*Universidade Federal do Espírito Santo*

*Centro Tecnológico*

*Departamento de Informática*

*Programa de Pós-Graduação em Informática*




# *Disciplina:* ***INF6008/7008 – Engenharia de Software***

*Prof.:* ***Monalessa Perini Barcellos***

*(monalessa@inf.ufes.br)*

1

# Conteúdo

**4. Métodos Ágeis**

4.1 Introdução

4.2 XP

4.3 Scrum



2

# 4.1 Introdução

- Os **métodos ágeis de desenvolvimento de software** tiveram seu início nos anos 80, como parte de um movimento contra os métodos burocráticos de desenvolvimento.
- **Objetivo:** superar os problemas presentes nas abordagens de desenvolvimento de software ditas 'tradicionais'.
- Por exemplo, o modelo cascata é considerado, pelos precursores dos métodos ágeis, lento e contraditório à maneira que engenheiros de software trabalham.



# 4.1 Introdução

**O Manifesto Ágil**

- Um grupo de profissionais veteranos na área de software decidiram se reunir em uma estação de esqui, nos EUA, para discutir formas de melhorar o desempenho de seus projetos.
- Embora cada envolvido tivesse suas próprias práticas e teorias sobre como fazer um projeto de software ter sucesso, cada qual com as suas particularidades, todos concordavam que, em suas experiências prévias, um pequeno conjunto de princípios sempre parecia ter sido respeitado quando os projetos davam certo.
- Com base nisso, em 2001, eles criaram o **Manifesto para o Desenvolvimento Ágil de Software**, frequentemente chamado apenas de **Manifesto Ágil**, e o termo **Desenvolvimento Ágil** (ou **Metodologia Ágil**) passou a descrever abordagens de desenvolvimento que seguem esses princípios.

# 4.1 Introdução

## O Manifesto Ágil

"Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar:

**Indivíduos e interação entre eles** mais que **processos e ferramentas**

**Software em funcionamento** mais que **documentação abrangente**

**Colaboração com o cliente** mais que **negociação de contratos**

**Responder a mudanças** mais que **seguir um plano**

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda."

*(http://manifestoagil.com.br/)*



# 4.1 Introdução

## O Manifesto Ágil

"Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar:

**Indivíduos e interação entre eles** mais que **processos e ferramentas**

**Software em funcionamento** mais que **documentação abrangente**

**Colaboração com o cliente** mais que **negociação de contratos**

**Responder a mudanças** mais que **seguir um plano**

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda."

*(http://manifestoagil.com.br/)*

Importante: **O Manifesto Ágil não prega a Anarquia!!!!!!!**

# 4.1 Introdução

O Manifesto Ágil estabelece 12 **princípios**:

1.“ Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
2.Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
3.Entregar software funcionando com frequencia, na escala de semanas até meses, com preferência aos períodos mais curtos.
4.Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
5.Construir projetos ao redor de indivíduos motivados, dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
6.O método mais eficiente e eficaz de transmitir informações para, e dentro de uma equipe de desenvolvimento, é através de uma conversa cara a cara.
7.Software funcionando é a medida primária de progresso.
8.Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente passos constantes.
9.Contínua atenção à excelência técnica e bom design aumentam a agilidade.
10.Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
11.As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
12.Em intervalos regulares, a equipe reflete em como ser mais efetiva, ajustando e otimizando seu comportamento adequadamente.”



7

# 4.1 Introdução

- O Manifesto Ágil está mais diretamente relacionado ao desenvolvimento de software.
- Outro grupo de profissionais se manifestou em 2005 em prol da agilidade no gerenciamento de projetos: **Agile Project Management “Declaration of Interdependence”**



8

# 4.1 Introdução

**Declaration of Interdependence (DOI)**

"Somos uma comunidade de líderes de projetos altamente bem sucedidos em entregar resultados. Para alcançar esses resultados:

Nós **entregamos resultados confiáveis** engajando clientes em interações frequentes e propriedade compartilhada.

Nós **esperamos incerteza** e a gerenciamos através de iterações, antecipação e adaptação.

Nós **desencadeamos criatividade e inovação** reconhecendo que indivíduos são fonte de valor irrevogável e criando um ambiente onde eles podem fazer a diferença.

Nós **impulsionamos desempenho** através de responsabilidade de grupo pelos resultados e responsabilidade compartilhada pela eficácia da equipe.

Nós **melhoramos eficácia e confiabilidade** através de estratégias, processos e práticas específicas para as situações.

*(http://pmdoi.org/)*



# 4.1 Introdução

## Para pensar…

**Os princípios estabelecidos no Manifesto Ágil são antagônicos aos princípios da Engenharia de Software dita tradicional?**

**Os princípios estabelecidos no Manifesto Ágil não vão ao encontro (ou até mesmo dependem) de princípios da Engenharia de Software dita tradicional?**

**Os Métodos Ágeis não são métodos da Engenharia de Software?**

# 4.1 Introdução

**Então:**

**Qualquer postura radical, seja em favor dos Métodos Ágeis ou da Engenharia de Software tradicional, ofusca o bom senso necessário para usar de forma equilibrada as melhores práticas para o desenvolvimento de software.**



11

# 4.1 Introdução

**Abordagem Tradicional** x **Abordagem Ágil**

- Influenciada pelo processo de fabricação industrial (etapas bem definidas, como o desenvolvimento de um carro)
- Calcada em características como: linearidade, determinismo, padronização e foco na execução.
- O custo para corrigir um erro aumenta cerca de dez vezes de uma etapa para outra.

- Influenciada pelo processo de produção intelectual (não se sabe muito bem o que fazer até que se faça e não há método sistemático específico – p.ex., escrever uma redação ou pintar um quadro)
- Calcada em características como: não linearidade, necessidade de revisões e ausência de determinismo.
- O custo para corrigir um erro é praticamente o mesmo ao longo de todo o projeto.

**Atenção**: Não há um consenso sobre o que é dito ser desenvolvimento *tradicional*:

- Para alguns, qualquer desenvolvimento que não seja claramente ágil é tradicional.
- Para outros, abordagens iterativas e evolucionistas não são desenvolvimento tradicional, mas sim ágil.



12

# 4.1 Introdução

- Atualmente dois métodos ágeis merecem destaque:

  **XP – *eXtreme Programming*** *(foco na programação)*

  **SCRUM** *(foco no gerenciamento de projetos)*



# 4.2 eXtreme Programming

- Abordagem para desenvolvimento de software que busca assegurar que o cliente receba o máximo de valor a cada dia de trabalho da equipe de desenvolvimento.

- Parte da premissa de que o cliente aprende sobre suas necessidades na medida em que é capaz de manipular o sistema sendo produzido. Com base no uso do sistema, o cliente reavalia suas necessidades e prioridades, gerando mudanças que devem ser incorporadas ao software pela equipe de desenvolvimento.

- XP é voltado para:
  - ✓ Projetos cujos requisitos são vagos e podem mudar com frequência
  - ✓ Desenvolvimento de sistemas orientados a objetos
  - ✓ Equipes pequenas (até 12 desenvolvedores)
  - ✓ Desenvolvimento iterativo/incremental (o sistema começa a ser implementado logo no início e vai ganhando novas funcionalidades ao longo do tempo).

# 4.2 eXtreme Programming

- XP é organizado em torno de um conjunto de **valores** e **práticas** que devem atuar de maneira coesa e harmônica.

- Valores do XP:

  - ✓ **Feedback:** quando o cliente aprende com o sistema que utiliza, ele reavalia as suas necessidades e dá feedback para a equipe de desenvolvimento.
  - ✓ **Comunicação:** a comunicação entre o cliente e a equipe permite que todos os detalhes do projeto sejam tratados com a atenção e agilidade que merecem.
  - ✓ **Simplicidade:** a simplicidade advoga que só deve ser implementado aquilo que é suficiente para atender as necessidades do cliente.
  - ✓ **Coragem:** a equipe precisa ter coragem e acreditar que, usando as práticas e valores do XP, será capaz de fazer o software evoluir com segurança e agilidade.



# 4.2 eXtreme Programming

- Práticas do XP:

**a) Cliente Presente**

O cliente deve conduzir o desenvolvimento a partir do feedback de uso do sistema.

**b) Jogo do Planejamento**

No XP um projeto é dividido em *releases* (módulos do sistema – 2 a 3 meses) e *iterações* (períodos de tempo – 1 a 2 semanas).

Em cada iteração é feito o *jogo do planejamento* para definir o que será feito e em quanto tempo. Todos participam, inclusive cliente.

- Requisitos (funcionalidades) são representados por meio de *histórias*
- As estimativas são feitas em uma unidade própria, chamada *ponto de história* (1 ponto pode ser 1 dia, 1 semana ou outra unidade que a equipe determinar)

# 4.2 eXtreme Programming

**Jogo do Planejamento**

- Antes de cada *release* é feita uma reunião para determinar o que será feito.
- As funcionalidades são apresentadas como histórias e podem ser escritas tanto pela equipe quanto pelo cliente. As histórias são escritas em cartões:

Apresentar para o usuário as dez tarifas mais baratas para uma determinada rota.

- Antes de cada iteração, é feita uma reunião para indicar o que vai ser feito da *release* na iteração corrente.
- As estimativas são feitas (ou revistas, caso tenham sido feitas no planejamento da release):
  - Primeiro, deve-se determinar quanto vale 1 ponto (p. ex.: 1 ponto = 1 dia de trabalho ideal por um par de desenvolvedores).
  - Em seguida estimam-se os números de pontos de cada história e registra-se no canto superior esquerdo do cartão.

●●●
Apresentar para o usuário as dez tarifas mais baratas para uma determinada rota.

  - No canto superior direito são registrados quantos pontos foram, de fato, consumidos na implementação da história.



17

# 4.2 eXtreme Programming

**c) Stand Up Meeting**

Diariamente a equipe se reúne para avaliar o que foi feito no dia anterior e priorizar o que será implementado no dia que se inicia.

**d) Programação em Pares**

O desenvolvimento das funcionalidades é feito em pares de programadores.

O código é permanentemente revisado enquanto é produzido.

**e) Desenvolvimento Orientado a Testes**


Os testes das funcionalidades são definidos antes que elas sejam implementadas.



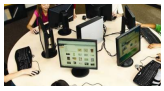
18

# 4.2 eXtreme Programming

**f) Refactoring**


Ato de alterar um código sem afetar a funcionalidade que ele implementa, visando à simplicidade e manutenibilidade.

**g) Código Coletivo**



Em um projeto XP todos os programadores têm acesso a todo o código e podem alterar o que julgarem importante, sem a necessidade de pedir autorização a outra pessoa. A alteração do código pode ser um *refactoring* ou a inclusão de novas partes.

**h) Código Padronizado**

Padrões de codificação são estabelecidos e seguidos.



# 4.2 eXtreme Programming

**i) Design Simples**


O design é o mais simples possível, sem se preocupar com generalizações para necessidades futuras.

**j) Metáfora**

Metáforas são utilizadas para facilitar a comunicação entre equipe de desenvolvimento e cliente.

**k) Ritmo Sustentável**

A equipe deve trabalhar apenas 8 horas por dia e evitar fazer horas extras. A qualidade do código depende da qualidade dos desenvolvedores e da capacidade de se manterem atentos e criativos. Poucas horas de descanso impactam negativamente na qualidade da equipe.

# 4.2 eXtreme Programming

**l) Integração Contínua** 

Os pares devem integrar seu código ao restante do sistema diversas vezes ao dia.

**m) Releases Curtas** 

A equipe produz um conjunto reduzido de funcionalidades e coloca em produção rapidamente para que o cliente possa utilizá-las no seu dia a dia.



# 4.2 eXtreme Programming

- Papéis envolvidos em um projeto XP:

**Gerente:** cuida das questões administrativas e do relacionamento com o cliente. Libera a equipe de questões não pertinentes ao desenvolvimento.

**Coach:** responsável técnico do projeto. Orienta as práticas do XP e assegura o bom funcionamento do processo.

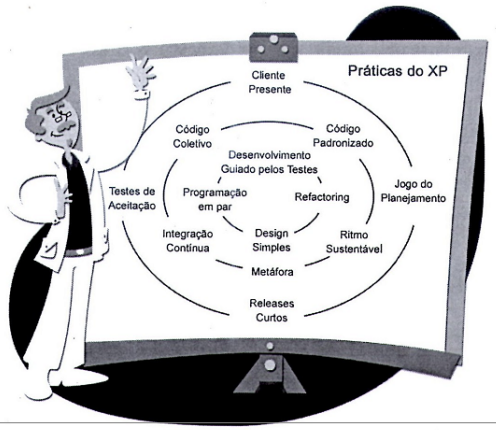
**Analista de Teste:** ajuda o cliente a escrever os testes de aceitação e faz com que sejam executados diversas vezes ao longo do projeto.

**Redator Técnico:** responsável pela documentação do sistema, para que os desenvolvedores se concentrem prioritariamente na implementação.

**Desenvolvedor:** responsável por analisar, projetar e codificar o sistema. Não há divisão entre analista, projetista e programador.

## 4.2 eXtreme Programming

- O XP não possui uma figura padrão que represente o método como um processo (sequência de passos).
- A figura clássica do XP é conhecida como *3 loops* e apresenta suas 13 práticas.






23

## 4.3 SCRUM

- Método ágil para gerenciamento de projetos.
- Criado por Jeff Sutherland, Ken Schwaber, Mike Beedle e Mike Cohn na década de 1990, baseada no desenvolvimento iterativo e incremental e novas estratégias de criação de produtos.
- Sua aplicação não está limitada a projetos de software.
- O nome foi inspirado em uma jogada de *Rugby*. Após uma "reunião" (agrupamento em torno da bola), o objetivo é retirar os obstáculos à frente do jogador que correrá com a bola, para que possa avançar o máximo possível no campo e marcar pontos.



24

# 4.3 SCRUM

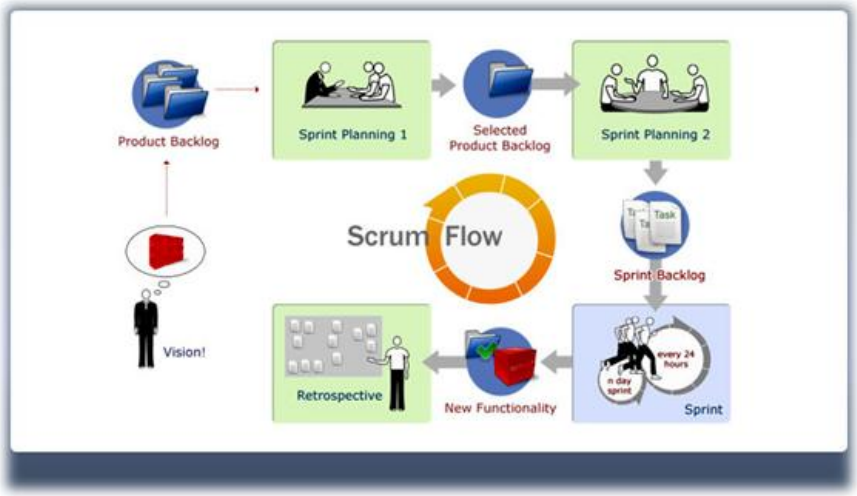
**Como funciona:**

1. Os projetos são divididos em ciclos (tipicamente mensais) chamados de **Sprints**. O Sprint representa uma janela de tempo dentro da qual um conjunto de atividades deve ser executado. Normalmente uma sprint dura de 2 a 4 semanas.
2. As funcionalidades a serem implementadas em um projeto são mantidas em uma lista chamada **Product Backlog.**
3. No início de cada Sprint, faz-se uma reunião de planejamento inicial (**Sprint Planning Meeting)** na qual **Product Owner** prioriza os itens do Product Backlog e a equipe seleciona as atividades que ela será capaz de implementar durante o Sprint que se inicia.
4. As atividades alocadas em um Sprint são transferidas do Product Backlog para o **Sprint Backlog.**
5. A cada dia de um Sprint, a equipe faz uma breve reunião (normalmente de manhã), chamada **Daily Scrum** (também é conhecida como **Stand up Meeting**) para disseminar conhecimento sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho do dia que se inicia.
6. Ao final de um Sprint, a equipe apresenta as funcionalidades implementadas em uma **Sprint Review Meeting.**
7. Finalmente, faz-se uma **Sprint Retrospective** e a equipe parte para o planejamento do próximo Sprint.



# 4.3 SCRUM

Funcionamento do SCRUM



# 4.3 SCRUM

**Papéis**

**Product Owner**

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

**Scrum Master**

- Representa a gerência do 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

**Equipe**

- Entre 5 e 9 pessoas
- Multifuncional (programadores, testadores, desenvolvedores de interfaces etc)
- Dedicação em tempo integral (raras exceções – ex.:DBA)
- Auto-organizável (idealmente, sem títulos ou hierarquia)
- Trocas só na mudança de Sprint.



# 4.3 SCRUM

**Product Backlog**

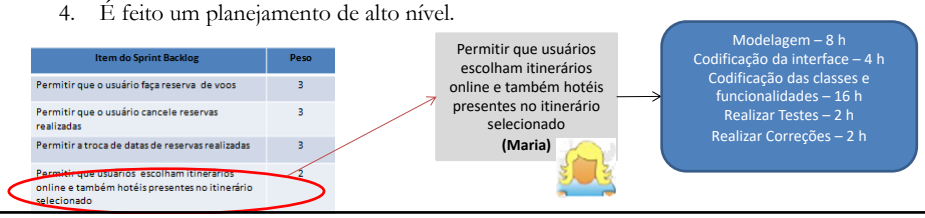
- Contém os requisitos do produto.
- Inclui uma lista de todo o trabalho necessário ao projeto.
- Cada item do Product Backlog tem um peso definido com base na vontade do cliente ou usuários.
- É priorizado pelo Product Owner.
- É repriorizado no início de cada Sprint.

| 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 empregadod do hotel gerem relatórios de lucratividade | 8 |
| ... | 8 |
| ... | 30 |
| ... | 50 |

# 4.3 SCRUM

***Sprint Planning Meeting***

1. ***Product Owner*** prioriza os itens do ***Product Backlog*** .
2. A **equipe** seleciona itens do ***Product Backlog*** com os quais se compromete a concluir durante a ***Sprint***. Cada membro indica o que ficará sob sua responsabilidade.
3. O ***Sprint Backlog*** é criado
    - Tarefas identificadas e estimadas (1 a 16 horas)
    - A criação do ***Sprint Backlog*** é feita de forma colaborativa, não apenas pelo ***ScrumMaster***
4. É feito um planejamento de alto nível.




29

# 4.3 SCRUM

***Daily Scrum (Stand up Meeting)***

- Realizada diariamente
- Todos em pé
- Dura 15 minutos
- Todos são convidados
- São feitas três perguntas (as respostas não são um "relatório" para o ***Scrum Master***, mas sim compromissos perante os pares)
    - a) O que fez para o projeto desde a última reunião?
    - b) O que fará para o projeto até a próxima reunião?
    - c) Há algum obstáculo para conseguir seu objetivo? Precisa de ajuda?
- Apenas os membros da **equipe**, ***Scrum Master*** e ***Product Owner*** podem falar
- Ajuda a evitar reuniões adicionais desnecessárias

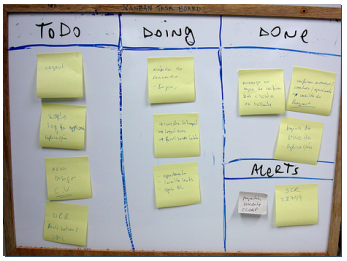


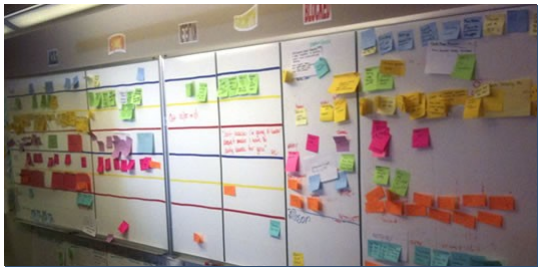
30

# 4.3 SCRUM

***Daily Scrum (Stand up Meeting)***

Uso do ***Kanban***





31

# 4.3 SCRUM

**Gerenciamento do *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 é identificado a partir do *Sprint*
- Se uma atividade não é clara, o *Scrum Master* deve definí-la como um item com uma quantidade maior de tempo e subdividí-la depois.
- Os itens são atualizados na medida em que se tornam mais conhecidos.

| Sprint 6 – 17/09 a 21/09 | | | | | | |
|---|---|---|---|---|---|---|
| **Tarefas** | | **Seg** | **Ter** | **Qua** | **Qui** | **Sex** |
| Implementar classes para registro de reserva | João | 4 | | | | |
| Implementar interface para registro de reserva | João | 4 | | | | |
| Implementar funcionalidade registro de reserva | João | | 8 | | | |
| Implementar classes para reserva de hotel | Pedro | 4 | | | | |
| ... | | ... | ... | ... | ... | ... |

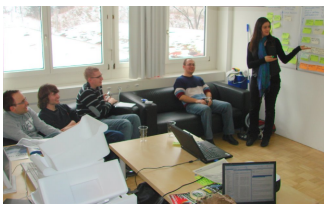


32

# 4.3 SCRUM

***Sprint Review Meeting***

- Equipe apresenta os resultados obtidos durante o *Sprint*.
- Tipicamente, inclui a demonstração de novas funcionalidades ou de sua arquitetura.
- Informal:
  - 2 horas de preparação
  - Sem slides
- Toda a equipe participa.
- Todos (cliente, diretoria etc.) são convidados.

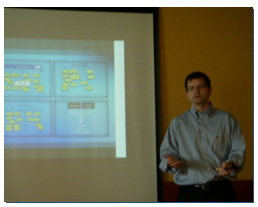




# 4.3 SCRUM

***Sprint Retrospective***

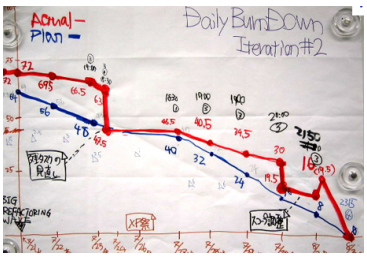
- Realizada para analisar o que deu certo e o que não deu no último *Sprint*.
- Tipicamente, dura de 15 a 30 minutos.
- Toda a equipe participa
- Todos (cliente, diretoria etc.) são convidados

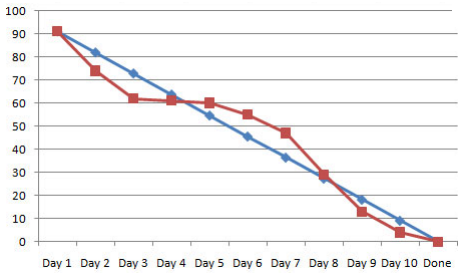


# 4.3 SCRUM

**Gerenciamento do Progresso**

Para acompanhar o progresso e a 'velocidade' da equipe de desenvolvimento, o Scrum utiliza um painel de progresso chamado de **Gráfico de Consumo** (*Burndown Chart*), que ilustra a quantidade de funcionalidades que foram desenvolvidas até o momento no Sprint. A inclinação da curva dá a noção de 'velocidade' da equipe.



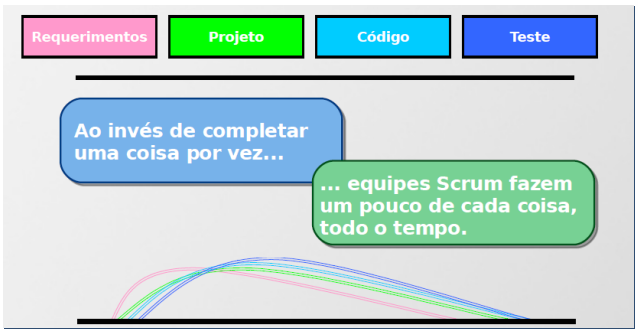








# 4.3 SCRUM

## Para pensar…




**Mas, isso não 'lembra' outros modelos da Engenharia de Software?**

# 4.3 SCRUM

## Para pensar…

**Como seria o uso do SCRUM/XP exatamente como a teoria orienta?**

**Todo tipo de empresa/equipe se adapta facilmente a esse tipo de abordagem?**

**Seria possível combinar práticas do SCRUM /XP com práticas 'tradicionais'? E com os modelos/normas de apoio à definição e melhoria de processo de software?**



## Referências


- MARÇAL, A. S., FREITAS, B. C. C., ET AL., 2007, Estendendo o SCRUM segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI, In: Proceedings of the XXXIII Conferencia Latinoamericana de Informática (CLEI 2007), San Jose, Costa Rica.
- http://manifestoagil.com.br/
- SCHWABER, K., 2004, Agile Project Management with Scrum, Paperback.

*Universidade Federal do Espírito Santo*

*Centro Tecnológico*

*Departamento de Informática*

*Programa de Pós-Graduação em Informática*




*Disciplina:* ***INF6008/7008 – Engenharia de Software***

*Prof.:* ***Monalessa Perini Barcellos***

*(monalessa@inf.ufes.br)*

39