Agilidade em Projetos de Desenvolvimento de Software

Prof. Rafael Dias Ribeiro, M.Sc, CSM, CSPO,PMP.
http://www.rafaeldiasribeiro.com.br

Agilidade

- s.f. Ligeireza, presteza, leveza; desembaraço.

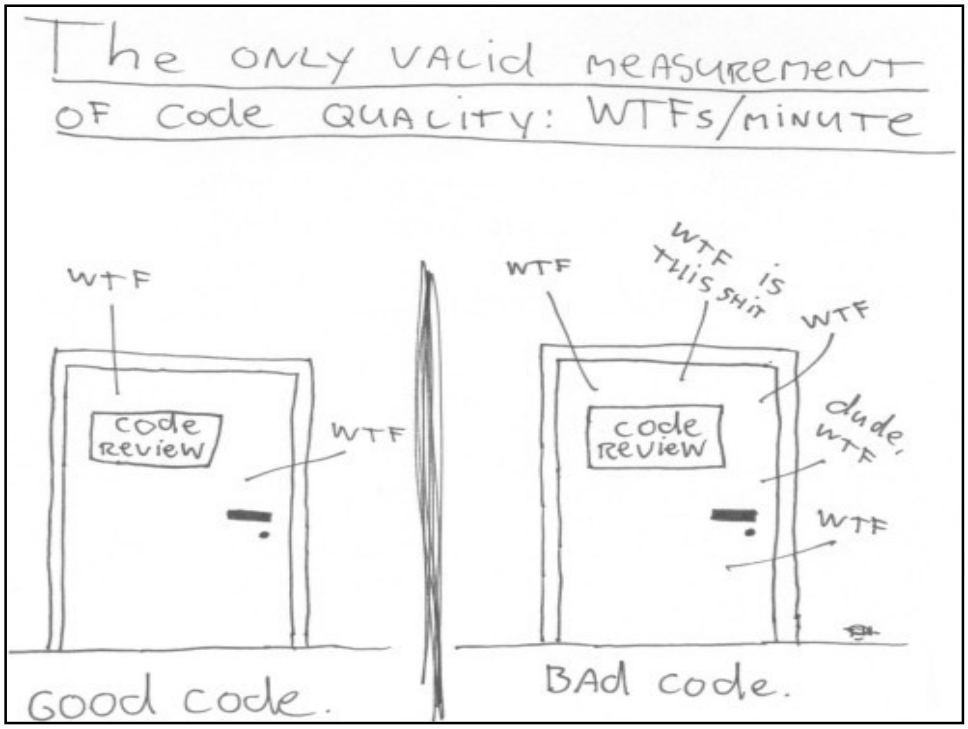
http://www.dicio.com.br/agilidade/

- é a capacidade de executar movimentos rápidos e ligeiros com mudanças nas direções

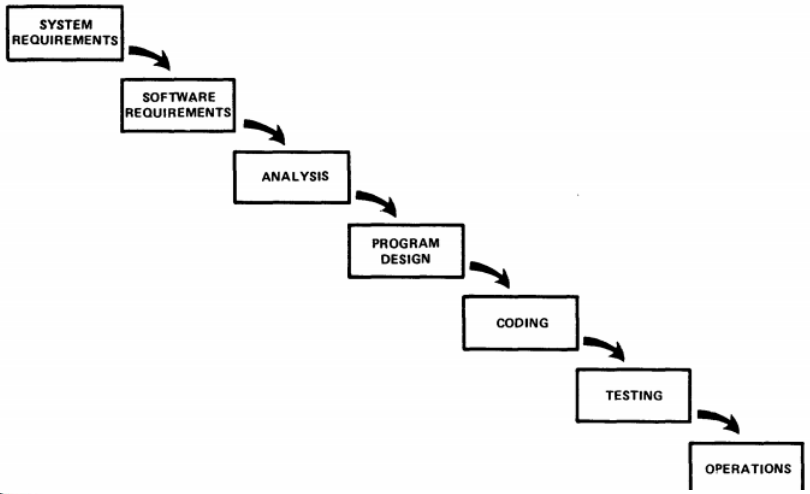
Adaptado de http://pt.wikipedia.org/wiki/Agilidade

Rapidez ≠ Agilidade

- Desenvolvimento ágil é diferente de desenvolvimento rápido de aplicações.
- A agilidade está em garantir a qualidade do método de se adaptar e solucionar problemas durante o desenvolvimento do produto.



Evolução Histórica: Waterfall
Proposta por Winston W. Royce em 1970



@ribeirord

Evolução Histórica: Waterfall – Winston W. Royce

"In order to procure a 5 million dollar hardware device, I would expect that a 30 pages specification would provide adequate detail to control the procurement. In order to procure 5 million dollars of software I would estimate 1500 pages specification is about right in order to achieve comparable control"

"A fim de obter um hardware de 5 milhões de dólares, eu esperaria que uma especificação de 30 páginas proporcionasse detalhes suficiente para controlar esta aquisição. A fim de obter um software de 5 milhões de dólares, eu estimaria 1500 páginas de especificação, a fim de conseguir o controle comparável"

Royce defendia que a documentação deve ser extensa...

Evolução Histórica: **Waterfall** – Winston W. Royce

"I believe in this concept, but the implementation described above is risky and invites failure. (...) The testing phase which occurs at the end of development cycles is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed"

"Eu acredito neste conceito, mas a implementação descrita acima é arriscada e convida falhas. (...) A fase de testes, que ocorre no final dos ciclos de desenvolvimento é o primeiro evento para o qual a temporização, o armazenamento, transferência de entrada/saída, etc., são experimentados como distinguido de analisados"

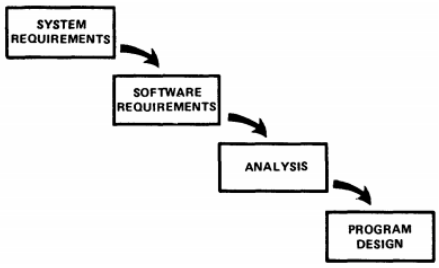
Se o projeto precisa estar pronto para ser testado o produto desenvolvido fatidicamente encontrará problemas (bugs, questões de performance, escalabilidade,...)

- DINÂMICA

Premissas / Suposições : Waterfall

- Um pessoa ou um grupo fala com o cliente e descobre o que ele deseja do software
- O mesmo grupo de analistas transmite a informação obtida com o cliente para a equipe técnica
- Os analistas verificam a viabilidade do projeto e identificam os riscos
- Os arquitetos de TI documentam (diagramas, etc) para orientar os desenvolvedores
- Os desenvolvedores criam os códigos de acordo com os documentos criados pelos arquitetos de TI
- Os *testers* verificam os documentos e validam os códigos gerados
- O software é implementado

- A distância do cliente durante o desenvolvimento do sistema muitas vezes faz perder o foco do que é importante para ele.
- As 4 primeiras fases consistem apenas em documentação, sem nenhuma produção de valor para o cliente.



- Em um período de desenvolvimento muitas tecnologias surgem assim como a própria necessidade do cliente , de acordo com a evolução do negócio.
- Será que os programadores são incapazes de tomar decisão sobre o código ?

RUP – Rational Unified Processes

- Criado pela Rational, mais tarde englobado pela IBM

" The Rational Unified Process activities create and maintain models. Rather than focusing on the production of large amount of paper documents, the Unified Process emphasize the development and maintenance of models – semanticly rich representations of the software system under development "

O *Rational Unified Process* tem a finalidade de criar e manter modelos. Ao invés de focar na produção de grande quantidade de documentos em papel, o Rational Unified Process enfatiza o desenvolvimento e manutenção de modelos – representações semanticamente ricas do sistema de software em desenvolvimento

RUP – Rational Unified Processes

" The Rational Unified Process is a configurable process. No single process is suitable for all software development. The Unified Process fits small development teams as well as large development organizations. The Unified Process is founded on a simple and clear process architecture that provides commonality across a family of processes. Yes, it can be to accommodate different situations. "

O *Rational Unified Process* é um processo configurável. Nenhum único processo é adequado para todo o desenvolvimento de software. O Processo Unificado se adequa a pequenas equipes de desenvolvimento, bem como organizações de desenvolvimento de grande porte. O Processo Unificado é fundamentada em uma arquitetura de processo simples e claros que fornece uma família de processos comuns. Sim, ele pode ser a de acomodar para situações diferentes.

RUP – Rational Unified Processes

Melhores Práticas:

- Desenvolva software iterativamente
- Gerencie requisitos
- Use arquiteturas baseadas em componentes
- Modele visualmente o software
- Verifique a qualidade do software
- Controle as mudanças no software

RUP – Rational Unified Processes

" Given today's sophisticated softwares systems, it is not possible to sequentially first define the entire problem, design the entire solution, build the software and then test the product at the end. Na interactive approach is required that allows an increasing understanding of the problem through successive refinements, and to incrementally grow an effective solution over multiple iterations. "

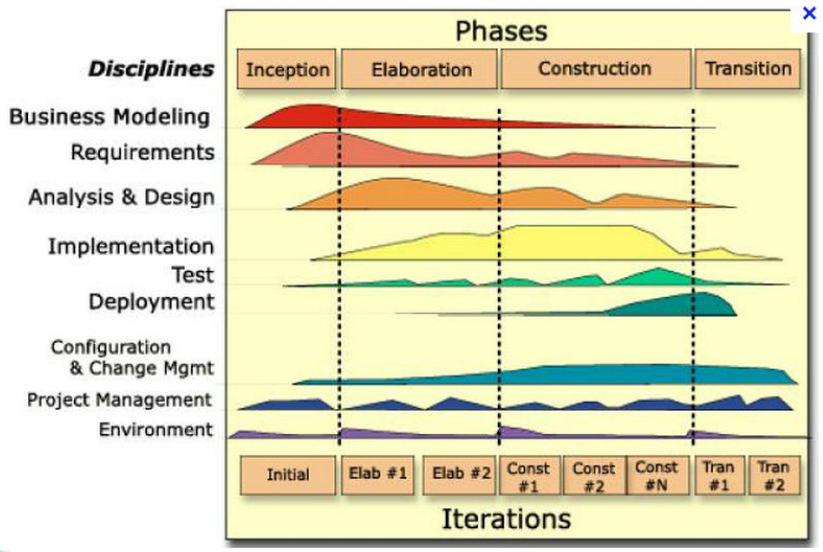
Devido a sofisticação dos softwares atuais, não é possível sequenciar o desenvolvimento de primeiro definir todo o problema, projetar toda a solução, construir o software e, em seguida, testar o produto no final. Na abordagem interativa é necessário permitir a compreensão crescente do problema através de refinamentos sucessivos, e gradativamente crescer na solução eficaz durante várias iterações.

RUP – Rational Unified Processes

- Desenvolvimento iterativo, começando sempre pelos pontos de maior risco ao projeto . Decisões de risco decididas no inicio do projeto são corrigíveis mais facilmente.

- Iterações curtas com releases internos durante a construção do projeto permitem um retorno mais rico do cliente e ajudam a controlar melhor o tempo estimado para a conclusão do desenvolvimento.

RUP – Rational Unified Processes



RUP – Rational Unified Processes

- O framework raramente é aplicado iterativamente...
- Apesar de esperar que mudanças aconteçam , não é muito dinâmico para tratá-las...

" While the process must always accommodate changes, the elaboration phase ensure that the architecture, requirements and plans are stable enough, and the risks are sufficiently mitigated, so you can predictably determine the cost and schedule for the completion of the development. Conceptually, his level of fidelity would correspond to the level necessary for na organization to commit to a fixed-price construction phase "

Embora o processo deve sempre acomodar as mudanças, a fase de elaboração deve garantir que a arquitetura, os requisitos e os planos sejam estáveis o suficiente e que os riscos foram suficientemente mitigados, para que você possa previsivelmente determinar o custo e o cronograma para a conclusão do desenvolvimento. Conceitualmente, o seu nível de fidelidade que corresponderia ao nível necessário para uma organização se comprometer com uma fase de construção a preço fixo.

Lean IT

- **Lean IT definition:** *Lean IT engages people, using a framework of Lean principles, systems, and tools, to integrate, align, and synchronize the IT organization with the business to provide quality information and effective information systems, enabling and sustaining the continuous improvement and innovation of processes. Lean IT has two aspects: outward facing, supporting the continuous improvement of business processes, and inward-facing, improving the performance of IT processes and services*

- Lean IT envolve as pessoas, utilizando um quadro de princípios do *Lean*, sistemas e ferramentas, para integrar, alinhar e sincronizar a organização de TI com o negócio para fornecer informação de qualidade e sistemas de informação eficazes, permitindo e manutenção da melhoria contínua e inovação de processos. Lean IT tem dois aspectos:.Para o exterior, apoiando a melhoria contínua dos processos de negócio, e para dentro (interna), melhorando o desempenho dos processos de TI e serviços

Lean IT

- O sistema Lean de produção foi desenvolvido pela Toyota e tem um princípio claro e simples: **Reduzir o desperdício**
- Desperdício é tudo o que não irá gerar valor para o cliente. Este conceito foi incorporado em alguns métodos ágeis e considera como desperdício documentação excessiva, tempo de desenvolvimento parado por falta de informações (ou falta de clareza), ou códigos e funcionalidades não solicitadas...

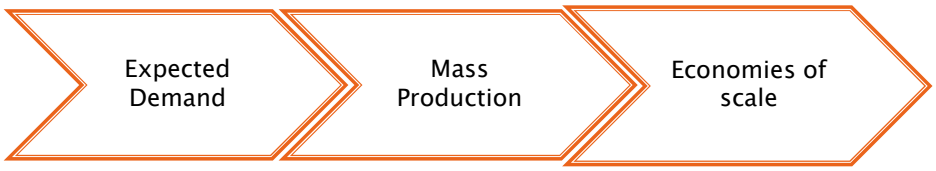
Lean IT

- Tipos de Desperdícios:
  - **Mura**: desperdício por tentar prever possíveis necessidades futuras. Evitar a mura significa reduzir ao máximo o inventário, isto é, as partes paradas no meio do processo, isto é, começando ou não terminando.
  - **Muri**: desperdícios que podem ser evitados por planejamento. Nessa categoria enquadra-se o excesso de burocracia ou de complexidade nem processo de produção.
  - **Muda**: desperdícios do dia a dia, criados por uma cultura anterior de trabalho.
    - Superprodução
    - Transporte desnecessário
    - Inventário
    - Locomoção
    - Defeitos
    - Super processamento
    - Espera

Waste Element	Examples	Business Outcome
Defects	• Unauthorized system and application changes. • Substandard project execution.	Poor customer service, increased costs.
Overproduction (Overprovisioning)	• Unnecessary delivery of low-value applications and services.	Business and IT misalignment, Increased costs and overheads: energy, data center space, maintenance.
Waiting	• Slow application response times. • Manual service escalation procedures.	Lost revenue, poor customer service, reduced productivity.
Non-Value Added Processing	• Reporting technology metrics to business managers.	Miscommunication.
Transportation	• On-site visits to resolve hardware and software issues. • Physical software, security and compliance audits.	Higher capital and operational expenses.
Inventory (Excess)	• Server sprawl, underutilized hardware. • Multiple repositories to handle risks and control. • Benched application development teams.	Increased costs: data center, energy; lost productivity.
Motion (Excess)	• Fire-fighting repeat problems within the IT infrastructure.	Lost productivity.
Employee Knowledge (Unused)	• Failing to capture ideas/innovation. • Knowledge and experience retention issues. • Employees spend time on repetitive or mundane tasks.	Talent leakage, low job satisfaction, increased support and maintenance costs.

Lean IT Sistema PUSH vs. PULL

- Sistema PUSH – upstream information



- No sistema Ford de produção, cada estação da linha de produção trabalha enquanto houver matéria prima para tal. A quantidade do que será produzido é regulada com base em provisões feitas sobre o mercado em um determinado período, mas não há ligação entre os pedidos reais e a linha de produção.
- Exemplo de Mura:

  Muitas peças produzidas esperando para serem utilizadas em outras estações, sem sequer saber se existe demanda pelo produto (inventário). Muito WIP (Work in Progress)
  - Dimensionamento do trabalho...

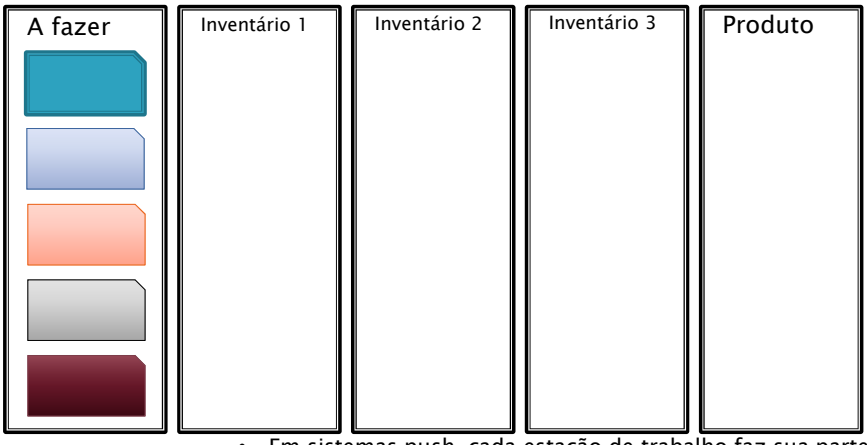
Lean IT Sistema PUSH vs. PULL

- Sistema PULL – downstream information

- Trabalha com o mínimo possível de inventário que ainda permite atender às demandas de clientes rapidamente. Sua concepção e prática são simples: há apenas o número suficiente de peças em inventário para um produto (ou lote) ser completo.
- A produção acontece de acordo com a demanda, reduzindo custos de armazenamento, de peças intermediárias e produtos não vendidos, além de flexibilizar a produção.

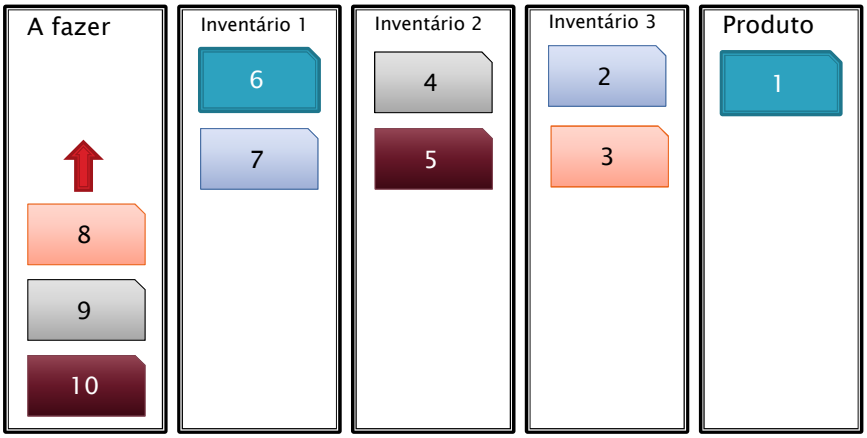
KanBan

Linha de Produção = sequência de passos para produzir algo



- Em sistemas push, cada estação de trabalho faz sua parte sem considerar fases anteriores e posteriores. O Kanban permite ver com mais clareza o acúmulo de inventário

KanBan



- No Kanban representamos o sistema pull de produção, há um limite de inventário claramente definido. Assim peças só são produzidas quando de fato há demanda e gasta-se menos em peças que não serão utilizadas ou em estoque de inventário.

@ribeirord

KanBan

- dispositivos físicos
- sinais de demanda ” downstream processes”
- regula a demanda em um pull system
- limita de trabalho em progresso (wip)
- controle visual
- auto direção

OBS: É uma excelente forma de visualizar o andamento da produção mas é apenas uma ferramenta que deve ser empregada para auxiliar (reforçar) a metodologia aplicada.

Systems Thinking

Após reduzir a mura, temos uma melhor visão de nossa linha de produção (trabalhos redundantes,...)

Quantas vezes não vimos consagrados processos de produção de software atrapalhando uma equipe em vez de ajudá-la ? Todas as fases do seu processo são realmente necessárias para o sucesso final ?

Systems Thinking é pensar e repensar durante todo o andamento do projeto no que poderia ser melhorado no próprio processo de desenvolvimento e nas interações entre as pessoas envolvidas

Work Cells

- Não é possível pensar em encontrar melhorias no processo se cada individuo está focado exclusivamente em uma tarefa o qual é especialista.
- O Lean entende não podem ser superespecialistas, i. e., não podem se limitar apenas a conhecimento em sua etapa. Devem conhecer todas elas e saber executar algumas delas.
- Cada membro da equipe é uma *Work Cell*: pessoa capaz de trabalhar no projeto como um todo, em algumas ou todas as suas partes. O conhecimento mais amplo sobre o projeto é incentivado, já que, ao conhecer o todo , melhores serão as críticas para melhoria.

Kaisen

- Em Lean acredita-se que não existe uma "bala de prata *" capaz de resolver todos os problemas, é preciso que cada membro da equipe adapte a metodologia e ferramentas à sua necessidade a todo momento.
- Kaisen é a palavra japonesa para melhoria, significa maximizar a função Ganho – Desperdício. Melhorias no processo, na forma de produção e no produto final são parte do dia a dia de quem trabalha com Lean

- Artigo de Frederich Brooks com o título *No Silver Bullet*, publicado na década de 70
- Leitura sugerida: http://www.4shared.com/office/kBZlRWok/TheMythicalManMonthFBrooks.html

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.

Princípios Ágeis

- 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 frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
- Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.

Princípios Ágeis

- 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.

Vantagens da Agilidade

- Feedback rápido
- Motivação da equipe
- "Sem Corpo Mole"
- Ver problemas mais cedo

SCRUM

" Waterfall and Spiral methodologies set the context and deliverable definition at the start of a project. Scrum and iterative methodologies initially plan the context and broad deliverable definition, and then envolve the deliverable during the project based on the environment. Scrum acknowledges that the underlying development processes are incompletely defined and uses control mechanisms to improve flexibility."

Metodologias de Cascata (*Waterfall)* e Espiral (*Spiral*) definem o contexto e a entrega no início de um projeto. Metodologias iterativas e o Scrum inicialmente planejam o contexto e a definição entrega ampla, e, em seguida, especifica a entrega durante o projeto com base no ambiente. Scrum reconhece que os processos de desenvolvimento subjacentes não são totalmente definidos e utiliza mecanismos de controle para melhorar a flexibilidade.

SCRUM

"The primary difference between the defined (waterfall, spiral and iterative) and empirical (SCRUM) approach is that the Scrum approach assumes that the analysis, design, and development process in the sprint phase are unpredictable. A control mechanism is used to manage the unpredictablity and control the risk. Flexibility, responsiveness, and reliability are the results."

A principal diferença entre processos definidos (waterfall, spiral and iterative) e a abordagem empírica (SCRUM) é que a abordagem Scrum assume que a análise, concepção, desenvolvimento e processo dentro da fase de sprint são imprevisíveis. Um mecanismo de controle é usado para gerenciar o falta de previsibilidade e controlar o risco. Flexibilidade, agilidade e confiabilidade são os resultados.

SCRUM

"Scrum: A framework within which people can address complex adaptative problems, while productively and creatively delivering products of the highest possible value(...) Scrum makes clear the relative efficacy of your product management and development practices so that can improve ."

Scrum: Um framework no qual as pessoas podem abordar de forma adaptativa os complexos problemas, de forma produtiva e criativa oferecendo produtos de maior valor possível (...) Scrum torna clara a eficácia relativa de seu gerenciamento de produtos e práticas de desenvolvimento.

A função do Scrum é evidenciar os problemas do projeto mais cedo para que se possa resolver o problema mais cedo.

SCRUM

- Scrum é fundamentado nas teorias empíricas de controle de processo, ou empirismo. O empirismo afirma que o **conhecimento vem da experiência e de tomada de decisões baseadas no que é conhecido**. O Scrum emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos.

SCRUM – Pilares que apoiam a implementação de controle de processo empírico

Transparência:

- Aspectos significativos do processo devem estar visíveis aos responsáveis pelos resultados. Esta transparência requer aspectos definidos por um padrão comum para que os observadores compartilharem um mesmo entendimento do que está sendo visto.

Inspeção

- Os usuários Scrum devem, frequentemente, inspecionar os artefatos Scrum e o progresso em direção ao objetivo, para detectar indesejáveis variações. Esta inspeção, não deve no entanto, ser tão frequente que atrapalhe a própria execução das tarefas. As inspeções são mais benéficas quando realizadas de forma diligente por inspetores especializados no trabalho a se verificar.

Adaptação

- Se um inspetor determina que um ou mais aspectos de um processo desviou para fora dos limites aceitáveis, e que o produto resultado será inaceitável, o processo ou o material sendo produzido deve ser ajustado. O ajuste deve ser realizado o mais breve possível para minimizar mais desvios.

SCRUM

Resumindo tudo isso...

- Scrum é um framework INCOMPLETO o qual as pessoas podem resolver problemas complexos e adaptáveis, enquanto entregam produtos de forma produtiva, criativa e com o maior valor possível !

SCRUM
Personagens

SCRUM – Personagens

- **O Product Owner**
- O Product Owner, ou dono do produto, é o responsável por maximizar o valor do produto e do trabalho da equipe de Desenvolvimento. Como isso é feito pode variar amplamente através das organizações, Times Scrum e indivíduos.
- O Product Owner pode fazer o trabalho acima, ou delegar para a Equipe de Desenvolvimento fazê-lo. No entanto, o Product Owner continua sendo o responsável pelos trabalhos.

SCRUM – Personagens

- O **Product Owner** é a única pessoa responsável por gerenciar o Backlog do Produto. O gerenciamento do Backlog do Produto inclui:
  - Expressar claramente os itens do Backlog do Produto;
  - Ordenar os itens do Backlog do Produto para alcançar melhor as metas e missões;
  - Garantir o valor do trabalho realizado pelo Time de Desenvolvimento;
  - Garantir que o Backlog do Produto seja visível, transparente, claro para todos, e mostrar o que o Time Scrum vai trabalhar a seguir; e,
  - Garantir que a Equipe de Desenvolvimento entenda os itens do Backlog do Produto no nível necessário.

SCRUM – Personagens

- O **Product Owner** é uma pessoa e não um comitê. O Product Owner pode representar o desejo de um comitê no Backlog do Produto, mas aqueles que quiserem uma alteração nas prioridades dos itens de Backlog devem convencer o Product Owner.
- Para que o Product Owner tenha sucesso, toda a organização deve respeitar as suas decisões. As decisões do Product Owner são visíveis no conteúdo e na priorização do Backlog do Produto. Ninguém tem permissão para falar com a Equipe de Desenvolvimento sobre diferentes configurações de prioridade, e o Time de Desenvolvimento não tem permissão para agir sobre o que outras pessoas disserem.

SCRUM – Personagens

- **Product Owner**

Fase	Atividade
PRE-GAME	• Identificar as necessidades Estratégicas do Projeto (Patrocinador, Time, Infraestrutura, Áreas Envolvidas, etc) • Realizar Kick-Off • Descobrir a visão do produto e elaborar artefatos necessários • Descobrir os requisitos para o Product Backlog • Organizar e priorizar o Product Backlog
GAME	• Participar de todas as Sprint Planning Meeting e Reviews. Quando necessário visitar a reunião diária e participar de Retrospective (geralmente, quando convidado pelo Time) • Estar disponível para o Time (ou garantir que alguém designado por ele esteja) • Elaborar Plano de Release • Manter o Product Backlog • Atualizar o Plano de Release
POST-GAME	• Project Retrospective • Tornar resultados visíveis para outros (e futuros) projetos Scrum na empresa; e/ou para a Enterprise Scrum

SCRUM – Personagens

- **Product Owner – Problemas Comuns...**
  - P.O. sem poder de decisão
  - P.O. com baixa disponibilidade
  - O "metade" P.O.
  - P.O. distante
  - P.O. "proxy"
  - P.O. "da sua parte"

SCRUM – Personagens

Equipe de Desenvolvimento

- A Equipe de Desenvolvimento consiste de profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto "Pronto" ao final de cada Sprint. Somente integrantes da Equipe de Desenvolvimento criam incrementos.
- As Equipes de Desenvolvimento são estruturadas e autorizadas pela organização para organizar e gerenciar seu próprio trabalho. A sinergia resultante aperfeiçoa a eficiência e a eficácia da Equipe de Desenvolvimento como um todo.

SCRUM – Personagens

- As **Equipes de Desenvolvimento** tem as seguintes características:
  - Eles são auto-organizadas. Ninguém (nem mesmo o Scrum Master) diz a Equipe de Desenvolvimento como transformar o Backlog do Produto em incrementos de funcionalidades potencialmente utilizáveis;
  - Equipes de Desenvolvimento são multifuncionais, possuindo todas as habilidades necessárias, enquanto equipe, para criar o incremento do Produto.
  - O Scrum não reconhece títulos para os integrantes da Equipe de Desenvolvimento que não seja o Desenvolvedor, independentemente do trabalho que está sendo realizado pela pessoa; Não há exceções para esta regra.
  - Individualmente os integrantes da Equipe de Desenvolvimento podem ter habilidades especializadas e área de especialização, mas a responsabilidade pertence à Equipe de Desenvolvimento como um todo; e,
  - Equipes de Desenvolvimento não contém sub-equipes dedicadas a domínios específicos de conhecimento, tais como teste ou análise de negócios.

SCRUM – Personagens

- **Tamanho da Equipe de Desenvolvimento**
  - O tamanho ideal da Equipe de Desenvolvimento é pequeno o suficiente para se manter ágil e grande o suficiente para completar uma parcela significativa do trabalho.
  - Menos de três integrantes na Equipe de Desenvolvimento diminuem a interação e resultam em um menor ganho de produtividade. Equipes de desenvolvimento menores podem encontrar restrições de habilidades durante a Sprint, gerando uma Equipe de Desenvolvimento incapaz de entregar um incremento potencialmente utilizável.
  - Havendo mais de nove integrantes é exigida muita coordenação. Equipes de Desenvolvimento grandes geram muita complexidade para um processo empírico gerenciar.
  - Os papéis de Product Owner e de Scrum Master não são incluídos nesta contagem, ao menos que eles também executem o trabalho do Backlog da Sprint.

SCRUM – Personagens

- **O Scrum Master**
- O Scrum Master é responsável por garantir que o Scrum seja entendido e aplicado. O Scrum Master faz isso para garantir que o Time Scrum adere à teoria, práticas e regras do Scrum. O Scrum Master é um servo-líder para o Time Scrum.
- O Scrum Master ajuda aqueles que estão fora do Time Scrum a entender quais as suas interações com o Time Scrum são úteis e quais não são. O Scrum Master ajuda todos a mudarem estas interações para maximizar o valor criado pelo Time Scrum.

SCRUM – Personagens

- **O Scrum Master trabalhando para o Product Owner**
- O Scrum Master serve o Product Owner de várias maneiras, incluindo:
  - Encontrando técnicas para o gerenciamento efetivo do Backlog do Produto;
  - Claramente comunicar a visão, objetivo e itens do Backlog do Produto para a Equipe de Desenvolvimento;
  - Ensinar a Time Scrum a criar itens de Backlog do Produto de forma clara e concisa;
  - Compreender a longo-prazo o planejamento do Produto no ambiente empírico;
  - Compreender e praticar a agilidade; e,
  - Facilitar os eventos Scrum conforme exigidos ou necessários.

SCRUM – Personagens

- **O Scrum Master trabalhando para a Equipe de Desenvolvimento**
- O Scrum Master serve a Equipe de Desenvolvimento de várias maneiras, incluindo:
  - Treinar a Equipe de Desenvolvimento em auto-gerenciamento e interdisciplinaridade;
  - Ensinar e liderar a Equipe de Desenvolvimento na criação de produtos de alto valor;
  - Remover impedimentos para o progresso da Equipe de Desenvolvimento;
  - Facilitar os eventos Scrum conforme exigidos ou necessários; e,
  - Treinar a Equipe de Desenvolvimento em ambientes organizacionais nos quais o Scrum não é totalmente adotado e compreendido.

SCRUM – Personagens

- **O Scrum Master trabalhando para a Organização**
- O Scrum Master serve a Organização de várias maneiras, incluindo:
  - Liderando e treinando a organização na adoção do Scrum;
  - Planejando implementações Scrum dentro da organização;
  - Ajudando funcionários e partes interessadas a compreender e tornar aplicável o Scrum e o desenvolvimento de produto empírico;
  - Causando mudanças que aumentam a produtividade do Time Scrum; e,
  - Trabalhando com outro Scrum Master para aumentar a eficácia da aplicação do Scrum nas organizações.

SCRUM – Personagens

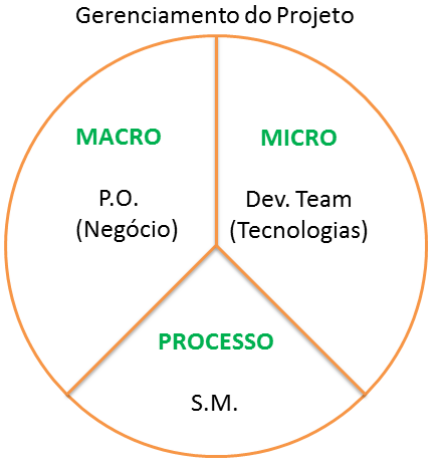
- **O Time Scrum**
- O Time Scrum é composto pelo **Product Owner**, a **Equipe de Desenvolvimento** e o **Scrum Master**.
- Times Scrum são auto-organizáveis e multifuncionais. Equipes auto-organizáveis escolhem qual a melhor forma para completarem seu trabalho, em vez de serem dirigidos por outros de fora da equipe.
- Equipes multifuncionais possuem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe.

SCRUM – Personagens

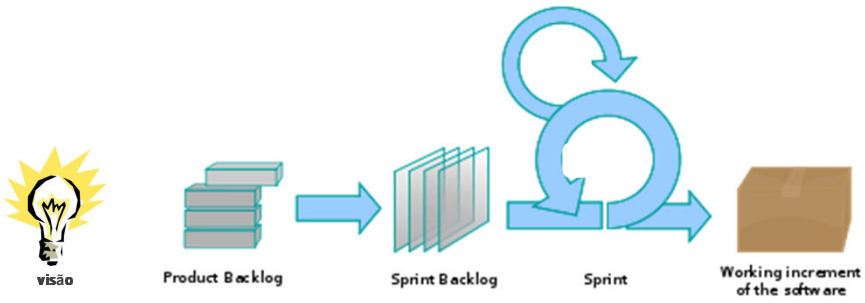
- **O Time Scrum**
- O modelo de equipe no Scrum é projetado para aperfeiçoar a flexibilidade, criatividade e produtividade.
- Times Scrum entregam produtos de forma iterativa e incremental, maximizando as oportunidades de realimentação.
- Entregas incrementais de produto "Pronto" garantem que uma versão potencialmente funcional do produto do trabalho esteja sempre disponível.

SCRUM

Resumindo tudo isso...



SCRUM



SCRUM – Visão

- **Visão é uma clara imagem que gera uma atração emocional entre pessoas e produto** (quando se fala a visão quem escuta deve ser capaz de imaginar como será o produto)
- Deve responder as seguintes perguntas:
  - Quem irá comprar este produto ? Quem é o cliente alvo ? Quem irá usar o produto ? Quem são os usuários alvo ?
  - Quais problemas do cliente (ou usuários) o produto pretende resolver ? Qual valor o produto adicionará ?
  - Quais atributos o produto deve possuir para resolver estes problemas e quais garantirão o sucesso do produto ?
  - Como o produto pode ser comparado a produtos ou alternativas existentes ? Quais são os pontos diferenciais deste produto ?
  - Qual o preço alvo do produto ? Como a empresa pretende ganhar dinheiro com este produto ? Quais serão as fontes de faturamento e qual o seu modelo de negócio ? (quando aplicável)

SCRUM – Visão

- Uma boa visão de produto permanece relativamente constante ao passo que o caminho para implementação da visão é frequentemente adaptado.

SCRUM – Visão

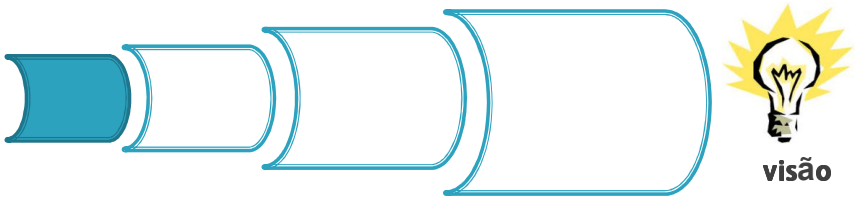
Elevator Statement

Para <*cliente/público-alvo*> que <*necessidade do cliente/público-alvo ou oportunidade*> , o <*nome do produto*> é um <*categoria do produto*> que <*principal benefício ou razão para comprar o produto*>. Diferentemente do <*principal competidor ou alternativa*> nosso produto <*principal diferenciação*> .

http://www.flyingsolo.com.au/marketing/business-marketing/preparing-your-elevator-statement

SCRUM – Visão

Elevator Statement



Praticando...

SCRUM – Visão

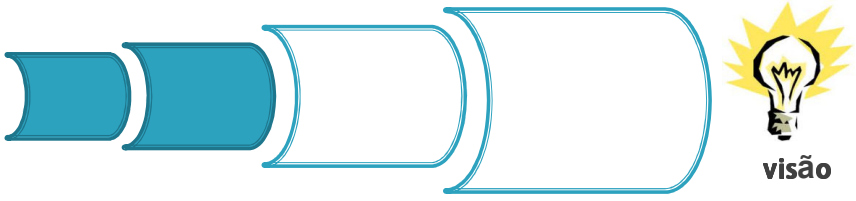
Product Vision Box

- Nome do Produto
- Gráficos
- Três ou quatro pontos chave (benefícios) para "vender" o produto
- Principais *features* no verso
- Principais requisitos operacionais

http://www.agile-ux.com/2011/03/04/a-day-in-life-of-an-agileux-practitioner-vision/

SCRUM – Visão

Product Vision Box



Praticando...

SCRUM – Visão

Product Road Map

Técnica: Remember the Future

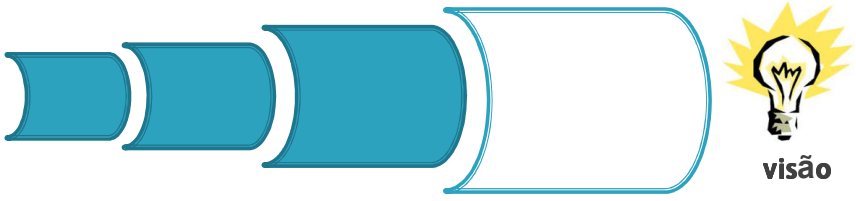
- Descobrir o entendimento do sucesso do cliente e iniciar a visualização do Road Map do produto/projeto
- Ao invés de olhar o passo a passo, se posicione no momento final desejado e "relembre" o que foi feito para chegarmos neste ponto.

OBS: NÃO é plano, NÃO é cronograma, NÃO é determinístico !

http://innovationgames.com/remember-the-future/

SCRUM – Visão

Product Road Map



Praticando...

SCRUM – Visão

Project Data Sheet

- Formalização da Visão

Project Name:	
Start Date:	
Clients	

Project Objective Statement:

Trade Off Matrix

	Fixed	Flexible	Accept	Target
Escope			X	1.250 pts
schedu		X		± 6 weeks
Budget			X	± U$5.000
Quality			X	
...				

Strategic Points

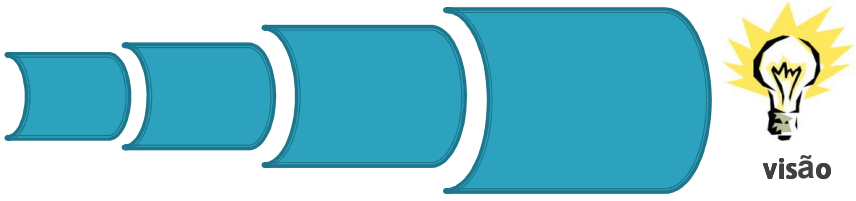
Client Benefits

Performance Attributes

Product Architecture

SCRUM – Visão

Project Data Sheet



Praticando...

SCRUM – Preparando o Product Backlog

User Story

Levantamento de
Requisitos Tradicional

Premissas

Cliente:
Linguagem de Negócio

Analista: Transforma
em linguagem técnica

Dev:
Linguagem técnica vira
códigos

SCRUM – Preparando o Product Backlog

User Story

Levantamento de
Requisitos Tradicional

Premissa

Clien
Lingu

Levantamento de Requisitos Ágeis

- Considera as premissas falsas !
- NÃO é necessário requisitos detalhados

Dev:
Linguagem técnica vira
códigos

SCRUM – Preparando o Product Backlog

User Story

3 C's

CARTÃO

CONVERSAS

CONFIRMAÇÃO

SCRUM – Preparando o Product Backlog

User Story

Independente
Negociável
Valiosa para usuários e clientes
Estimável
Small(pequena)
Testável

SCRUM – Preparando o Product Backlog

User Story

QUEM ?

Como um <PERFIL> eu posso /gostaria/devo <FUNÇÃO> para <VALOR AO NEGÓCIO>

O QUE ?

POR QUE ?

Como um P.O. eu devo REVISAR OS CONCEITOS DE SCRUM para FACILITAR O APRENDIZADO DE TÓPICOS AVANÇADOS

SCRUM – Preparando o Product Backlog

History Case

A ideia é gerar "Persona" e responder:

Como um ...

Quero ...

Para ...

Ex: Panela para preparar Salmão...

Como um Cozinheiro **(Usuário)**

Quero panela de inox, com fundo oval e ... **(Funcionalidade)**

Para cozinhar um salmão **(Valor de Negócio)**

SCRUM – Preparando o Product Backlog

History Case

A ideia é gerar "Persona" e responder:

Pagamento de Boleto: **RUIM**

Para que o comprador possa pagar sem cartão de crédito
Como comprador
Quero que o sistema de suporte a emissão de boletos

Pagamento de Boleto:

Para que eu consiga comprar produtos nessa loja
Como comprador que não tem cartão de crédito
Quero que o sistema de suporte a pagamento em boleto

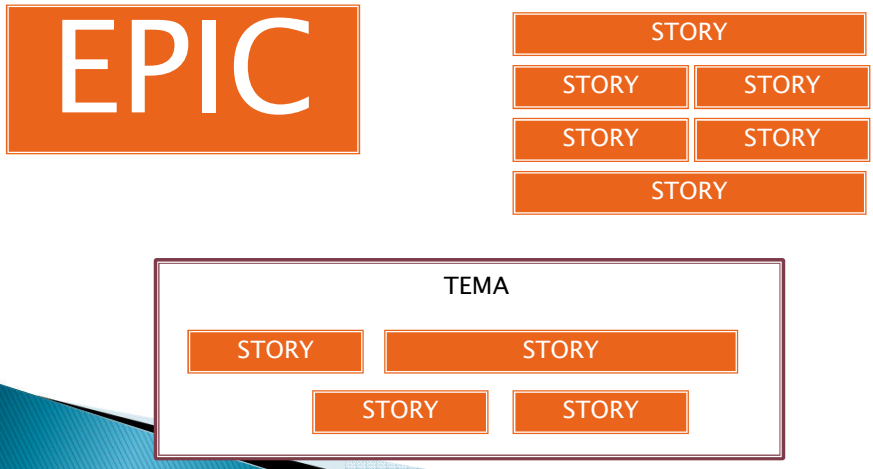
SCRUM – Preparando o Product Backlog

Teste de Aceitação

- O P.O., com a colaboração de quem achar necessário, é quem deve escrever os Testes de Aceitação, e deve fazê-lo antes da codificação

SCRUM – Preparando o Product Backlog

Stories, Temas e Epics



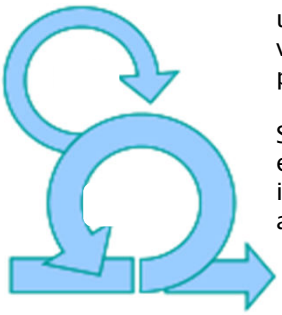
SCRUM – Preparando o Product Backlog

Priorização do Product Backlog

- **Valor**: Um item é valioso se é necessário para que o produto seja lançado.
- **Conhecimento, incerteza e riscos**: Como esses itens influenciam o sucesso do produto, eles devem ser sempre considerados como de alta prioridade
- **Capacidade para lançamento**: Itens que nos permitam mais rapidamente lançar um release de produto devem possuir boa prioridade
- **Dependência**: dependência de itens em um Product Backlog é um fato, e deve ser considerado ...

- A priorização por TEMAS deve ser considerado

SCRUM



Sprint

O coração do Scrum é a **Sprint**, um *time-box* de um mês ou menos, durante o qual um "Pronto", versão incremental potencialmente utilizável do produto, é criado.

Sprints tem durações coerentes em todo o esforço de desenvolvimento. Uma nova Sprint inicia imediatamente após a conclusão da Sprint anterior.

As Sprints são compostas por:

- uma reunião de planejamento da Sprint,
- reuniões diárias, (Daily Scrum)
- o trabalho de desenvolvimento,
- uma revisão da Sprint, (Review)
- restrospectiva da Sprint. (Retrospective)

SCRUM

Sprint

Durante a Sprint:

- Não são feitas mudanças que podem afetar o objetivo da Sprint;
- A composição da Equipe de Desenvolvimento permanecem constantes;
- As metas de qualidade não diminuem; e,
- O escopo pode ser esclarecido e renegociado entre o Product Owner e a Equipe de Desenvolvimento.

Cada Sprint tem a definição do que é para ser construído, um plano projetado e flexível que irá guiar a construção, o trabalho e o resultado do produto.

SCRUM – Planning Meeting

- O Planning Meeting é o time-boxed e deve ocupar não mais de 5% do tempo do Sprint, se o Sprint é de 2 semanas, essa reunião não deve consumir mais de 4 horas.
- O objetivo é definir as histórias que serão feitas no Sprint que acaba de começar:
- Apresentação da História
  - O P.O. apresenta a visão de negócio dos itens mais prioritários do Product Backlog aos desenvolvedores. (Ex: History Cases or Use Cases)
- Dúvidas do Negócio
  - Os desenvolvedores tiram suas dúvidas sobre as histórias, em termos da lógica de negócio – não entram em questões técnicas
- Opcional
  - O Product Owner deve sair da sala, caso permaneça ele não deve emitir opiniões para o próximo passo

SCRUM – Planning Meeting

- Pontuação
  - Os desenvolvedores dão pontos à cada uma das histórias, neste momento se considera a parte técnica. (Ex: Pontuação por Planning Poker)
- Sprint Backlog
  - Os desenvolvedores apresentam a pontuação para o P.O. e baseado na pontuação e na capacidade de atendimento por pontos por Sprint da equipe escolhe as histórias mais prioritárias (negociando com os desenvolvedores, se necessário).
- Definição de Metas
  - Se não tiver sido definida durante o processo, o P.O. define a meta do Sprint.

SCRUM – Daily scrum

Reuniões diárias de no máximo 15 minutos. Reunião breve e informal , deve acontecer sempre no mesmo horário e local combinados e dela participam apenas o TIME.

Funcionamento:

- No horário combinado, cada membro vai ao local combinado
- Todos de PÉ, respondem as seguintes perguntas:
  - O que fiz desde o último Scrum
  - O que farei até o próximo Scrum
  - Quais problemas estão me atrapalhando
- Se alguém tiver uma sugestão breve, se identifica para que, após o daily scrum, os interessados se reunam para resolver o problema juntos.

SCRUM – Review

A Revisão da Sprint é executada no final da Sprint para inspecionar o incremento e adaptar o Backlog do Produto se necessário.

Durante a reunião de Revisão da Sprint o Time Scrum e as partes interessadas colaboram sobre o que foi feito na Sprint. Com base nisso e em qualquer mudança no Backlog do Produto durante a Sprint, os participantes colaboram nas próximas coisas que precisam ser prontas.

Esta é uma reunião informal, e a apresentação do incremento destina-se a motivar e obter comentários e promover a colaboração.

Esta é uma reunião tem um *time-boxed* de 2.5% do Sprint. Por exemplo, uma Sprint de duas semanas tem Reuniões de Revisão de duas horas.

SCRUM – Review

A Reunião de Revisão inclui os seguintes elementos:

· O Product Owner identifica o que foi "Pronto" e o que não foi "Pronto";

· A Equipe de Desenvolvimento discute o que foi bem durante a Sprint, quais problemas ocorreram dentro da Sprint, e como estes problemas foram resolvidos;

· A Equipe de Desenvolvimento demonstra o trabalho que está "Pronto" e responde as questões sobre o incremento;

· O Product Owner discute o Backlog do Produto tal como está. Ele (ou ela) projeta as prováveis datas de conclusão baseado no progresso até a data; e,

· O grupo todo colabora sobre o que fazer a seguir, e é assim que a Reunião de Revisão da Sprint fornece valiosas entradas para a Reunião de Planejamento da próxima Sprint.

SCRUM – Review

· O resultado da Reunião de Revisão da Sprint é um Backlog do Produto revisado que define o provável Backlog do Produto para a próxima Sprint.

O Backlog do Produto pode também ser ajustado completamente para atender novas oportunidades.

Dicas:
- O sucesso do Review é baseado na meta do Sprint
- Garantir que cliente, mas principalmente usuário aprove o produto
- Apresentar o produto sem influenciar o usuário
- Usuário deve dar feedback do uso
- **NÃO !** Se codifica no Review
- Caso encontre BUGs, eles e novas ideias voltam para o Backlog

SCRUM – Retrospective

- Esta é uma reunião tem um *time-boxed* de 3.75% do Sprint.
- É o momento de reflexão e exposição de problemas de um time e, portanto , é o momento no qual se melhora o processo, evidencia-se e resolve-se problemas que afligem a equipe.

O propósito da Retrospectiva da Sprint é:

- Inspecionar como a última Sprint foi em relação as pessoas, relações, processos e ferramentas;
- Identificar e ordenar os principais itens que foram bem e as potenciais melhorias; e,
- Criar um plano para implementar melhorias no modo que o Time Scrum faz seu trabalho;

SCRUM – Retrospective

O Scrum Master encouraja o Time Scrum a melhorar, dentro do processo do *framework* do Scrum, o processo de desenvolvimento e as práticas para fazê-lo mais efetivo e agradável para a próxima Sprint.

Durante cada Retrospectiva da Sprint, o Time Scrum planeja formas de aumentar a qualidade do produto, adaptando a definição de "Pronto" quando apropriado.

Ao final da Retrospectiva da Sprint, o Time Scrum deverá ter identificado melhorias que serão implementadas na próxima Sprint.

A implementação destas melhorias na próxima Sprint é a forma de adaptação à inspeção que o Time Scrum faz a si próprio.

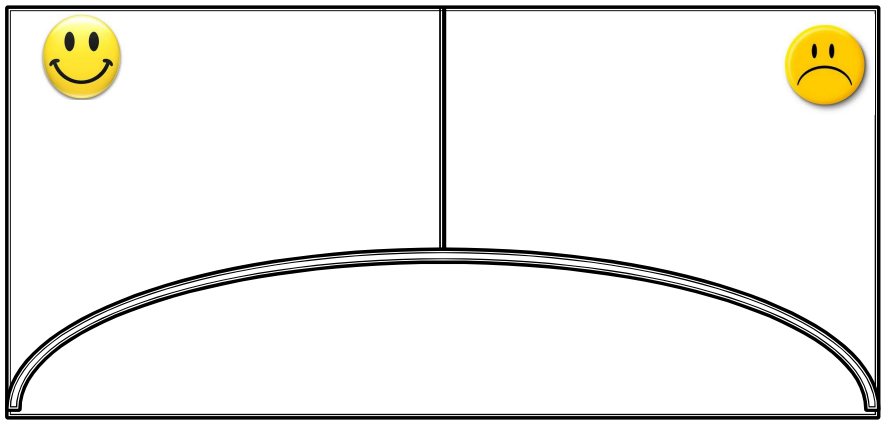
A Retrospectiva da Sprint fornece um evento dedicado e focado na inspeção e adaptação, no entanto, as melhorias podem ser adotadas a qualquer momento.

SCRUM – Retrospective

Existem diversas formas de trabalhar na retrospectiva, uma das mais adotadas no Brasil funciona da seguinte forma:

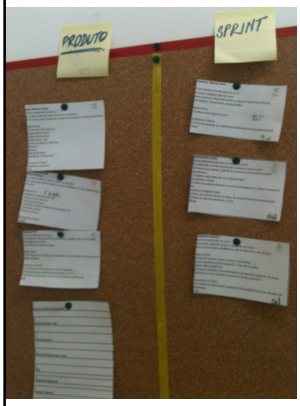
- Cada membro da caneta ganha caneta e post-it;
- Cada um, sem olhar opiniões ou conversar com outros membros do time , escreve os pontos positivos e negativos do Sprint;
- Um membro da equipe agrupa os post-its com opiniões parecidas e narra o que foi descrito. Normalmente os problemas mais sérios aparecerão mais vezes;
- A equipe discute como resolver os problemas apontados já para o próximo Sprint. Evitar problemas recorrentes
- Anota-se as soluções discutidas e as mantem visíveis durante o Sprint, como lembrete

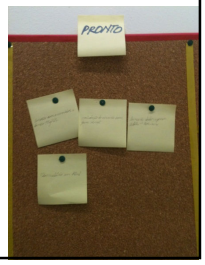
SCRUM – Retrospective



SCRUM

SCRUM





SCRUM

- O que pode alterar o tamanho da Sprint ?

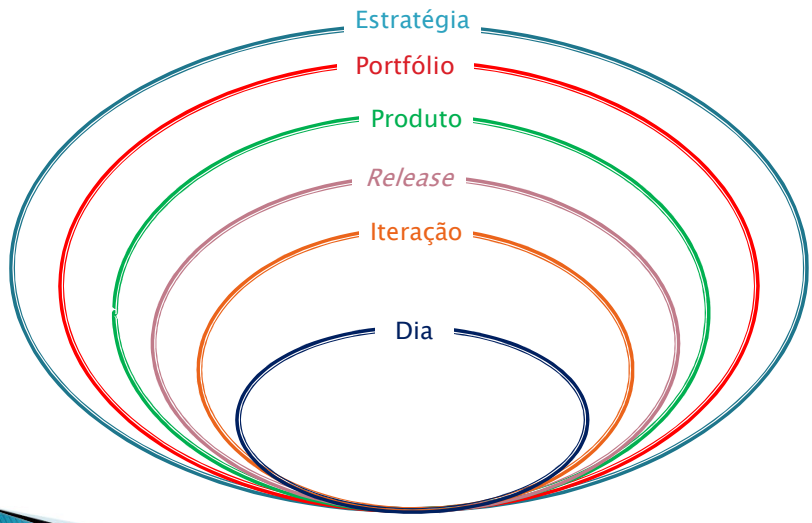
SCRUM

- O que pode alterar o tamanho da Sprint ?
  - Escopo
  - Tamanho do time
  - Disponibilidade do cliente
  - Conhecimento do time sobre agilidade
  - Conhecimento do time sobre a tecnologia
  - Times novos
  - ...

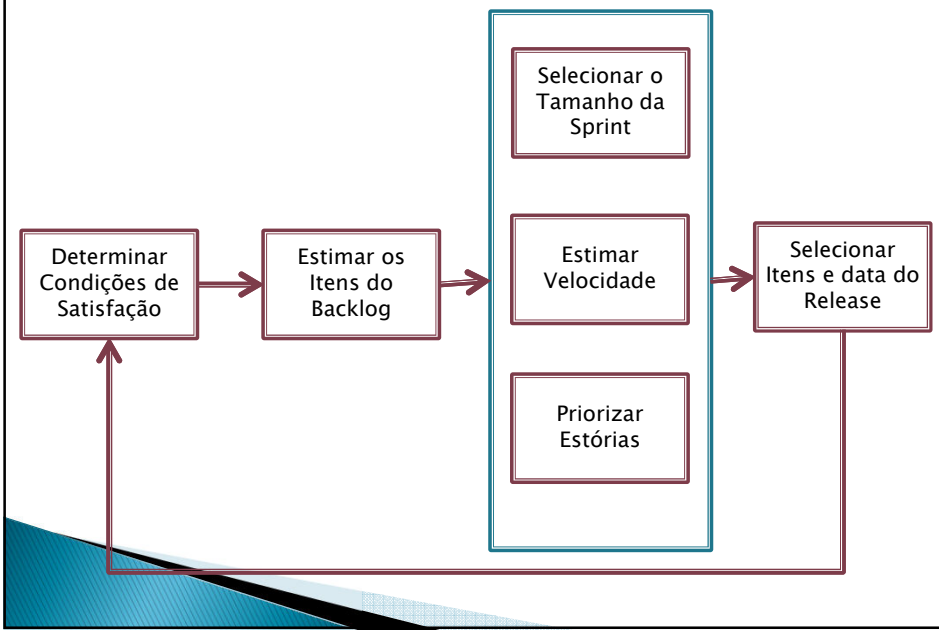
SCRUM

- Dica:
  - Quanto maior o tempo de feedback pior será... O lean possui um conceito de Fail Fast que afirma que quanto mais rápido falhar, melhor para a mudança de estratégia.

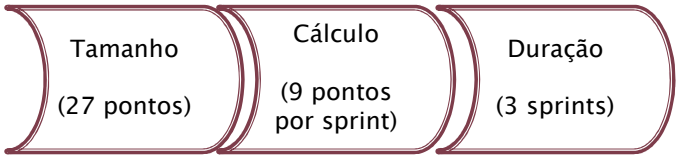
SCRUM – Cebola do Planejamento



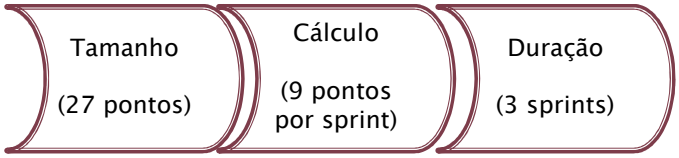
SCRUM – Planejamento de Release



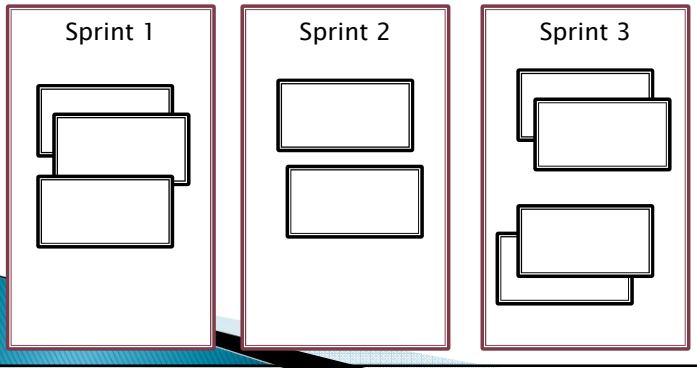
SCRUM – Planejamento de Release



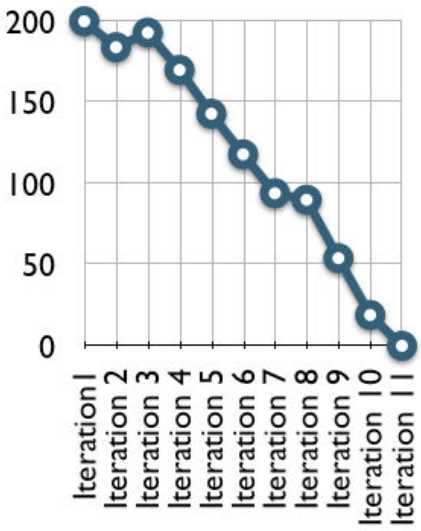
SCRUM – Planejamento de Release



Resultado



SCRUM – Release Burndown



Fonte: http://www.mountaingoatsoftware.com/scrum/release-burndown

SCRUM – Sprint Burndown

Fonte: http://www.scrumalliance.org/articles/39-glossary-of-scrum-terms#1116

eXtreme Programming

Os Valores em XP são conceitos não tangíveis que acredita-se fazer uma grande diferença na qualidade final do produto e na motivação dos times. Eles são:

- Comunicação
- Feedback
- Coragem
- Simplicidade

eXtreme Programming

Valores:

- **Comunicação**
  - O valor da comunicação é visto dentro do time, entre seus membros, e entre o time e o cliente. Ambos tem igual importância.
  - A comunicação deve ser:
    - DIRETA
    - EFICAZ
    - ESCLARECEDORA

eXtreme Programming

Valores:

- **Feedback**
  - É um valor que engloba as relações interpessoais, mas também se refere ao feedback que o próprio código do projeto devolve aos membros do time
  - Para que o feedback do código funcione bem, são necessários testes automatizados de unidade e um servidor de integração contínua para que os testes mais longos sejam rodados com frequência e, se quebrarem, sinalizem uma inconsistência.
    - Com o feedback o cliente pode:
      - Identificar erros rapidamente
      - Definir prioridades

eXtreme Programming

Valores:

- **Coragem**
  - O XP prega que os desenvolvedores precisam ter coragem para refatorar o código em prol de melhorias em clareza e design – e nada melhor para dar coragem do que testes automatizados.
  - Coragem é também apagar o código, mesmo funcionalidades inteiras, não importa o trabalho que tenha sido empregado para desenvolve-la.
  - Coragem para não tentar prever o futuro, mas sim focar no que é realmente necessário no momento. XP associa a essa idéia a sigla YAGNI (you ain't gonna need it)

eXtreme Programming

Valores:

- **Simplicidade**
  - Considere que , na média, o tempo de construção de um software é cerca de 30% do tempo investido nele. Os outros 70% são dedicados a manutenção do sistema.
  - Num cenário como esse, a simplicidade é essencial para tornar esse período maior muito agradável.
  - O desenvolvedor deve:
    - Implementar apenas o básico
    - Não antecipar funcionalidades

eXtreme Programming

Valores:

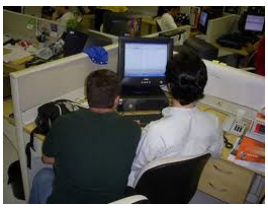
- **Propriedade Coletiva do Código**
- **Programação Pareada**
- **Testes Automatizados e test first**
- **Test Driven Design (TDD)**
- **Integração contínua e Deploy Contínuo**
- **Refatoração Constante**

eXtreme Programming

Valores:

- **Propriedade Coletiva do Código**
  - É comum que desenvolvedores trabalhem em partes independentes do código. A consequência desta abordagem é que cada desenvolvedor se sente responsável apenas por sua parte...
  - O ideal é o sentimento de time onde todos são responsáveis pelo código. Assim, um desenvolvedor é livre para interferir em qualquer parte do código sem irritar o "dono" do código.

eXtreme Programming



Valores:

- **Programação Pareada (pair programming)**
  - A programação em par é uma forma eficaz de reduzir a incidência de bugs em um sistema.
  - Quem trabalha continuamente com programação em par se habitua a corrigir e ter seu trabalho corrigido dezenas de vezes ao dia. A incidência de erros identificados pelo colega costuma ser tão elevada que surpreende quem não está acostumado ao uso da técnica.
  - Equipes que trabalham em par conseguem reduzir drasticamente a inserção de defeitos em seus códigos.
  - A programação em par ajuda os desenvolvedores a criarem soluções mais simples, mais rápidas de implementar e mais fáceis de manter.

eXtreme Programming

Valores:

- **Programação Pareada (pair programming)**
  - A programação em par também é uma forma de fazer com que o desenvolvedor tenha mais confiança no código que produz.
  - O conjunto de características apresentadas acima faz com que a programação em par acelere o desenvolvimento significativamente, embora à primeira vista pareça o contrário.

Programar em par exige que as pessoas envolvidas sejam receptivas, compreensivas umas com as outras, engajadas e, sobretudo, humildes. É necessário aceitar que somos falíveis para que possamos programar em par. Weinberg criou o termo egoless programming, ou seja, programação sem ego.

eXtreme Programming

Valores:

- **Testes Automatizados e test first**
  - Um dos grandes desafios é trabalhar em códigos antigos.... O XP prega a refatoração constante ! Mas quanto maior o projeto, maior é a quantidade de código não escrita por nós ou antigo, o que aumenta a insegurança de refatorar o código...impedindo a evolução do projeto.

eXtreme Programming

Valores:

- **Testes Automatizados e test first**
  - O XP prega o uso extensivo de testes automatizados que descrevem o comportamento de uma funcionalidade, preferencialmente escritos antes mesmo do código que eles testam, prática que recebe o nome de **desenvolvimento dirigido por testes (Test Driven Development – TDD).**
  - Os testes automatizados tem 2 funções importantes:
    - **Permitir refatoração** : podemos refatorar o código com mais segurança. Podemos alterar o código e verificar automaticamente se o software continua funcionando.
    - **Documentar**: Os testes devem ter nomes que explicam quais funcionalidades eles testam, assim ao executar o código, o desenvolvedor sabe quais funcionalidades foram implementadas e qual o comportamento esperado pelo código.

eXtreme Programming

Valores:

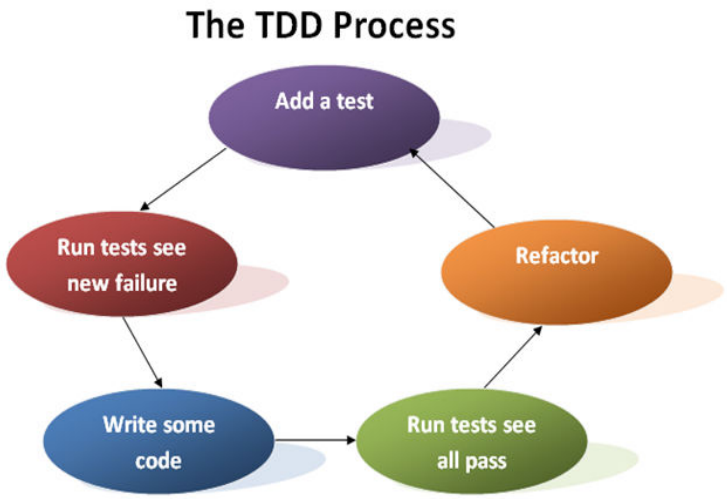
- **Test Driven Design (TDD)**
  - **" Queremos que os nossos testes guiem o próprio design das classes do sistema "**

O processo TDD funciona da seguinte maneira:

- Faz-se o teste automatizado para o caso mais simples
- Roda-se o teste (que não deverá passar pois a funcionalidade ainda não foi implementada)
- Implementa-se através da mudança mais simples possível que faça o teste passar
- Se o código não estiver o melhor possível:
  - Refatora
  - Certifique-se que os testes continuem passando
- Se o código estiver bom
  - Volte para o primeiro item com o próximo teste mais simples

@ribeirord

eXtreme Programming



eXtreme Programming

Valores:

- Integração contínua e Deploy Contínuo

eXtreme Programming

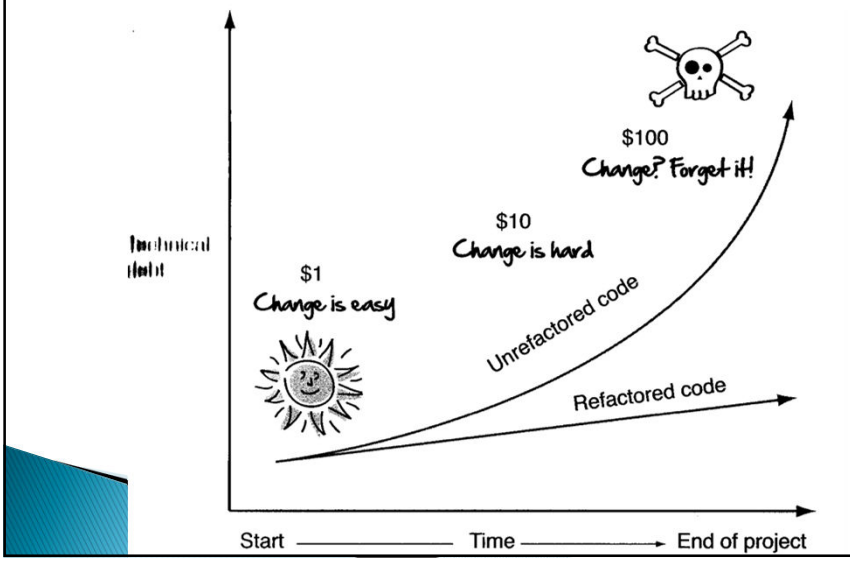
Valores:

- Refatoração Constante
  - Com o aumento do projeto é comum que pequenas partes de código mal escrito se acumulem e, quando menos se esperar, compromete todo o projeto. (*Big Ball of Mud – Joe Yoder*)
  - A melhor forma de evitar este problema é através de pequenas refatorações constantes.

"Refatoração é uma técnica controlada para reestruturar um trecho de código existente, alterando sua estrutura interna sem modificar seu comportamento externo. " (Martin Fowler)

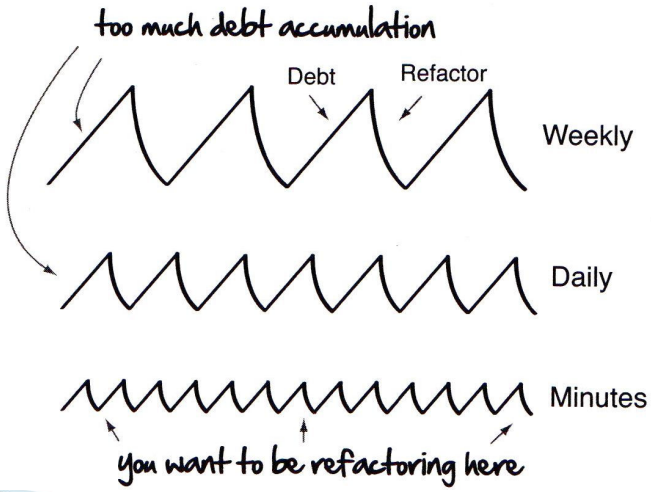
eXtreme Programming

- **Refatoração Constante**



eXtreme Programming

- **Refatoração Constante**



Estudo de Caso – Simulador de BlackJack

O blackjack é jogado com um ou mais baralhos de **52 cartas**, para cujos valores será designado um total de pontos.

As cartas de 2 a 10 terão o valor dos números. Reis, damas e valetes valem 10 pontos cada e ases poderão ser usados tanto como 1 ou 11.

O objetivo para o jogador será comprar cartas até um total de 21 pontos (sem exceder), vencendo o total das cartas do distribuidor.

Estudo de Caso – Simulador de BlackJack

Design a deck of cards

```
public class Deck
{
  private readonly Ilist<Card> cards = new List<Card>( );

public Deck( )
{
cards.Add(Card.Dois_de_Copas);
cards.Add(Card.Tres_de_Copas);
//..restante de copas

cards.Add(Card.Dois_de_Ouros);
cards.Add(Card.Tres_de_Ouros);
//..restante de ouros

cards.Add(Card.Dois_de_Espadas);
cards.Add(Card.Tres_de_Espadas);
//..restante de Espadas

cards.Add(Card.Dois_de_Paus);
cards.Add(Card.Tres_de_Paus);
//..restante de paus

//Coringa
cards.Add(Card.Coringa);
}
```

Estudo de Caso – Simulador de BlackJack

BUG DESCOBERTO !

Não existe **Coringa** no Black Jack

Como testar automaticamente em cada novo incremento para que problemas assim não ocorram ?

http://junit.sourceforge.net/doc/testinfected/testing.htm

Estudo de Caso – Simulador de BlackJack

```
public class DeckTest
{

    public void Verify_Deck_contains_52_cards( )
    {
        var deck = new Deck( );
        Assert.AreEqual(52,deck.Count( ) );
     }
}
```

Estudo de Caso – Simulador de BlackJack

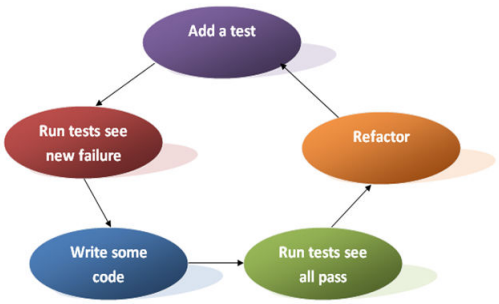
Criar Baralho de Cartas .
.
.
.

Critérios de Teste .
52 cartas no Baralho.
13 Cartas de Cada Naipe.
Não pode existir carta Coringa.
Uma carta de Cada Tipo

Estudo de Caso – Simulador de BlackJack

Critérios de Teste .
52 cartas no Baralho.
13 Cartas de Cada Naipe.
Não pode existir carta Coringa.
Uma carta de Cada Tipo

The TDD Process



Estudo de Caso – Simulador de BlackJack

Critérios de Teste .
52 cartas no Baralho. OK
13 Cartas de Cada Naipe.
Não pode existir carta Coringa.
Uma carta de Cada Tipo

```
public class DeckTest2
{

    public void Verify_Deck_contains_52_cards( )
    {
       var deck = new Deck( );
       Assert.AreEqual(52,deck.Count( ) );
     }
}
```

Estudo de Caso – Simulador de BlackJack

```
public class DeckTest2
{

    public void Verify_Deck_contains_52_cards( )
    {
       var deck = new Deck( );
       Assert.AreEqual(52,deck.Count( ) );
     }


 public void Verify_Deck_contains_13_cards_for_each_suit( )
    {
       var deck = new Deck( );
       Assert.AreEqual(13,deck.NumberofCopas( ) );
       Assert.AreEqual(13,deck.NumberofEspadas( ) );
       Assert.AreEqual(13,deck.NumberofOuros( ) );
       Assert.AreEqual(13,deck.NumberofPaus( ) );

     }
}
```

Critérios de Teste .
52 cartas no Baralho. OK
13 Cartas de Cada Naipe. OK
Não pode existir carta Coringa.
Uma carta de Cada Tipo

Estudo de Caso – Simulador de BlackJack

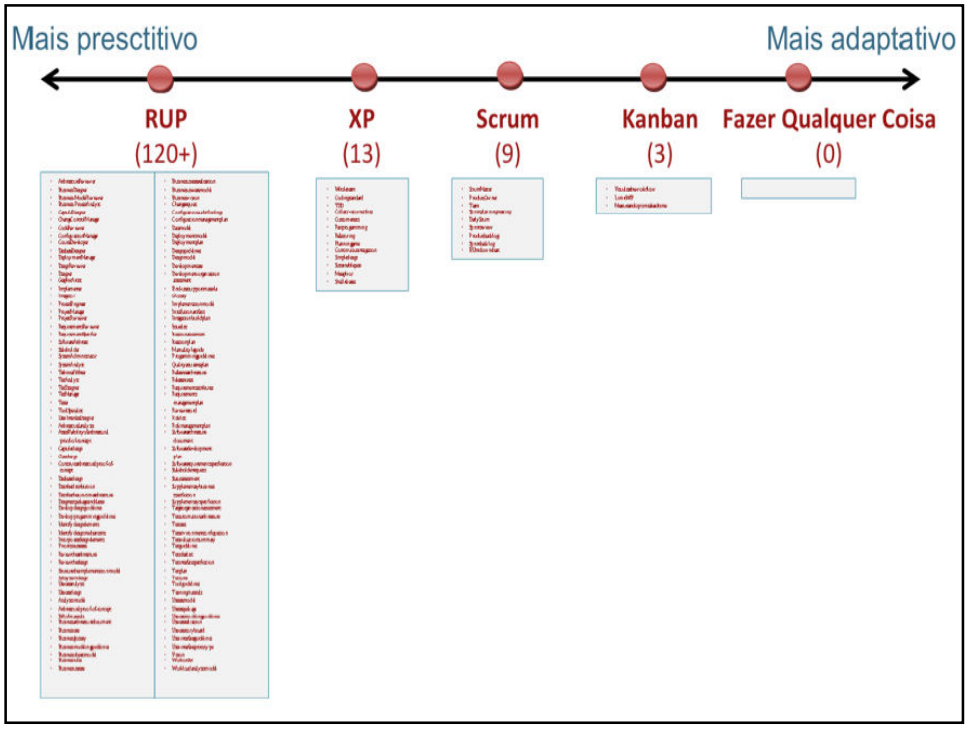
```
public class DeckTest2
{

//..Continuação

 public void Verify_Deck_contains_no_joker( )
    {
       var deck = new Deck( );
       Assert.IsFalse(deck.Contains(Card.Coringa ) );
     }


}
```

Critérios de Teste .
52 cartas no Baralho. OK
13 Cartas de Cada Naipe. OK
Não pode existir carta Coringa. OK
Uma carta de Cada Tipo

Estudo de Caso – Simulador de BlackJack

```
public class DeckTest2
{

//..Continuação

 public void Verify_Every_Card_in_the_Deck ( )
    {
       var deck = new Deck( );
       Assert.IsTrue(deck.Contains(Card. Dois_de_Copas ) );
       Assert.IsTrue(deck.Contains(Card. Tres_de_Copas ) );

       //...Testar todas as cartas válidas para cada naipe

     }


}
```

Critérios de Teste .
52 cartas no Baralho. OK
13 Cartas de Cada Naipe. OK
Não pode existir carta Coringa. OK
Uma carta de Cada Tipo OK



Planning Poker

A ideia principal por traz do Planning Poker é permitir que todos os membros do time de desenvolvimento que estão comprometidos na implementação do Sprint participem colocando a sua visão de complexidade para que juntos possam chegar a um indicador de complexidade comum para o time.

A escala de complexidade é baseada na sequência de Fibonacci (1, 2, 3, 5, 8, 13,21, ∞), ou em outra escala definida pela equipe...

0	1/2	1
2	3	5
8	13	20
40	100	∞
	?	

Planning Poker

Cada participante do time que estiver comprometido recebe um conjunto de cartas sendo cada uma com o número de complexidade.

O grupe se reúne geralmente na reunião Sprint Planning e esclarece as histórias com o PO (Product Owner) para depois estimar uma a uma.

Seguindo a ordem de sequencia das às histórias já priorizadas pelo PO.

Então o time conta até três e cada um apresenta uma das cartas ao mesmo tempo. Esse é um momento importante, pois nenhum membro pode influenciar o outro na hora de mostrar as cartas.

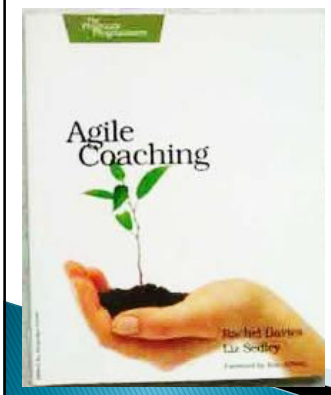
Planning Poker

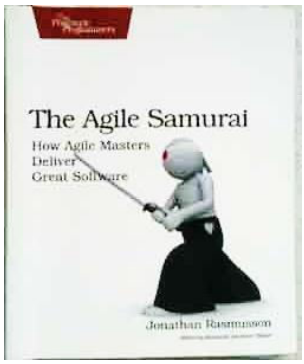
Após apresentada as cartas confere-se os números das mesmas para ver se deu tudo igual ou ocorreu divergências. Em caso de divergência cada membro pode argumentar o que o levou a pensar diferente dos demais e nesse momento pode usar os argumentos que justifique aquele item ser mais complexo ou mais simples.

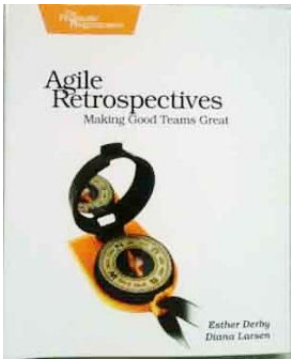
Com o tempo você vai observar opiniões do tipo “Eu já implementei uma rotina parecida em um projeto anterior” que trazem a tona o grande valor dessa técnica que é justamente fazer as pessoas serem ouvidas.

Após a apresentação dos argumentos cada pessoa que ficou na dúvida pode propor uma nova votação e mudar o seu voto para mais ou para menos conforme o novo entendimento da questão. Por isso um item que estava complexo pode ser finalizado com mais simples ou vice-versa prevalecendo o entendimento do time sobre a questão.

Leitura Complementar







Dúvidas ou Sugestões de Melhorias, entre em contato: rafaeldiasribeiro@gmail.com