenvolvedores a propor métodos ágeis;
- Essa filosofia é refletida no **Manifesto Ágil**.



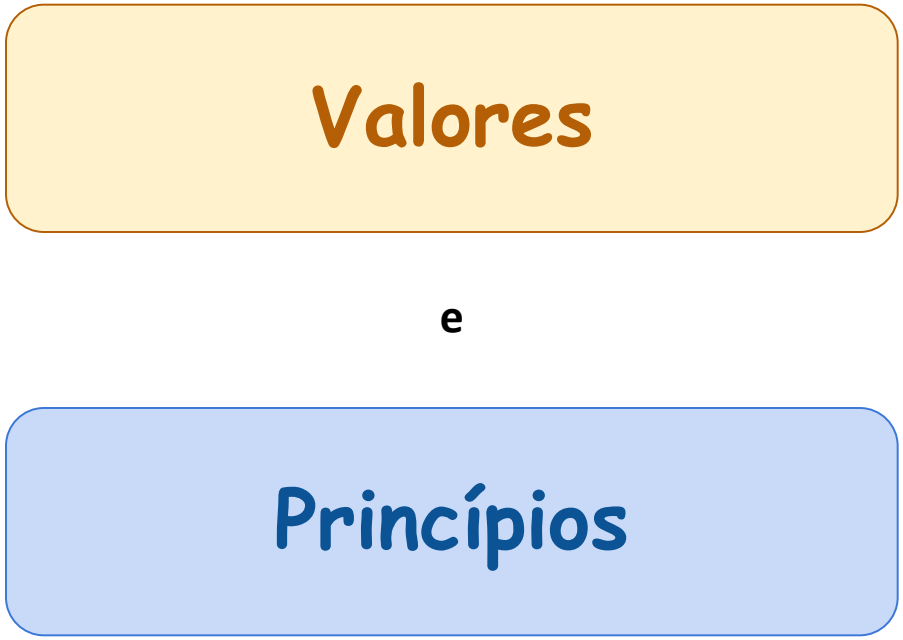
Manifesto Ágil

- Documento assinado em 2001 por **17** pesquisadores da área;
- **http://www.manifestoagil.com.br**
- Pesquisadores:

Kent Beck	Ward Cunningham	Andrew Hunt	Robert C. Martin
Mike Beedle	**Martin Fowler**	Ron Jeffries	Steve Mellor
Arie van Bennekum	James Grenning	Jon Kern	Ken Schwaber
Alistair Cockburn	**Jim Highsmith**	Brian Marick	Jeff Sutherland
Dave Thomas			

Manifesto Ágil

- Documento formado por:

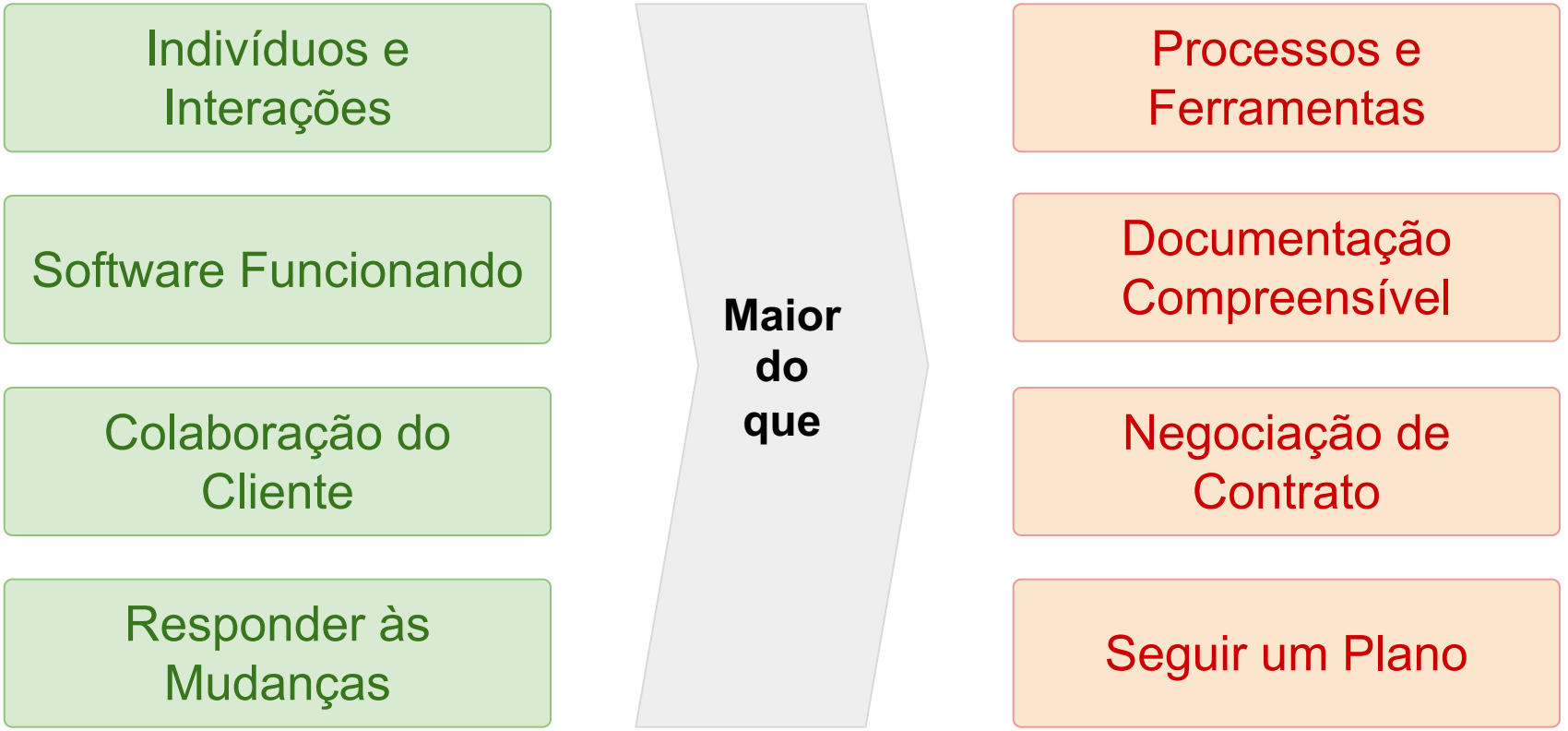


Manifesto Ágil

“Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo”.

Manifesto Ágil

- Valores



"Embora os itens à direita sejam importantes, valorizamos mais os que estão na esquerda"

Manifesto Ágil

- Princípios

Nossa maior prioridade é **satisfazer o cliente**, através da entrega adiantada e contínua de software de valor

Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.

Entregar software funcionando com freqüencia, na escala de semanas até meses, com preferência aos períodos mais curtos.

Pessoas relacionadas a negócios e desenvolvedores devem **trabalhar em conjunto** e diariamente, durante todo o curso do projeto.

Manifesto Ágil

- Princípios

Construir projetos ao redor de **indivíduos motivados**. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.

O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma **conversa cara a cara**.

Software funcional é a medida primária de progresso.

Processos ágeis promovem um **ambiente sustentável**. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.

Manifesto Ágil

- Princípios

Contínua atenção à **excelência técnica** e bom design, aumenta a agilidade.

Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.

As melhores arquiteturas, requisitos e designs emergem de times **auto-organizáveis**.

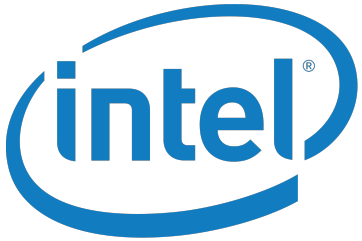
Em intervalos regulares, o **time reflete** em como ficar mais efetivo, então, se **ajusta** e otimiza seu comportamento de acordo.

Métodos Ágeis

- Modelos de desenvolvimento de software considerados ágeis:
  - Feature-Driven Development (FDD)
  - Dynamic Systems Development Method (DSDM)
  - Scrum
  - XP
  - Crystal Clear
  - Adaptative Software Development (ASD)

Métodos Ágeis

- Empresas que usam:



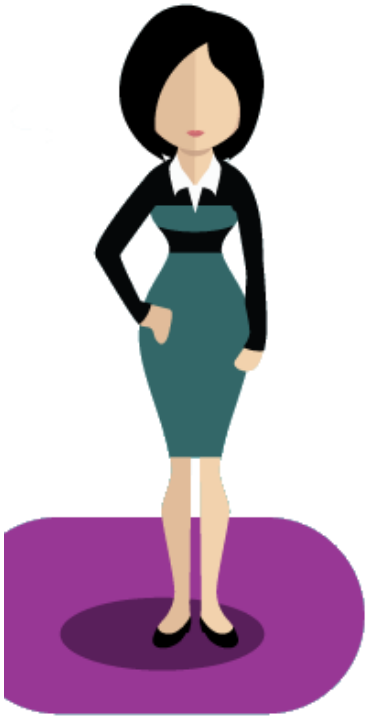

Scrum

Scrum

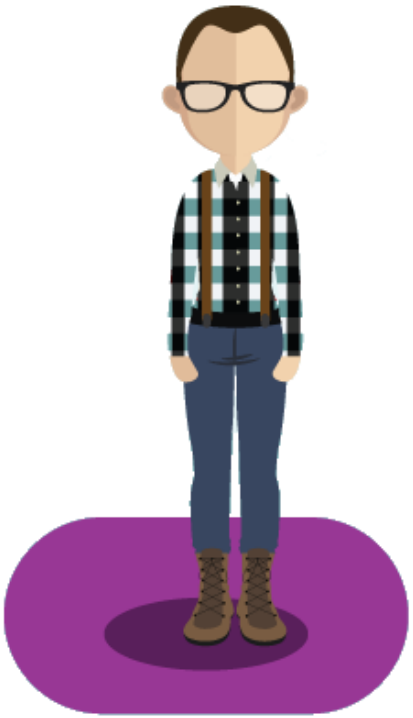
- Modelo ágil de gestão de projetos;
- Conceito mais importante chama-se **sprint** (ou ciclo);
- Origem na indústria automobilítica;
- Livro de Schwaber e Beedle (2001) explica de forma completa e sistemática;

Perfis Importantes no Scrum

Product Owner



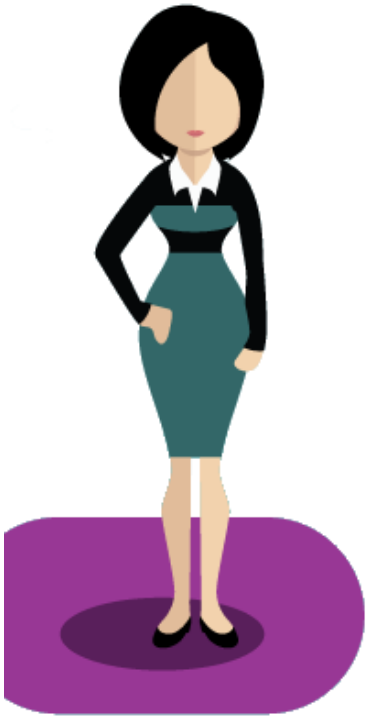
Scrum Master



Scrum Team

Perfis Importantes no Scrum

Product Owner

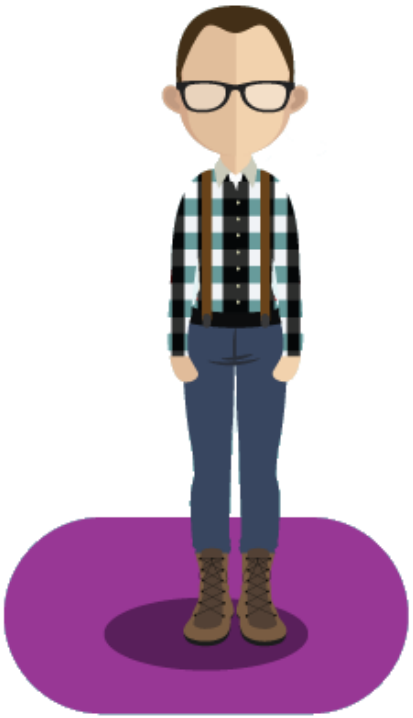


- Responsável pelo projeto em si;
- Indicar quais requisitos são os mais importantes em cada ciclo;
- Responsável por conhecer e avaliar as necessidades do cliente;

Perfis Importantes no Scrum

- Não é gerente;
- Não é lider.

Scrum Master



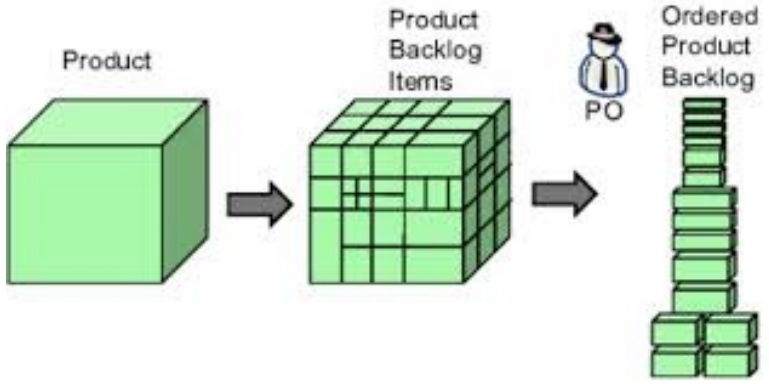
- É um facilitador;
- Conhece bem o modelo;
- Solucionador de conflitos;

Perfis Importantes no Scrum

- Equipe de desenvolvimento;
- Não necessariamente dividida em papéis (analista, designer…);
- Todos interagem para desenvolver o produto em conjunto;
- Recomendado equipes de 6 a 10 pessoas.

Product Backlog

- Lista contendo as funcionalidades a serem implementadas em cada projeto (requisitos ou histórias de usuário);
- Não precisa ser completo (do Manifesto Ágil, **adaptação** em vez de planejamento);
- Não precisa fazer um levantamento inicial excessivamente superficial;
- Tentar obter do cliente o maior número possível de informações sobre suas necessidades



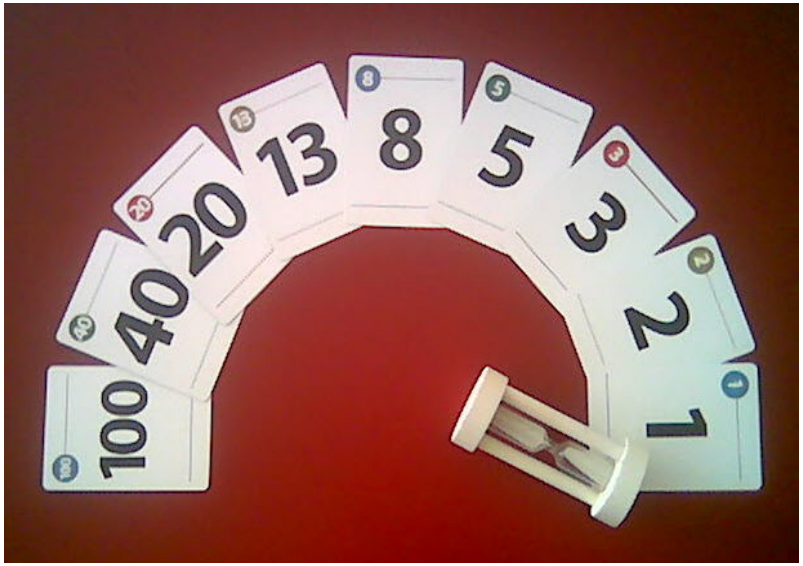
Product Backlog

Exemplo

ID	Nome	Imp	PH	Como demonstrar	Notas
1	Depósito	30	5	Logar, abrir página de depósito, depositar R$ 10,00, ir para a página de saldo e verificar que ele aumentou em R$ 10,00	Precisa de um diagrama de sequência UML.
2	Ver extrato	10	8	Logar, clicar em “Transações”. Fazer um depósito. Voltar para “Transações”, ver que o depósito aparecue.	Usar paginação para evitar consultas grandes ao BD.

- **Imp:** Importância da história de usuário;
- **PH:** Estimativa de esforço necessário para transformar a história em software; Valor dado em Pontos de História;
- **Como demonstrar:** considerar a história efetivamente implementada.

Planning Poker



- Definido pela primeira vez por James Grenning em 2002;
- Obtém estimativas por meio de um jogo de cartas;
- Realizadas rodadas para obter a estimativa de um cartão que possui uma estória ou tarefa a desenvolver;
- PO é responsável por tirar todas as possíveis dúvidas evitando assim o retrabalho.

Sprint

- Ciclo de desenvolvimento de poucas semanas de duração (2 a 4 semanas);
- No início é feito um **sprint planning meeting**
  - Prioriza os elementos do product backlog e transfere para o sprint backlog.
- Equipe se compromete em desenvolver as atividades do sprint backlog;
- Product Owner se compromete a **não trazer** novas funcionalidades durante o mesmo sprint;

Product Backlog
Requisitos em alto nível e voltado as necessidades do cliente

Sprint Backlog
Visão dos requisitos voltada a maneira como a equipe vai desenvolvê-los

Quadro de Andamento de Atividades

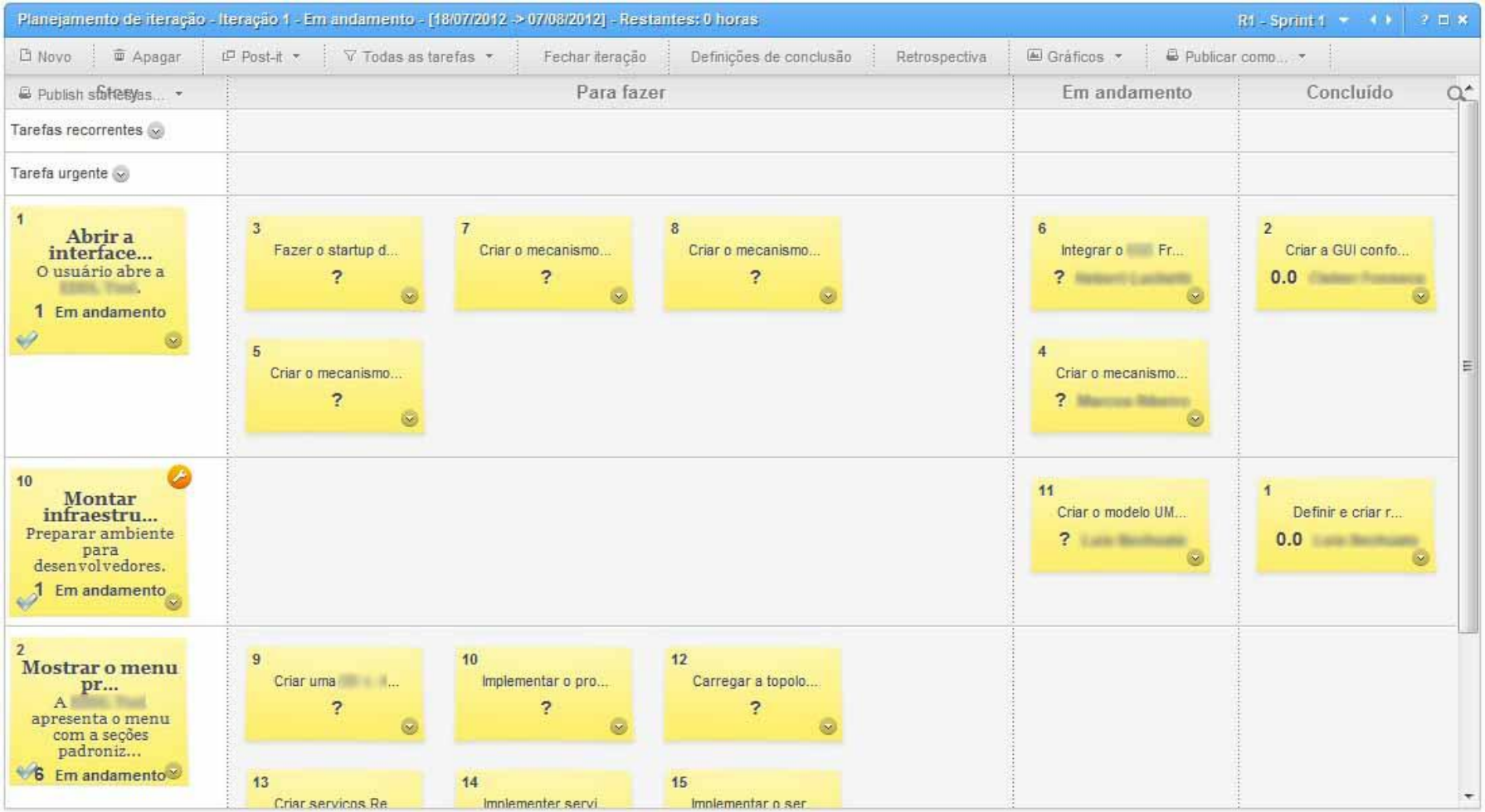


Diagrama Sprint Burndown

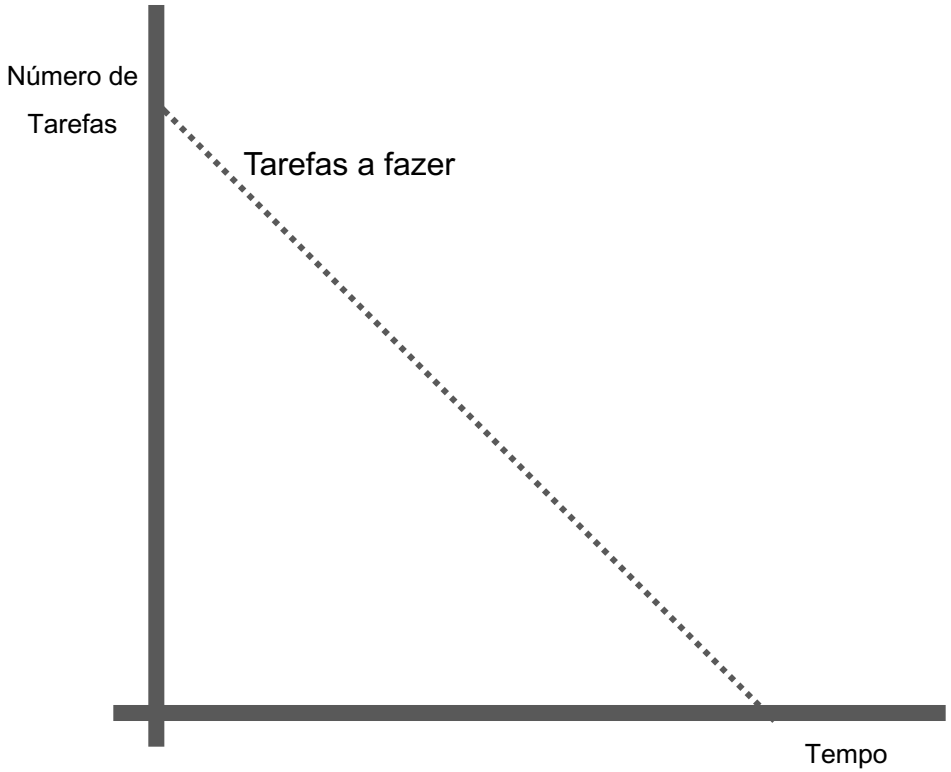
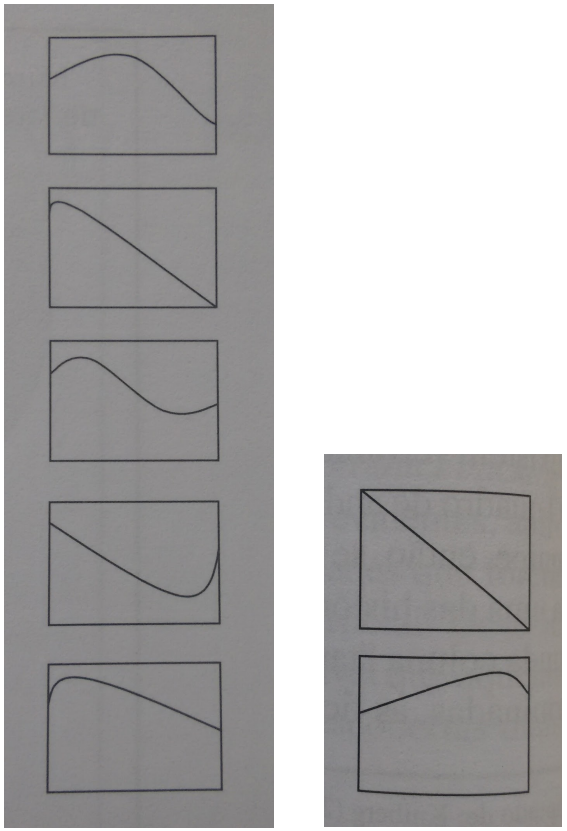
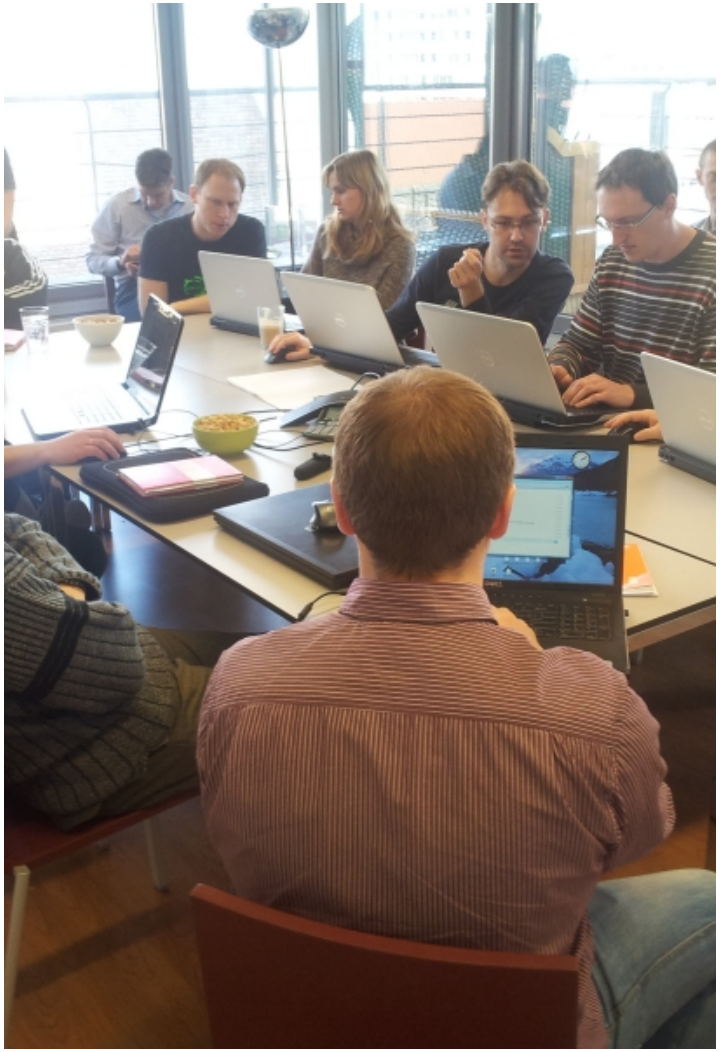


Diagrama Ideal



Sprint

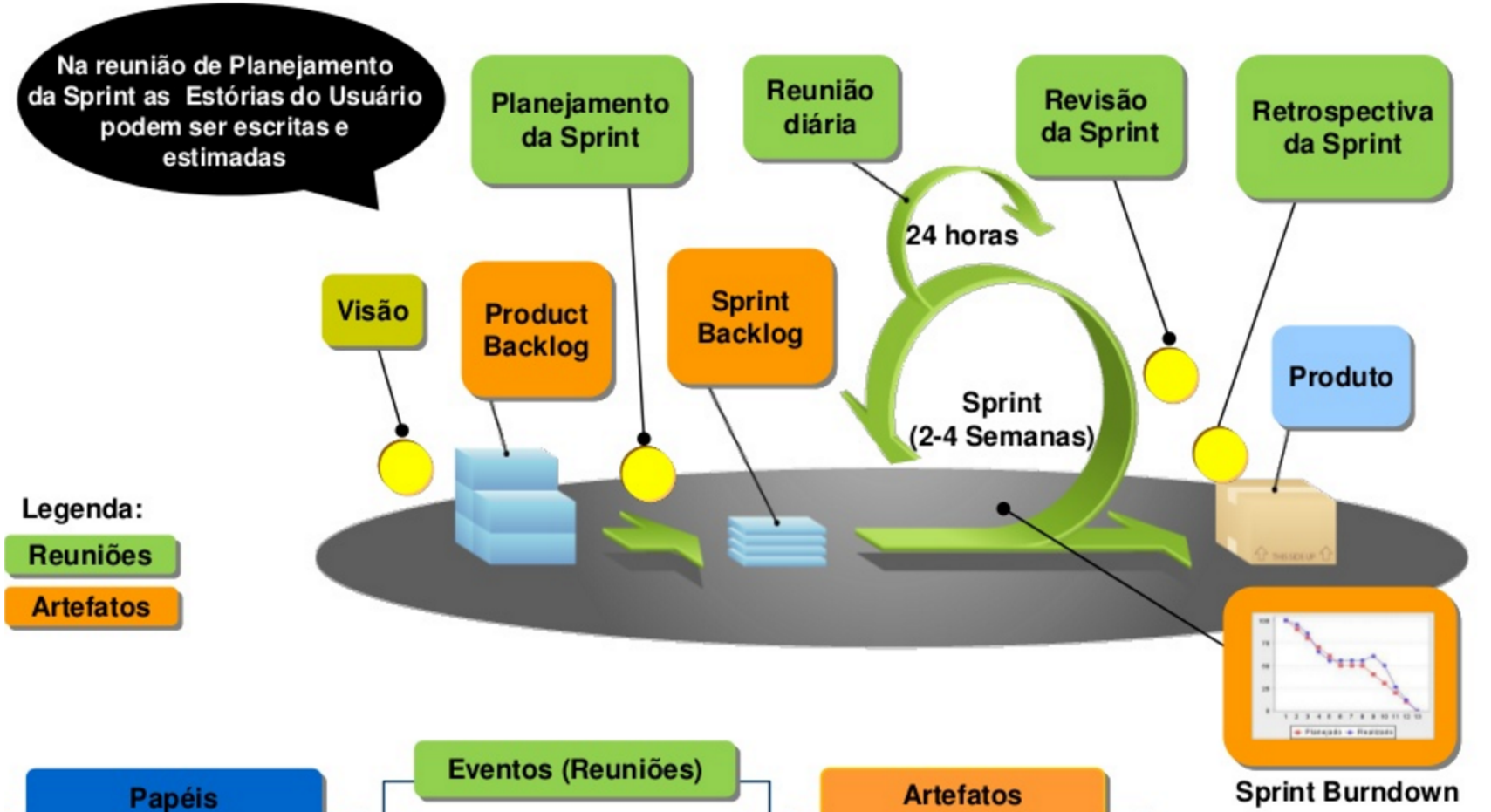
- Final da sprint, equipe deve realizar:
  - Sprint Review Meeting
  - Sprint Retrospective
- Sprint Review Meeting
  - Verificar o que foi feito e, então, partir para uma nova sprint
- Sprint Retrospective
  - Avaliar a equipe e os processos (impedimentos, problemas, dificuldades, ideias novas…)



Daily Scrum

- Modelo sugere reuniões diárias chamada **Daily Scrum**;
- Objetivo:
  - Falar o que fez no dia anterior;
  - O que vai fazer no dia seguinte;
  - O que impede de prosseguir.
- Reuniões rápidas e em **pé** em frente ao quadro de anotações;
- Boa maneira de dissipar o cansaço.

Visão Geral do Scrum




XP
Extreme Programming

Extreme Programming

- Também conhecido como XP;
- Surgiu nos Estados Unidos no final da década de 1990;
- Inicialmente adequada a equipes pequenas e médias;
- Codificação é a principal tarefa;
- Baseada em diversos valores, princípios e regras;
- Principais valores do XP:
  - Simplicidade;
  - Respeito;
  - Comunicação;
  - Feedback;
  - Coragem.

Simplicidade

- Concentrar nas atividades efetivamente necessárias e não naquelas que poderiam ser;
- Assuma a solução **mais simples** como a melhor;
- Use as tecnologias, algoritmos e técnicas mais simples que permitirão atender aos requisitos do usuário-final;
- **Design**, **processo** e **código** podem ser simplificados a qualquer momento.

Respeito

- Respeito entre os membros da equipe, assim como entre a equipe e o cliente;

Comunicação

- XP prioriza comunicação de boa qualidade preferindo encontros presenciais. Quanto mais **pessoal** e **expressiva**, melhor;
- Encontro presenciais > videoconferências > telefonemas > e-mails;
- Dê preferência a comunicação mais ágil.

Feedback

- Buscar obter feedback o quanto antes para evitar eventuais **falhas** de comunicação e aumento do **custo** da correção;
- Cliente sabe se o produto que está sendo desenvolvido **atende** às suas necessidades;

Coragem

- Coragem de abraçar as **inevitáveis** mudanças em vez de simplesmente ignorá-las por estarem fora do contrato formal ou por serem difíceis de acomodar;
- Testes, integração contínua, programação em pares e outras práticas de XP **aumentam** a confiança do programador e ajudam-no a ter coragem para:
  - Melhorar o código que está funcionando;
  - Investir tempo no desenvolvimento de testes;
  - Pedir ajuda aos que sabem mais.

Princípios Básicos do XP

- A partir do valores, os princípios básicos do XP são definidos:
  - Feedback Rápido;
  - Presumir Simplicidade;
  - Mudanças Incrementais;
  - Abraçar Mudanças;
  - Trabalho de Alta Qualidade.
- Priorização das funcionalidades mais importantes.

Princípios Básicos do XP

- **Feedback Rápido**
  - Modele um pouco, mostre ao cliente e então modele novamente
- **Presumir Simplicidade**
  - Deixe o modelo tão simples quanto possível;
- **Mudanças Incrementais**
  - Os problemas devem ser solucionados com um conjunto de pequenas modificações
- **Abraçar Mudanças**
  - Aceite as mudanças e tenha coragem para reconstruir
- **Trabalho de Alta Qualidade**
  - A qualidade do trabalho nunca deve ser comprometida. Importância da codificação e dos testes antes da programação

Atividades do XP

- Escutar
- Testar
- Codificar
- Projetar

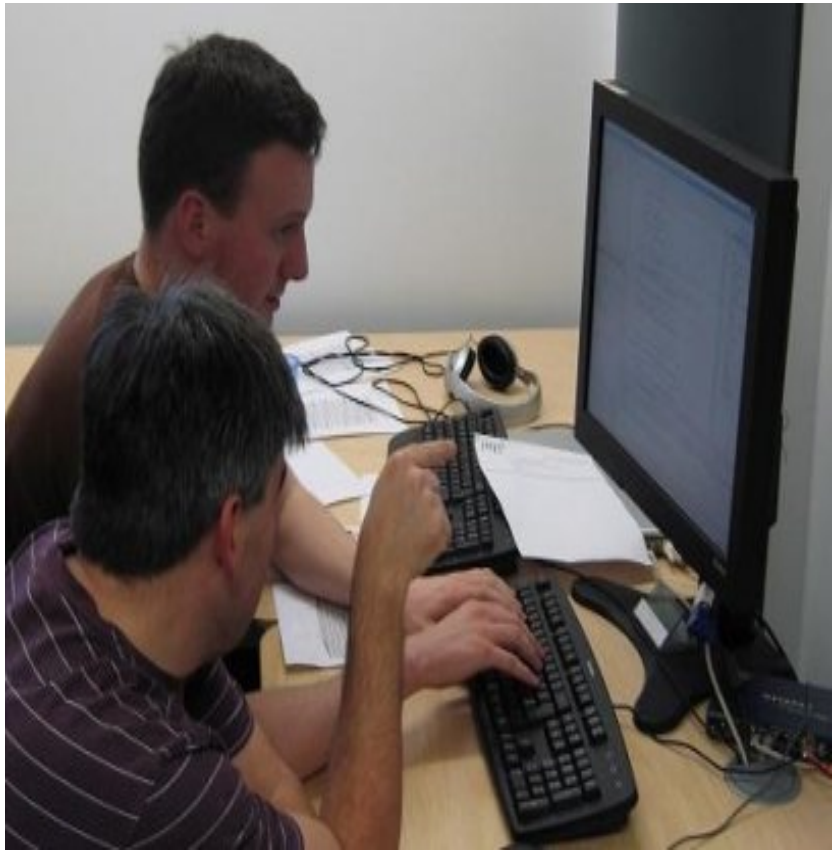
Práticas XP

- Jogo de Planejamento;
- Metáfora;
- Equipe Coesa;
- Reuniões em Pé;
- Design Simples;
- Versões Pequenas;
- Ritmo Sustentável;
- Posse Coletiva;
- Programação em Pares;
- Padrões de Codificação;
- Testes de Aceitação;
- Desenvolvimento orientado a testes (TDD);
- Refatoração;
- Integração Contínua.

* As práticas do XP não são consenso entre os desenvolvedores;

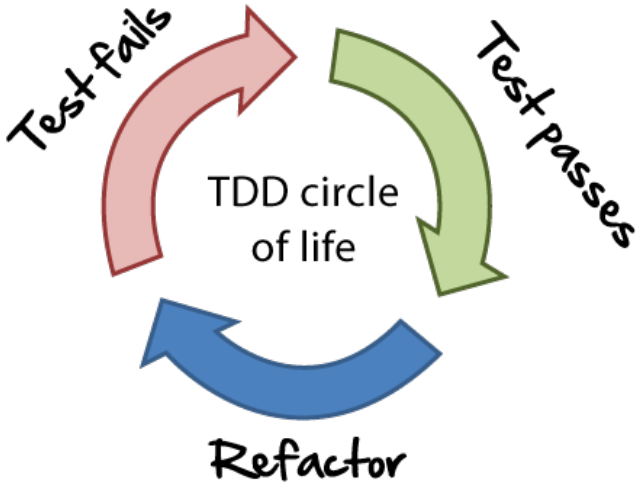
Programação em Pares

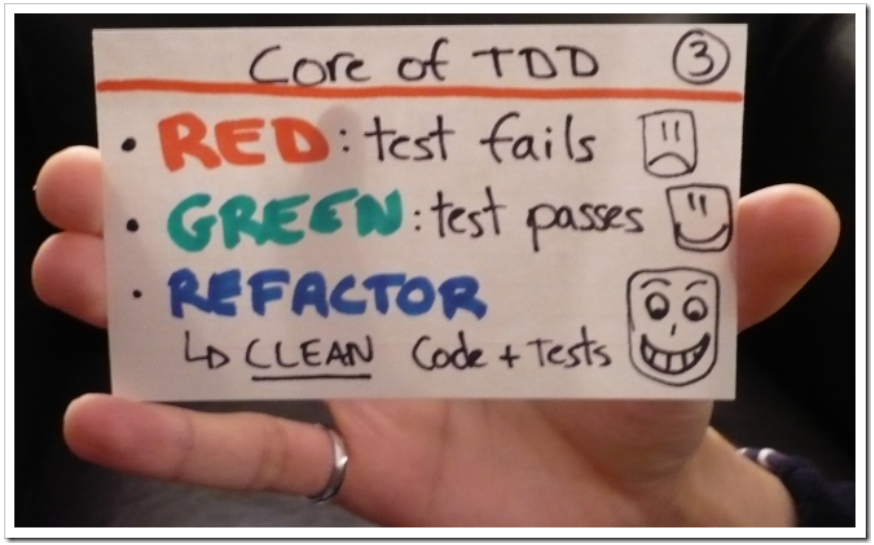
- Todo o desenvolvimento em XP é feito em pares
  - Um computador, um teclado, dois programadores
  - Um piloto, um co-piloto
  - Papéis são alternados frequentemente
  - Pares são trocados periodicamente
- Benefícios
  - Melhor qualidade do design, código e testes
  - Revisão constante do código
  - Nivelamento da equipe
  - Maior comunicação



TDD

- Desenvolvimento orientado a Testes;
- "*Test first, then code*";
- 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%;
- Testes são a documentação executável do sistema;

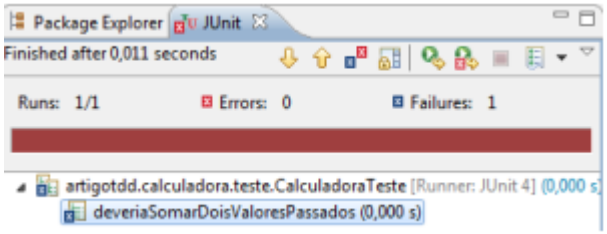




TDD

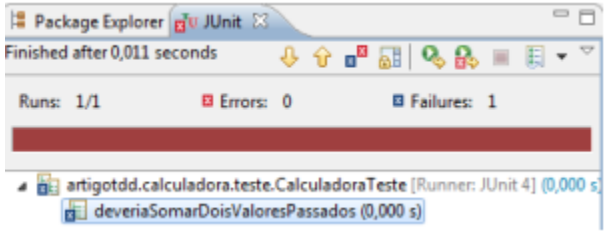
```
public class CalculadoraTeste {
	@Test
	public void deveriaSomarDoisValoresPassados() throws Exception {
		int valorA = 1;
		int valorB = 2;
		Calculadora calculadora = new Calculadora();
		int soma = calculadora.soma(valorA, valorB);

		assertEquals(3, soma);
	}
}
```



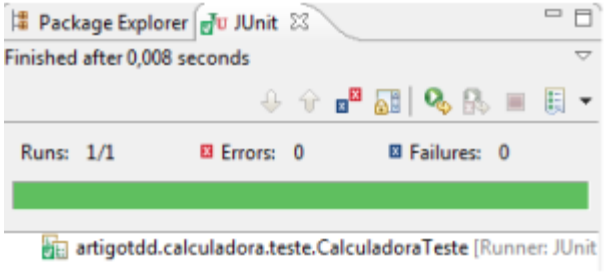

```
public class Calculadora {

	public int soma(int valorA, int valorB) {
		return 0;
	}
}
```



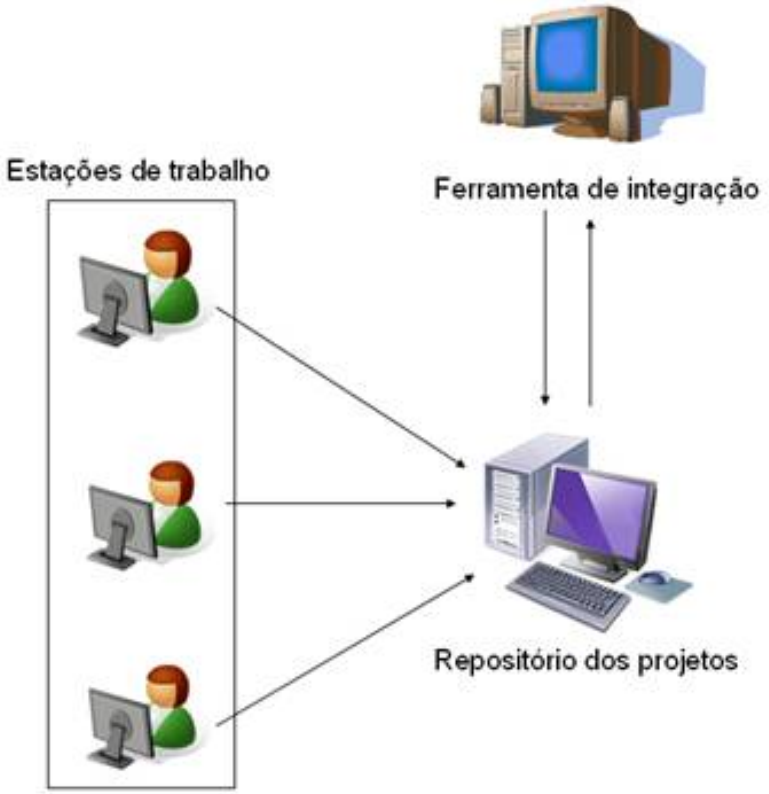

```
public class Calculadora {

	public int soma(int valorA, int valorB) {
		return valorA + valorB;
	}
}
```



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
- Benefícios:
  - Expõe o estado atual do desenvolvimento;
  - Oferece *feedback* sobre todo o sistema;
  - Permite encontrar problemas de design;



Dificuldades

- Vencer barreiras culturais;
- Deixar alguém mexer no seu código;
- Trabalhar em pares e ter coragem de admitir que não sabe;
- Vencer hábitos antigos:
  - Manter as coisas simples;
  - Jogar fora código desnecessário;
  - Escrever testes antes de codificar;
  - Refatoração com frequência (vencer o medo).

Conclusão

- Para implementar XP não é preciso usar diagramas ou processos formais;
- A maior parte das práticas do XP devem ser seguidas
- Não é a bala de prata

Quando não usar Métodos Ágeis

- Equipes grandes e espalhadas geograficamente:
  - Comunicação é um valor fundamental de XP;
  - Não é fácil garantir o nível de comunicação requerido em projetos XP em grandes equipes.
- Situações onde o feedback é demorado:
  - Testes muito difíceis, arriscados e que levam tempo;
  - Programadores espalhados em ambientes físicos distantes e sem comunicação eficiente.
- Sistemas que precisam passar por regulamentação;