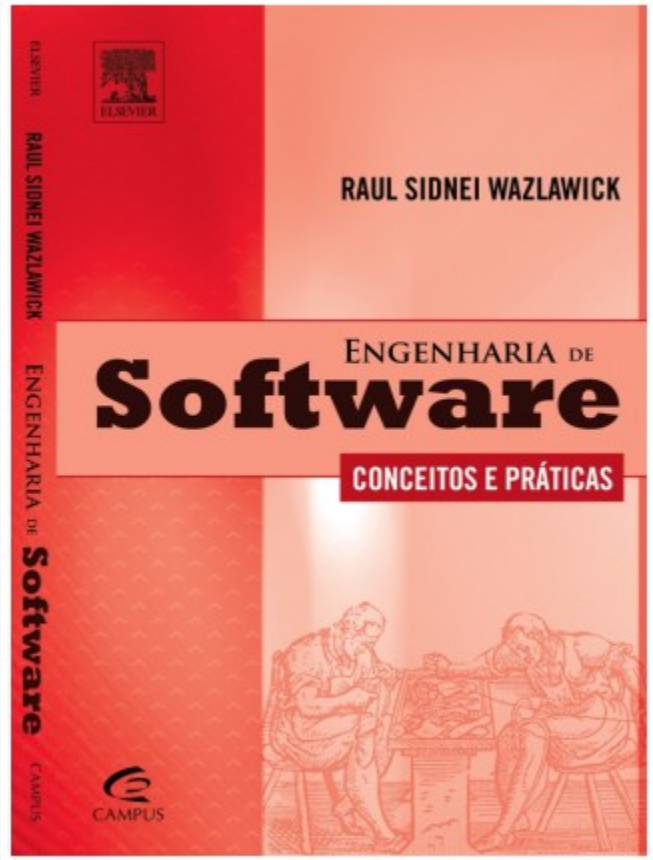

UFV

SIN 221 - Engenharia de Software I

Profa. Liziane Santos Soares

liziane.soares@ufv.br

UFV



MODELOS ÁGEIS

Engenharia de Software: Conceitos e Práticas

Prof. Raul Sidnei Wazlawick

UFSC-CTC-INE

Elsevier, 2013

UFV

Manifesto ágil

Nós estamos descobrindo formas melhores de desenvolver software fazendo e ajudando outros a fazer. Através deste trabalho chegamos aos seguintes valores:

- ***Indivíduos e interações estão acima de processos e ferramentas.***
- ***Software funcionando está acima de documentação compreensível.***
- ***Colaboração do cliente está acima de negociação de contrato.***
- ***Responder às mudanças está acima de seguir um plano.***

Ou seja, enquanto forem valorizados os itens à esquerda, os itens à direita valerão mais.

UFV

Princípios ágeis (1/3)

- **Princípio1:** Nossa maior prioridade é **satisfazer o cliente** através da entrega rápida e contínua de software com valor.
- **Princípio2: Mudanças nos requisitos são bem vindas**, mesmo nas etapas finais do projeto. Processos ágeis usam a mudança como um diferencial competitivo para o cliente.
- **Princípio3: Entregar software frequentemente**, com intervalos que variam de duas semanas a dois meses, preferindo o intervalo mais curto.
- **Princípio4:** Administradores (*business people*) e desenvolvedores devem **trabalhar juntos diariamente** durante o desenvolvimento do projeto.

UFV

Princípios ágeis (2/3)

- **Princípio5:** Construa projetos em torno de **indivíduos motivados**. Dê a eles o ambiente e o suporte e confie que eles farão o trabalho.
- **Princípio6:** O meio mais eficiente e efetivo de tratar a comunicação entre e para a equipe de desenvolvimento é a **conversa face a face.**
- **Princípio7: Software funcionando** é a medida primordial de progresso.
- **Princípio8:** Processos ágeis promovem **desenvolvimento sustentado**. Os financiadores, usuários e desenvolvedores devem ser capazes de manter o ritmo indefinidamente.

UFV

Princípios ágeis (3/3)

- **Princípio9:** Atenção contínua à **excelência técnica** e bom design melhora a agilidade.
- **Princípio10: Simplicidade** – a arte de maximizar a quantidade de trabalho não feito – é essencial.
- **Princípio11:** As melhores arquiteturas, requisitos e projetos emergem de **equipes auto-organizadas**.
- **Princípio12:** Em intervalos regulares a equipe **reflete** sobre como se tornar mais efetiva e então ajusta seu comportamento de acordo.

UFV

SCRUM

UFV

SCRUM

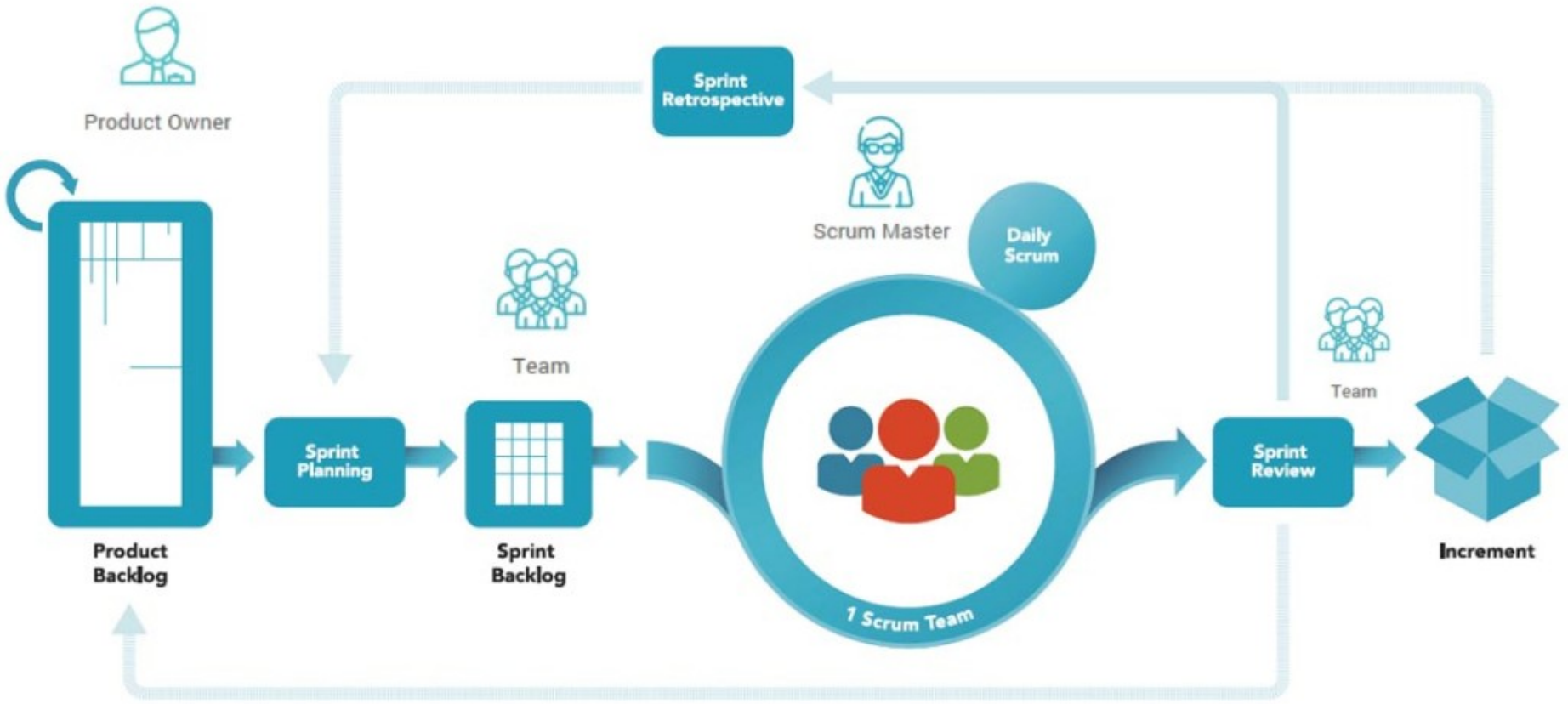
Scrum: *termo emprestado do rugby onde consiste em um método de reinício de jogada, onde os jogadores dos dois times se juntam com a cabeça abaixada e se empurram com o objetivo de ganhar a posse de bola*

SCRUM

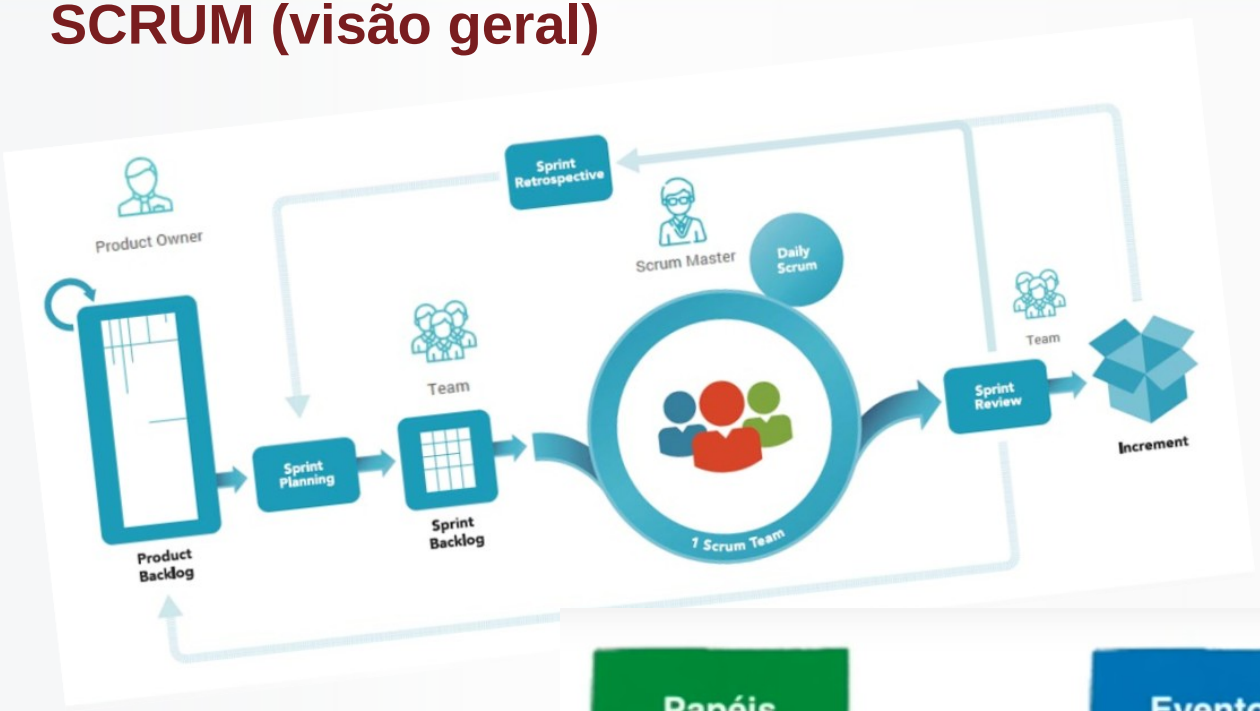
- A concepção inicial do Scrum deu-se na indústria automobilística (Takeuchi & Nonaka, 1986), e o modelo pode ser adaptado a várias outras áreas diferentes da produção de software.
- Modelo ágil, cujo conceito mais importante é o sprint: ciclo de desenvolvimento com duração de duas semanas a um mês
- Denominado por algumas expressões: modelo ágil, processo de software, framework de gerência de projeto

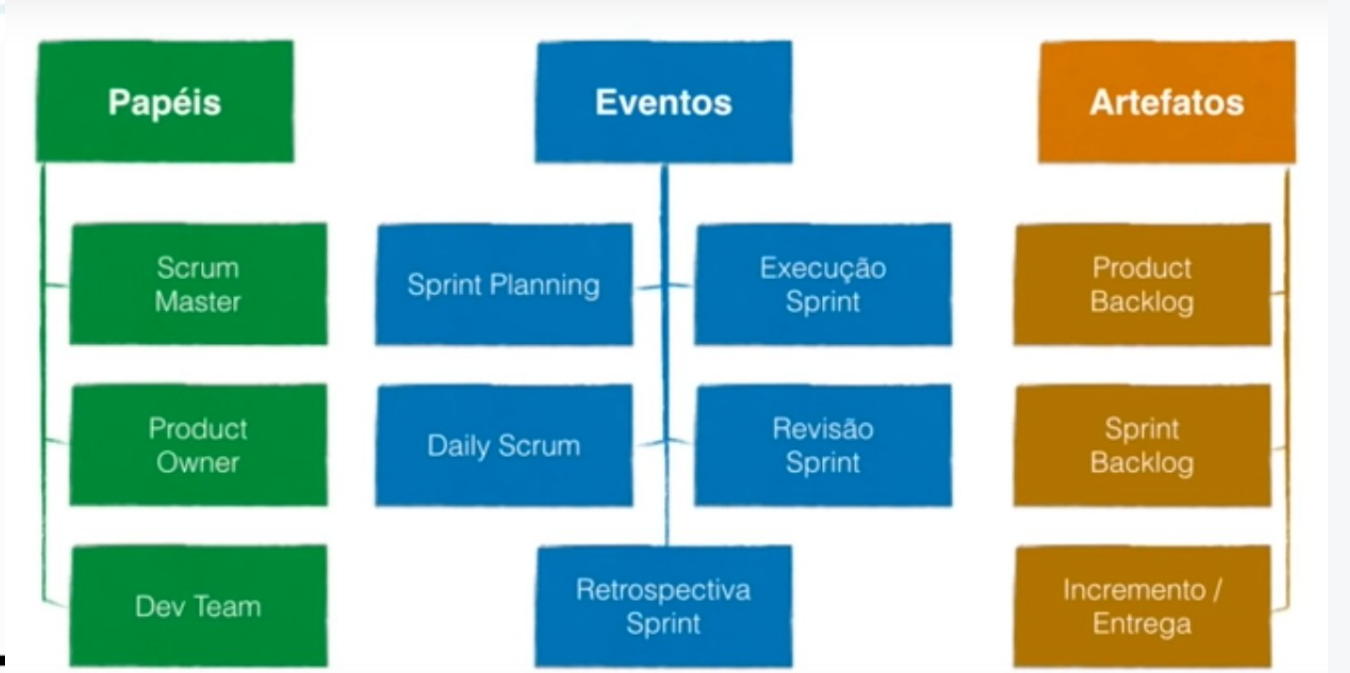
UFV

SCRUM (visão geral)

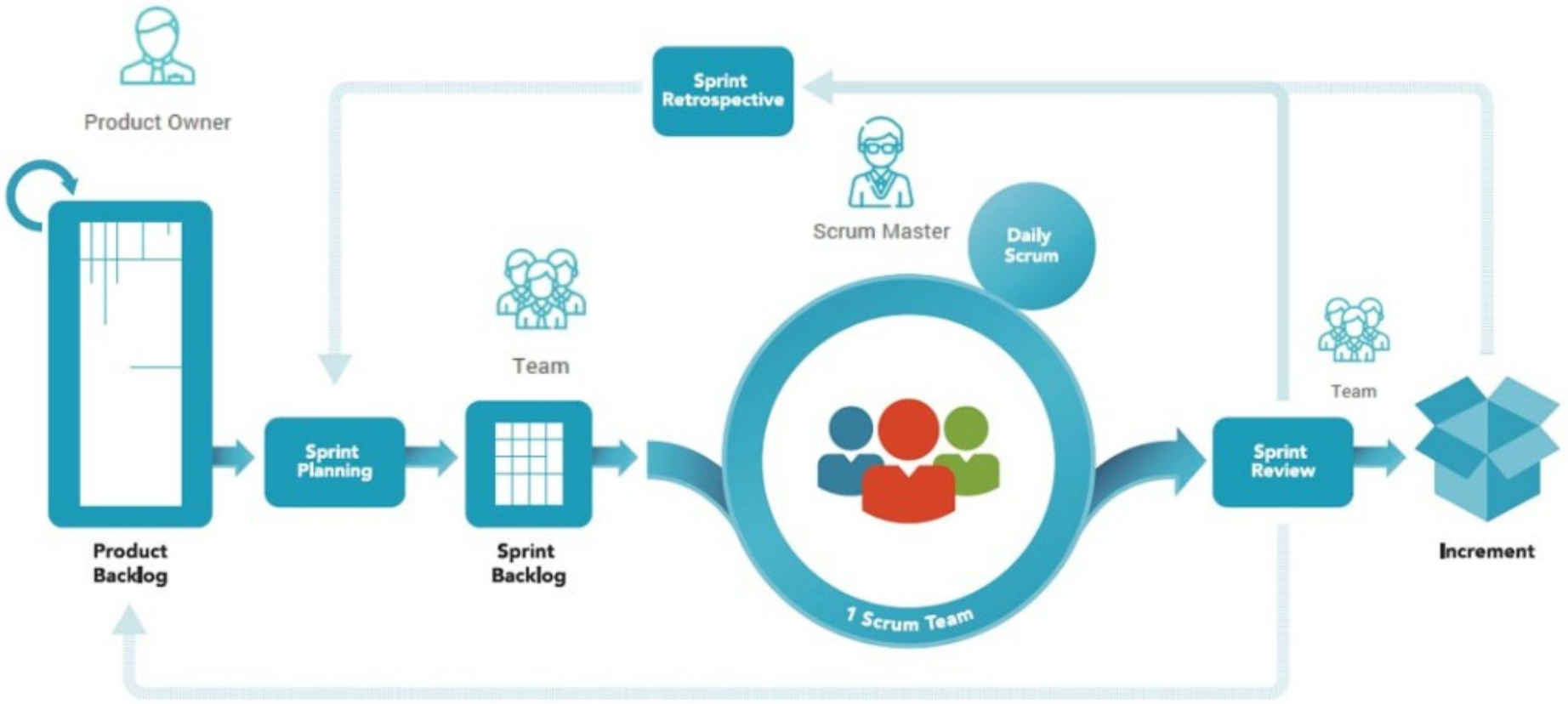


SCRUM (visão geral)



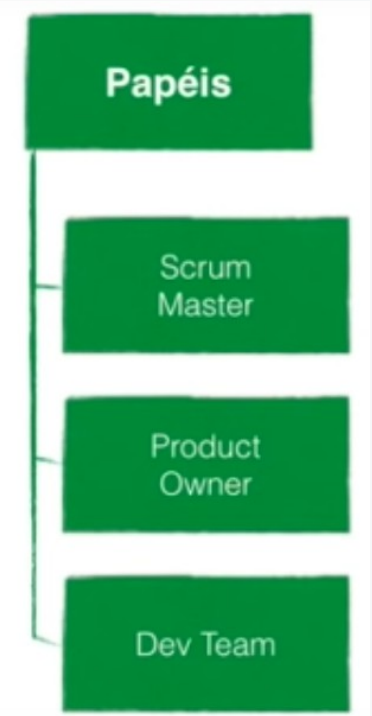


SCRUM (visão geral)



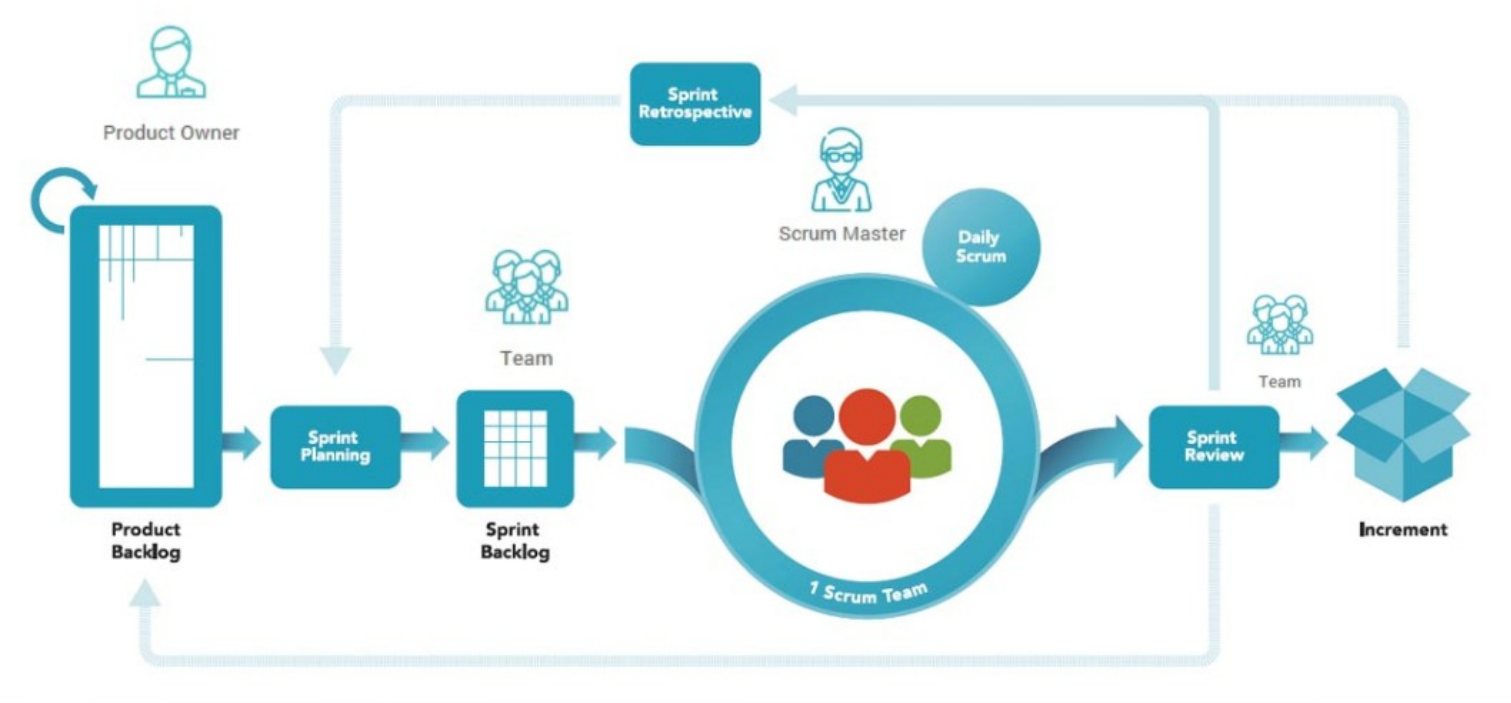
Papéis

- O **product owner** responsável por conhecer e avaliar as necessidades do cliente e deixar claro para a equipe o que está sendo buscado com o projeto. Responsável por decidir que recursos e funcionalidades serão desenvolvidos e em que ordem, priorizando os itens do *Product Backlog*.
- O **Scrum master**, que não é um gerente no sentido dos modelos prescritivos. O *Scrum master* não é um líder, já que as equipes são auto-organizadas, mas um **facilitador** (pessoa que conhece bem o modelo) e resolvedor de conflitos. Responsável por ajudar a equipe a entender e abraçar os valores e princípios do *Scrum*.
- O **Scrum team** (Dev Team) que é a equipe de desenvolvimento. Essa equipe não é necessariamente dividida em papeis como analista, designer e programador, mas todos interagem para se autoorganizar e desenvolver o produto em conjunto, da melhor maneira possível. Usualmente são recomendadas equipes de 6 a 10 pessoas.



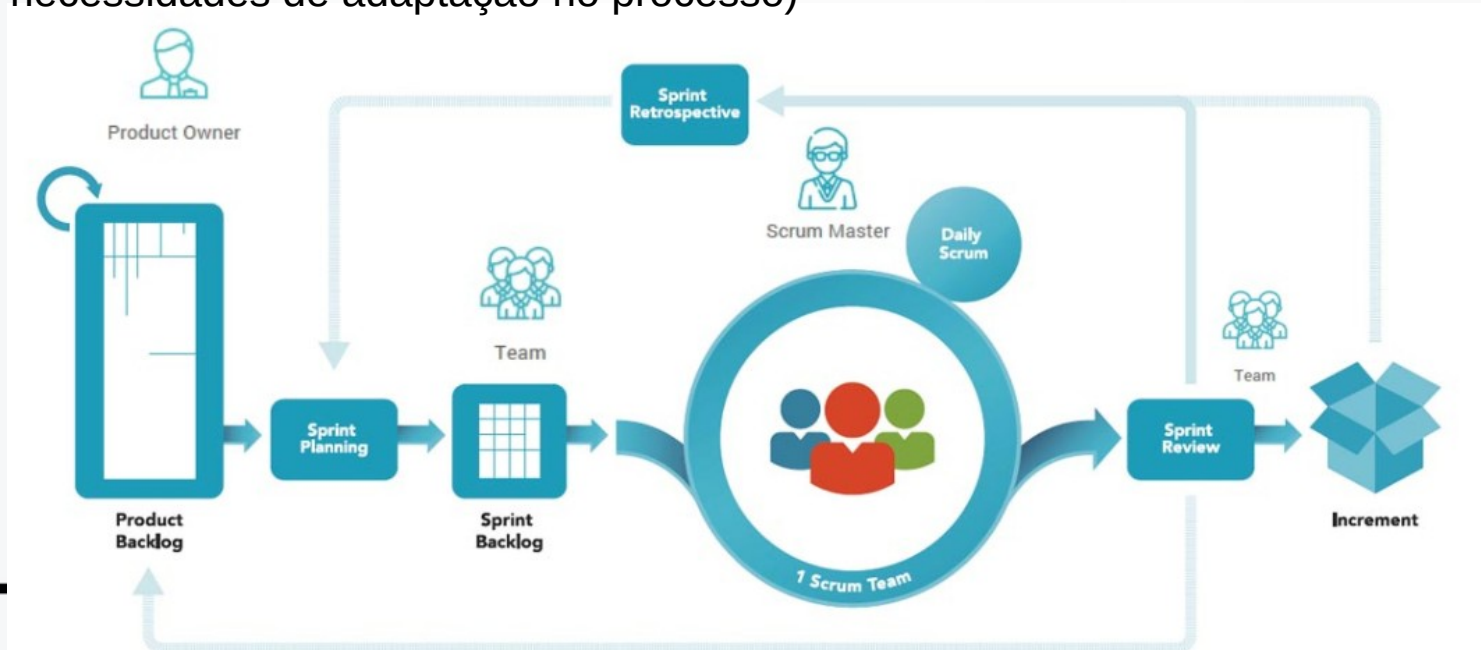
Eventos

- O **Product Owner** provê uma visão do produto (business case, macro planejamento: objetivos principais do projeto, etc)
- A visão do projeto é desmembrada em histórias de usuário (funcionalidades), as quais são **priorizadas,** formando o **Product Backlog** (Product Owner auxiliado pelo Scrum Master)
- Antes de cada sprint, é feita a **Sprint Planning** para planejamento do sprint.



Eventos

- São realizadas sprints com tempo fixo (2 a 4 semana).
- Durante a **Execução da sprint,** é realizada diariamente a **Daily Scrum,** reunião onde se responde**: o que fez ontem? O que fará hoje? Algum impedimento?** => Visão do andamento do projeto (pode ser usado o Kanban Board e o BurnDown Chart)
- No final da sprint, realiza-se a **Revisão de Sprint** (revisão sobre o sprint validação do produto sendo desenvolvido, atualização do Product Backlog). E a **Retrospectiva Sprint** (verificar necessidades de adaptação no processo)



Sprint

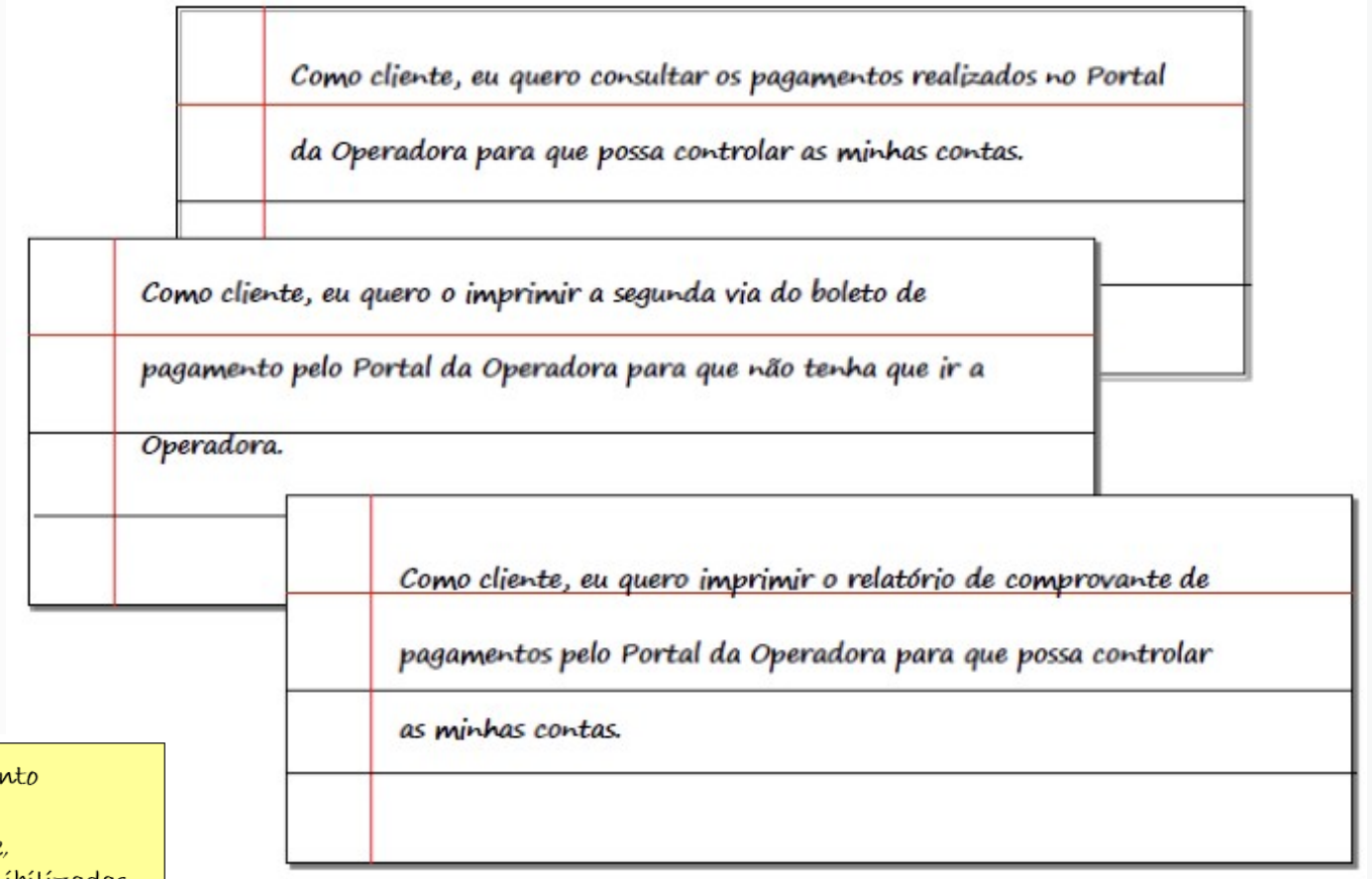
- O sprint é o ciclo de desenvolvimento de poucas semanas de duração sobre o qual se estrutura o Scrum.
- No início de cada sprint é feito um ***sprint planning meeting***, no qual a equipe **prioriza** os elementos do **product backlog** a serem implementados, e transfere estes elementos do *product backlog* para o ***sprint backlog,*** ou seja, a lista de funcionalidades a serem implementadas no ciclo que se inicia.
  - **Product backlog:** requisitos de alto nível, necessidades diretas do cliente
  - **Sprint Backlog**: requisitos de forma mais voltada à maneira como a euquipe vai desenvolvê-los
- A equipe se compromete a desenvolver as funcionalidades, e o ***product owner*** se compromete a não trazer novas funcionalidades durante o mesmo sprint. **Se novas funcionalidades forem descobertas, serão abordadas em sprints posteriores.**

Product backlog

- As funcionalidades a serem implementadas em cada projeto (requisitos ou histórias de usuário) são mantidas em uma lista chamada de ***product backlog***.

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 página de saldo e verificar que ele aumentou em R$10,00.	Precisa de um diagrama de sequência UML. Não há necessidade de se preocupar com criptografia por enquanto.
2	Ver extrato	10	8	Logar, clicar em "transações". Fazer um depósito. Voltar para transações, ver que o depósito apareceu.	Usar paginação para evitar consultas grandes ao BD. *Design* similar para visualizar página de usuário.

Exemplos de histórias de usuário



Formas de Pagamento

Como um Cliente,
Eu quero que sejam disponibilizadas diversas formas de pagamento
Para pagar meu pedido.

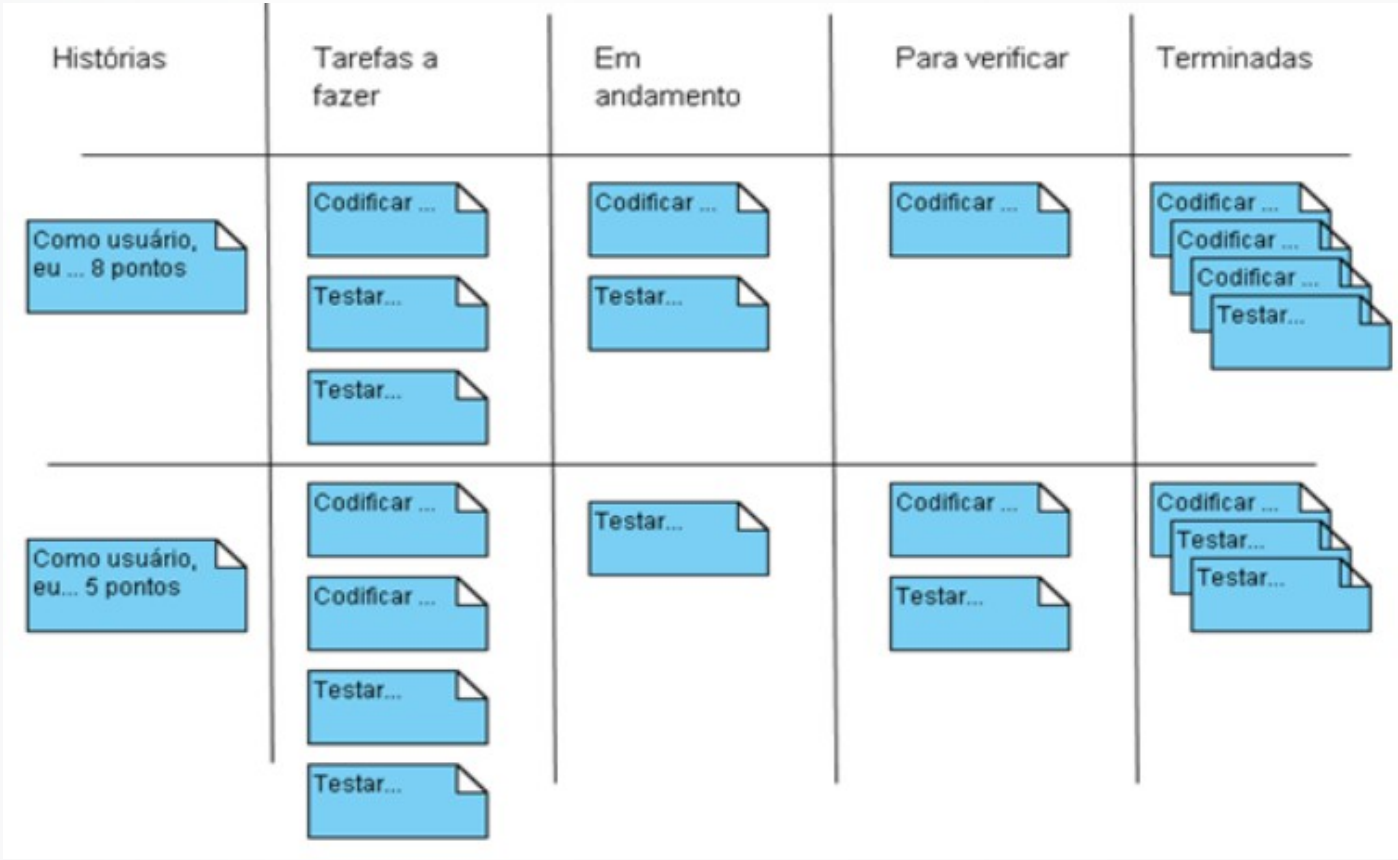
Product backlog

- **Id**: identificador numérico (manter identificação do requisitos no caso de mudança de Nome)
- **Nome**: nome curto e representativo da história
- **Imp**.: importância da historia de usuário, mais alto implica em mais importante (mas sem refletir proporção)
- **PH**: estimativa de esforço para transformar a historia (*ph – pontos de história*)
- **Como demonstrar**: o que deve ser possível fazer para considerar a história efetivamente implementada. Pode incluir passos de teste para a história
- **Notas**: quaisquer outras observações pertinentes

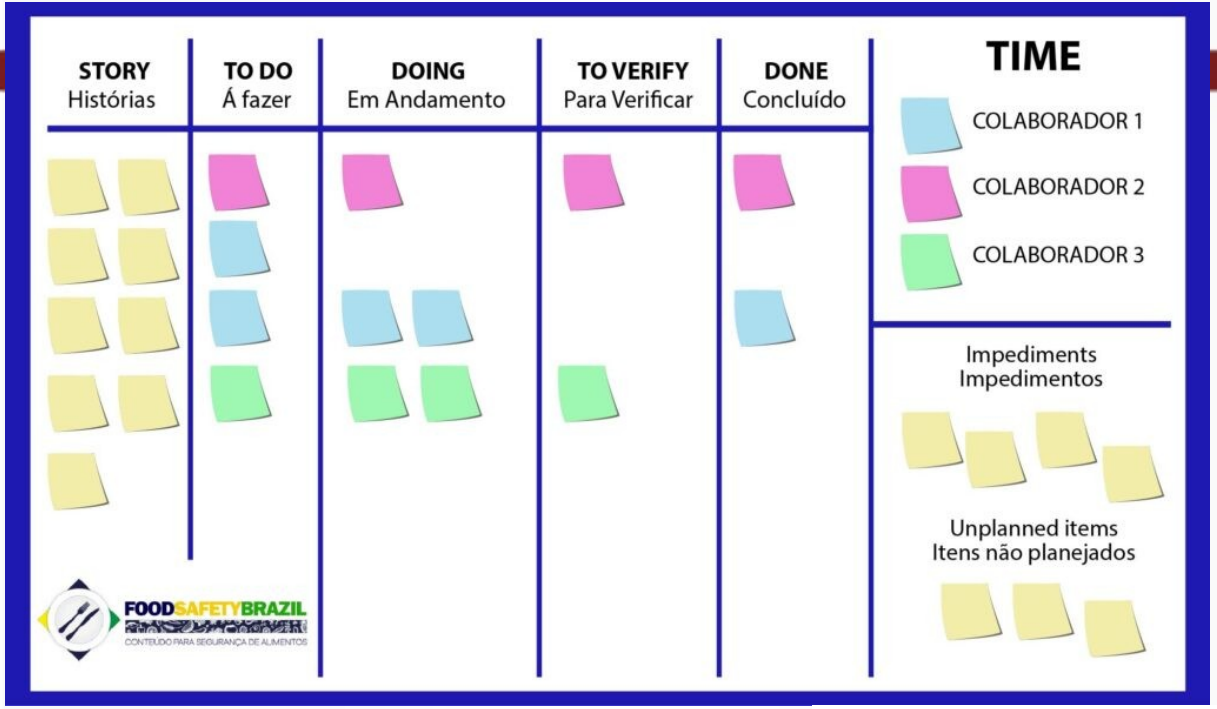
Product Backlog **(exemplo)**					
ID	**Nome**	**Imp.**	**PH**	**Como demonstrar**	**Notas**
1	Depósito	30	5	Logar, abrir página de depósito, depositar R$10,00, ir para página de saldo e verificar que ele aumentou em R$10,00.	Precisa de um diagrama de sequência UML. Não há necessidade de se preocupar com criptografia por enquanto.
2	Ver extrato	10	8	Logar, clicar em "transações". Fazer um depósito. Voltar para transações, ver que o depósito apareceu.	Usar paginação para evitar consultas grandes ao BD. *Design* similar para visualizar página de usuário.

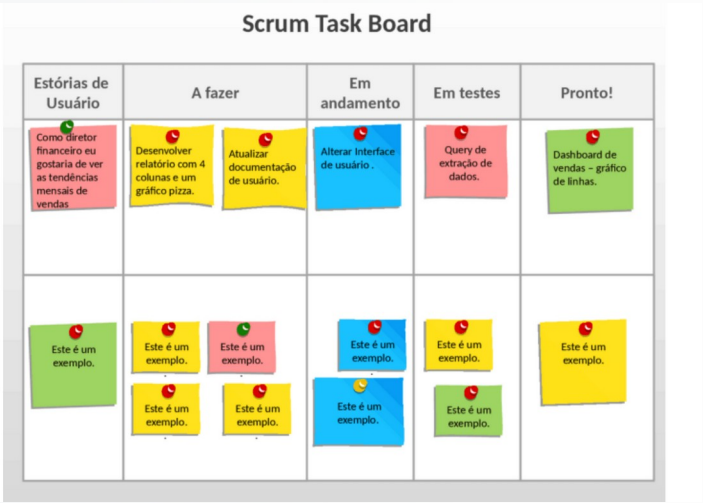
Sprint backlog

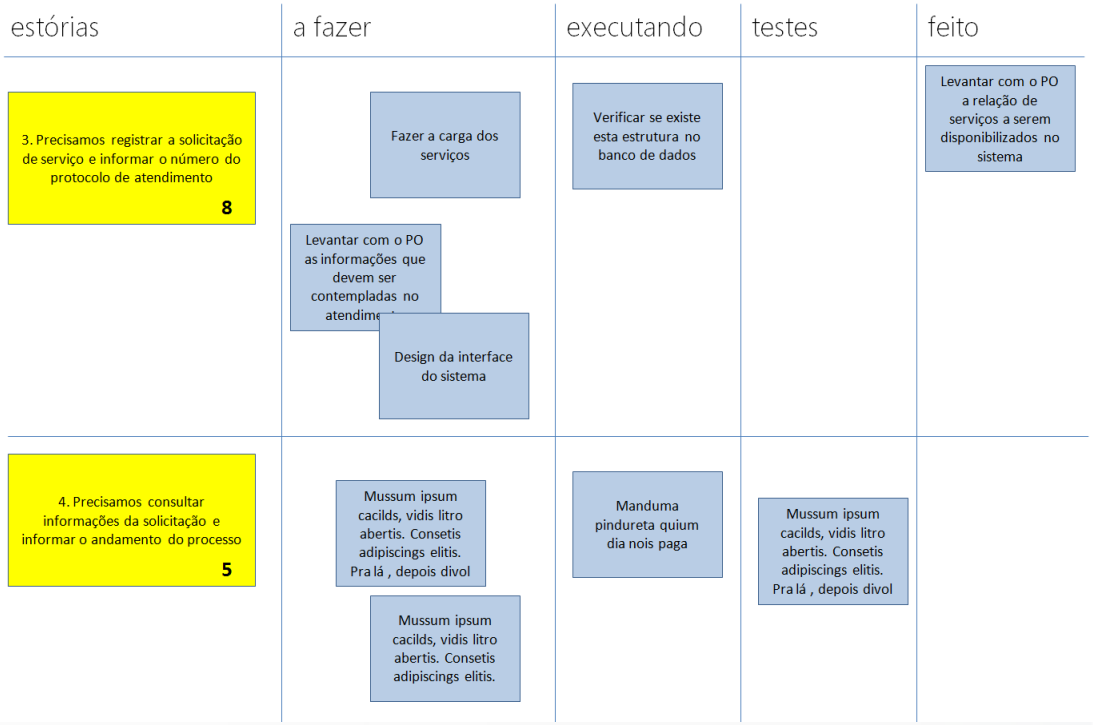
- O *product owner* mantêm o *sprint backlog* atualizado, indicando tarefas concluídas e a concluir , mostradas diariamente no quadro à vista de todos

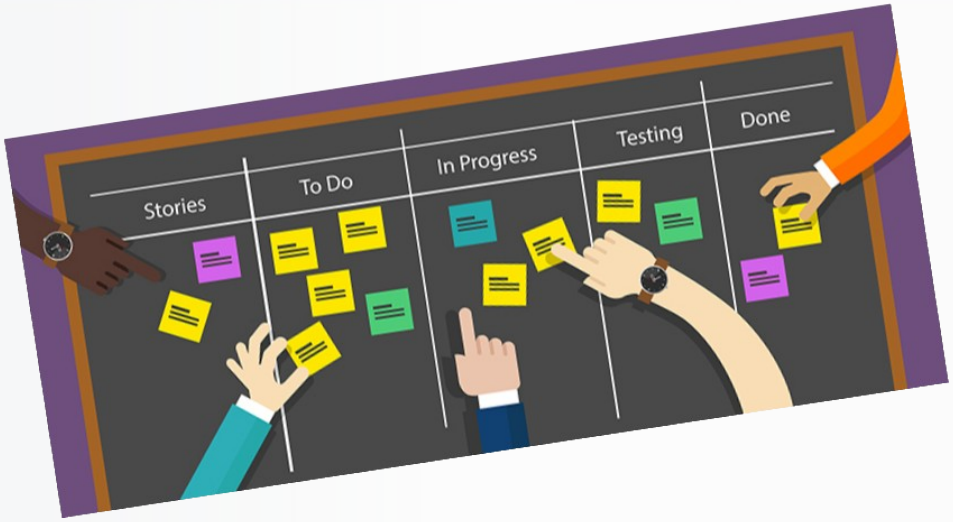


Kanban Chart







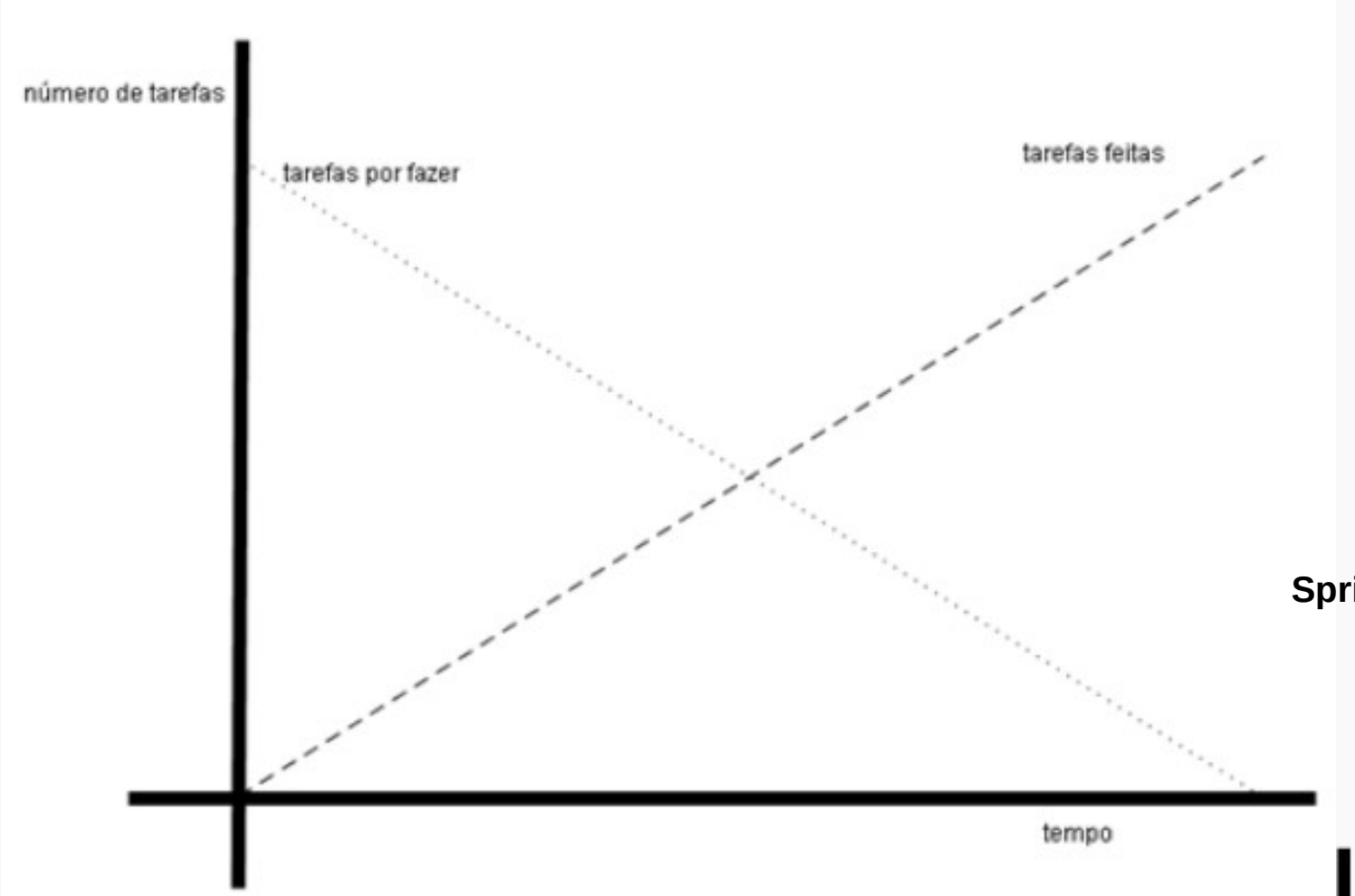


- Quadro inicializado com as histórias de usuário (1ª coluna) retiradas do *product backlog* (enfoque do usuário / cliente)
- Equipe defini atividades para necessárias para implementar cada história (enfoque do desenvolvedor)
- Essa lista de atividades constitui o *sprint backlog,* e é usada para compor a 2ª coluna "*tarefas a fazer*"
- A medida que as atividades vão sendo realizadas e verificadas, vão sendo migradas para as outras colunas.
- Atividades não prevista para implementar as histórias em andamento, podem ser adicionadas à coluna "a fazer". Mas novas histórias não (durante o ciclo corrente)

Daily scrum

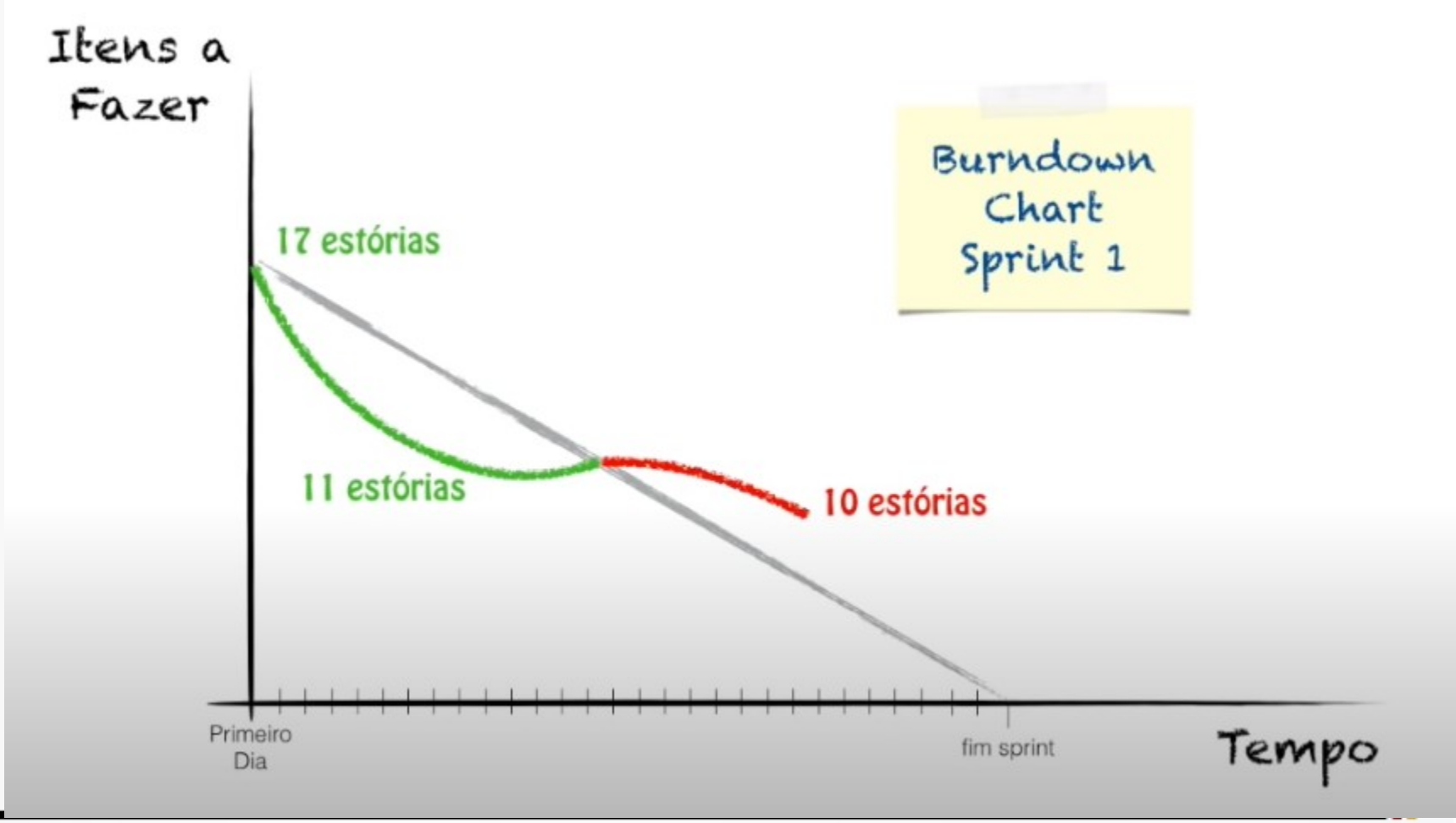
- O modelo sugere que a equipe realize uma reunião diária, chamada ***daily scrum***, onde o objetivo é que **cada membro da equipe fale sobre o que fez no dia anterior, o que vai fazer no dia que se segue** e, se for o caso, o **que o impede de prosseguir.**
- Essas **reuniões** devem ser **rápidas**. Por isso, se sugere que sejam feitas com as pessoas **em pé** e em **frente a um quadro de anotações**. Além disso, recomenda-se que sejam feitas logo **após o almoço**, quando os participantes estarão mais imersos nas questões do trabalho (longe dos problemas pessoais), além de ser uma boa maneira de dissipar o cansaço que atinge os desenvolvedores no início da tarde.

Sprint burndown chart



Sprint burndown

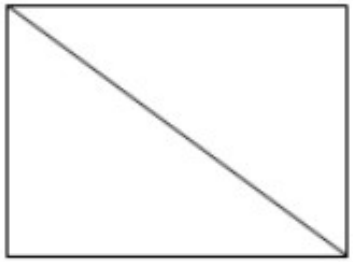
Sprint burndown chart



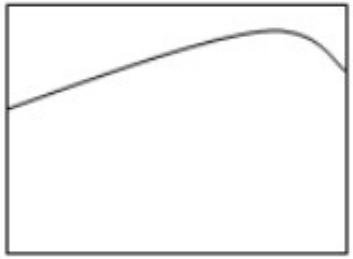
Perfis de equipe

- Pode ser feita uma análise sobre a linha de tarefas por fazer , identificando sete tipos de comportamento de equipes conforme seus *sprint burndown*:

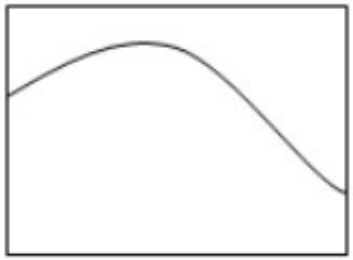
Fakey-fakey, caracterizado por uma linha reta e regular, indicando que provavelmente a equipe não está sendo muito honesta porque o mundo real é bem complexo e dificilmente projetos se comportam com tanta regularidade.



Late-learner, indicando um acúmulo de tarefas até perto do final do *sprint*. É típico de equipes iniciantes que ainda estão tentando aprender o funcionamento do *Scrum*. **(Iniciantes)**

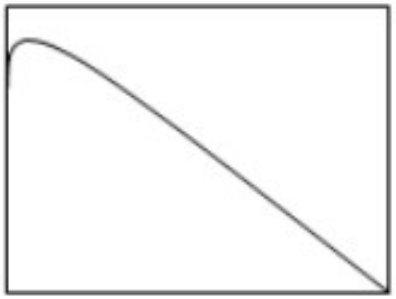


Middle-learner, indicando que a equipe pode estar amadurecendo e trazendo para mais cedo as atividades de descoberta e, especialmente, as necessidades de testes.

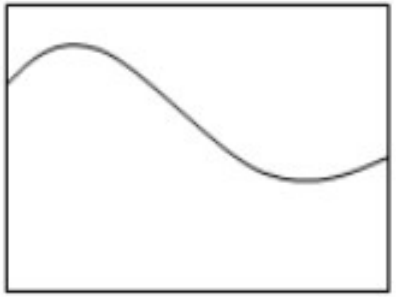


Perfis de equipe

Early-learner, indicando uma equipe que procura logo no início do *sprint* descobrir todas as necessidades e posteriormente desenvolve-las gradualmente até o final do ciclo.

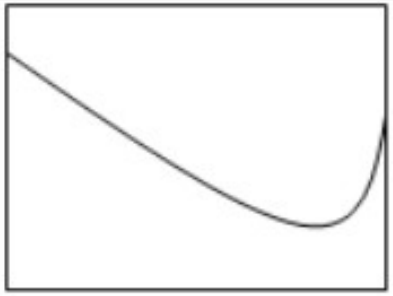


Plateau, indicando uma equipe que tenta balancear o aprendizado cedo e tardio, o que acaba levando o ritmo de desenvolvimento a um platô. Inicialmente fazem bom progresso, mas não conseguem manter o ritmo até o final do *sprint*.

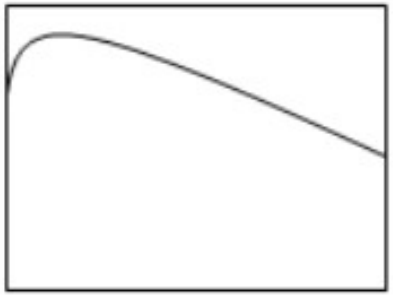


Perfis de equipe

Never-never, indicando uma equipe que acaba tendo surpresas desagradáveis no final de um *sprint*.



Scope increase, indicando uma equipe que percebe um súbito aumento na carga de trabalho por fazer. Usualmente a solução nestes casos é tentar renegociar o escopo da *sprint* com o *product owner*, mas não se descarta também uma finalização da *sprint* para que seja feito um replanejamento do *product backlog*.



Finalização de sprint

- Ao final de cada sprint a equipe deve fazer um ***sprint review meeting*** (ou sprint demo) para verificar o que foi feito e, a partir daí, partir para uma nova sprint. O ***sprint review meeting*** é a demonstração e avaliação do produto do sprint.
- Outra reunião que pode ser feita ao final de uma sprint é a ***sprint retrospective***, cujo objetivo é avaliar a equipe e os processos (impedimentos, problemas, dificuldades, ideias novas etc.).

UFV

Assista o vídeo abaixo do MindMaster, que fornece um bom resumo sobre a dinâmica do Scrum

https://www.youtube.com/watch?v=XfvQWnRgxG0

XP- Extremme Programming

UFV

XP – Extreme Programming

- É um modelo ágil inicialmente adequado a equipes pequenas e médias que é baseado em uma série de valores, princípios e regras.
- XP surgiu no final da década de 1990, nos Estados Unidos, tendo sido criada por Kent Beck.

XP is a lightweight methodology for small to medium sized teams developing software in the face of vague or rapidly changing requirements.

-- Kent Beck.

UFV

Valores XP

- **Simplicidade**: pesquisas mostram que metade das funcionalidades introduzidas em sistemas nunca são usadas. Focar nas funcionalidades efetivamente necessárias.
- **Respeito**: entre os membros da equipe. Sem respeito a comunicação falha e projeto “afunda”
- **Comunicação**: essencial para que o cliente consiga dizer o que precisa, preferencia por comunicação verbal .
- **Feedback**: obter feedbacks o quanto antes para que eventuais falhas de comunicação sejam corrigidas.
- **Coragem**: confiar nos mecanismos de mudança para se ter coragem de abraçar as inevitáveis modificações.

WHAT IS XP?

Lightweight

Discipline

Humanistic

Software development

WHAT IS XP?

Processo é reduzido ao mínimo possível

Lightweight

Inclui práticas a serem seguidas

Discipline

Foco nas pessoas: desenvolvedores, clientes, ...

Humanistic

Objetivo chave

Software development

Princípios XP

- Feedback rápido.
- Presumir simplicidade.
- Mudanças incrementais.
- Abraçar mudanças.
- Trabalho de alta qualidade.

Práticas XP

- Jogo de planejamento (planning game).

  – **Semanalmente** a equipe deve se **reunir com o cliente** para **priorizar** as **funcionalidades** a serem desenvolvidas. Cabe ao cliente identificar as principais necessidades e à equipe de desenvolvimento estimar quais podem ser implementadas **no ciclo semanal que se inicia**. Ao final da semana essas funcionalidades são entregues ao cliente. Esse tipo de modelo de relacionamento com o cliente é adaptativo, em oposição aos contratos rígidos usualmente estabelecidos.

Práticas XP

- **Metáfora (metaphor).**

  – É preciso conhecer a linguagem do cliente e seus significados. A equipe deve aprender a se comunicar com o cliente na linguagem que ele compreende.

- **Equipe coesa (whole team).**

  – O cliente faz parte da equipe de desenvolvimento e a equipe deve ser estruturada de forma que eventuais barreiras de comunicação sejam eliminadas.

- **Reuniões em pé (stand-up meeting).**

  – Como no caso de Scrum, reuniões em pé tendem a serem mais objetivas e efetivas.

UFV

Práticas XP

- **Design simples (simple design).**

  – Isso implica em **atender** a **funcionalidade** solicitada pelo **cliente sem sofisticar desnecessariamente**. Deve-se fazer o que o cliente precisa, não o que o desenvolvedor gostaria que ele precisasse. Por vezes, **design simples** pode ser confundido com design fácil. **Nem sempre o design simples é o mais fácil de implementar.** O design fácil pode não atender às necessidades, ou pode gerar problemas de arquitetura.

- **Versões pequenas (small releases).**

  – A l**iberação de versões pequenas** do sistema pode **ajudar o cliente a testar as funcionalidades de forma contínua.** XP leva ao extremo este princípio, sugerindo versões ainda menores do que as de outros processos incrementais como UP e Scrum.

- **Ritmo sustentável (sustainable pace).**

  – **Trabalhar com qualidade um número razoável de horas por dia** (não mais do que 8). Horas extras só são recomendadas quando efetivamente trouxerem um aumento de produtividade, mas não podem ser rotina.

Práticas XP

- **Posse coletiva (collective ownership).**

  – O código não tem dono e não é necessário pedir permissão a ninguém para modificá-lo.

- **Programação em pares (pair programming).**

  – A **programação** é sempre **feita por duas pessoas** em cada computador. Usualmente trata-se de um **programador** mais experiente e um **aprendiz**. O aprendiz deve usar a máquina enquanto o mais experiente o ajuda a evoluir em suas capacidades. Com isso, o código gerado terá sempre sido verificado por pelo menos duas pessoas, reduzindo drasticamente a possibilidade de erros. Existem sugestões também de que a programação em pares seja feita por desenvolvedores de mesmo nível de conhecimento, os quais devem se alternar na pilotagem do computador (Bravo, 2010).

Práticas XP

- **Padrões de codificação (coding standards).**

  – A equipe deve e**stabelecer e seguir padrões de codificação**, de forma que parecerá que o código foi todo desenvolvido pela mesma pessoa, mesmo que tenham sido dezenas.

- **Testes de aceitação (customer tests).**

  – São testes planejados e conduzidos pela equipe em conjunto com o cliente para verificar se os requisitos foram atendidos.

- **Desenvolvimento orientado a testes (test driven development).**

  – Antes de programar uma unidade deve-se definir e implementar os testes pelos quais ela deverá passar.

UFV

Práticas XP

- **Refatoração (refactoring).**

  – Não se deve fugir da refatoração quando necessária. Ela permite manter a complexidade do código em um nível gerenciável. É um investimento que traz benefícios a médio e longo prazo.

- **Integração contínua (continuous integration).**

  – Nunca esperar até ao final do ciclo para integrar uma nova funcionalidade. Assim que estiver viável ela deve ser integrada ao sistema para evitar surpresas.

UFV

Regras de planejamento

- Escrever histórias de usuário.
- O planejamento de entregas cria o cronograma de entregas.
- Faça entregas pequenas frequentes.
- O projeto é dividido em iterações.
- O planejamento da iteração inicia cada iteração.

Regras de gerenciamento

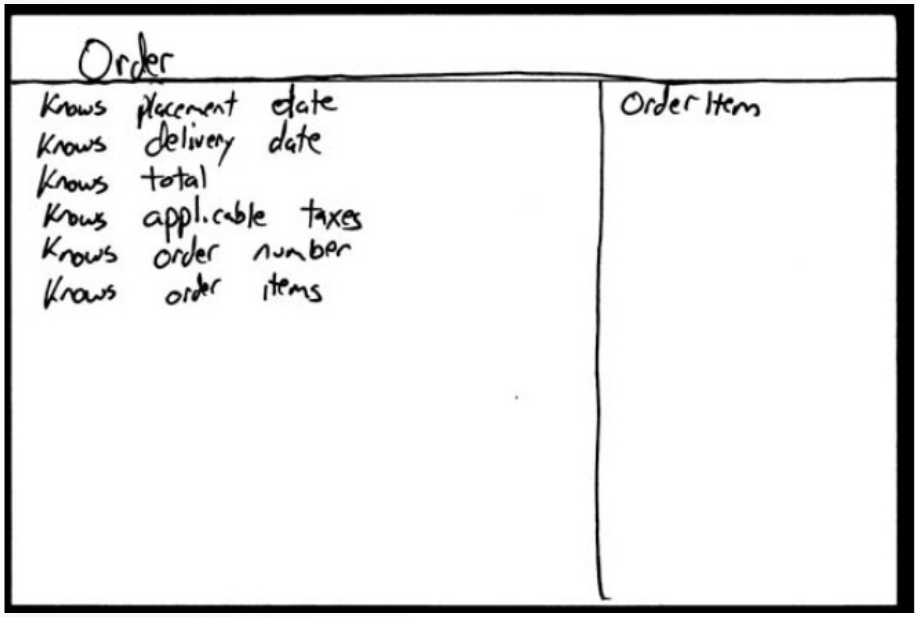
- Dê à equipe um espaço de trabalho aberto e dedicado.
- Defina uma jornada sustentável.
- Inicie cada dia com uma reunião em pé.
- A velocidade do projeto é medida.
- Mova as pessoas.
- Conserte XP quando for inadequado.

Regras de design

- **Simplicidade**.
- Escolha uma **metáfora de sistema**: ajuda a explicar o funcionamento do sistema a alguém fora do projeto.
- Use **Cartões CRC** (cartões de responsabilidade e colaboração) durante reuniões de projeto: equipe identifica responsabilidades do objeto e colaborações do objeto. Cartões representam instâncias de diferentes classes
- Crie **soluções afiadas** (spikes) para reduzir risco: explorar riscos do projeto e buscar soluções afiadas (spike) para os problemas identificados
- **Nenhuma funcionalidade** é adicionada **antes da hora**.
- Use **refatoração** sempre e onde for possível

UFV

Cartões CRC (cartões de responsabilidade e colaboração)



Transcript	
See the prototype Determine average mark	Student Seminar Professor Enrollment

Professor	
Name Address Phone number Email address Salary Provide information Seminars instructing	Seminar

Seminar	
Name Seminar number Fees Waiting list Enrolled students Instructor Add student Drop student	Student Professor

Student Schedule	
See the prototype	Seminar Professor Student Enrollment Room

Student	
Name Address Phone number Email address Student number Average mark received Validate identifying info Provide list of seminars taken	Enrollment

Room	
Building Room number Type (Lab, class, ...) Number of Seats Get building name Provide available time slots	Building

Student	
Student number Name Address Phone number Enroll in a seminar Drop a seminar Request transcripts	Seminar

Building	
Building Name Rooms Provide name Provide list of available rooms for a given time period	Room

Regras de codificação

- O **cliente** está **sempre disponível**.
- O **código** deve ser escrito **de acordo** com **padrões** aceitos.
- Escreva o **teste** de **unidade primeiro**.
- Todo o **código** é **produzido por pares**.
- **Só um par** faz **integração de código** de **cada vez**.
- **Integração** deve ser **frequente**.
- Defina um **computador exclusivo** para **integração**.
- A **posse** do **código** deve ser **coletiva**.

UFV

Regras de teste

- Todo o código deve ter **testes de unidade**.
- **Todo código** deve passar pelos **testes** de unidade antes de ser entregue.
- Quando um **erro** de funcionalidade é **encontrado**, **testes são criados.**
- **Testes de aceitação** são **executados** com **frequência** e os resultados são publicados.

Até a próxima!

Campus Viçosa:
Avenida Peter Henry Rolfs, s/n
CEP 36570-900
Viçosa - MG - Brasil | + 55 31 3899-2200

Campus Florestal:
Rodovia LMG 818, km 6
CEP 35690-000
Florestal - MG - Brasil | + 55 31 3536-3300

Campus Rio Paranaíba:
Rodovia MG-230, Km 8
CEP 38810-000
Rio Paranaíba- MG - Brasil | + 55 34 3855-9300

www.ufv.br

UFV
Universidade Federal de Viçosa