



Unidade 3 – Princípios de Desenvolvimento Ágil

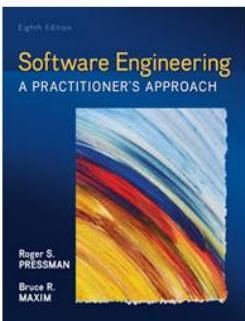


Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

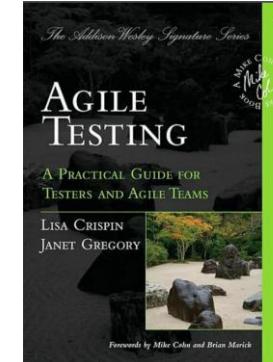
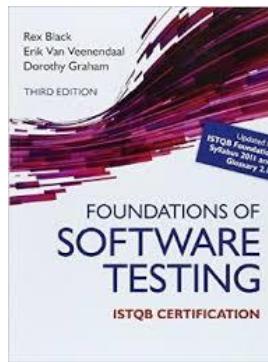
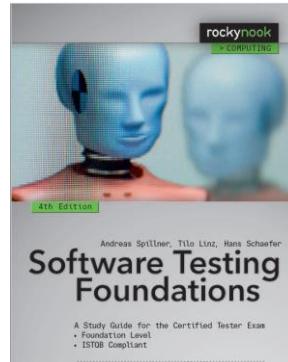


Bibliografia

- **Software Engineering – A Practitioner's Approach – Roger S. Pressman – Eight Edition – 2014**
- **Software Engineering – Ian Sommerville – 10th edition - 2015**
- Engenharia de Software – Uma abordagem profissional – Roger Pressman - McGraw Hill, Sétima Edição - 2011
- Engenharia de Software – Ian Sommerville – Nona Edição – Addison Wesley, 2007
- Software Testing Foundations – Andreas Spillner , 2014 – 4th Edition
- Foundations of Software Testing ISTQB Certification – Rex Black, 2010
- Agile Testing – Lisa Crispin, Janet Gregory, Mike Cohn Books, 2009
- **Syllabus – ISTQB – CTFL – AT – versão 2014BR**



[Software Engineering: A Practitioner's Approach, 8/e](#)





Certificação CTFL-AT

- O **Agile Tester Extension Foundation** (**CTFL-AT**) é uma abordagem relativamente nova para teste de software que segue os princípios do desenvolvimento ágil de software, conforme Manifesto Ágil;
- A certificação provê os conhecimentos necessários para que um profissional possa integrar uma equipe de testes ágeis e atingir alta performance;
- Para se submeter ao exame, o candidato deve ter a certificação **CTFL** (**Foundation Level**).



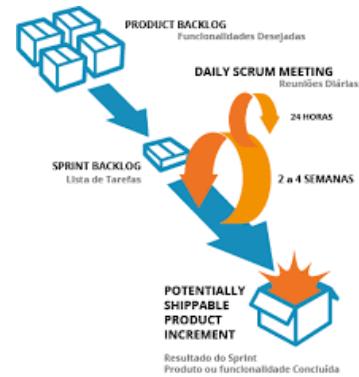


Fundamentos do Desenvolvimento de Software Ágil

Syllabus 1.1



- O teste de software é parte integrante do processo de desenvolvimento de software;
- No processo tradicional de desenvolvimento de software, a abordagem é tipicamente sequencial (waterfall), no qual os prazos típicos de entrega do produto levam meses, até anos;
- Dadas as necessidades de negócio e competitividade das empresas atuais, tais prazos são proibitivos e não recomendados;
- Com métodos ágeis, as entregas são mais sistemáticas (contínuas e frequentes) e aderentes ao negócio do cliente, tipicamente de 2 a 4 semanas entregam-se incrementos de software.
- Dentro dessa visão surgiram os processos e testes ágeis para desenvolvimento de software.





Qual a diferença entre um testador tradicional e um testador Ágil?



Testador Tradicional x Testador Ágil

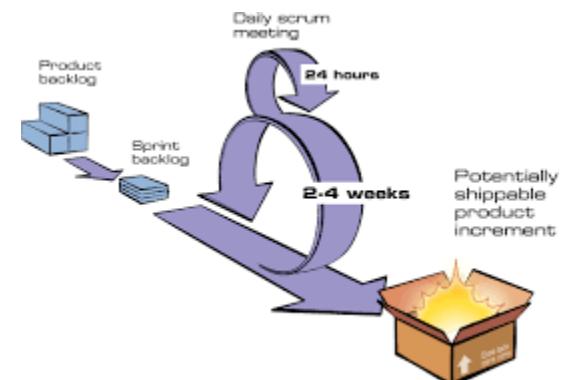
- O testador tradicional, em geral, é mais **passivo** e **menos pró-ativo**, uma vez que segue o processo tradicional de desenvolvimento de software, no qual software é gerado com atividades **sequenciais** (waterfall);
- O testador ágil, por natureza, é **pró-ativo**, uma vez que trabalha num time de projeto ágil participando de forma mais **integrada** ao time e ao projeto;
- Em projetos ágeis há uma maior troca de experiências entre o testador e os outros membros do time. O testador passa conceitos de teste ao demais profissionais e recebe também desses informações de negócio e de desenvolvimento, contribuindo assim numa **sinergia favorável** ao projeto como um todo.





Equipes Ágeis

- São **multidisciplinares**;
- São formadas por testadores, gerentes de projetos, desenvolvedores, representantes de negócio (PO – product owner), etc;
- As equipes ágeis se caracterizam por intensa sinergia, troca de experiências e devem trabalhar de forma integrada e conjunta visando a entrega do software com qualidade e dentro dos prazos;
- Foca-se na **descoberta precoce de defeitos** para privilegiar a **qualidade** do produto a ser desenvolvido.





Comprometimento x Envolvimento

- Equipe de porcos ou de frangos?



- Cada membro da equipe ágil deve estar **comprometido** com o projeto e não apenas **envolvido**...





Manifesto Ágil



5 a 10% da prova

- Não foi criado por jovens estudantes ou profissionais inexperientes!





Manifesto Ágil – Declaração de Valores



5 a 10% da prova

- Valorizam-se mais indivíduos e interações que processos e ferramentas;
- Valorizam-se mais softwares que já funcionam que documentação abrangente;
- Valorizam-se mais colaboração do cliente que negociação contratual;
- Valorizam-se mais respostas à mudanças que seguimento de plano.

O Manifesto Ágil argumenta que, embora os conceitos da direita sejam importantes, os da esquerda têm mais valor...





Indivíduos e Interações

- Desenvolvimento ágil é muito focado em **pessoas**;
- Software é feito **por** e **para pessoas**;
- Times de pessoas desenvolvem o software por meio de **intensa** e **contínua interação**;
- **Menor** dependência de **processos** ou **ferramentas**;
- **Resultado:** O trabalho se torna mais **eficaz**!





Software funcionando...

- Na perspectiva do cliente é muito mais útil e valioso **software funcionando**, mesmo com funcionalidade reduzida, do que documentação excessivamente detalhada;
- Permite **feedback** rápido à equipe de desenvolvimento;
- **Time-to-Market ?**
- Permite mudanças rápidas de ambientes de **negócio**;
- Cria ambiente mais propício para **inovação**;
- Colabora para **soluções** de problemas não esclarecidos.





Colaboração com o cliente...

- Os clientes, usualmente, encontram grandes **dificuldades** em **transmitir** para a equipe o que eles precisam;
- **Colaborar** diretamente com o cliente aumenta a probabilidade de compreender exatamente o que ele quer;
- Embora celebrar **contratos** com os clientes seja importante, trabalhar em colaboração regular e próxima com eles pode trazer mais **succeso** ao projeto.





Resposta à Mudança

- **Mudanças** são **inevitáveis** em projetos de software;
- **Raramente** um projeto de software termina da mesma forma pelo qual foi **planejado** !!!
- O **ambiente** em que a empresa opera, a **legislação**, a **concorrência** do negócio e diversos outros **fatores** podem ter grandes influências sobre o projeto e seus objetivos;
- Esses fatores devem ser **acomodados** pelo processo de desenvolvimento de software;
- Como tal, ter **flexibilidade** nas práticas para a adoção da mudança é mais importante que simplesmente seguir um plano de forma rígida.





No mundo dos negócios, as mudanças ocorrem cada vez mais rápido. Para caminhar para trás, basta ficar parado...

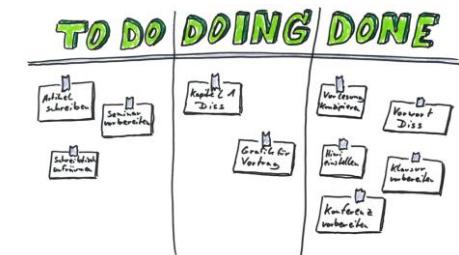
(Carlos Hilsdorf)

kdfrases.com



12 princípios do Manifesto Ágil

- Prioridade de entrega de software com satisfação ao cliente com valor agregado;
- Acolhimento aos requisitos de mudança, mesmo no final do desenvolvimento;
- Entrega contínua de versões de software, em pequena escala de tempo;
- Desenvolvedores e Pessoas de negócio devem trabalhar em conjunto;
- Equipe deve estar motivada em ambiente confortável e seguro;
- Comunicação face-a-face (o corpo fala...);
- Software em funcionamento é medida primária de progresso;
- Desenvolvedor e cliente devem estar sempre disponíveis;
- Atenção contínua à excelência técnica e bom design;
- Simplicidade;
- Trabalho em equipe (com ritmo produtivo e constante durante o projeto);
- Avaliação e ajuste da equipe em intervalos regulares.





Equipes Ágeis

- Um time precisa ter todas as **competências** e **habilidades** necessárias para o desenvolvimento do projeto;
- Cada membro do time colabora com uma parcela desse conhecimento. A soma do conhecimento de cada membro comporá o **conhecimento do time**;
- **Não há**, portanto, um membro que carrega o time nas costas;
- O time deve, portanto, ser equilibrado e atuar de forma cooperativa e colaborativa;
- Equipes ágeis são relativamente pequenas, em torno de 5 a 6 pessoas.



Agile
Software
Development





Equipes Ágeis

- A **equipe** é a **responsável** pela qualidade de um projeto ágil;
- A culpa de um fracasso **não** é de uma pessoa específica do time;
- Testadores, desenvolvedores e os representantes das empresas trabalham **juntos** em todas as etapas do processo de desenvolvimento;
- Testadores vão trabalhar em estreita **colaboração** com os desenvolvedores e representantes das empresas para garantir que os níveis de qualidade sejam alcançados.





Equipes Ágeis

- O ideal é que se compartilhe o mesmo espaço de trabalho, pois a co-localização facilita fortemente a comunicação e a interação;
- Reuniões diárias envolvendo todos os membros do time, onde o progresso do trabalho é comunicado, e quaisquer impedimentos ao progresso são realçados;
- Deve-se promover uma dinâmica de equipe mais eficaz e eficiente;
- É um dos principais benefícios do desenvolvimento ágil.





Benefícios da abordagem em equipe

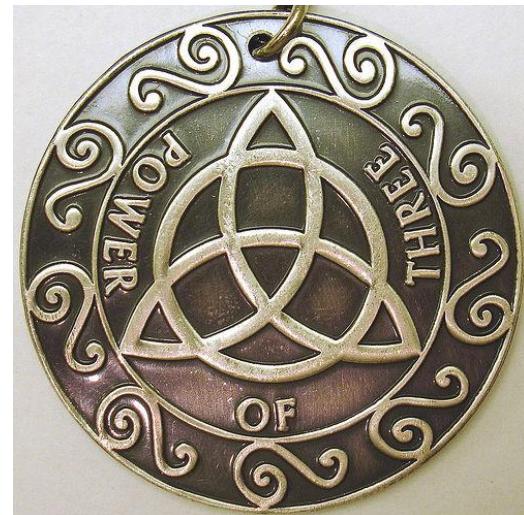
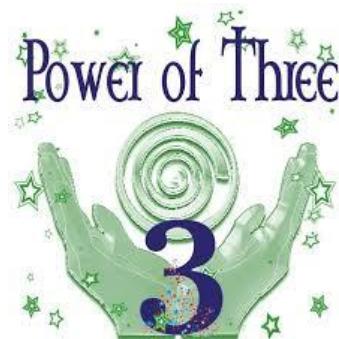
- Melhorar a comunicação e colaboração dentro da equipe;
- Ativar os vários conjuntos de habilidades dentro da equipe para serem aproveitados em benefício do projeto;
- Promover qualidade na responsabilidade de cada elemento.





O poder dos Três

- A equipe inteira está envolvida em consultas ou reuniões em que as características do produto são apresentadas, analisadas ou estimadas;
- O conceito que envolve testadores, desenvolvedores e representantes de negócio em todas as discussões é conhecido como o Poder dos Três.





Feedback inicial e Frequente

- Projetos ágeis têm **iterações curtas**, permitindo que a equipe do projeto receba **feedback** avançado e contínuo sobre a qualidade do produto durante todo o ciclo de desenvolvimento;
- No Scrum, estas iterações curtas é conhecida por **Sprint**;
- Quanto mais cedo a equipe receber feedback, de forma mais ágil o projeto será **ajustado** para atender as necessidades do cliente;
- Uma das formas de se prover feedback rápido é pela **integração contínua**.





Feedback inicial e Frequente

- Quando as abordagens de desenvolvimento **sequenciais** são utilizadas, muitas vezes o cliente **não** vê o produto até que o projeto esteja quase concluído;
- Nesse ponto, muitas vezes é **tarde** demais para a equipe de desenvolvimento tratar – de forma eficaz – todos os problemas que o cliente pode ter.





Benefícios do Feedback



- Evitar **mal-entendidos** nos requisitos, que somente podem ser detectados tarde no ciclo de desenvolvimento, quando são mais **caros** para serem resolvidos;
- Esclarecer solicitações de funcionalidades dos clientes, tornando-os disponíveis antecipadamente para uso do cliente. Desta maneira, **o produto reflete melhor o que o cliente quer**;
- Descobrir (via **integração contínua**), isolar e resolver os problemas de **qualidade** mais cedo;
- Providenciar informações para a equipe ágil quanto à sua **produtividade** e capacidade de desenvolvimento;
- Promover **fluxo de projeto consistente**.





Aspectos de Abordagem Ágil

Syllabus 1.2



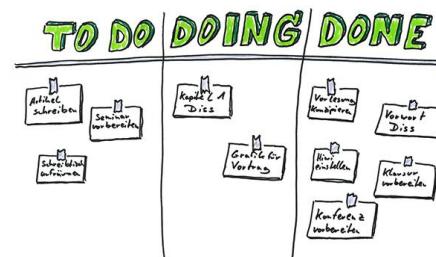
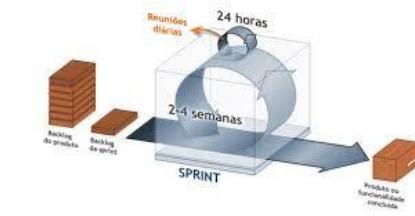
- Atualmente existem diversas abordagens ágeis em uso nas organizações;
- Práticas comuns incluem:
 - ✓ Criação colaborativa da estória do usuário;
 - ✓ Retrospectivas (rever o que foi feito);
 - ✓ Integração Contínua;
 - ✓ Planejamento para cada iteração e para liberação total.





Abordagens para CTFL-AT

- Existem várias abordagens ágeis, sendo que cada qual implementa os valores e princípios do Manifesto Ágil;
- O Syllabus **CTFL-AT** considera:
 - ✓ eXtreme Programming;
 - ✓ Scrum;
 - ✓ Kanban
- Algumas empresas interpretam erroneamente alguns desses princípios e valores.





eXtreme Programming (XP)



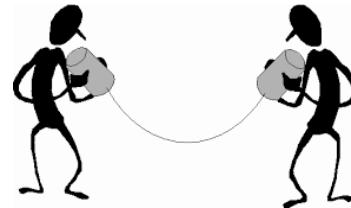
- Criado por **Kent Beck** (final da década de 90);
- É uma abordagem ágil para desenvolvimento de software descrito por:
 - ✓ 5 **valores**;
 - ✓ 14 **princípios**;
 - ✓ 13 **práticas de Desenvolvimento**.





Valores do XP

- ✓ **Comunicação;**
- ✓ **Simplicidade;**
- ✓ **Feedback;**
- ✓ **Coragem;**
- ✓ **Respeito.**





14 Princípios do XP



- | | |
|----------------------------|------------------------------------|
| ✓ Humanidade | ✓ Fluxo |
| ✓ Economia | ✓ Oportunidade |
| ✓ Benefício Mútuo | ✓ Redundância |
| ✓ Auto similaridade | ✓ Falha |
| ✓ Aperfeiçoamento | ✓ Qualidade |
| ✓ Diversidade | ✓ Primeiros Passos |
| ✓ Reflexão | ✓ Responsabilidade Assumida |



13 práticas do XP

Extreme
Programming
Agile

- ✓ Sentar-se juntos
- ✓ A equipe inteira
- ✓ Espaço de trabalho informativo
- ✓ Trabalho energizado
- ✓ Programação em pares
- ✓ Estórias
- ✓ Ciclo Semanal
- ✓ Ciclo Trimestral
- ✓ Folga
- ✓ Elaboração de Dez Minutos
- ✓ Integração Contínua
- ✓ Programação do Teste Primeiro (TDD)
- ✓ Design Incremental





Influência do XP em outras abordagens



- ✓ A maioria das abordagens de desenvolvimento do software Ágil em uso atualmente são influenciadas pelo **XP** e seus valores e princípios;
- ✓ Por exemplo, as equipes ágeis que seguem o **Scrum**, frequentemente incorporam as práticas **XP**.





SCRUM

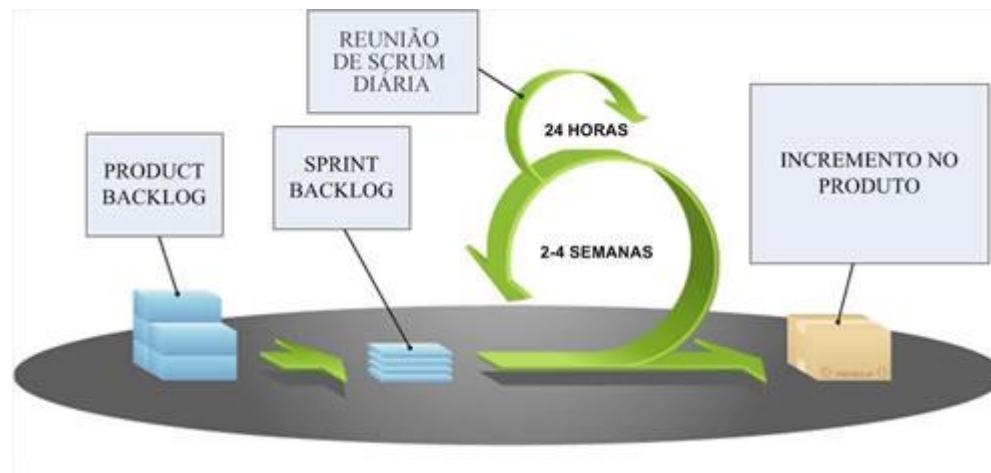
- ✓ A mais famosa das abordagens ágeis;
- ✓ Muito usada em desenvolvimento de software;
- ✓ O termo “**Scrum**” teve origem no **Rugby**. Em **Rugby**, **Scrum** corresponde a um método de reinício de jogada, onde os jogadores dos dois times se juntam com a cabeça abaixada e se empurram com o objetivo de ganhar a posse de bola;





SCRUM

- ✓ É uma estrutura de gestão **ágil**, que contém instrumentos e **práticas** iniciadas no início da década de 90.

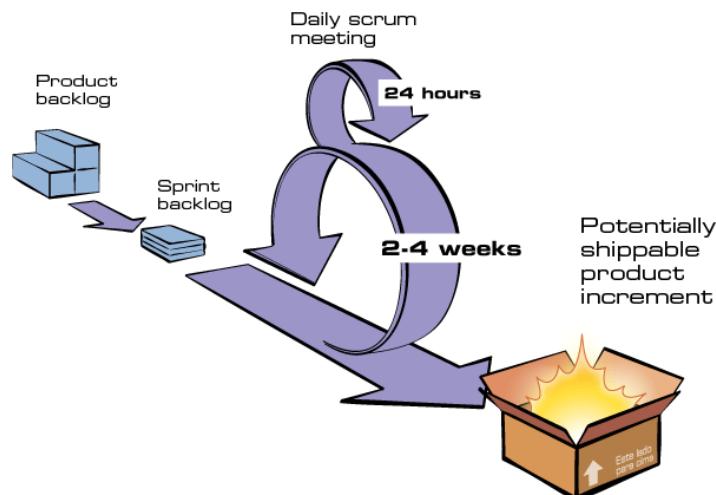




Sprint



- ✓ **Scrum** divide um projeto em iterações, chamadas **Sprint**;
- ✓ Um **Sprint** tem duração fixa, geralmente duas a quatro semanas;
- ✓ Cada **Sprint** resulta em um produto potencialmente liberável (entregável) chamado incremento.

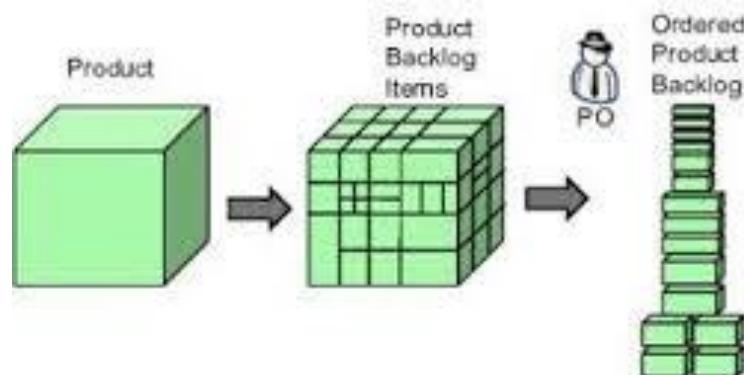




Backlog do Produto

SCRUM

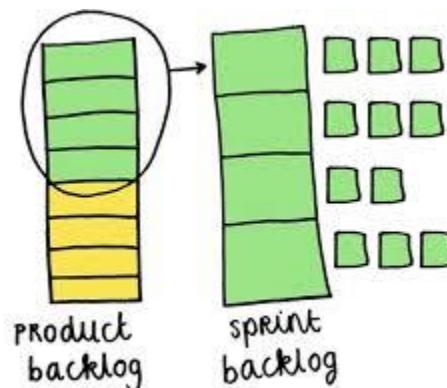
- ✓ Lista priorizada de itens de produtos planejados, gerenciada pelo **PO** – Product Owner, Representante do Produto;
- ✓ O backlog do produto evolui de Sprint para Sprint, chamado refinamento de Backlog.





Backlog do Sprint

- ✓ No início de cada Sprint, a equipe **Scrum** seleciona um conjunto de itens de prioridade mais elevada;
- ✓ A seleção é feita com base no princípio da atração, no qual prioriza-se os itens que estão relacionados visando agilidade do processo.





Timeboxing



- ✓ Apenas as tarefas, requisitos ou funcionalidades que a equipe espera concluir no Sprint fazem parte do backlog do Sprint;
- ✓ Há uma preocupação grande em se fechar as tarefas em blocos de tempo (**Timeboxing**);
- ✓ As tarefas do **Backlog Sprint** são preferencialmente divididas em tarefas menores de mesma duração, tipicamente **1 dia**;
- ✓ **Timeboxing** não se aplica apenas às tarefas, mas em outras situações, como por exemplo, início e fim de reuniões.

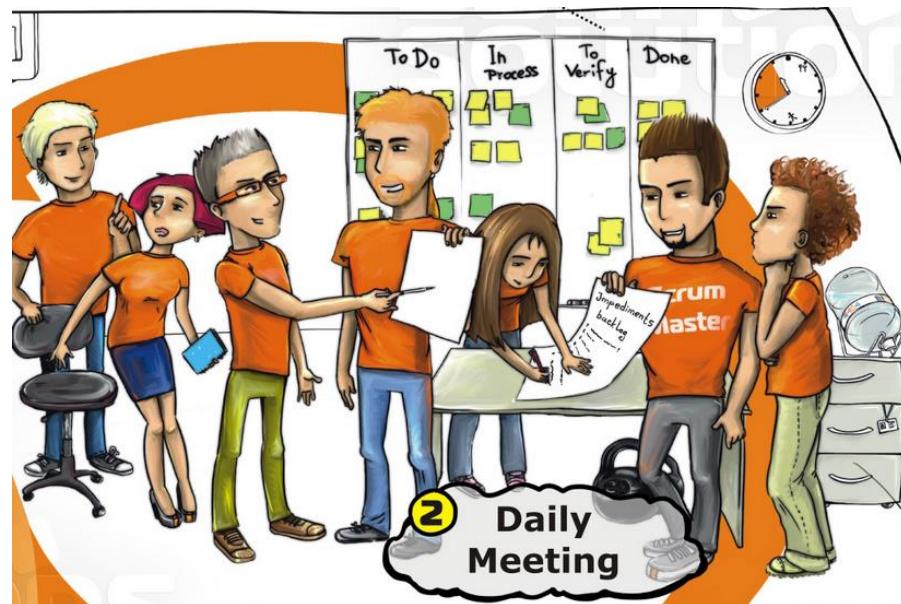




Transparência



- ✓ A equipe de desenvolvimento descreve e atualiza os status de **Sprint** em uma base diária, em reunião diariamente ocorrida;
- ✓ Esse procedimento faz com que o conteúdo e o andamento do **Sprint** atual, incluindo resultados, sejam visíveis para a equipe, gestão e todas as partes interessadas;
- ✓ Por exemplo, a equipe de desenvolvimento pode mostrar o status de **Sprint** em um quadro branco;





Time Auto Organizado e Multifuncional



- ✓ Scrum Master e Proprietário do Produto (**PO – Product Owner**) não são líderes;
- ✓ Num time de futebol, todos desempenham suas funções sem que um exerça autoridade formal sobre outro, nem mesmo o capitão do time;
- ✓ O time é auto organizado: não há líder de equipe, pois a equipe toma as decisões;
- ✓ Decisões são tomadas por consenso (em conjunto);
- ✓ A equipe é multifuncional.





As três funções do Scrum – Scrum Master



- ✓ O **Scrum Master** garante que as práticas e regras do **Scrum** sejam implementadas e seguidas;
- ✓ Resolve violações, questões de funcionalidades, ou outros impedimentos que possam impedir a equipe de seguir as práticas e regras;
- ✓ Age como um facilitador (consultor) do **Scrum** e não como um gerente de projetos (líder) no sentido habitual de metodologias não ágeis;
- ✓ É o treinador das práticas **Scrum**.





As três funções do Scrum – Product Owner (PO)



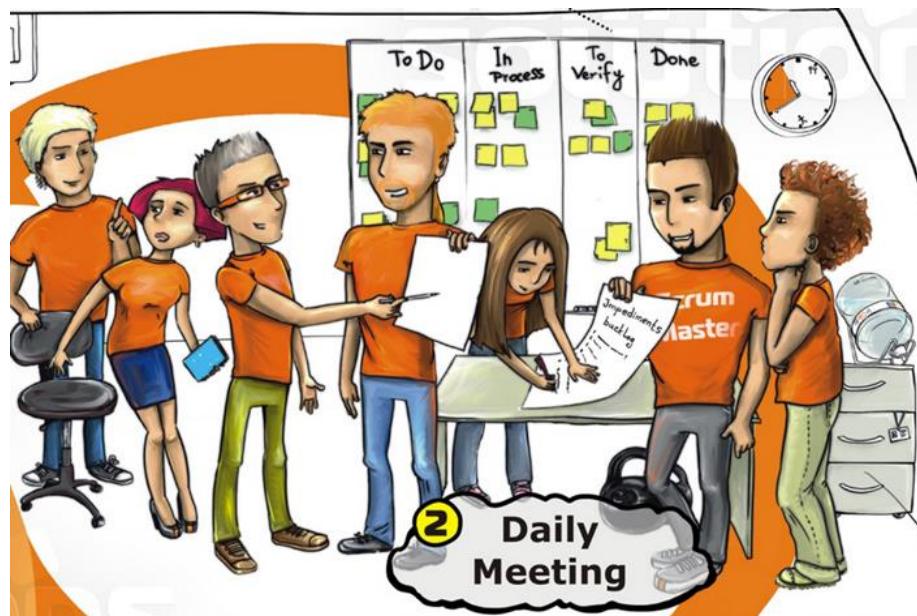
- ✓ Representa o **cliente**, e gera, mantém e prioriza o **backlog** do produto;
- ✓ Zela pelo ponto de vista e pelas **preocupações** do cliente;
- ✓ Negocia com a equipe de desenvolvimento o backlog do **Sprint** frente ao backlog do produto;
- ✓ Sensibiliza a equipe sobre a importância dos itens do backlog do produto;
- ✓ Lida com visões do cliente, riscos e prioridades do produto.





As três funções do Scrum – Equipe de Desenvolvimento

- ✓ Responsável pelo desenvolvimento e teste do produto;





Qual o foco do Scrum?



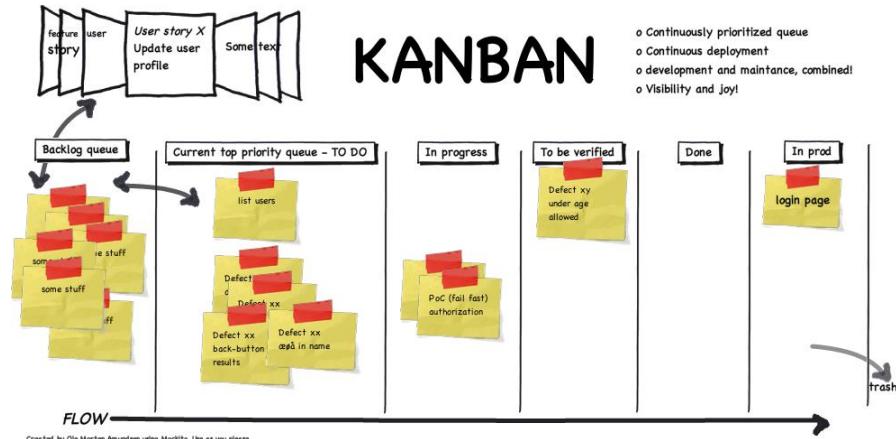
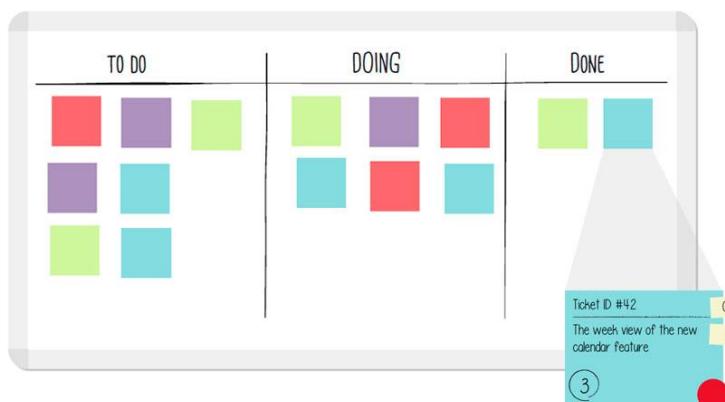
- ✓ O **Scrum** foca em questões de gestão e, portanto, não especifica – de forma tecnológica – como uma atividade deve ser desenvolvida;
- ✓ Essa abordagem difere do **XP**, que considera questões técnicas de programação, como por exemplo, integração contínua, etc;
- ✓ Assim, **Scrum** não detalha – do ponto de vista técnico – como o teste deve ser executado no desenvolvimento do software;





Kanban

- ✓ Abordagem criada na década de 70 (Toyota);
- ✓ Empregada originalmente como ferramenta para controle de produção;
- ✓ **Kanban** significa quadro (espaço para anotações);
- ✓ Trazida para as metodologias ágeis a partir do ano 2000;
- ✓ Seu objetivo é apresentar uma visualização do fluxo de trabalho do projeto em desenvolvimento;





Kanban



- ✓ Quadro **Kanban** mostra o fluxo de trabalho do projeto;
- ✓ Cada coluna exibe uma estação (conjunto de tarefas relacionadas);
- ✓ As tarefas são simbolizadas por **post-its** (blilhetes) que se movem da esquerda para a direita;

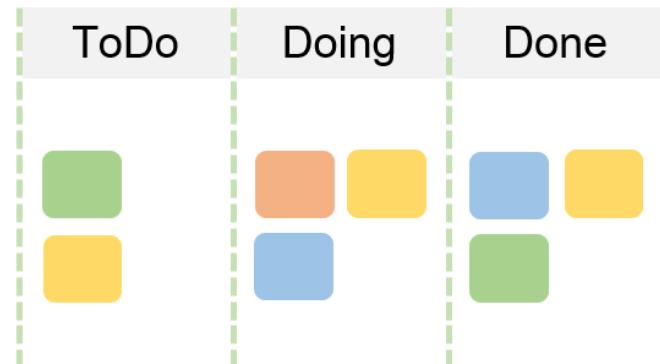




Controle de Tarefas

KANBAN

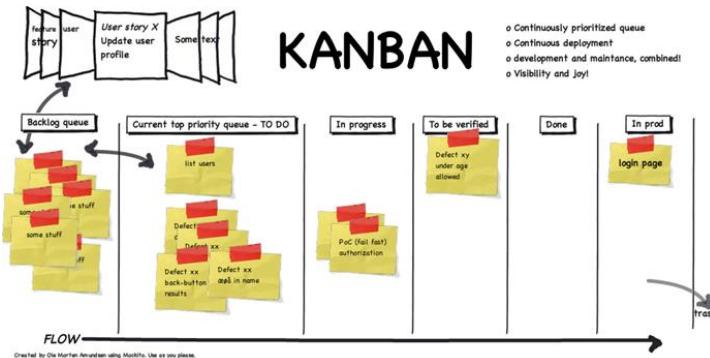
- ✓ A quantidade de tarefas ativas paralelas é estritamente limitada;
- ✓ Esse controle é efetuado pelo número máximo de passagens permitidas para uma estação ou de forma global para o quadro;
- ✓ Na ocorrência de alguma estação com capacidade livre, tira-se um bilhete da estação anterior, promovendo-o.





Scrum e Kanban

- ✓ Tanto **Scrum** quanto **Kanban** empregam um quadro para visualizar as tarefas ativas;
- ✓ Fornecem, dessa forma, transparência de conteúdos e exibem o andamento das tarefas;
- ✓ Tarefas em espera no backlog são transferidas para o quadro **Kanban** tão logo apareça um novo espaço disponível.





Scrum e Kanban

- ✓ **Kanban** não necessariamente trabalha com iterações ou sprints;
- ✓ Assim, **Kanban** foi concebido para apresentar o fluxo de forma linear (contínua);
- ✓ **Scrum** é fortemente orientado a controle de projetos (como **PMI**) enquanto que **Kanban** é orientado a demanda (produção);
- ✓ **Kanban** não necessariamente emprega o conceito de Timeboxing utilizado no **Scrum**.





Especificações em Projetos

- ✓ Problemas em Especificação são, muitas vezes, uma das principais razões para o fracasso de um projeto de software;
- ✓ Esses problemas podem ser oriundos de falhas de comunicação entre usuários e equipe de projeto, falta de uma visão global do projeto, funcionalidades redundantes ou contraditórias, etc.





Requisitos em projetos ágeis

- ✓ No desenvolvimento ágil, **estórias de usuários** são escritas para capturar os requisitos a partir das perspectivas de desenvolvedores, testadores e representantes de negócio;
- ✓ Revisões informais frequentes são realizadas enquanto os requisitos são capturados.
- ✓ No desenvolvimento tradicional (sequencial) a visão comum de uma funcionalidade é realizada através de análises formais após a elaboração dos requisitos;
- ✓ As **estórias de usuários** abordam características funcionais e não-funcionais.

ESTÓRIAS DO USUÁRIO

Note Title 24/5/2006

NOME DA ESTÓRIA	A QUAL ITERAÇÃO PERTENCE
GERAR RELATÓRIO	ITERAÇÃO 4 IMPORTÂNCIA 5 ESFORÇO 8
- EU QUERO GERAR UM RELATÓRIO A PARTIR DOS DADOS DOS CLIENTES DO TÍPO ANEXAR A ISSAS CANTAS.	
ANEXAR TUDO O QUE FOR ÚTIL AO DESENVOLVIMENTO DA ESTÓRIA	

NOME DA ESTÓRIA
 A QUAL ITERAÇÃO PERTENCE
 QUAL A IMPORTÂNCIA DADA PELO CLIENTE
 QUAL O ESFORÇO ESTIMADO PELOS DESENVOLVEDORES
 DESCRIÇÃO DA FUNCIONALIDADE



Critérios de Aceitação

- ✓ As estórias de usuários incluem critérios de aceitação;
- ✓ Critérios de aceitação correspondem aos resultados esperados de uma estória;
- ✓ Esses critérios são definidos em colaboração entre os representantes de negócio, desenvolvedores e testadores;
- ✓ Oferecem aos desenvolvedores e testadores uma visão ampliada da característica que os representantes de negócio vão validar;
- ✓ Uma equipe ágil considera uma tarefa concluída (**done**) quando um conjunto de critérios de aceitação foi atendido.

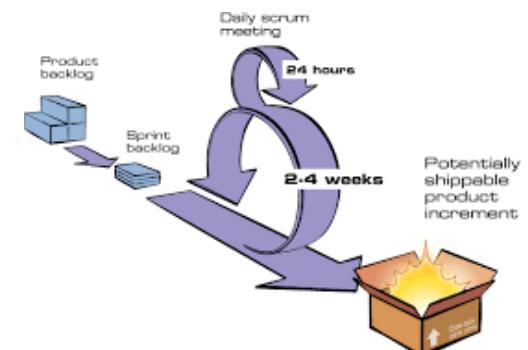




Contribuições do Testador



- ✓ Um testador pode contribuir:
 - Identificando detalhes **ausentes** ou requisitos não funcionais;
 - Formulando perguntas abertas aos representantes de negócio sobre a estória do usuário;
 - Propondo **diferentes formas** de testar a estória do usuário;
 - Confirmando os critérios de aceitação.





Estória do Usuário – Conceito 3C

- ✓ Uma estória de Usuário corresponde à composição de 3 elementos:
 - **Card** (cartão post-it)
 - **Conversation** (conversação)
 - **Confirmation** (confirmação)

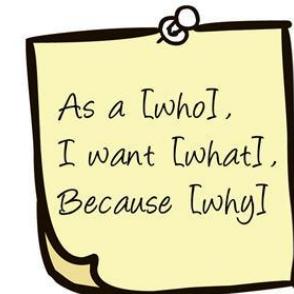
3C's
·card ·conversation
·confirmation





Cartão

- ✓ Meio físico que descreve uma estória de usuário;
- ✓ Identifica:
 - **Exigência**
 - **Criticidade**
 - **Desenvolvimento esperado**
 - **Duração do Teste**
 - **Critérios de Aceitação**
- ✓ A descrição deve ser **precisa**, uma vez que será utilizada no **backlog** do produto.





Conversação



- ✓ A conversa:
 - Explica como o software será usado;
 - Pode ser documentada ou verbal;
 - Inicia durante a fase de planejamento de lançamento e continua quando a estória é programada.
- ✓ Os testadores, tendo um ponto de vista diferente em relação aos desenvolvedores e representantes de negócio, gera um elemento contributivo valioso para a troca de ideias, opiniões e experiências.



Confirmação

- ✓ Os critérios de aceitação, discutidos na conversa:
 - São utilizados para confirmar que a estória foi realizada;
 - Podem se estender por várias estórias de usuário.
- ✓ Para confirmar a realização de uma estória, os critérios de aceitação definidos devem ser atendidos;
- ✓ Vários membros da equipe podem desempenhar papéis de testadores. Pode-se também incluir no time, mesmo temporariamente, especialistas na área de segurança, carga, stress, desempenho, etc. ;
- ✓ Ambos os testes (**positivos** e **negativos**) devem ser utilizados para abranger os critérios.

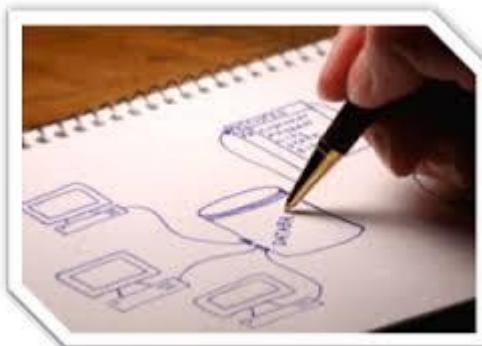


Testes com mensagens de erro!



Documentação da Estória de Usuário

- ✓ As equipes ágeis variam em termos de como documentam estórias de usuário;
- ✓ Independentemente da abordagem adotada, deve ser **concisa, suficiente e necessária.**



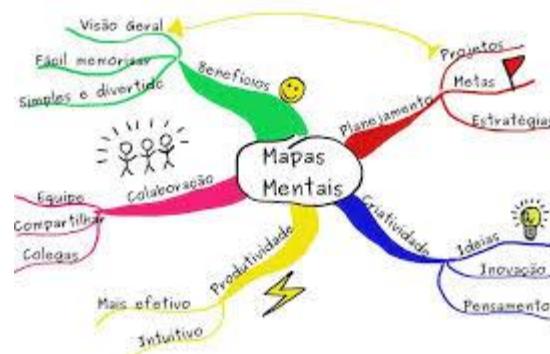
- ✓ **Concisa** significa sumarizada, resumida, que vá direto ao ponto;
- ✓ **Suficiente** significa que não deixe de apresentar detalhes importantes;
- ✓ **Necessária** significa que a documentação tem valor agregado.



Técnicas para Criação de Estórias

- ✓ Estórias podem ser criadas por técnicas tais como: **brainstorming** e **mapas mentais**;
- ✓ O testador pode usar a técnica **INVEST**:

- Independente;
- Negociável;
- Valioso;
- Estimável;
- Pequeno (**Small**);
- Testável.





Retrospectivas

- ✓ Reunião realizada no final de cada iteração para discutir:

- **O que foi bem sucedido?**
- **O que poderia ser melhorado?**
- **Como incorporar as melhorias e preservar os êxitos em iterações futuras?**





Retrospectivas

- ✓ Abrangem temas tais como:
 - **Processo**
 - **Pessoas**
 - **Organizações**
 - **Relacionamentos**
 - **Ferramentas**
- ✓ Sua realização **regular**, em conjunto com atividades de acompanhamento adequadas, são fundamentais para a auto-organização e melhoria contínua de desenvolvimento e testes;
- ✓ Podem resultar em decisões de melhorias relacionadas com o teste;
- ✓ Podem abordar a capacidade de teste dos Aplicativos, Estórias de Usuário, Funcionalidades e Interfaces do Sistema.
- ✓ O tempo e a organização da retrospectiva dependem do método ágil empregado no projeto;
- ✓ Em alguns casos, a equipe do projeto pode convidar outros participantes para a reunião.





Papel do Testador na Retrospectiva

- ✓ Os **testadores** devem desempenhar um papel importante nas retrospectivas, uma vez que **fazem parte da equipe**;
- ✓ O **teste ocorre em cada Sprint** e contribui vitalmente para o sucesso;
- ✓ **Todos** os membros da equipe, testadores e não-testadores, podem fornecer informações sobre as atividades de teste e não-teste.





O ambiente da Retrospectiva

- ✓ As retrospectivas devem ocorrer dentro de um ambiente profissional caracterizado pela confiança mútua;
- ✓ A reunião de retrospectiva não é um julgamento ou tribunal onde se penaliza algum membro do time;
- ✓ Deve-se avaliar o trabalho de forma global, sem, portanto, constranger qualquer membro do time.
- ✓ Deve ser uma reunião franca com objetivo de melhoria coletiva;
- ✓ O que deu certo? O que deu errado? O que fazer para melhorar?





Integração Contínua

- ✓ É um dos mais **sólidos conceitos** dentro da metodologia ágil;
- ✓ Contribui para que os problemas de modificação de código sejam resolvidos;
- ✓ No final de cada Sprint, deve-se entregar **incrementos confiáveis integrados**;
- ✓ A integração contínua aborda esse desafio através da **fusão** de todas as alterações feitas no software e integração de todos os componentes alterados regularmente, pelo menos uma vez por dia.;
- ✓ Tem por objetivo garantir **estabilidade** do software de modo que em cada novo incremento do produto, garanta-se a permanência de software **confiável**;
- ✓ Esse procedimento baseia-se em **testes de regressão**.

INTEGRAÇÃO CONTÍNUA

— PARA TESTADORES —





Processo Repetitivo e Automatizado

- ✓ Os desenvolvedores constantemente:
 - Constroem
 - Integram o seu trabalho
 - Testam
- ✓ Com o objetivo de detectar os defeitos no código mais rapidamente, um processo **unificado, repetitivo e automatizado** deve ser criado e mantido, envolvendo:
 - Gestão de Configuração
 - Compilação do Componente
 - Compilação de Módulos (Software)
 - Implementação
 - Testes





Quando são executadas as atividades

- ✓ Na sequência da codificação dos desenvolvedores, depuração e check-in de código em um repositório de código-fonte comum, deve ser executado um processo de **integração contínua** composto pelas seguintes atividades automatizadas:
 - **Análise Estática** (checagem do código sem executá-lo e identifica pontos de melhoria)
 - **Compilação** (ligar o código e gerar arquivos executáveis)
 - **Teste da Unidade** (verificar a cobertura de código e relatar resultados de testes)
 - **Implantar** (instalar o projeto em um ambiente de teste)
 - **Teste de Integração** (executar os testes de integração e relatar resultados)
 - **Relatório** (Dashboard) (postar o status das atividades em local publicamente visível)





Testes Automatizados

- ✓ Um processo de construção e teste automatizado ocorre em uma **base diária** e detecta erros de integração de modo antecipado e rápido;
- ✓ A integração contínua permite que os testadores ágeis realizem testes automatizados regularmente, em alguns casos, como parte do próprio processo de **integração contínua**;
- ✓ Permite também que se envie **feedback** rápido para a equipe sobre a **qualidade** do código.





Testes de Regressão Automatizados

- ✓ Consideram a maior quantidade de funcionalidades possível, incluindo estórias de usuários desenvolvidos nas iterações anteriores.
- ✓ Todos os **membros da equipe** podem **visualizar** os resultados, especialmente quando os relatórios automatizados são integrados no processo;
- ✓ Podem ser **contínuos** ao longo da iteração;
- ✓ Desenvolvimento e teste de grandes sistemas integrados são beneficiados com o emprego de **boa cobertura nos testes de regressão automatizados**;
- ✓ Automatizando-se o teste de regressão, os testadores ágeis ficam livres para se concentrar seus testes manuais em novas funcionalidades, mudanças implementadas, teste de **confirmação** (reteste de defeitos corrigidos).





Benefícios da Integração Contínua

- ✓ Detecção mais **precoce** e análise mais fácil da causa raiz de problemas de integração;
- ✓ **Feedback** regular (diariamente) para a equipe sobre o funcionamento do código;
- ✓ Todo o dia o cliente terá uma **versão testada** do software sendo desenvolvido;
- ✓ **Reduz** o risco de regressão (rápido reteste da base de código após cada alteração);
- ✓ Promove **confiança**, uma vez que o trabalho é feito em base sólida;
- ✓ Realiza progresso em direção a conclusão do incremento do produto visível (**encorajamento** da equipe)
- ✓ Elimina **riscos** de programação associados à integração do big-bang;
- ✓ Disponibilidade constante de software **executável** em todo Sprint;
- ✓ Reduz as atividades de testes **manuais** repetitivas.





Riscos e Desafios – Integração Contínua

- ✓ Ferramentas de integração contínua devem ser introduzidas e mantidas;
- ✓ O processo de integração contínua deve ser definido e estabelecido;
- ✓ A automação de teste exige funcionalidades adicionais e pode ser complexa para se estabelecer;
- ✓ A cobertura completa de teste é essencial para alcançar vantagens de teste automatizado;
- ✓ As equipes, por vezes, confiam excessivamente nos testes de unidade e realizam muito pouco do teste de sistema e de aceitação.





Integração Contínua – Dependência de Ferramentas

- ✓ A integração contínua requer o uso de ferramentas, incluindo:
 - Ferramentas para **Testes**;
 - Ferramentas para Automatizar o Processo de **Construção**;
 - Ferramentas para Controle de **Versão**.

- ✓ Requer **perfil** de profissional que domine ferramentas de automação.





Planejamento de Iteração e de Lançamento

- ✓ Nos processos ágeis, planejamento é uma atividade em curso (contínua);
- ✓ Nos ciclos de vida ágil, dois tipos de planejamento ocorrem: **planejamento de iteração** e **planejamento de lançamento**;





Planejamento de Lançamento

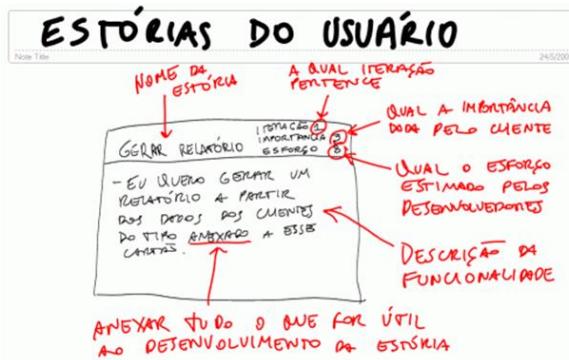
- ✓ O Planejamento de **Lançamento** foca a **liberação** de um produto, muitas vezes, alguns meses antes do início de um projeto. Define e redefine o **backlog** do produto, e pode envolver o refinamento de estórias de usuários maiores em uma coleção de estórias menores;
- ✓ O Planejamento de Lançamento tem a visão de conjunto (envolvendo **vários** Sprints);
- ✓ O backlog do produto está associado ao Planejamento de **Lançamento**;
- ✓ Fornece a base para uma abordagem de teste e plano de teste que abrange todas as iterações.
- ✓ Planos de **Lançamento** são de alto nível (visão macro).





Planejamento de Lançamento

- ✓ No planejamento de Lançamento a equipe -- juntamente com os representantes das empresas – estabelecem e priorizam as estórias de usuários para o lançamento;
- ✓ Com base nessas estórias de usuários, riscos associados ao projeto e qualidade são identificados, e uma estimativa de alto nível é desenvolvida;





Planejamento de Lançamento – Envolvimento dos Testadores

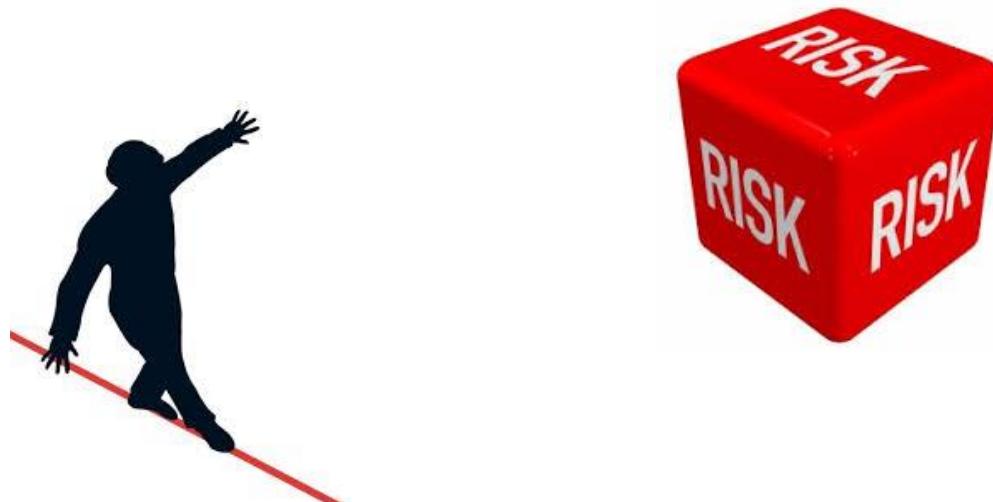
- ✓ Os testadores são envolvidos no Planejamento de Lançamento e agregam valor nas seguintes atividades:
 - Definir estórias de usuários **testáveis**, incluindo critérios de aceitação;
 - Participação no projeto e análise do **risco** da qualidade;
 - Estimativa de esforço de teste associado às estórias do usuário;
 - Definir os níveis de testes (unitário, integração, sistema e aceite) necessários;
 - Planejar o teste para o lançamento.





Planejamento de Iteração

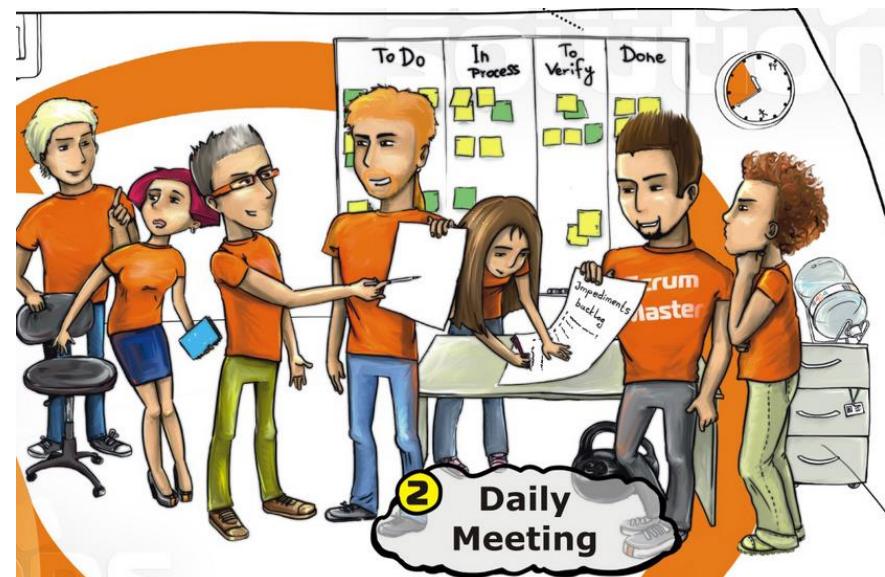
- ✓ Após a realização do Planejamento de Lançamento (visão geral), o Planejamento de Iteração (ciclo ou Sprint) para a primeira iteração é iniciado;
- ✓ O Planejamento de Iteração foca o final de uma **única iteração** e está relacionado ao backlog da iteração;
- ✓ No Planejamento de Iteração, a equipe seleciona estórias de usuário do backlog de liberação priorizados, elabora uma análise de **risco** para as estórias de usuários e **estima** o trabalho necessário para cada estória de usuário.





Planejamento de Iteração

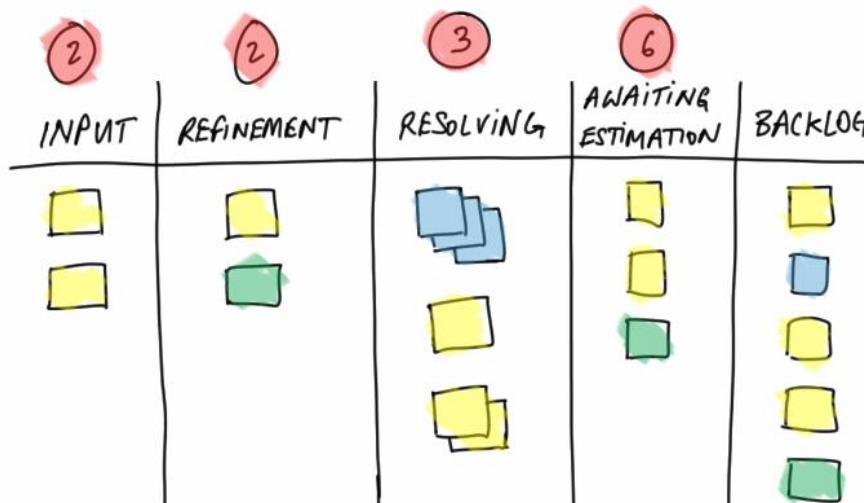
- ✓ Se uma estória de usuário é demasiado **vago**, a equipe pode se recusar a aceitá-la e utilizar a próxima estória de usuário baseada na prioridade, até que a primeira seja esclarecida;
- ✓ Os representantes de negócio devem responder às perguntas da equipe sobre cada estória, de modo que a equipe possa **entender** o que eles devem implementar e **como testar** cada estória.





Planejamento de Iteração

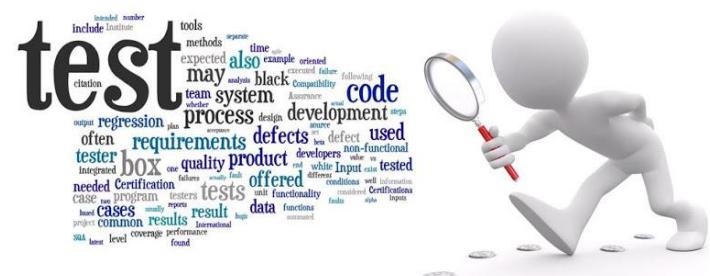
- ✓ O número de estórias selecionadas é baseado na velocidade da equipe estabelecida e no tamanho estimado de estórias de usuários selecionadas;
- ✓ Após a finalização do conteúdo da iteração, as estórias de usuários são divididas em tarefas, que serão realizadas pelos membros apropriados da equipe.





Planejamento de Iteração – Envolvimento do Testador

- ✓ Os testadores são envolvidos no Planejamento de Iteração, e especialmente, agregar valor nas seguintes atividades:
 - Participar da análise de risco detalhada de estórias de usuários;
 - Determinar a testabilidade das estórias de usuários;
 - Criar testes de aceitação (casos de teste) para estórias de usuários;
 - Dividir estórias de usuários em tarefas (particularmente tarefas de teste);
 - Estimativa de esforço de teste para todas as tarefas de teste;
 - Identificar os aspectos funcionais e não-funcionais do sistema a ser testado;
 - Apoiar e participar em automação de testes em vários níveis de teste.





Planejamento de Lançamento e Iteração

- ✓ Ambos devem ser representados por um Plano de Teste;
- ✓ Questões comumente encontradas nesse Plano de Teste:
 - O que se vai testar? (Escopo do Teste)
 - Quais os membros da equipe realizarão as atividades de Teste?
 - Qual o ambiente de teste?
 - Há dependências e pré-requisitos para o teste?
 - Quais os riscos?
 - Quais as estimativas de esforço de teste?

