International Software Testing Qualifications Board

Certified Tester

Foundation Level Extension
Syllabus Agile Tester

Versão 2014br

Comissão Internacional para Qualificação de Teste de Software

---

Tradução realizada pela TAG01 - Documentação do BSTQB baseada na versão 2014 do *Certified Tester Foundation Level Syllabus – Agile Tester Extended* do ISTQB.

Brazilian Software Testing Qualifications Board

Grupo de Trabalho do Foundation Level Extension Agile Tester: Rex Black (Presidente), Bertrand Cornanguer (Vice-Presidente), Gerry Coleman (Líder dos Objetivos de Aprendizagem), Debra Friedenberg (Líder de Exame), Alon Linetzki (Líder de Resultados de Negócios e de Marketing), Tauhida Parveen (Editor), e Leo van der Aalst (Líder de Desenvolvimento).

Autores: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh, e Stephan Weber.

Revisores internos: Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael e Erik van Veenendaal; 2013-2014.

Histórico de Revisões

Versão	Data	Observação
Syllabus v0.1	26/06/2013	Seções autônomas
Syllabus v0.2	16/09/2013	Comentários da avaliação WG em v01 incorporado
Syllabus v0.3	20/10/2013	Comentários da avaliação WG em v02 incorporado
Syllabus v0.7	16/12/2013	Comentários da avaliação Alpha em v03 incorporado
Syllabus v0.71	20/12/2013	Atualizações de grupo de trabalho em v07
Syllabus v0.9	30/01/2014	Versão Beta
Syllabus 2014	31/05/2014	Versão GA
Syllabus 2014br	15/12/2017	Revisão da Língua Portuguesa

Sumário

Agradecimentos

Este documento foi produzido por uma equipe do Grupo de Trabalho ISTQB Foundation Level.

A equipe *Agile Extension* agradece a equipe de revisão e os Conselhos Nacionais por suas sugestões e contribuições.

Na época, o *Syllabus Foundation Level Agile Extension* foi concluído, o Grupo de Trabalho *Agile Extension* apresentou a seguinte associação: Rex Black (Presidente), Bertrand Cornanguer (Vice-Presidente), Gerry Coleman (Líder dos Objetivos de Aprendizagem), Debra Friedenberg (Líder de Exame), Alon Linetzki (Líder de Resultados de Negócios e de Marketing), Tauhida Parveen (Editor), e Leo van der Aalst (Líder de Desenvolvimento).

Autores: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh, e Stephan Weber.

Revisores internos: Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael e Erik van Veenendaal; .

A equipe agradece também as seguintes pessoas dos Conselhos Nacionais e da comunidade de especialistas Ágil, que participaram da revisão, comentário e votação da *Foundation Agile Extension Syllabus*: Dani Almog, Richard Berns, Stephen Bird, Monika Bögge, Afeng Chai, Josephine Crawford, Tibor Csöndes, Huba Demeter, Arnaud Foucal, Cyril Fumery, Kobi Halperin, Inga Hansen, Hanne Hinz, Jidong Hu, Phill Isles, Shirley Itah, Martin Klonk, Kjell Lauren, Igal Levi, Rik Marselis, Johan Meivert, Armin Metzger, Peter Morgan, Ninna Morin, Ingvar Nordstrom, Chris O'Dea, Klaus Olsen, Ismo Paukamainen, Nathalie Phung, Helmut Pichler, Salvatore Reale, Stuart Reid, Hans Rombouts, Petri Säilynoja, Soile Sainio, Lars-Erik Sandberg, Dakar Shalom, Jian Shen, Marco Sogliani, Lucjan Stapp, Yaron Tsubery, Sabine Uhde, Stephanie Ulrich, Tommi Välimäki, Jurian Van de Laar, Marnix Van den Ent, António Vieira Melo, Wenye Xu, Ester Zabar, Wenqiang Zheng, Peter Zimmerer, Stevan Zivanovic, and Terry Zuo.

Este documento foi formalmente aprovado para liberação pela Assembleia Geral da ISTQB® em 31 de maio de 2014.

Introdução a este Syllabus

Finalidade deste Documento

Este programa constitui a base para a Qualificação do Teste de Software Internacional no Nível Fundamental para o Testador Ágil. O ISTQB® fornece este programa da seguinte forma:

- Para Conselhos Nacionais, para traduzir em seu idioma local e credenciar prestadores de treinamento. Conselhos Nacionais podem adaptar o programa às suas necessidades linguísticas específicas e modificar as referências para se adaptar às suas publicações locais.
- Para Conselhos de Exame, para derivar questões do exame em seu idioma local adaptado para os objetivos de aprendizagem para cada programa.
- Para prestadores de treinamento, para a produção de material didático e determinar os métodos de ensino adequados.
- Para os candidatos à certificação, para se preparar para o exame (como parte de um curso de formação ou de forma independente).
- Para a comunidade internacional de software e sistemas de engenharia, para o avanço da profissão de software e testes de sistemas, e como base para livros e artigos.

O ISTQB® pode permitir que outras entidades possam utilizar este programa para outros fins, desde que busque e obtenha permissão prévia por escrito.

Visão Geral

O documento Visão Geral do *Foundation Level Agile Tester* [ISTQB_FA_OVIEW] inclui as seguintes informações:

- Resultados de negócios para o programa
- Resumo para o programa
- Relações entre os programas
- Descrição dos níveis cognitivos (níveis K)
- Apêndices

Objetivos de aprendizagem examináveis

Os Objetivos de aprendizagem apoiam os Resultados do Negócio e são utilizados para criar o exame para a obtenção da certificação *Certified Tester Foundation Level—Agile Tester*. Em geral, todas as partes deste programa são examináveis em um nível K1. Ou seja, o candidato irá reconhecer, lembrar e recordar um termo ou conceito. Os objetivos de aprendizagem específicos a níveis K1, K2 e K3 são mostrados no início do capítulo pertinente.

Capítulo 1: Desenvolvimento do software ágil (150 min)

Palavras-chave.

Manifesto Ágil, desenvolvimento de software ágil, modelo de desenvolvimento incremental, modelo de desenvolvimento iterativo, ciclo de vida do software, automação de testes, base em testes, desenvolvimento orientado a testes, oráculo de teste, estória do usuário

Objetivos de aprendizagem para Desenvolvimento do Software Ágil

1.1 Os Fundamentos do Desenvolvimento do Software Ágil

FA-1.1.1 (K1) Relembrar o conceito básico de desenvolvimento do software Ágil baseado no manifesto Ágil

FA-1.1.2 (K2) Compreender as vantagens da abordagem de equipe inteira

FA-1.1.3 (K2) Compreender os benefícios do feedback inicial e frequente

1.2 Aspectos de abordagens do Ágil

FA-1.2.1 (K1) Relembrar abordagens de desenvolvimento do software Ágil

FA-1.2.2 (K3) Escrever estórias de usuários testáveis em colaboração com os desenvolvedores e os representantes de negócio

FA-1.2.3 (K2) Compreender como as retrospectivas podem ser utilizadas como um mecanismo para a melhoria de processos em Projetos Ágeis

FA-1.2.4 (K2) Compreender o uso e propósito de integração contínua

FA-1.2.5 (K1) Conhecer as diferenças entre planejamento de iteração e de liberação, e como um testador agrega valor em cada uma dessas atividades

1.1 Os fundamentos do desenvolvimento de software ágil

Um testador em um projeto Ágil vai trabalhar de forma diferente do que um testador trabalhando em um projeto tradicional. Os testadores devem compreender os valores e princípios que sustentam os projetos no formato ágil, e como farão parte integrante de uma abordagem de equipe junto com desenvolvedores e representantes de negócio. Os membros de um projeto do Ágil se comunicam entre si com antecedência e com frequência, o que ajuda na remoção de defeitos precoces e desenvolvimento de um produto de qualidade.

1.1.1 Desenvolvimento do software ágil e do Manifesto Ágil

Em 2001, um grupo de indivíduos, representando as metodologias de desenvolvimento de software mais utilizados, acordou um conjunto comum de valores e princípios que ficou conhecido como o Manifesto para Desenvolvimento do Software Ágil ou o Manifesto Ágil [Agilemanifesto]. O Manifesto Ágil contém quatro declarações de valores:

- **Indivíduos e interações** sobre *processos e ferramentas*
- **Software funcionando** sobre *documentação mais abrangente*
- **Colaboração com o cliente** sobre *negociação de contratos*
- **Resposta às mudanças** sobre o *seguimento de um plano*

O Manifesto Ágil argumenta que, embora os conceitos à direita tenham valor, os da esquerda têm maior valor.

Indivíduos e interações

O desenvolvimento do modelo ágil é muito centrado em pessoas. Equipes de pessoas desenvolvem o software, e é através de uma comunicação contínua e interação, ao invés de uma dependência de ferramentas ou processos, que as equipes podem trabalhar de forma mais eficaz.

Software funcionando

A partir da perspectiva do cliente, o software funcionando é muito mais útil e valioso do que a documentação excessivamente detalhada, e fornece uma oportunidade para dar o feedback rápido à equipe de desenvolvimento. Além disso, ter um software funcionando, embora com funcionalidade reduzida, estará disponível muito antes no ciclo de vida de desenvolvimento, o desenvolvimento ágil pode apresentar vantagem significativa no *time-to-market*. O desenvolvimento ágil é, portanto, especialmente útil em mudanças rápidas de ambientes de negócios onde os problemas e/ou soluções não são esclarecidos, ou onde a empresa pretende inovar em novos domínios de problemas.

Colaboração do cliente

Os clientes muitas vezes encontram grande dificuldade em especificar o sistema que necessitam. Colaborar diretamente com o cliente aumenta a probabilidade de compreender exatamente o que o cliente quer. Embora celebrar contratos com os clientes seja importante, trabalhar em colaboração regular e próxima com eles pode trazer mais sucesso para o projeto.

Resposta à mudança

A mudança é inevitável em projetos de software. O ambiente em que a empresa opera, a legislação, atividade concorrente, avanços tecnológicos e outros fatores podem ter grandes influências sobre o projeto e seus objetivos. Estes fatores devem ser acomodados pelo processo de desenvolvimento. Como tal, ter flexibilidade nas práticas de trabalho para aceitar a mudança é mais importante do que simplesmente aderir rigidamente a um plano.

Princípios

Os valores fundamentais do Manifesto Ágil são apresentados em doze princípios:

- Nossa maior prioridade é satisfazer o cliente através do desenvolvimento avançado e contínuo de software de valor.
- Acatar mudanças de requisitos, mesmo no final do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
- Desenvolver software funcionando com frequência, em intervalos de algumas semanas a alguns meses, com preferência para a escala de tempo mais curta.
- Pessoas relacionadas ao negócio e desenvolvedores devem trabalhar juntos diariamente durante o projeto.
- Criar projetos em torno de indivíduos motivados. Providenciar a eles o ambiente e suporte necessários, confiando que realizarão seu trabalho.
- O método mais eficiente e eficaz de transmitir informações para e dentro de uma equipe de desenvolvimento é a conversa cara a cara.
- Software funcionando é a principal medida de progresso.
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
- A atenção contínua à excelência técnica e bom desenho intensifica a agilidade.
- Simplicidade - a arte de maximizar a quantidade de trabalho não realizado - é essencial.
- As melhores arquiteturas, requisitos e projetos emergem de equipes auto organizáveis.
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz, e então ajustar e otimizar seu comportamento em conformidade.

As diferentes Metodologias Ágeis proporcionam práticas prescritivas para colocar esses valores e princípios em ação.

1.1.2 Abordagem da equipe inteira

A abordagem de equipe inteira envolve a todos com o conhecimento e as habilidades necessárias para garantir o sucesso do projeto. A equipe inclui representantes do cliente e de outras partes interessadas de negócios que determinam as características do produto. A equipe deve ser relativamente pequena; equipes bem-sucedidas têm sido observadas com apenas três pessoas e no máximo nove. O ideal é que toda a equipe compartilhe o mesmo espaço de trabalho, pois a localização facilita fortemente a comunicação e a interação. A abordagem da equipe inteira é suportada através de reuniões diárias (ver capítulo 2.2.1), envolvendo todos os membros da equipe, onde o progresso do trabalho é comunicado, e quaisquer impedimentos ao progresso são realçados. A abordagem da equipe inteira promove uma dinâmica de equipe mais eficaz e eficiente.

O uso de uma abordagem de equipe inteira para desenvolvimento do produto é um dos principais benefícios do desenvolvimento ágil. Seus benefícios incluem:

- 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.

A equipe inteira é responsável pela qualidade nos projetos ágeis. A essência da abordagem de equipe inteira reside nos testadores, desenvolvedores e os representantes das empresas que trabalham juntos em todas as etapas do processo de desenvolvimento. Os testadores vão trabalhar em estreita colaboração com os desenvolvedores e representantes das empresas para garantir que os níveis de qualidade desejados sejam alcançados. Isso inclui apoiar e colaborar com os representantes de negócio para ajudá-los a criar testes de aceite adequados, trabalhando com desenvolvedores para chegar a acordos sobre a estratégia de teste, e decidir sobre as abordagens de automação de testes. Os testadores podem, portanto, transferir e ampliar o conhecimento de testes para outros membros da equipe e influenciar o desenvolvimento do produto.

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 [Crispin08].

1.1.3 Feedback inicial e frequente

Os projetos ágeis têm iterações curtas, permitindo a equipe do projeto receber feedback avançado e contínuo sobre a qualidade do produto durante todo o ciclo de desenvolvimento. Uma forma de prover feedback rápido é pela integração contínua (ver capítulo 1.2.4).

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 eficazmente todos os problemas que o cliente pode ter. Ao receber o frequente feedback do cliente no decorrer do projeto, as equipes ágeis podem incorporar a maioria das novas alterações no processo de desenvolvimento do produto. O feedback avançado e frequente ajuda no foco da equipe quanto às funcionalidades com o valor comercial mais elevado ou risco associado, e esses são desenvolvidos primeiramente para o cliente. Da mesma forma, ajuda a gerenciar melhor a equipe, pois a capacidade de todos é transparente. Por exemplo, qual volume de trabalho podemos realizar em um *sprint* ou iteração? O que poderia nos ajudar a avançar mais rápido? O que nos impede de fazer isso?

Os benefícios do feedback inicial e frequente incluem:

- Evitar mal-entendidos nos requisitos, que somente podem ser detectados tardiamente no ciclo de desenvolvimento, quando serão mais caros para se reparar.
- Esclarecer solicitações de funcionalidades dos clientes, tornando-os disponíveis antecipadamente para uso. 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.

1.2 Aspectos de abordagens ágeis

Existe uma série de abordagens ágeis em uso nas organizações. Práticas comuns incluem criação colaborativa da estória do usuário, retrospectivas, integração contínua e planejamento para cada iteração, bem como para a liberação total. Esta subcapítulo descreve algumas das abordagens ágeis.

1.2.1 Abordagens de desenvolvimento do software ágil

Existem várias abordagens ágeis, sendo que cada qual implementa valores e princípios do Manifesto Ágil de maneiras diferentes. Neste programa, três representantes de abordagens ágeis são considerados: *Extreme Programming* (XP), *Scrum*, e *Kanban*.

Extreme Programming

Extreme Programming (XP), originalmente introduzido por Kent Beck [Beck04], é uma abordagem ágil para desenvolvimento de software descrito por certos valores, princípios e práticas de desenvolvimento.

O XP engloba cinco valores para orientar o desenvolvimento: comunicação, simplicidade, feedback, coragem e respeito.

O XP descreve um conjunto de princípios como diretrizes adicionais: humanidade, economia, benefício mútuo, auto similaridade, aperfeiçoamento, diversidade, reflexão, fluxo, oportunidade, redundância, falha, qualidade, primeiros passos e responsabilidade assumida.

O XP descreve treze práticas principais: sentar-se junto, 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 e design incremental.

A maioria das abordagens de desenvolvimento do software ágil em uso hoje são influenciadas pelos valores e princípios do XP. Por exemplo, as equipes ágeis que seguem o *Scrum* frequentemente incorporam as práticas XP.

Scrum

Scrum é uma estrutura de gestão ágil, que contém os seguintes instrumentos e práticas constituídas [Schwaber01]:

- *Sprint*: O *Scrum* divide um projeto em iterações (chamadas *sprints*) de duração fixa (geralmente de duas a quatro semanas).
- *Incremento do produto*: Cada *sprint* resulta em um produto potencialmente entregável (chamado de incremento).
- *Backlog do produto*: O proprietário do produto gerencia uma lista priorizada de itens de produtos planejados (chamada *backlog* 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 (o chamado *backlog* de *sprint*) do *backlog* do produto. Uma vez

que a equipe *Scrum*, e não o proprietário do produto, seleciona os itens a serem realizados no *sprint*, a seleção é referida como sendo o princípio de puxar e não o princípio de empurrar estes itens.

- *Definição de Pronto*: Para verificar se há produto potencialmente liberável no final de cada *sprint*, a equipe *Scrum* discute e define critérios adequados para a conclusão do *sprint*. A discussão aprofunda a compreensão da equipe dos itens do *backlog* e os requisitos do produto.
- *Timeboxing*: Apenas as tarefas, requisitos ou funcionalidades que a equipe espera concluir no *sprint* fazem parte do *backlog* do *sprint*. Se a equipe de desenvolvimento não consegue terminar uma tarefa em um *sprint*, as características de produtos associadas são removidas do *sprint* em andamento e a tarefa é restituída ao *backlog* do produto. O *Timeboxing* não se aplica apenas às tarefas, mas em outras situações (p.e., no início e fim das reuniões).
- *Transparência*: A equipe de desenvolvimento descreve e atualiza os status dos *sprint* em uma base diária, em reunião de *Scrum* convocada diariamente. Isso 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 do *sprint* em um quadro branco.

Scrum define três funções:

- *Scrum Master*: garante que as práticas e regras do *Scrum* sejam implementadas e seguidas, resolvendo quaisquer violações, questões de funcionalidades, ou outros impedimentos que possam interferir no prosseguimento do projeto. Esta pessoa não é o líder da equipe, mas um treinador.
- *Proprietário do Produto*: representa o cliente, e gera, mantém e prioriza o *backlog* do produto. Esta pessoa não é o líder da equipe.
- *Equipe de Desenvolvimento*: Desenvolve e testa o produto. O time é auto organizado: não há líder da equipe, pois toda a equipe toma as decisões. A equipe também é multifuncional (ver capítulos 2.3.2 e 3.1.4).

O *Scrum* (em oposição ao XP) não dita técnicas de desenvolvimento de software específicos (p.e., o primeiro teste de programação). Além disso, o *Scrum* não fornece orientação sobre como o teste deve ser realizado em um projeto.

Kanban

Kanban [Anderson13] é uma abordagem de gestão que às vezes é utilizada em projetos ágeis. O objetivo geral é visualizar e otimizar o fluxo de trabalho em uma cadeia de valor agregado. *Kanban* utiliza três instrumentos [Linz14]:

- *Quadro Kanban*: A cadeia de valor a ser gerenciada é visualizada por um quadro de *Kanban*. Cada coluna mostra uma estação, que é um conjunto de atividades relacionadas, por exemplo, desenvolvimento ou teste. Os itens a serem produzidos ou tarefas a serem processadas são simbolizados por bilhetes se movendo da esquerda para a direita através do quadro nas colunas de valor.
- *Limite do Trabalho em Andamento*: O volume de tarefas ativas paralelas é estritamente limitado. Este é controlado pelo número máximo de passagens permitidas para uma coluna

de valor e/ou globalmente para o quadro. Sempre que uma estação tem capacidade livre, o trabalhador move um bilhete da coluna de valor antecessora.
- *Tempo de espera*: *Kanban* é utilizado para otimizar o fluxo contínuo de tarefas, minimizando o tempo de espera (média) para concluir o fluxo de valor.

O *Kanban* apresenta algumas semelhanças com *Scrum*. Em ambos os quadros, visualizar as tarefas ativas (p.e., em um quadro branco público) fornece transparência de conteúdo e andamento das tarefas. As tarefas não agendadas estão aguardando em um *backlog* e são transferidas para o quadro *Kanban*, assim que surge um novo espaço (capacidade de produção) disponível.

Iterações ou *sprints* são opcionais no *Kanban*. O processo de *Kanban* permite liberar seu item de entregas por item, e não como parte de uma liberação. O *Timeboxing* como um mecanismo de sincronização, portanto, é opcional, diferentemente do *Scrum*, que sincroniza todas as tarefas em um *sprint*.

1.2.2 Criação colaborativa da estória do usuário

Especificações ineficientes são muitas vezes uma das principais razões para o fracasso de um projeto. Problemas de especificação podem resultar da falta de compreensão dos usuários sobre as suas verdadeiras necessidades, ausência de uma visão global do sistema, funcionalidades redundantes ou contraditórias, e outras falhas de comunicação. No desenvolvimento ágil, estórias de usuários são escritas para capturar os requisitos a partir da perspectiva de desenvolvedores, testadores e representantes de negócio. No desenvolvimento sequencial, esta visão comum de uma funcionalidade é realizada através de análises formais após a elaboração dos requisitos; no desenvolvimento ágil, esta visão compartilhada é realizada através de revisões informais frequentes enquanto os requisitos estão sendo elaborados.

As estórias do usuário devem abordar características funcionais e não-funcionais. Cada estória inclui critérios de aceite para essas características. Esses critérios devem ser definidos em colaboração entre os representantes de negócio, desenvolvedores e testadores. Eles oferecem aos desenvolvedores e testadores uma visão ampliada da característica que os representantes de negócio vão validar. Uma equipe ágil considera uma tarefa concluída quando um conjunto de critérios de aceite foi atendido.

Normalmente, a perspectiva única do testador irá melhorar a estória do usuário, identificando detalhes ausentes ou requisitos não-funcionais. Um testador pode contribuir, formulando perguntas abertas aos representantes de negócio sobre a estória do usuário, propondo formas de testá-la, e confirmar os critérios de aceite.

A autoria colaborativa da estória do usuário pode usar técnicas como *brainstorming* e mapas mentais. O testador pode usar a técnica INVEST [INVEST]:

- **I**ndependente
- **N**egociável
- **V**alioso
- **E**stimável
- Pequeno (***S****mall*)
- **T**estável

De acordo com o conceito 3C [Jeffries00], uma estória de usuário é a conjunção de três elementos:

- *Cartão*: O cartão é o meio físico que descreve uma estória de usuário. Ele identifica a exigência, sua criticidade, desenvolvimento esperado, duração do teste e os critérios de aceite para essa estória. A descrição deve ser precisa, uma vez que será utilizada no *backlog* do produto.
- *Conversacional*: A conversa explica como o software será usado. A conversa pode ser documentada ou verbal. Os testadores, tendo um ponto de vista diferente em relação aos desenvolvedores e representantes de negócio [ISTQB_FL_SYL], gera um elemento contributivo valioso para a troca de ideias, opiniões e experiências. A conversa começa durante a fase de planejamento de lançamento e continua quando a estória é programada.
- *Confirmação*: Os critérios de aceite, discutidos na conversa, são utilizados para confirmar que a estória foi realizada. Estes critérios de aceite podem se estender por várias estórias de usuários. Ambos os testes positivos e negativos devem ser utilizados para abranger os critérios. Durante a confirmação, vários participantes desempenham a função de um testador. Estes podem incluir desenvolvedores, bem como especialistas com foco em desempenho, segurança, interoperabilidade e outras características de qualidade. Para confirmar a realização de uma estória, os critérios de aceite definidos devem ser testados e atendidos.

As equipes ágeis variam em termos de como eles documentam estórias de usuários. Independentemente da abordagem adotada para documentar estórias de usuário, ela deve ser concisa, suficiente e necessária.

1.2.3 Retrospectivas

No desenvolvimento ágil, uma retrospectiva é uma reunião realizada no final de cada iteração para discutir o que foi bem-sucedido, o que poderia ser melhorado e como incorporar as melhorias e preservar os êxitos em iterações futuras. Retrospectivas abrangem temas como processo, pessoas, organizações, relacionamentos e ferramentas. Reuniões de retrospectivas são regularmente realizadas, quando ocorrem atividades de acompanhamento adequadas, sendo fundamentais para a auto-organização e melhoria contínua do desenvolvimento e testes.

As retrospectivas podem resultar em decisões de melhoria relacionadas com o teste, focadas na eficiência, produtividade, qualidade do caso de teste e satisfação da equipe. Eles também podem abordar a capacidade de teste dos aplicativos, estórias de usuários, funcionalidades ou interfaces do sistema. Análise de causa raiz de defeitos podem conduzir testes e desenvolvimento de melhorias. Em geral, as equipes devem implementar apenas algumas melhorias em cada iteração. Isto permite a melhoria contínua a um ritmo sustentado.

O tempo e a organização da retrospectiva, depende do método ágil específico seguido. Representantes de negócio e da equipe assistem cada retrospectiva como participantes, enquanto o facilitador organiza e dirige a reunião. Em alguns casos, as equipes podem convidar outros participantes para a reunião.

Os testadores devem desempenhar um papel importante nas retrospectivas. Os testadores são parte da equipe e promovem sua perspectiva única [ISTQB_FL_SYL], Capítulo 1.5. 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.

As retrospectivas devem ocorrer dentro de um ambiente profissional caracterizado pela confiança mútua. Os atributos de uma retrospectiva de sucesso são os mesmos do que qualquer outro, conforme discutido no programa *Foundation Level* [ISTQB_FL_SYL], Capítulo 3.2.

1.2.4 Integração contínua

O desenvolvimento de um incremento de produto requer software confiável integrado, atuando no final de cada *sprint*. A integração contínua aborda este desafio através da fusão de todas as alterações feitas no software e a integração de todos os componentes alterados regularmente, pelo menos uma vez por dia. A gestão da configuração, compilação, compilação de software, implementação e testes são envolvidos em um único processo repetitivo e automatizado. Visto que os desenvolvedores integram o seu trabalho constantemente, constroem constantemente, e testam constantemente, os defeitos no código são detectados mais rapidamente.

Na sequência da codificação dos desenvolvedores, depuração e *check-in* de código em um repositório de código-fonte, o processo de integração contínua é composto pelas seguintes atividades automatizadas:

- *Análise estática de código*: execução de análise estática de código e relatórios de resultados.
- *Compilação*: compilar e ligar o código, gerando os arquivos executáveis.
- *Teste da unidade*: executar os testes de unidade, verificando a cobertura de código e relatando os resultados dos testes.
- *Implantar*: instalar o projeto em um ambiente de teste.
- *Teste de integração*: executar os testes de integração e relatar os resultados.
- *Relatório* (*dashboard*): postar o status de todas essas atividades em um local publicamente visível ou status de e-mail visível para a equipe.

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, e enviar feedback rápido para a equipe sobre a qualidade do código. Estes resultados de testes são visíveis para todos os membros da equipe, especialmente quando os relatórios automatizados são integrados no processo. Teste de regressão automatizada pode ser contínuo ao longo da iteração. Testes de regressão automatizados abrangem a maior funcionalidade possível, incluindo estórias de usuários desenvolvidas nas iterações anteriores. A boa cobertura nos testes de regressão automatizados ajuda no desenvolvimento (e teste) de grandes sistemas integrados. Quando o teste de regressão é automatizado, os testadores ágeis são livres para concentrar seus testes manuais em novas funcionalidades, em mudanças, e em teste de confirmação de correções de defeitos.

Além de testes automatizados, as organizações que utilizam integração contínua normalmente utilizam ferramentas de desenvolvimento para implementar o controle de qualidade contínua. Além de realizar os testes unitários e de integração, tais ferramentas podem realizar testes dinâmicos e estáticos adicionais, medida e desempenho do perfil, extração e formatação de documentação a partir do código-fonte, e facilitar os processos manuais de garantia da qualidade. Esta aplicação

contínua de controle de qualidade tem por objetivo melhorar a qualidade do produto, bem como reduzir o tempo de desenvolvimento do mesmo, substituindo a prática tradicional de aplicação de controle da qualidade após completar todo o desenvolvimento.

As ferramentas de desenvolvimento podem ser ligadas a ferramentas de implantação automática, que podem buscar o desenvolvimento adequado da integração contínua ou desenvolver e implantá-lo em um ou mais de um servidor de desenvolvimento, teste, estágio, ou até mesmo ambientes de produção. Isso reduz os erros e atrasos associados à confiança na equipe especializada ou programadores para instalar os lançamentos nesses ambientes.

A integração contínua pode fornecer os seguintes benefícios:

- Permite a detecção mais precoce e análise mais fácil da causa raiz de problemas de integração e mudanças conflitantes.
- Dá o feedback regular à equipe de desenvolvimento se o código estiver funcionando.
- Mantém a versão do software que está sendo testada dentro de um dia da versão que está sendo desenvolvida.
- Reduz o risco de regressão associado com a reconstrução do código do desenvolvedor devido ao rápido reteste da base de código após cada pequeno conjunto de alterações.
- Promove confiança de que o trabalho de desenvolvimento de cada dia é fundamentado em uma base sólida.
- Realiza progresso em direção a conclusão do incremento do produto visível, encorajando desenvolvedores e testadores.
- Elimina os riscos de programação associada à integração do *big-bang*.
- Fornece disponibilidade constante de software executável em todo o *sprint* para fins de teste, demonstração ou de treinamento.
- Reduz as atividades de teste manuais repetitivas.
- Fornece feedback rápido sobre as decisões tomadas para melhorar a qualidade e os testes.

No entanto, a integração contínua tem seus riscos e desafios:

- 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 seu estabelecimento.
- A cobertura completa de teste é essencial para alcançar vantagens nos testes automatizados.
- As equipes, por vezes, confiam excessivamente nos testes de unidade e realizam muito pouco do teste de sistema e de aceite.

A integração contínua requer o uso de ferramentas, incluindo ferramentas para testes, para automatização, para o processo de construção e controle de versão.

1.2.5 Planejamento de iteração e lançamento

Como mencionado no programa *Foundation Level* [ISTQB_FL_SYL], o planejamento é uma atividade em curso, e este é o caso nos ciclos de vida ágil. Nos ciclos de vida ágil, dois tipos de planejamento ocorrem, planejamento de lançamento e planejamento de iteração.

O planejamento do lançamento foca a liberação de um produto, muitas vezes, alguns meses antes do início do projeto. O planejamento do lançamento define e redefine o *backlog* do produto, e pode envolver o refinamento das maiores estórias de usuários em uma coleção de estórias menores. Este planejamento fornece a base para uma abordagem de teste e plano de teste que abrange todas as iterações. Os planos de lançamento são de alto nível.

No planejamento do lançamento, os representantes da empresa estabelecem e priorizam as estórias de usuários para o lançamento do produto, em colaboração com a equipe (ver capítulo 1.2.2). Com base nas estórias de usuários, riscos associados ao projeto e qualidade são identificados, e uma estimativa de esforço de alto nível é desenvolvida (ver capítulo 3.2).

Os testadores são envolvidos no planejamento do lançamento para, especialmente, agregar valor nas seguintes atividades:

- Definir estórias de usuários testáveis, incluindo os critérios de aceite.
- Participação no projeto e análise do risco da qualidade.
- Estimativa de esforço de teste associado às estórias do usuário.
- Definir os níveis de testes necessários.
- Planejar o teste para o lançamento.

Após a realização do planejamento do lançamento, o planejamento de iteração para a primeira iteração é iniciado. O planejamento de iteração foca o final de uma única iteração e está relacionado ao *backlog* desta iteração.

No planejamento de iteração, a equipe seleciona estórias de usuário priorizadas do *backlog* de liberação, elabora estórias de usuários, realiza uma análise de risco para as estórias de usuários e estima o trabalho necessário para cada uma. Se uma estória de usuário é demasiadamente vaga e tenta esclarecer que falhou, a equipe pode se recusar a aceitá-la passando a utilizar a próxima estória de usuário baseada na prioridade. 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 uma.

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

Os testadores são envolvidos no planejamento de iteração para, 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 aceite 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 na automação de testes nos vários níveis de testes.

Os planos de lançamento podem mudar à medida que o projeto avança, incluindo alterações em estórias de usuários individuais no *backlog* do produto. Estas alterações podem ser desencadeadas por fatores internos ou externos. Os fatores internos incluem capacidade de desenvolvimento,

velocidade e questões técnicas. Os fatores externos incluem a descoberta de novos mercados e oportunidades, novos concorrentes, ou ameaças de negócios que podem mudar os objetivos de lançamento e/ou prazos. Além disso, os planos de iteração podem mudar durante uma iteração. Por exemplo, uma estória de usuário particular, que foi considerada relativamente simples durante a estimativa pode ser mais complexa do que o esperado.

Essas mudanças podem ser um desafio para os testadores. Os testadores devem compreender o panorama do lançamento para fins de planejamento do teste, e devem ter uma base de teste e oráculo adequados para cada iteração para fim de desenvolvimento do teste, como discutido no programa *Foundation Level* [ISTQB_FL_SYL], capítulo 1.4. As informações solicitadas devem estar disponíveis com antecedência para o testador, e ainda a mudança deve ser adotada de acordo com os princípios ágeis. Este dilema requer decisões cuidadosas sobre as estratégias de teste e documentação de teste. Para saber mais sobre desafios de testes ágeis, ver [Black09], capítulo 12.

Os planejamentos de lançamento e iteração devem abordar o plano de teste, bem como o planejamento das atividades de desenvolvimento. As questões relacionadas especificamente com a abordagem do teste incluem:

- O escopo dos testes, a extensão dos testes para as áreas de escopo, os objetivos do teste, e as razões para tais decisões.
- Os membros da equipe que realizarão as atividades de teste.
- O ambiente e dados de teste necessários, e quaisquer acréscimos ou alterações em um deles ocorrerão antes ou durante o projeto.
- O tempo, sequenciamento, dependências e pré-requisitos para as atividades de teste funcional e não-funcional (p.e., a frequência de realização dos testes de regressão, cujas funcionalidades dependem de outras funcionalidades ou dados de teste, etc.), incluindo a forma como as atividades de teste se relacionam e dependem das atividades de desenvolvimento.
- O projeto e riscos da qualidade a serem abordados (ver Capítulo 3.2.1).

Além disso, a maior estimativa de esforço da equipe deve ter em consideração o tempo e o esforço necessários para completar as atividades de testes.

Capítulo 2: Princípios fundamentais do teste ágil, práticas e processos (105 min)

Palavras-chave.

Elaborar teste de verificação, item de configuração, gestão da configuração

Objetivos de aprendizagem

2.1 As diferenças entre os testes em abordagens tradicionais e do ágil

FA-2.1.1 (K2) Descrever as diferenças entre as atividades de teste em projetos ágeis e projetos não ágeis.

FA-2.1.2 (K2) Descrever como as atividades de desenvolvimento e teste são integradas nos projetos ágeis

FA-2.1.3 (K2) Descrever a função dos testes independentes em projetos ágeis

2.2 Status de testes em projetos ágeis

FA-2.2.1 (K2) Descrever as ferramentas e técnicas utilizadas para comunicar o status de teste em um projeto ágil, incluindo a evolução de teste e qualidade do produto

FA-2.2.2 (K2) Descrever o processo de evolução de testes em várias iterações e explicar por que a automação de teste é importante para gerir o risco de regressão em projetos ágeis

2.3 Função e habilidades de um testador em uma equipe ágil

FA-2.3.1 (K2) Compreender as habilidades (pessoas, domínio e teste) de um testador em uma equipe ágil

FA-2.3.2 (K2) Compreender a função de um testador na equipe ágil

2.1 As Diferenças entre os testes em abordagens tradicionais e no ágil

Conforme descrito no programa *Foundation Level* [ISTQB_FL_SYL] e em [Black09], atividades de teste estão relacionadas com as atividades de desenvolvimento e, portanto, o teste varia em diferentes ciclos de vida. Os testadores devem compreender as diferenças entre os testes em modelos de ciclo de vida tradicionais (sequencial, por exemplo, tais como o modelo V ou interativo, tais como RUP) e ciclos de vida ágil, a fim de trabalhar de forma eficaz e eficiente. Os modelos ágeis diferem quanto o modo pelo qual as atividades de teste e desenvolvimento são integradas, os produtos do projeto de trabalho, os nomes, critérios de entrada e saída utilizados para vários níveis de testes, o uso de ferramentas, e como o teste independente pode ser efetivamente utilizado.

Os testadores devem se lembrar de que as organizações variam consideravelmente em sua implementação de ciclos de vida. Os desvios dos ideais de ciclos de vida ágil (ver Capítulo 1.1) podem representar personalização inteligente e adaptação das práticas. A capacidade de adaptar-se ao contexto de um dado projeto, incluindo as práticas de desenvolvimento de software, na verdade é um fator chave de sucesso para os testadores.

2.1.1 Atividades de teste e desenvolvimento

Uma das principais diferenças entre os ciclos de vida tradicionais e os ciclos de vida ágil é a ideia de iterações muito curtas, cada iteração resultando em um software que oferece funcionalidades de valor para as partes interessadas. No início do projeto, há um período de planejamento de lançamento. Isto é seguido por uma sequência de iterações. No início de cada iteração, há um período de planejamento de iteração. Após o estabelecimento do escopo da iteração, as estórias de usuários selecionadas são desenvolvidas, integradas com o sistema, e testadas. Estas iterações são altamente dinâmicas, com o desenvolvimento, integração e atividades de teste que ocorrem ao longo de cada iteração, e com paralelismo e sobreposição consideráveis. As atividades de teste ocorrem durante toda a iteração, e não como uma atividade final.

Os testadores, desenvolvedores e as partes interessadas do negócio têm uma função nos testes, assim como nos ciclos de vida tradicionais. Os desenvolvedores executam testes de unidade conforme desenvolvem as funcionalidades das estórias de usuários. Os testadores testam essas funcionalidades. As partes interessadas também testam as estórias durante a implementação. As partes interessadas podem usar casos de teste escritos, mas também podem simplesmente experimentar usando a funcionalidade, a fim de fornecer um feedback rápido para a equipe de desenvolvimento.

Em alguns casos, as iterações de amadurecimento ou de estabilização ocorrem periodicamente para resolver quaisquer defeitos remanescentes e outras formas de dívida técnica. No entanto, a melhor prática é que nenhuma funcionalidade seja considerada finalizada até que tenha sido integrada e testada com o sistema [Goucher09]. Outra boa prática é tratar defeitos remanescentes da iteração anterior no início da próxima iteração, como parte do *backlog* da iteração (referido como "corrigir bugs em primeiro lugar"). No entanto, alguns queixam-se que esta prática resulta numa situação em que o trabalho total a ser feito na iteração é desconhecido, e que vai ser mais difícil estimar quando as funcionalidades restantes serão realizadas. No final da sequência de iterações, pode

haver um conjunto de atividades de lançamento para obter o software pronto para entrega, embora em alguns casos, ocorra a liberação no fim de cada iteração.

Quando o teste baseado em risco é usado como uma das estratégias de teste, uma análise de risco de alto nível ocorre durante o planejamento do lançamento, conduzidas muitas vezes por testadores. No entanto, os riscos específicos de qualidade associados a cada iteração são identificados e avaliados no planejamento da iteração. Esta análise de risco pode influenciar a sequência de desenvolvimento, assim como a prioridade e profundidade do teste das funcionalidades. Da mesma forma, influencia a estimativa do esforço de teste necessário para cada função (ver capítulo 3.2).

Em algumas práticas ágeis (p.e., *Extreme Programming*), o emparelhamento é usado. O emparelhamento pode envolver testadores trabalhando juntos em pares para testar a funcionalidade. O emparelhamento também pode envolver um testador atuando em colaboração com um desenvolvedor para desenvolver e testar uma funcionalidade. O emparelhamento pode ser difícil quando a equipe de teste está distribuída, mas os processos e as ferramentas podem ajudar a viabilizá-lo. Para obter mais informações sobre o trabalho distribuído, ver [ISTQB_ALTM_SYL], capítulo 2.8.

Os testadores também podem servir como treinadores de teste e qualidade dentro da equipe, compartilhando conhecimentos e apoiando o trabalho de garantia de qualidade dentro da equipe. Isto promove um senso de responsabilidade coletiva na qualidade do produto.

A automação em todos os níveis de testes ocorre em muitas equipes ágeis, e isso pode significar que os testadores gastam mais tempo na criação, execução, monitoramento e manutenção de testes automatizados e em resultados. Devido ao pesado uso da automação de testes, uma percentagem mais elevada do teste manual nos projetos ágeis tende a ser feita usando técnicas baseadas em experiência e defeitos, tais como ataques de software, testes exploratórios, e suposição de erro (ver [ISTQB_ALTA_SYL], capítulos 3.3 e 3.4 e [ISTQB_FL_SYL], capítulo 4.5). Enquanto os desenvolvedores se concentram na criação de testes de unidade, os testadores devem se concentrar na criação de testes de integração, testes de sistema e integração de sistemas automatizados. Isto leva a uma tendência das equipes ágeis em favorecer os testadores com uma sólida formação técnica e conhecimento em automação de testes.

Um princípio fundamental ágil é que pode ocorrer mudança durante o projeto. Portanto, uma documentação do produto de trabalho leve é favorecida nos projetos ágeis. Mudanças nas funcionalidades existentes têm implicações nos testes, especialmente implicações nos testes de regressão. O uso de testes automatizados é uma forma de gerenciar o volume de esforço de teste associado com a mudança, no entanto, é importante que a taxa de variação não exceda a capacidade da equipe do projeto de lidar com os riscos associados a essas mudanças.

2.1.2 Produtos de trabalho do projeto

Os produtos de trabalho do projeto de interesse imediato para testadores ágeis tipicamente se enquadram em três categorias:

1. Produtos de trabalho orientados para o negócio que descrevem o que é necessário (p.e., requisitos de especificações) e como usá-los (p.e., a documentação do usuário).

2. Produtos de trabalho de desenvolvimento que descrevem como o sistema é construído (p.e., diagramas de banco de dados entidade-relacionamento), que na verdade implementam o sistema (p.e., código) ou que avaliam códigos individuais (p.e., testes de unidade automatizados).
3. Produtos de trabalho de teste que descrevem como o sistema é testado (p.e., estratégias e planos de teste), que realmente testam o sistema (p.e., testes manuais e automatizados), ou que apresentam os resultados do teste (p.e., painéis de teste, conforme discutidos no capítulo 2.2.1).

Em um projeto ágil típico é uma prática comum evitar a produção de grandes volumes de documentação. Ao contrário, o foco é mais em ter um software funcionando em conjunto com os testes automatizados que demonstrem o cumprimento das exigências. Este incentivo para reduzir a documentação se aplica apenas à documentação que não atribui valor ao cliente. Em um projeto ágil bem-sucedido, estabelece-se um equilíbrio entre o aumento da eficiência através da redução de documentação e fornecimento de documentação suficiente para apoiar as atividades de negócios, testes, desenvolvimento e manutenção. A equipe deve tomar uma decisão durante o planejamento do lançamento sobre quais produtos de trabalho são necessários e qual o nível de documentação do produto do trabalho.

Os produtos de trabalho típicos orientados para o negócio nos projetos ágeis incluem estórias de usuários e critérios de aceite. Estórias de usuários são os formulários ágeis de especificações de requisitos, e devem explicar como o sistema deve se comportar em relação a uma única e coerente funcionalidade ou função. Uma estória de usuário deve definir uma funcionalidade suficientemente pequena para ser concluída em uma única iteração. As maiores coleções de funcionalidades relacionadas, ou uma coleção de sub-funcionalidades que compõem uma única característica complexa, podem ser referidas como "épicos". Épicos podem incluir estórias de usuários para diferentes equipes de desenvolvimento. Por exemplo, uma estória de usuário pode descrever o que é necessário no nível da API (middleware), enquanto uma outra estória descreve o que é necessário ao nível UI (aplicativo). Essas coleções podem ser desenvolvidas através de uma série de *sprints*. Cada épico e suas estórias de usuários devem ter critérios de aceite associados.

Os produtos típicos de trabalho do desenvolvedor em projetos ágil incluem código. Os desenvolvedores ágeis também muitas vezes criam testes de unidade automatizados após o desenvolvimento do código. Em alguns casos, no entanto, os desenvolvedores criam testes de modo incremental antes que cada parte do código seja escrita, afim de proporcionar um modo de verificação de que aquela parte do código escrita funciona como o esperado. Embora esta abordagem seja referida como primeiro teste ou desenvolvimento orientado a testes, na realidade, estes testes são mais uma forma de baixo nível de execução das especificações de projeto ao invés de testes [Beck02].

Os produtos típicos de trabalho do testador em projetos ágil incluem testes automatizados, bem como planos de teste, catálogos de risco de qualidade, testes manuais, relatórios de defeitos e logs de resultados de testes. Os documentos são capturados da forma mais leve possível, o que muitas vezes ocorre com estes documentos nos ciclos de vida tradicionais. Os testadores também vão produzir métricas de teste ao partir de relatórios de defeitos e *logs* de resultados de testes, e novamente, há uma ênfase em uma abordagem leve.

Em algumas implementações ágeis, especialmente regulatórias, de segurança crítica, projetos e produtos distribuídos ou altamente complexos, é necessária uma maior formalização dos produtos de trabalho. Por exemplo, algumas equipes transformam estórias de usuários e critérios de aceite em mais especificações de requisitos formais. Relatórios de rastreabilidade vertical e horizontal podem ser preparados para atender os auditores, regulamentos e outros requisitos.

2.1.3 Níveis de teste

Os níveis de teste são atividades de teste logicamente relacionadas, muitas vezes, pela maturidade ou integridade do item em teste.

Nos modelos do ciclo de vida sequencial, os níveis de teste são geralmente definidos de modo que os critérios de saída de um nível sejam parte dos critérios de entrada para o próximo nível. Em alguns modelos iterativos, esta regra não se aplica. Os níveis de teste se sobrepõem. As especificações de requisitos, de modelagem, e das atividades de desenvolvimento podem sobrepor-se com os níveis de teste.

Em alguns ciclos de vida ágeis, esta sobreposição ocorre porque as mudanças nos requisitos, projeto e código podem acontecer a qualquer momento em uma iteração. Embora o *Scrum*, em tese, não permita alterações nas estórias de usuários após o planejamento da iteração, na prática, essas mudanças ocorrem ocasionalmente. Durante uma iteração, qualquer estória de usuário geralmente progride sequencialmente através das seguintes atividades de teste:

- Teste de unidade, normalmente feito pelo desenvolvedor.
- Teste de aceite da funcionalidade, que é às vezes dividido em duas atividades:
    - Testes de verificação de funcionalidades, que muitas vezes é automatizado, podem ser feitos por desenvolvedores ou testadores, e envolve testes contra os critérios de aceite da estória do usuário.
    - Testes de validação de funcionalidades, que normalmente é manual e pode envolver desenvolvedores, testadores e partes interessadas que trabalham de forma colaborativa para determinar se a funcionalidade está apta para uso, para melhorar a visibilidade dos progressos realizados, e receber feedback real das partes interessadas.

Além disso, há muitas vezes um processo paralelo de testes de regressão que ocorre em toda a iteração. Trata-se de reexecutar os testes de unidade automatizados e testes de verificação da funcionalidade da iteração atual e iterações anteriores, geralmente através de uma estrutura de integração contínua.

Em alguns projetos ágeis, pode haver um nível de teste do sistema, uma vez que se inicia a primeira estória de usuário pronto para tais testes. Isso pode envolver a execução de testes funcionais, bem como testes não-funcionais de desempenho, confiabilidade, usabilidade, e outros tipos de teste pertinentes.

As equipes ágeis podem empregar diversas formas de testes de aceite (usando o termo conforme explicado no programa *Foundation Level* [ISTQB_FL_SYL]). Testes alfa internos e testes beta externos podem ocorrer, quer no final de cada iteração, após a conclusão de cada iteração, ou após uma série de iterações. Os testes de aceite: do usuário, operacionais, regulamentação e contrato

também podem ocorrer, ou no final de cada iteração após sua conclusão, ou após uma série de iterações.

2.1.4 Gestão de testes e configuração

Os projetos ágeis frequentemente envolvem o uso pesado de ferramentas automatizadas para desenvolver, testar e gerenciar o desenvolvimento de software. Os desenvolvedores usam ferramentas para análise estática, testes unitários e cobertura de código. Eles verificam continuamente o código e os testes de unidade em um sistema de gerenciamento de configuração, utilizando estruturas automatizadas de desenvolvimento e teste. Estas estruturas permitem a integração contínua do novo software com o sistema, com a análise estática e testes de unidade realizados repetidamente conforme o novo software é verificado [Kubaczkowski].

Estes testes automatizados também podem incluir testes funcionais nos níveis de integração e de sistema. Tais testes automatizados funcionais podem ser criados usando equipamentos de teste funcional, interface de usuário *opensource*, ferramentas de teste funcionais, ou ferramentas de negócio, e podem ser integrados com os testes automatizados executados como parte da estrutura de integração contínua. Em alguns casos, devido à duração, os testes funcionais são separados dos testes de unidade e executados com menos frequência. Por exemplo, os testes de unidade podem ser executados cada vez que um novo software é verificado, enquanto os testes funcionais mais longos são executados apenas por alguns dias.

Um dos objetivos dos testes automatizados é confirmar que o projeto está funcionando e é instalável. Se algum teste automatizado falhar, a equipe deve corrigir o defeito a tempo para a próxima verificação do código. Isto requer um investimento em relatórios de teste em tempo real para fornecer uma boa visibilidade dos resultados dos testes. Essa abordagem ajuda a reduzir os ciclos caros e ineficientes de "projetar-instalar-falhar-reprojetar-reinstalar" que pode ocorrer em muitos projetos tradicionais, uma vez que as mudanças que ocasionam as falhas no projeto de software na instalação são detectadas rapidamente.

As ferramentas automatizadas de teste e projeto ajudam a gerir o risco de regressão associados às frequentes mudanças que muitas vezes ocorrem nos projetos ágeis. No entanto, a excessiva dependência de teste de unidade automatizada isolada para administrar esses riscos pode ser um problema, pois o teste de unidade muitas vezes tem limitado a eficácia da detecção dos defeitos [Jones11]. Os testes automatizados nos níveis de integração e de sistema também são necessários.

2.1.5 Opções organizacionais para teste independente

Como discutido no programa *Foundation Level* [ISTQB_FL_SYL], os testadores independentes são muitas vezes mais eficazes na detecção dos defeitos. Em algumas equipes ágeis, os desenvolvedores criam muitos testes no formato automatizado. Um ou mais testadores podem ser agregados à equipe, realizando muitas das tarefas de teste. No entanto, dada a posição desses testadores dentro da equipe, há um risco da perda de independência e avaliação objetiva.

Outras equipes ágeis mantêm equipes de teste totalmente separadas e independentes, e atribuem testadores sob demanda durante os dias finais de cada *sprint*. Isso pode preservar a independência, e esses testes podem fornecer uma avaliação objetiva e imparcial do software. No entanto, as

pressões de tempo, a falta de compreensão das novas funcionalidades do produto e problemas de relacionamento com as partes interessadas e desenvolvedores muitas vezes levam a problemas com essa abordagem.

Uma terceira opção é ter uma equipe de teste separada e independente onde os testadores são designados para as equipes ágeis em uma base de longo prazo, no início do projeto, permitindo-lhes manter a sua independência ao adquirir um bom entendimento do produto e das relações fortes com outros membros da equipe. Além disso, a equipe de teste independente pode ter testadores especializados fora das equipes ágeis para trabalhar em atividades independentes de longo prazo e/ou iteração, tais como o desenvolvimento de ferramentas de teste automatizado, a realização de teste não funcional, criar e apoiar ambientes de teste e de dados, e realização de níveis de teste que pode não se encaixar em um *sprint* (p.e., teste de integração do sistema).

2.2 Status de teste em projetos ágeis

A mudança ocorre rapidamente nos projetos ágeis. Esta mudança significa que o status do teste, seu progresso e a qualidade do produto evoluam constantemente, onde os testadores devem elaborar maneiras de obter essas informações para a equipe, para que possam tomar decisões para se manterem no caminho certo para a conclusão de cada iteração. Além disso, a mudança pode afetar as funcionalidades existentes de iterações anteriores.

Portanto, os testes manuais e automatizados devem ser atualizados para lidar eficazmente com risco de regressão.

2.2.1 Comunicação do status, progresso de teste e qualidade do produto

As equipes ágeis progridem tendo o software funcionando no final de cada iteração. Para determinar quando a equipe estará trabalhando com o software, eles precisam monitorar o progresso de todos os itens de trabalho na iteração e lançamento. Os testadores nas equipes ágeis utilizam vários métodos para registrar o progresso e o status do teste, incluindo os resultados dos testes de automação, progressão das tarefas de teste e estórias sobre o quadro de tarefas ágeis e gráficos *burndown* que mostram o progresso da equipe. Estes podem então ser compartilhados ao resto da equipe, usando a mídia, como painéis de *wiki* e emails do tipo painel, bem como verbalmente durante as reuniões. Equipes ágeis podem usar ferramentas que geram automaticamente relatórios de status com base em resultados de testes e progresso da tarefa, que por sua vez atualizam *dashboards* e emails no estilo *wiki*. Este método de comunicação reúne também as métricas do processo de teste, que podem ser utilizadas na melhoria do processo. A comunicação sobre o status dos testes de forma automatizada libera o tempo dos testadores para se concentrar na concepção e execução de mais casos de teste.

As equipes podem usar gráficos *burndown* para acompanhar o progresso do lançamento e em cada iteração. Um gráfico *burndown* [Crispin08] representa o volume de trabalho a ser realizado contra o tempo alocado para o lançamento ou iteração.

Para fornecer uma representação visual instantânea e detalhada do status atual de toda a equipe, incluindo o status de teste, as equipes podem usar quadros de tarefas ágeis. Os cartões de estória, tarefas de desenvolvimento, tarefas de teste e outras tarefas criadas durante o planejamento da iteração (ver capítulo 1.2.5) são capturados no quadro de tarefas, muitas vezes utilizando cartões

de cores coordenadas para determinar o tipo de tarefa. Durante a iteração, o progresso é gerido através do deslocamento dessas tarefas no quadro de tarefas em colunas tais como trabalho a realizar, trabalho em progresso, verificação e trabalho realizado. As equipes ágeis podem utilizar ferramentas para manter seus cartões de estória e quadros de tarefas ágeis, automatizados em *dashboards* com atualizações de status.

As tarefas de teste no quadro de tarefas se relacionam com os critérios de aceite definidos para as estórias de usuários. Como *scripts* de automação de teste, testes manuais e testes exploratórios para uma tarefa de teste atingem um status concluído, a tarefa passa para a coluna de trabalho realizado do quadro de tarefas. A equipe inteira analisa o status do quadro de tarefas regularmente, muitas vezes durante as reuniões diárias, para assegurar que as tarefas estão se movendo em todo o quadro a um ritmo aceitável. Se alguma tarefa (incluindo tarefas de teste) não estiver se movendo ou estiver se movendo muito lentamente, a equipe comenta e aborda todas as questões que possam estar bloqueando o progresso destas tarefas.

A reunião diária inclui todos os membros da equipe ágil, incluindo testadores. Nessa reunião, eles comunicam seu status atual. A agenda de cada membro é [Guia do Ágil Alliance]:

- O que você concluiu desde a última reunião?
- O que você planeja concluir até a próxima reunião?
- O que está bloqueando o seu caminho?

Quaisquer problemas que possam bloquear o progresso dos testes são comunicados durante as reuniões diárias, para que toda a equipe esteja ciente dos problemas e possa resolvê-los em conformidade.

Para melhorar a qualidade geral do produto, muitas equipes ágeis realizam pesquisas de satisfação dos clientes para receber feedback sobre se o produto atende as expectativas dos clientes. As equipes podem utilizar outras métricas semelhantes às capturadas em metodologias de desenvolvimento tradicionais, tais como taxas de aprovação/reprovação de teste, taxas de detecção de defeitos, resultados de teste de confirmação e regressão, densidade de defeitos, defeitos detectados e corrigidos, cobertura de requisitos, cobertura de riscos, cobertura de código, e rotatividade do código para melhorar a qualidade do produto.

Como acontece com qualquer ciclo de vida, as métricas capturadas e relatadas devem ser relevantes e ajudar na tomada de decisões. As métricas não devem ser utilizadas para premiar, punir ou afastar quaisquer membros da equipe.

2.2.2 Gestão de risco de regressão com evolução dos casos de teste manuais e automatizado

Em um projeto ágil, assim que cada iteração é concluída, o produto cresce. Por conseguinte, no âmbito dos testes, eles também aumentam. Juntamente com o teste das alterações no código na iteração atual, os testadores também precisam verificar se alguma regressão foi introduzida nas funcionalidades que foram desenvolvidas e testadas em iterações anteriores. O risco de introduzir uma regressão no desenvolvimento ágil é alto, devido ao código extenso (linhas de código adicionadas, modificadas ou apagadas de uma versão para outra). Visto que responder à mudança é um princípio chave ágil, as mudanças também podem ser feitas em funcionalidades anteriormente

desenvolvidas para atender às necessidades comerciais. A fim de manter a velocidade sem incorrer em um grande volume de dívida técnica, é fundamental que as equipes invistam em automação de testes em todos os níveis o mais cedo possível. Também é fundamental que todos os ativos de teste, tais como testes automatizados, casos de teste manuais, dados de teste e outros artefatos de teste sejam mantidos atualizados com cada iteração. É altamente recomendável que todos os ativos de teste sejam mantidos em uma ferramenta de gestão de configuração, a fim de permitir o controle de versão, para assegurar a facilidade de acesso por todos os membros da equipe, e para apoiar as alterações necessárias devido à mudanças de funcionalidade e ainda preservar a informação histórica dos ativos de teste.

Como a repetição completa de todos os testes raramente é possível, especialmente em projetos ágeis com cronograma apertado, os testadores precisam alocar tempo em cada iteração para rever casos de teste manuais e automatizados de iterações anteriores e atuais para selecionar casos de teste que podem ser candidatos ao teste de regressão, e para retirar casos de teste que não são mais relevantes. Testes escritos em iterações anteriores para verificar funcionalidades específicas podem ter pouco valor em iterações posteriores devido a alterações nas funcionalidades ou em novas que alteram a forma como as anteriores se comportam.

Ao rever os casos de teste, os testadores devem considerar a adequação para automação. A equipe precisa automatizar o máximo de testes possíveis de iterações anteriores e atuais. Isso permite que os testes de regressão automatizados reduzam o risco de regressão, com menos esforço que os testes de regressão manual exigiriam. Este esforço de teste de regressão reduzido libera os testadores para testar mais a fundo novas funcionalidades e funções na iteração atual.

É fundamental que os testadores tenham a capacidade de identificar rapidamente e atualizar os casos de teste a partir de iterações e/ou versões anteriores que são afetadas pelas alterações feitas na iteração atual. Definir como a equipe projeta, escreve e armazena casos de teste deve ocorrer durante o planejamento do lançamento. Boas práticas para a modelagem e implementação dos testes precisam ser aprovadas no início e aplicadas de forma consistente. Os prazos mais curtos para testes e a mudança constante em cada iteração aumentará o impacto do projeto de teste e das práticas de implementação ineficientes.

O uso de automação de teste, em todos os níveis de teste, permite que equipes ágeis forneçam feedback rápido sobre a qualidade do produto. Testes automatizados bem escritos fornecem um documento vivo da funcionalidade do sistema [Crispin08]. O uso do sistema de gerenciamento de configuração para verificação dos testes automatizados e dos resultados dos testes, em conformidade com a versão do produto, permite que as equipes ágeis possam rever a funcionalidade testada e os resultados dos testes de uma determinada configuração a qualquer momento.

Testes de unidade automatizados são executados antes que o código fonte seja marcado na linha principal do sistema de gestão de configuração para garantir que as alterações de código não prejudiquem o desenvolvimento do software. Para reduzir os prejuízos no desenvolvimento, que possam retardar o progresso de toda a equipe, o código não deve ser verificado, a menos que todos os testes de unidade automatizados sejam aprovados. Os resultados dos testes de unidade automatizados fornecem feedback imediato quanto ao código e a qualidade do projeto, mas não quanto à qualidade do produto.

Testes de aceite automatizados são executados regularmente, como parte da integração contínua completa do projeto do sistema. Estes testes são executados com referência à um projeto do sistema completo, pelo menos diariamente, mas geralmente não são realizados com cada verificação do código, pois eles levam mais tempo para executar do que os testes de unidade automatizados e podem atrasar a verificação do código. Os resultados dos testes de aceite automatizados fornecem feedback sobre a qualidade do produto em relação a regressão desde a última compilação, mas não fornecem status da qualidade geral do produto.

Os testes automatizados podem ser executados de forma contínua em relação ao sistema. Um subconjunto inicial de testes automatizados para cobrir a funcionalidade crítica do sistema e pontos de integração deve ser criado imediatamente após uma nova compilação ser implantada no ambiente de teste. Estes testes são comumente conhecidos como testes de verificação de compilação. Os resultados dos testes de verificação de compilação vão fornecer um feedback instantâneo sobre o software após a implantação, para que as equipes não percam tempo testando uma configuração instável.

Os testes automatizados contidos no conjunto de testes de regressão são geralmente executados como parte da compilação principal diária no ambiente de integração contínua, e novamente quando uma nova compilação é implantada no ambiente de teste. Quando um teste de regressão automatizado falha, a equipe interrompe as atividades e investiga as razões da falha do teste. O teste pode ter falhado devido a alterações funcionais legítimas na iteração atual, neste caso, a estória do usuário e/ou seu teste pode precisar ser atualizado para refletir os novos critérios de aceite. Alternativamente, o teste poderá ser retirado se outro teste foi compilado para cobrir as modificações. No entanto, se o teste falhou devido a um defeito, é uma boa prática para a equipe corrigir o defeito antes de avançar com novas funcionalidades.

Além da automação de teste, as seguintes tarefas também podem ser automatizadas:

- Geração de dados de teste.
- Carregamento dos dados de teste nos sistemas.
- Implantação de compilações para os ambientes de teste.
- Restauração de um ambiente de teste (p.e., os arquivos de dados do banco de dados ou site) para uma linha de base.
- Comparação das saídas de dados.

A automação destas tarefas reduz a sobrecarga e permite que a equipe tenha tempo para desenvolver e testar novas funcionalidades.

2.3 Função e habilidades de um testador em uma equipe ágil

Em uma equipe ágil, testadores devem colaborar estreitamente com todos os outros membros da equipe e com as partes interessadas. Isso tem uma série de implicações em termos das habilidades que um testador deve ter e as atividades que eles realizam em uma equipe ágil.

2.3.1 Habilidades do testador ágil

Os testadores ágeis devem ter todas as habilidades mencionadas no programa *Foundation Level* [ISTQB_FL_SYL]. Além dessas habilidades, um testador em uma equipe do ágil deve ser competente

em automação de testes, desenvolvimento orientado a testes, desenvolvimento orientado a aceite, caixa-branca, caixa-preta, e testes baseados na experiência.

Como as metodologias do ágil dependem muito da colaboração, comunicação e interação entre os membros da equipe, bem como das partes interessadas fora da equipe, os testadores em uma equipe do ágil devem ter boas habilidades interpessoais. Nestas equipes os testadores devem:

- Serem positivos e orientados para solução com os membros da equipe e partes interessadas.
- Mostrar pensamento crítico e cético orientado para a qualidade do produto.
- Ativamente adquirir informações das partes interessadas (ao invés de confiar inteiramente em especificações escritas).
- Avaliar e relatar com precisão o progresso e o resultado dos testes e a qualidade do produto.
- Trabalhar efetivamente para definir estórias de usuários testáveis, especialmente os critérios para o aceite, com representantes dos clientes e partes interessadas.
- Colaborar dentro da equipe, trabalhando em pares com os programadores e outros membros da equipe.
- Responder rapidamente às mudanças, incluindo alterações, adições ou melhorias dos casos de teste.
- Planejar e organizar o seu próprio trabalho.

O crescimento contínuo de competências, incluindo o crescimento de habilidades interpessoais, é essencial para todos os testadores, incluindo aqueles das equipes do ágil.

2.3.2 Função de um testador em uma equipe do ágil

A função de um testador em uma equipe ágil inclui atividades que geram e fornecem feedback, não só no status de teste, progresso de teste e qualidade do produto, mas também na qualidade do processo. Além das atividades descritas em outra parte deste *syllabus*, essas atividades incluem:

- Compreender, implementar e atualizar a estratégia de teste.
- Medir e informar a cobertura do teste em todas as dimensões de coberturas aplicáveis.
- Garantir o uso adequado do ferramental de teste.
- Configurar, utilizar e gerenciar os ambientes de teste e os dados de teste.
- Relatar os defeitos e trabalhar com a equipe para resolvê-los.
- Treinar outros membros da equipe em aspectos relevantes aos testes.
- Assegurar que as tarefas adequadas de teste sejam programadas durante lançamento e planejamento da iteração.
- Colaborar ativamente com desenvolvedores e partes interessadas para esclarecer requisitos, especialmente em termos de testabilidade, consistência e completude.
- Participar ativamente de retrospectivas da equipe, sugerindo e implementando melhorias.

Dentro de uma equipe do ágil, cada membro da equipe é responsável pela qualidade do produto e desempenha um papel na execução de tarefas relacionadas com o teste.

As organizações ágeis podem detectar alguns riscos organizacionais relacionados com o teste:

- Os testadores trabalham tão estreitamente com os desenvolvedores que eles perdem a mentalidade apropriada de um testador.

- Os testadores se tornam tolerantes ou ausentes sobre práticas ineficientes, ineficazes, ou de baixa qualidade na equipe.
- Os testadores não podem manter o ritmo com as alterações realizadas em iterações com limitações de tempo. Para mitigar esses riscos, as organizações podem considerar variações para preservar a independência discutida no capítulo 2.1.5.

Capítulo 3: Técnicas, ferramentas e métodos de teste ágil (480 min)

Palavras-chave.

Critérios de aceite, testes exploratórios, testes de desempenho, risco do produto, risco de qualidade, testes de regressão, abordagem de teste, gráfico de teste, estimativa de teste, automação de execução do teste, estratégia de teste, desenvolvimento orientado a testes, estrutura de teste da unidade

Objetivos de Aprendizado

3.1 Métodos de teste do ágil

FA-3.1.1 (K1) Relembrar os conceitos de desenvolvimento orientado a testes, desenvolvimento orientado para teste, e desenvolvimento orientado a comportamento.

FA-3.1.2 (K1) Relembrar os conceitos da pirâmide de teste.

FA-3.1.3 (K2) Resumir os quadrantes de teste e suas relações com os níveis e tipos de teste.

FA-3.1.4 (K3) Para um determinado projeto ágil, praticar a função de um testador em uma equipe *Scrum.*

3.2 Avaliação de riscos de qualidade e estimativa do esforço de teste

FA-3.2.1 (K3) Avaliar os riscos de qualidade em um projeto ágil.

FA-3.2.2 (K3) Estimar o esforço do teste com base no conteúdo da iteração e nos riscos da qualidade.

3.3 Técnicas nos Projetos ágeis

FA-3.3.1 (K3) Interpretar as informações relevantes para apoiar as atividades de teste.

FA-3.3.2 (K2) Explicar às partes interessadas da empresa como definir critérios de aceite testáveis.

FA-3.3.3 (K3) escrever casos de teste de desenvolvimento orientado para teste de aceite a partir de estórias de usuário.

FA-3.3.4 (K3) Para o comportamento funcional e não-funcional, descrever casos de teste usando a técnica caixa-preta de projeto de teste com base em determinadas estórias de usuários.

FA-3.3.5 (K3) Realizar teste exploratório para apoiar o teste de um projeto ágil.

3.4 Ferramentas em Projetos ágeis

FA-3.4.1 (K1) Relembrar diferentes ferramentas disponíveis para testadores de acordo com sua finalidade e atividades nos projetos ágeis.

3.1 Métodos de teste do ágil

Existem certas práticas de teste que podem ser seguidas em todos os projetos de desenvolvimento (ágil ou não) para produzir produtos de qualidade. Estas práticas incluem em descrever os testes com antecedência, para expressar um comportamento adequado, com foco na prevenção precoce do defeito, detecção e remoção, e garantir que os tipos de teste corretos sejam realizados no momento certo e como parte do nível correto de teste. Os profissionais do ágil visam introduzir essas práticas mais cedo. Os testadores nos projetos ágeis desempenham um papel fundamental na orientação do uso dessas práticas de testes em todo o ciclo de vida.

3.1.1 Desenvolvimentos orientados para teste, teste de aceite e comportamento

Desenvolvimento orientado para teste, desenvolvimento orientado para teste de aceite, e desenvolvimento orientado para o comportamento são três técnicas complementares em uso entre as equipes ágeis para realizar testes entre os vários níveis de teste. Cada técnica é um exemplo de um princípio fundamental de teste, o benefício de teste e atividades de controle de qualidade (CQ), uma vez que os testes são definidos antes que o código seja escrito.

Desenvolvimento orientado para teste

O desenvolvimento orientado para teste (TDD - *Test Driven Development*) é usado para desenvolver código guiado por casos de testes automatizados. O processo de desenvolvimento orientado para testes é:

- Adicionar um teste que captura o conceito do programador do funcionamento desejado de uma pequena parte do código.
- Realizar o teste, o qual falhará uma vez que o código não existe.
- Escrever o código e realizar o teste em um *loop* estreito até o teste seja aprovado.
- Decompor o código após a aprovação do teste, reexecutar o teste para garantir a continuidade da aprovação do código decomposto.
- Repetir esse processo para a próxima pequena parte do código, realizando os testes anteriores, bem como os testes adicionados.

Os testes escritos são principalmente no nível de unidade e são focados no código, embora os testes também possam ser escritos nos níveis de integração ou de sistema. O TDD adquiriu sua popularidade através de *Extreme Programming* [Beck02], mas também é utilizado em outras metodologias do ágil e às vezes em ciclos de vida sequenciais. Ele ajuda os desenvolvedores a focar em resultados esperados claramente definidos. Os testes são automatizados e são utilizados na integração contínua.

Desenvolvimento orientado para teste de aceite

O desenvolvimento orientado para o teste de aceite (ATDD - *Acceptance Test Driven Development*) [Adzic09] define critérios e testes de aceite durante a criação das estórias de usuários (ver capítulo 1.2.2). O desenvolvimento orientado para teste de aceite é uma abordagem colaborativa que permite que todas as partes interessadas compreendam como o componente de software tem que se comportar e o que os desenvolvedores, testadores e as partes interessadas precisam para

garantir esse comportamento. O processo de desenvolvimento orientado para o teste de aceite desenvolvimento é explicado no capítulo 3.3.2.

O ATDD cria testes reutilizáveis para teste de regressão. Ferramentas específicas apoiam a criação e execução de tais testes, muitas vezes dentro do processo de integração contínua. Estas ferramentas podem se conectar a dados e serviços de camadas de aplicação, que permitem que os testes sejam realizados no nível do sistema ou aceite. O desenvolvimento orientado para teste de aceite permite a resolução rápida de defeitos e validação de comportamento da funcionalidade. Ele ajuda a determinar se os critérios de aceite são cumpridos para a funcionalidade.

Desenvolvimento orientado para o comportamento

O desenvolvimento orientado para o comportamento (BDD - *Behavior Driven Development*) [Chelimsky10] permite que um desenvolvedor se concentre em testar o código com base no comportamento esperado do software. Como os testes são baseados no comportamento exibido do software, os testes são geralmente mais fáceis para o entendimento de outros membros da equipe e partes interessadas.

As estruturas específicas de desenvolvimento orientado para o comportamento podem ser utilizadas para definir os critérios de aceite com base no formato dado/quando/então:

> *Dado algum contexto inicial,*
>
> *Quando ocorre um evento,*
>
> *Então aguarde alguns resultados.*

A partir desses requisitos, a estrutura de desenvolvimento orientado para o comportamento gera um código que pode ser usado por desenvolvedores para criar casos de teste. O BDD ajuda o desenvolvedor a colaborar com outras partes interessadas, incluindo os testadores para definir testes de unidade acurados focados nas necessidades comerciais.

3.1.2 Pirâmide de teste

Um sistema de software pode ser testado em diferentes níveis. Os típicos são, a partir da base da pirâmide para o topo, unidade, integração, sistema e aceite (ver [ISTQB_FL_SYL], capítulo 2.2). A pirâmide de teste enfatiza muitos testes para os níveis mais baixos (base da pirâmide) e, conforme o desenvolvimento se move para níveis superiores, o número de testes diminui (topo da pirâmide). Normalmente, os testes de unidade e no nível de integração são automatizados e são criados usando ferramentas baseadas em API. Nos níveis de sistema e de aceite, os testes automatizados são criados usando as ferramentas baseadas em GUI. O conceito de pirâmide teste é baseado no princípio de testes de controle de qualidade (ou seja, a eliminação de defeitos o mais cedo possível no ciclo de vida).

3.1.3 Quadrantes de teste, níveis de teste e tipos de teste

Os quadrantes de teste, definidos por *Brian Marick* [Crispin08], alinham os níveis de teste com os tipos de testes apropriados na metodologia ágil. O modelo de quadrantes de teste e suas variantes ajudam a assegurar que todos os tipos de testes importantes e os níveis de teste sejam incluídos no ciclo de vida de desenvolvimento. Este modelo também fornece uma maneira de diferenciar e

descrever os tipos de testes a todas as partes interessadas, incluindo desenvolvedores, testadores e representantes de negócio.

Nos quadrantes de teste, os testes podem ser um negócio (usuário) ou tecnologia (desenvolvedor). Alguns testes apoiam o trabalho realizado pela equipe ágil e confirmam o comportamento do software. Outros testes podem verificar o produto. Os testes podem ser totalmente manuais, totalmente automatizados, uma combinação de ambos, mas apoiados por ferramental. Os quatro quadrantes de teste são:

- O quadrante Q1 é o nível da unidade, voltado para tecnologia apoiando os desenvolvedores. Este quadrante contém testes de unidade. Estes testes devem ser automatizados e incluídos no processo de integração contínua.
- O quadrante Q2 é o nível do sistema, voltado para negócios, e confirma o comportamento do produto. Este quadrante contém testes funcionais, exemplos, testes de estória, protótipos de experiência do usuário, e simulações. Estes testes verificam os critérios de aceite e podem ser manuais ou automatizados. Eles são muitas vezes criados durante o desenvolvimento da estória do usuário e, assim, melhorar a qualidade das estórias. Eles são úteis na criação de suítes de teste automatizados de regressão.
- O quadrante Q3 é o nível de aceite do sistema ou do usuário, voltado para o negócio, e contém testes que criticam o produto, utilizando cenários e dados realistas. Este quadrante contém testes exploratórios, cenários, fluxos de processos, testes de usabilidade, teste de aceite do usuário, testes alfa e beta. Estes testes são muitas vezes manuais e orientados para o usuário.
- O quadrante Q4 é o nível de aceite operacional ou do sistema, orientado para tecnologia, e contém testes que criticam o produto. Este quadrante contém desempenho, carga, estresse e testes de escalabilidade, testes de segurança, manutenção, gestão de memória, compatibilidade e interoperabilidade, migração de dados, infraestrutura, e testes de recuperação. Estes testes são muitas vezes automatizados.

Durante uma determinada iteração, os testes de algum ou todos os quadrantes podem ser necessários. Os quadrantes de teste se aplicam a testes dinâmicos ao invés de testes estáticos.

3.1.4 A Função de um testador

Ao longo deste programa, foi feita referência geral a métodos e técnicas ágeis, e a função de um testador dentro de vários ciclos de vida do ágil. Este subcapítulo analisa especificamente a função de um testador em um projeto seguindo um ciclo de vida do *Scrum* [Aalst13].

Trabalho em equipe

Trabalho em equipe é um princípio fundamental no desenvolvimento ágil. O Ágil enfatiza a abordagem da equipe inteira composta por desenvolvedores, testadores e representantes do negócio que trabalham juntos. As melhores práticas organizacionais e comportamentais nas equipes *Scrum* são:

- *Multifuncional*: Cada membro da equipe traz um conjunto diferente de habilidades para a equipe. A equipe trabalha em conjunto na estratégia de teste, no planejamento de testes,

na especificação do teste, na execução dos testes, na avaliação dos teste, e no relato dos testes (relatórios).

- *Auto-organização*: A equipe pode consistir apenas de desenvolvedores, mas, como mencionado no capítulo 2.1.5, o ideal é que haja um ou mais testadores.
- *Co-localizado*: Os testadores se reúnem com os desenvolvedores e o gestor do produto.
- *Colaborativo*: Os testadores colaboram com os membros da equipe, com outras equipes, as partes interessadas, o gestor do produto, e o *Scrum Master*.
- *Capacitado*: As decisões técnicas de projeto e teste são tomadas pela equipe como um todo (desenvolvedores, testadores e *Scrum Master*), em colaboração com o gestor do produto e outras equipes, se necessário.
- *Comprometido*: O testador tem o compromisso de questionar e avaliar o comportamento e as características do produto em relação às expectativas e necessidades dos clientes e usuários.
- *Transparente*: O desenvolvimento e progresso dos testes é visível no quadro de tarefas do Ágil (ver capítulo 2.2.1).
- *Credibilidade*: O testador deve garantir a credibilidade da estratégia de testes, sua implementação e execução, caso contrário, as partes interessadas não vão confiar nos resultados do teste. Isso é muitas vezes feito através do fornecimento de informações às partes interessadas sobre o processo de teste.
- *Aberto ao feedback*: O feedback é um aspecto importante para ser bem-sucedido em qualquer projeto, especialmente em projetos ágeis. As retrospectivas permitem que as equipes aprendam com os sucessos e com os fracassos.
- *Resiliente*: Os testes devem ser capazes de responder à mudança, como todas as outras atividades nos projetos ágeis.

Estas melhores práticas maximizam a probabilidade de testes bem-sucedidos nos projetos *Scrum*.

Sprint Zero

O *Sprint Zero* é a primeira iteração do projeto, onde muitas atividades de preparação ocorrem (ver capítulo 1.2.5). O testador colabora com a equipe nas seguintes atividades durante esta iteração:

- Identifica o escopo do projeto (ou seja, o *backlog* do produto).
- Cria uma arquitetura inicial do sistema e protótipos de alto nível.
- Planeja, adquire e instala as ferramentas necessárias (p.e., para gerenciamento de testes, gerenciamento de defeitos, automação de testes e integração contínua).
- Cria uma estratégia de teste inicial para todos os níveis de teste, abordando (entre outros tópicos) o escopo de teste, os riscos técnicos, os tipos de teste (ver capítulo 3.1.3) e as metas de cobertura.
- Realiza uma análise de risco inicial de qualidade (ver capítulo 3.2.1).
- Define métricas de teste para medir o processo de teste, seu progresso no projeto, e a qualidade do produto.
- Especifica a definição de "realizado".
- Cria o quadro de tarefas (ver capítulo 2.2.1)
- Define quando continuar ou interromper o teste antes de entregar o sistema para o cliente.

O *Sprint Zero* define o rumo que o teste precisa alcançar e como o teste precisa alcançá-lo ao longo dos *sprints*.

Integração

Nos projetos ágeis, o objetivo é agregar valor para o cliente em uma base contínua (de preferência em cada *sprint*). Para permitir isso, a estratégia de integração deve considerar a concepção e teste. Para permitir uma estratégia de teste contínua para a funcionalidade e características apresentadas, é importante identificar todas as dependências entre as funções e características subjacentes.

Planejamento de teste

Visto que o teste está totalmente integrado na equipe ágil, o seu planejamento deve começar durante a sessão do planejamento de lançamento e ser atualizado durante cada *sprint*. O planejamento de teste para o lançamento e cada *sprint* deve abordar as questões discutidas no capítulo 1.2.5.

O planejamento do *sprint* resulta em um conjunto de tarefas a serem inseridas no quadro de tarefas, onde cada tarefa deve ter a duração de um ou dois dias de trabalho. Além disso, quaisquer questões relacionadas a testes devem ser monitoradas para manter um fluxo constante dos testes.

Práticas de teste ágil

Muitas práticas podem ser úteis para testadores em uma equipe *Scrum*, algumas das quais incluem:

- *Empilhamento*: Dois membros da equipe (p.e., um testador e um desenvolvedor, dois testadores, ou um testador e um gestor do produto) se reúnem em uma estação de trabalho para realizar um teste ou outra tarefa do *sprint*.
- *Projeto de teste incremental*: Os casos de teste e gráficos são gradualmente desenvolvidos das estórias de usuários e outras bases de testes, começando com testes simples e passando para testes mais complexos.
- *Mapeamento Mental*: O Mapa Mental é uma ferramenta útil no teste [Crispin08]. Por exemplo, os testadores podem usar o mapa mental para identificar quais sessões de teste realizar, para mostrar estratégias de teste, e para descrever os dados de teste.

Estas práticas são suplementares às outras práticas discutidas neste programa e no capítulo 4 do programa *Foundation Level* [ISTQB_FL_SYL].

3.2 Avaliação de riscos de qualidade e estimativa do esforço de teste

Um objetivo típico de testes em todos os projetos, ágeis ou tradicionais, é reduzir o risco de problemas de qualidade do produto a um nível aceitável antes do lançamento. Testadores em projetos ágeis podem usar os mesmos tipos de técnicas utilizadas em projetos tradicionais para identificar riscos de qualidade (ou riscos de produtos), para avaliar o nível de risco associado, para estimar o esforço necessário para reduzir suficientemente esses riscos, e depois mitigar esses riscos através de projeto de teste, para implementação e execução. No entanto, dadas as iterações curtas e a taxa de mudança em projetos ágeis, são necessárias algumas adaptações dessas técnicas.

3.2.1 Avaliar os riscos de qualidade em projetos ágeis

Um dos muitos desafios dos testes é a seleção, alocação e priorização adequadas das condições de teste. Isso inclui determinar o volume adequado de esforço para alocar a fim de cobrir cada condição com testes, e sequenciar os testes resultantes de modo a otimizar a eficácia e eficiência do trabalho a ser feito. A identificação dos riscos, a análise e as estratégias de mitigação de risco podem ser utilizadas pelos testadores nas equipes ágeis para ajudar a determinar um número aceitável de casos de teste a serem realizados, embora muitas restrições e variáveis que interagem possam exigir comprometimentos.

Nos projetos ágeis, a análise de risco da qualidade ocorre em dois lugares.

O risco é a possibilidade de um resultado ou evento negativo ou indesejável. O nível de risco é detectado através da avaliação da probabilidade de sua ocorrência e do impacto causado. Quando o efeito primário do problema potencial é a qualidade do produto, os potenciais problemas são chamados de riscos de qualidade e riscos de produto. Quando o efeito primário do problema potencial é sobre o sucesso do projeto, os problemas potenciais são referidos como riscos do projeto ou planejamento de riscos [Black07] [vanVeenendaal12].

- *Planejamento do lançamento*: representantes de negócio que conhecem as funcionalidades no lançamento fornecem uma visão geral de alto nível dos riscos, e toda a equipe, incluindo o testador(res), podem auxiliar na identificação e avaliação dos riscos.
- *Planejamento de iteração*: toda a equipe identifica e avalia os riscos de qualidade.

Exemplos de riscos de qualidade para um sistema incluem:

- Cálculos incorretos em relatórios (um risco funcional relacionado com acurácia).
- Resposta lenta a entrada do usuário (um risco não funcional relacionados com a eficiência e com o tempo de resposta).
- Dificuldade na compreensão de telas e campos (um risco não funcional relacionado com a usabilidade e inteligibilidade).

Como mencionado anteriormente, uma iteração começa com o seu planejamento, que culmina em tarefas estimadas em um quadro de tarefas. Estas tarefas podem ser priorizadas em parte com base no nível de risco de qualidade associados às mesmas. As tarefas associadas aos riscos mais elevados devem começar mais cedo envolvendo mais esforço de teste. As tarefas associadas a riscos menores devem começar mais tarde envolvendo menos esforço de teste.

Um exemplo de como o processo de análise de risco de qualidade em um projeto ágil pode ser realizado durante o planejamento da iteração é descrito nas seguintes etapas:

1. Reunir os membros da equipe ágil, incluindo os testadores.
2. Listar todos os itens do *backlog* da iteração atual (p.e., em um quadro de tarefas)
3. Identificar os riscos de qualidade associados a cada item, considerando-se todas as características relevantes de qualidade.
4. Avaliar cada risco identificado, que inclui duas atividades: categorizar o risco e determinar o seu nível com base no impacto e na probabilidade de defeitos.
5. Determinar a extensão do teste proporcional ao nível de risco.

6. Escolher a técnica de teste apropriada para mitigar cada risco, com base no risco, no seu nível, e na característica de qualidade pertinente.

O testador então projeta, implementa e executa os testes para mitigar os riscos. Isso inclui a totalidade de funcionalidades, comportamentos, características de qualidade e atributos que afetam o cliente, o usuário e a satisfação das partes interessadas.

Ao longo do projeto, a equipe deve manter-se consciente de informações adicionais que podem alterar o conjunto de riscos e/ou o nível de risco associado a riscos de qualidade conhecidos. Deve ocorrer periodicamente a análise do risco de qualidade resultando em adaptações dos testes. Os ajustes incluem a identificação de novos riscos, a reavaliação do nível dos riscos existentes e a avaliação da eficácia das atividades de mitigação dos riscos.

Os riscos de qualidade também podem ser atenuados antes do início da execução dos testes. Por exemplo, se os problemas com as estórias dos usuários são detectados durante a identificação do risco, a equipe do projeto pode rever completamente estas estórias como uma estratégia de mitigação.

3.2.2 Estimativa do esforço de teste com base no conteúdo e risco

Durante o planejamento de lançamento, a equipe ágil estima o esforço necessário para completar o lançamento. A estimativa aborda também o esforço de teste. A técnica comum para estimativa utilizada nos projetos ágeis é o *planning poker*, uma técnica baseada no consenso. O gestor do produto ou cliente lê uma estória de usuário para os avaliadores. Cada avaliador tem um baralho de cartas com valores semelhantes para a sequência de Fibonacci (ou seja, 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...) ou qualquer outra progressão de sua escolha (p.e., os tamanhos da camisa que variam de extra pequeno a extra-extra-grande). Os valores representam o número de pontos da estória, dias de esforço, ou outras unidades nas quais a equipe estima. A sequência de Fibonacci é recomendada, pois os números na sequência refletem que a incerteza cresce proporcionalmente com o tamanho da estória. A estimativa alta significa geralmente que a estória não é bem compreendida ou deve ser dividida em várias estórias menores.

Os estimadores discutem a funcionalidade e fazem perguntas para o gestor do produto conforme necessário. Aspectos como desenvolvimento e esforço de teste, complexidade da estória e escopo dos testes desempenham um papel fundamental na estimativa. Portanto, é aconselhável incluir o nível de risco de um item de *backlog*, além da prioridade especificada pelo gestor do produto, antes do início da sessão de *planning poker*. Quando a funcionalidade é amplamente discutida, cada membro seleciona privativamente uma carta para representar a sua estimativa. Todos os cartões são então revelados ao mesmo tempo. Se todos os membros selecionaram o mesmo valor, este se torna a estimativa. Caso contrário, os membros discutem as diferenças nas estimativas e ao término a rodada é repetida até que seja alcançado um acordo, seja por consenso ou pela aplicação de regras (p.e., usar a mediana, use a maior pontuação) para limitar o número de rodadas. Essas discussões garantem uma estimativa confiável do esforço necessário para completar itens de *backlog* do produto solicitados pelo proprietário do produto e ajudam a melhorar o conhecimento coletivo do que tem que ser feito [Cohn04].

3.3 Técnicas nos projetos ágeis

Muitas das técnicas de teste e níveis de testes que se aplicam a projetos tradicionais também podem ser aplicadas nos projetos ágeis. No entanto, nos projetos ágeis, existem algumas considerações específicas e variações de técnicas de teste, terminologias e documentação que devem ser consideradas.

3.3.1 Critérios de aceite e cobertura adequada, e outras informações para testes

Os projetos ágeis descrevem requisitos iniciais como estórias de usuário em um *backlog* priorizado no início do projeto. Os requisitos iniciais são curtos e geralmente seguem um formato pré-definido (ver capítulo 1.2.2). Os requisitos não funcionais, tais como usabilidade e desempenho também são importantes e podem ser especificados como estórias exclusivas de usuários, ou ligados a outras estórias de usuários funcionais. Os requisitos não funcionais podem seguir um formato pré-definido ou padrão, como [ISSO-25000], ou uma norma específica do setor.

As estórias de usuários servem como uma base importante de teste. Outras bases de teste possíveis incluem:

- Experiência em projetos anteriores.
- Funções, funcionalidades e características de qualidade dos sistemas existentes.
- Código, arquitetura e modelagem.
- Perfis de usuário (contexto, configurações de sistema e comportamento).
- Informações sobre defeitos de projetos existentes e anteriores.
- Categorização de defeitos em uma taxonomia de defeito.
- Normas aplicáveis (p.e., [DO-178B] para software de aviação).
- Riscos de qualidade (ver capítulo 3.2.1).

Durante cada iteração, os desenvolvedores criam código que implementa as funções e funcionalidades descritos nas estórias de usuários com as características de qualidade relevantes, e esse código é verificado e validado através de testes de aceite. Para ser testável, os critérios de aceite devem abordar os seguintes tópicos [Wiegers13]:

- *Comportamento funcional*: o comportamento observável externamente, com as ações do usuário como entrada operando sob certas configurações.
- *Características de qualidade*: como o sistema executa o comportamento especificado. As características também podem ser chamadas de atributos de qualidade ou requisitos não funcionais. As características mais comuns da qualidade são desempenho, confiabilidade, usabilidade, etc..
- *Cenários* (casos de uso): uma sequência de ações entre um ator externo (muitas vezes um usuário) e o sistema, a fim de realizar uma tarefa ou objetivo específico do negócio.
- *Regras de negócios*: atividades que só podem ser executadas no sistema sob certas condições definidas por procedimentos e restrições externas (p.e., os procedimentos utilizados por uma companhia de seguros para lidar com reclamações de seguros).

- *Interfaces externas*: descrições das conexões entre o sistema a ser desenvolvido e o mundo exterior. As interfaces externas podem ser divididas em diferentes tipos (interface do usuário, interface com outros sistemas, etc.).
- *Restrições*: qualquer projeto e restrição de implementação que restrinja as opções para o desenvolvedor. Os equipamentos com software embutido muitas vezes devem respeitar as restrições físicas, tais como tamanho, peso e conexões de interface.
- *Definições de dados*: o cliente pode descrever o formato, os tipos de dados, os valores permitidos e os valores padrão para um item de dados na composição de uma estrutura complexa de dados de negócios (p.e., o CEP em um endereço de correio dos EUA).

Além das estórias de usuários e seus critérios associados de aceite outras informações relevantes para o testador, incluem:

- Como se espera que o sistema atue e seja utilizado.
- As interfaces do sistema, que podem ser utilizadas/acessadas para testar o sistema.
- Se o suporte atual da ferramenta é suficiente.
- Se o testador tem conhecimento e habilidade suficiente para realizar os testes necessários.

Os testadores, muitas vezes, descobrem a necessidade de informações adicionais (p.e., a cobertura de código) ao longo das iterações e devem trabalhar em colaboração com o resto dos membros da equipe ágil para obter estas informações. As informações relevantes desempenham um papel fundamental para determinar se uma atividade específica pode ser considerada realizada. Este conceito da definição de finalizado é crucial nos projetos ágeis e aplica-se em uma série de maneiras diferentes, como discutido nas seguintes subseções.

Níveis de teste

Cada nível de teste tem sua própria definição de finalizado. A lista a seguir dá exemplos que podem ser relevantes para os diferentes níveis de teste.

Teste da unidade:

- 100% de cobertura de decisão, sempre que possível com revisões cuidadosas de todos os caminhos inviáveis.
- Análise estática realizada em todo o código.
- Os defeitos importantes não resolvidos (classificados com base na prioridade e gravidade).
- Não é conhecida nenhuma pendência técnica inaceitável restante no projeto e no código [Jones11].
- Todos os códigos, testes de unidade, e resultados de unidade de teste avaliados.
- Todos os testes de unidade automatizados.
- As características importantes estão dentro dos limites acordados (p.e., desempenho).

Teste de integração

- Todos os requisitos funcionais foram testados, incluindo os testes positivos e negativos, com o número de testes baseados no tamanho, complexidade e riscos.
- Todas as interfaces entre as unidades testadas.
- Todos os riscos de qualidade cobertos de acordo com a medida acordada de testes.
- Inexistência de defeitos importantes não solucionados (priorizados de acordo com o risco e importância).

- Todos os defeitos encontrados foram relatados.
- Todos os testes de regressão automatizados, sempre que possível, com todos os testes automatizados armazenados em um repositório comum.

Teste de sistema

- Testes completos de estórias de usuários, funcionalidades e funções.
- Todos os usuários cobertos.
- As características mais importantes de qualidade do sistema cobertas (p.e., desempenho, robustez, fiabilidade).
- Testes realizados em um ambiente de produção, incluindo todo o hardware e software para todas as configurações de suporte, na medida do possível.
- Todos os riscos de qualidade cobertos de acordo com a medida acordada de testes.
- Todos os testes de regressão automatizados, sempre que possível, com todos os testes automatizados armazenados em um repositório comum.
- Todos os defeitos encontrados relatados e possivelmente resolvidos.
- Inexistência de defeitos importantes não solucionados (priorizados de acordo com o risco e importância).

Estória do Usuário

A definição de finalizado para estórias de usuários pode ser determinada pelos seguintes critérios:

- As estórias de usuários selecionadas para a iteração estão completas, compreendidas pela equipe, e com critérios de aceite detalhados e testáveis.
- Todos os elementos da estória do usuário foram especificadas e avaliadas, incluindo a conclusão dos testes de aceite da estória do usuário.
- As tarefas necessárias para implementar e testar as estórias de usuários selecionadas foram identificadas e estimadas pela equipe.

Funcionalidade

A definição de pronto para funcionalidades, que pode se estender por várias estórias de usuários ou épicos, pode incluir:

- Todas as estórias de usuários e seus critérios de aceite que constituem a funcionalidade foram definidas e aprovadas pelo cliente.
- O projeto está completo, sem nenhuma pendência técnica conhecida.
- O código está completo, sem pendências técnicas conhecidas ou fatoração inacabada.
- Os testes de unidade foram realizados e alcançaram o nível definido de cobertura.
- Os testes de integração e do sistema para a funcionalidade foram realizados de acordo com os critérios definidos de cobertura.
- Nenhum grande defeito deverá ser corrigido.
- A documentação das funcionalidades está completa, o que pode incluir notas de lançamento, manuais do usuário e funções de ajuda online.

Iteração

A definição de finalização para a iteração pode incluir o seguinte:

- Todas as funcionalidades para a iteração estão prontas e testadas individualmente de acordo com os critérios de nível de funcionalidade.
- Todos os defeitos não críticos que não podem ser corrigidos dentro dos limites da iteração foram adicionados e priorizados ao *backlog* do produto.
- A integração de todas as funcionalidades para a iteração foi concluída e testada.
- A documentação foi escrita, avaliada e aprovada

Neste ponto, o software é potencialmente desmembrável, pois a iteração foi concluída com êxito, mas nem todas as iterações resultam em um lançamento.

Lançamento

A definição de "finalizado" para um lançamento, pode se estender por várias iterações, e podem incluir as seguintes áreas:

- *Cobertura*: todos os elementos da base de teste relevantes para o conteúdo do lançamento foram cobertos por testes. A adequação da cobertura é determinada pelo que é novo ou alterado, sua complexidade e tamanho, e os riscos de falhas associadas.
- *Qualidade*: A intensidade de defeito (p.e., quantos defeitos são encontrados por dia ou por transação), a densidade de defeitos (p.e., o número de defeitos encontrados em comparação com o número de estórias de usuários, esforço, ou atributos de qualidade), o número estimado de permanência dos defeitos dentro dos limites aceitáveis, as consequências de defeitos não resolvidos e restantes (p.e., a gravidade e de prioridade) foram compreendidas e aceitáveis, o nível residual de risco associado a cada um dos riscos identificados de qualidade foram compreendidos e são aceitáveis.
- *Tempo*: Se a data de entrega pré-determinada foi cumprida, as considerações comerciais associadas ao lançamento ou não, devem ser consideradas.
- *Custo*: O custo estimado do ciclo de vida deve ser usado para calcular o retorno sobre o investimento para o sistema de entrega (ou seja, o custo de desenvolvimento e manutenção calculado deve ser consideravelmente menor do que as vendas totais esperadas do produto). A maior parte do custo do ciclo de vida vem muitas vezes da manutenção após o lançamento do produto, devido ao número de defeitos que escapam à produção.

3.3.2 Desenvolvimento orientado para o teste de aceite

O desenvolvimento orientado para o teste de aceite é uma abordagem de testar primeiro. Os casos de teste são criados antes de implementar a estória do usuário. Os casos são criados pela equipe ágil, incluindo o desenvolvedor, o testador, e os representantes de negócio [Adzic09] e podendo serem manuais ou automatizados. A primeira etapa é uma oficina de especificação, onde a estória do usuário é analisada, discutida e escrita por desenvolvedores, testadores e representantes do negócio. Quaisquer insuficiências, ambiguidades ou erros na estória do usuário são corrigidos durante este processo.

A próxima etapa é criar os testes. Isto pode ser feito pela equipe, em conjunto ou individualmente pelo testador. Em qualquer caso, uma pessoa independente, tal como um representante comercial, valida os testes. Os testes são exemplos que descrevem as características específicas da estória do usuário. Estes exemplos ajudarão a equipe na correta implementação da estória. Uma vez que os

exemplos e testes são os mesmos, estes termos são usados alternadamente. O trabalho começa com exemplos básicos e questões abertas.

Normalmente, os primeiros testes são os testes positivos, confirmando o comportamento correto, sem condições de exceção ou erro, que incluem a sequência de atividades executadas, se tudo correr conforme esperado. Após a realização dos testes positivos, a equipe deve escrever os testes negativos abrangendo os atributos não funcionais (p.e., desempenho, usabilidade). Os testes são expressos de tal forma que as partes interessadas sejam capazes de compreender, contendo frases em linguagem natural que envolvem as pré-condições necessárias, se for o caso, incluindo as entradas e as saídas relacionadas.

Os exemplos devem cobrir todas as características da estória do usuário e não devem ser adicionados à estória. Isto significa que não deve existir um exemplo que descreva um aspecto da estória do usuário que não foi documentado na própria estória. Além disso, dois exemplos não devem descrever as mesmas características da estória do usuário.

3.3.3 Projeto de teste funcional e não funcional de caixa-preta

Nos testes ágeis, muitos testes são criados por testadores simultaneamente com as atividades de programação dos desenvolvedores. Assim como os desenvolvedores realizam a programação com base nas estórias de usuários e critérios de aceite, os testadores criam testes baseados nas estórias de usuários e seus critérios de aceite (alguns testes, tais como os testes exploratórios e alguns outros testes baseados na experiência, são criados mais tarde, durante a realização do teste, conforme explicado no capítulo 3.3.4.). Os testadores podem aplicar técnicas tradicionais de projeto de teste caixa-preta, tais como partição de equivalência, análise do valor limite, tabelas de decisão, e teste de transição de estado. Por exemplo, a análise do valor limite pode ser utilizada para selecionar os valores de teste quando um cliente é limitado no número de itens que escolherem para compra.

Em muitas situações, os requisitos não funcionais podem ser documentados como estórias de usuário. As técnicas de projeto de teste caixa-preta (como análise de valor limite) também podem ser utilizadas para criar testes para características de qualidade não funcionais. A estória de usuário pode conter requisitos de desempenho ou confiabilidade. Por exemplo, uma determinada execução não pode exceder um determinado limite de tempo, ou um número de operações não pode ser inferior a um determinado número de vezes.

Para obter mais informações sobre o uso das técnicas de projeto do teste caixa-preta, consulte os syllabi *Foundation Level* [ISTQB_FL_SYL] e *Advanced Test Analyst Level* [ISTQB_ALTA_SYL].

3.3.4 Teste exploratório e teste ágil

O teste exploratório é importante em projetos ágeis, devido ao pouco tempo disponível para análise do teste e as informações limitadas sobre as estórias de usuários. A fim de alcançar os melhores resultados, o teste exploratório deve ser combinado com outras técnicas baseadas na experiência como parte de uma estratégia de teste reativo, misturado com outras estratégias de testes, tais como testes de análise baseada no risco, baseados em requisitos analíticos, baseados em modelo, e testes de regressão. As estratégias de teste e a mistura da estratégia de teste são discutidas no syllabus *Foundation Level* [ISTQB_FL_SYL].

Em testes exploratórios, o projeto de teste e a realização do teste ocorre ao mesmo tempo, guiado por um gráfico de teste. Um gráfico de teste fornece as condições para cobrir durante uma sessão de testes *time-boxed*. Durante o teste exploratório, os resultados dos testes mais recentes guiam o próximo teste. As mesmas técnicas de teste caixa-branca e caixa-preta podem ser utilizadas para projetar os testes, na realização de testes pré-concebidos.

Um gráfico de teste pode incluir as seguintes informações:

- *Ator*: usuário a que se destina o sistema.
- *Objetivo*: o tema do gráfico, incluindo o objetivo que o ator pretende atingir, ou seja, as condições de teste.
- *Configuração*: o que precisa ser posto em prática a fim de iniciar a realização do teste.
- *Prioridade*: a importância relativa deste gráfico, com base na prioridade da estória do usuário associado ao nível de risco.
- *Referências*: especificações (p.e., estória de usuário), riscos ou outras fontes de informação.
- *Dados*: todos os dados que são necessários para elaborar o gráfico.
- *Atividades*: uma lista de ideias do que o ator pode querer fazer com o sistema (p.e., "fazer logon no sistema como um super usuário") e o que seria interessante testar (ambos os testes positivos e negativos).
- *Notas de Oráculo*: como avaliar o produto para determinar os resultados corretos (p.e., captar o que acontece na tela e comparar com o que está escrito no manual do usuário).
- *Variações*: ações alternativas e avaliações para complementar as ideias descritas nas atividades.

Para gerenciar o teste exploratório, um método chamado gestão de testes baseado em sessão pode ser usado. Uma sessão é definida como um período ininterrupto de teste que pode durar de 60 a 120 minutos. As sessões de testes incluem:

- Sessão de pesquisa (aprender seu funcionamento).
- Sessão de análise (avaliar funcionalidades ou características).
- Profundidade da cobertura (casos laterais, cenários, as interações) A qualidade dos testes depende da capacidade dos testadores em fazer perguntas pertinentes sobre o que testar. Exemplos incluem:
  - O que é mais importante para saber mais sobre o sistema?
  - De que forma o sistema pode falhar?
  - O que acontece se?
  - O que deve acontecer quando?
  - As necessidades dos clientes, requisitos e expectativas são cumpridas?
  - É possível instalar o sistema (e remover se necessário) em todos os caminhos de atualização com suporte?

Durante a execução do teste, o testador usa a criatividade, a intuição, cognição e habilidade para encontrar possíveis problemas com o produto. O testador também precisa ter bom conhecimento e compreensão do software em teste, o domínio do negócio, como o software é usado, e como determinar quando o sistema falha.

Um conjunto heurístico pode ser aplicado durante o teste. A heurística pode orientar o testador em como realizar os testes e avaliar os resultados [Hendrickson]. Os exemplos incluem:

- Limites.
- CRUD (Criar, Ler, Atualizar, Deletar).
- Variações de configuração.
- Interrupções (p.e., fazer *logoff*, desligar ou reiniciar).

É importante para o testador documentar o processo tanto quanto possível, caso contrário, seria difícil voltar atrás e ver como um problema no sistema foi detectado. A lista a seguir fornece exemplos de informações que podem ser úteis ao documento:

- *Cobertura de teste*: quais dados de entrada foram utilizados, quanto foi coberto, e quanto ainda resta ser testado.
- *Notas de avaliação*: observações durante os testes, estabilidade na execução do sistema e funcionalidades em teste, detecção de defeitos, o que está previsto como a próxima etapa de acordo com as observações atuais, e qualquer outra lista de ideias.
- *Lista de risco/estratégia*: quais riscos foram cobertos e quais permanecem entre os mais importantes, qual será a estratégia inicial ser seguida pois não precisa de quaisquer mudanças.
- *Questões, perguntas e anomalias*: qualquer comportamento inesperado, alguma dúvida sobre a eficácia da abordagem, qualquer preocupação com as ideias/tentativas de teste, ambiente de teste, dados de teste, a incompreensão da função, script de teste ou o sistema em teste.
- *Comportamento atual*: registro do comportamento real do sistema que precisa ser salvo (p.e., vídeo, capturas de tela, arquivos de dados de saída).

As informações registradas devem ser capturadas e/ou resumidas em alguma ferramenta de status de gestão (p.e., ferramentas de gerenciamento de teste, ferramentas de gerenciamento de tarefas, o quadro de tarefas), de uma forma que facilite para as partes interessadas a compreensão do status atual de da realização de todos os testes.

3.4 Ferramentas em projetos ágeis

As ferramentas descritas no syllabus *Foundation Level* [ISTQB_FL_SYL] são relevantes e utilizadas pelos testadores nas equipes ágeis. Nem todas as ferramentas são usadas da mesma maneira e algumas são mais relevantes para projetos ágeis do que para projetos tradicionais. Por exemplo, embora as ferramentas de gerenciamento de teste, ferramentas de gerenciamento de requisitos e ferramentas de gerenciamento de incidentes (ferramentas de rastreio de defeito) possam ser utilizadas por equipes ágeis, algumas optam por uma ferramenta inclusiva (p.e., gestão do ciclo de vida do aplicativo ou gestão de tarefas) que fornece funcionalidades relevantes para o desenvolvimento ágil, tais como quadros de tarefas, gráficos *burndown* e estórias de usuários. As ferramentas de gerenciamento de configuração são importantes para testadores em equipes ágeis, devido ao elevado número de testes automatizados em todos os níveis e à necessidade de armazenar e gerenciar os artefatos associados aos testes automatizados.

Além das ferramentas descritas no syllabus *Foundation Level* [ISTQB_FL_SYL], os testadores nos projetos ágeis também podem utilizar as ferramentas descritas nas subseções a seguir. Estas ferramentas são usadas por toda a equipe para garantir a colaboração e compartilhamento de informações, que são fundamentais para as práticas do Ágil.

3.4.1 Ferramentas de gestão e rastreamento de tarefas

Em alguns casos, as equipes ágeis usam quadros de estória/tarefas físicas (p.e., quadro branco, quadro de cortiça) para gerenciar e rastrear estórias de usuários, testes e outras tarefas ao longo de cada *sprint*. Outras equipes vão usar software de gerenciamento de ciclo de vida e gerenciamento de tarefas, incluindo quadros eletrônicos de tarefa. Essas ferramentas servem para os seguintes fins:

- Registrar as estórias e suas tarefas relevantes de desenvolvimento e teste para garantir que nada seja perdido durante um *sprint*.
- Capturar as estimativas das tarefas dos membros da equipe e calcular automaticamente o esforço necessário para implementar uma estória, para apoiar com mais eficiência as sessões de planejamento de iteração.
- Associar as tarefas de desenvolvimento com as tarefas de teste de mesma estória, para fornecer um quadro completo do esforço necessário da equipe para implementar a estória.
- Agregar as atualizações de status da tarefa de desenvolvedores e testadores ao concluírem os seus trabalhos, fornecendo um resumo calculado do status atualizado de cada estória, iteração e lançamento global.
- Apresentar visualmente (através de métricas, gráficos e *dashboards*) o estado atual de cada estória de usuário, iteração e lançamento, permitindo que todas as partes interessadas, incluindo as pessoas em equipes geograficamente distribuídas, possam verificar rapidamente o status.
- Integração com ferramentas de gerenciamento de configuração, o que pode permitir o registro automático de verificações do código e realizações das tarefas, e, em alguns casos, atualizações de status automatizado para tarefas.

3.4.2 Ferramentas de comunicação e compartilhamento de informações

Além de e-mails, documentos e comunicação verbal, as equipes ágeis muitas vezes usam três tipos adicionais de ferramentas para apoiar a comunicação e o compartilhamento de informações: *wikis*, mensagens instantâneas e compartilhamento de *desktop*.

Os *Wikis* permitem que as equipes desenvolvam e compartilhem uma base de conhecimento online sobre vários aspectos do projeto, incluindo:

- Diagramas de funcionalidades do produto, discussões sobre as funcionalidades, diagramas de protótipos, fotos de discussões do quadro branco e outras informações.
- Ferramentas e/ou técnicas úteis para o desenvolvimento e teste por outros membros da equipe.
- Métrica, gráficos e *dashboards* sobre o status do produto, o que é especialmente útil quando o *wiki* está integrado com outras ferramentas, tais como o servidor de desenvolvimento e o gerenciamento de tarefas, uma vez que a ferramenta pode atualizar o status do produto automaticamente.
- As conversas entre os membros da equipe, semelhantes a mensagens instantâneas e e-mail, mas de uma forma que é compartilhada com todos os outros membros da equipe.

Mensagens instantâneas, teleconferência de áudio e vídeo, e ferramentas de chat oferecem os seguintes benefícios:

- Permite a comunicação direta em tempo real entre os membros da equipe, em especial em equipes distribuídas.
- Envolve equipes distribuídas em reuniões diárias.
- Reduz contas de telefone através do uso de tecnologia de voz sobre IP, eliminando restrições de custo que podem reduzir a comunicação do membro da equipe em ambientes distribuídos.

Compartilhamento de desktop e ferramentas de captura fornecem os seguintes benefícios:

- Em equipes distribuídas, demonstrações de produtos, revisões de código, e até mesmo emparelhamento podem ocorrem.
- Captura de demonstrações de produtos, no final de cada iteração, que podem ser enviadas para o *wiki* da equipe.

Essas ferramentas devem ser utilizadas para complementar e ampliar, e não substituir, a comunicação face-a-face em equipes ágeis.

3.4.3 Desenvolvimento do software e ferramentas de distribuição

Como discutido anteriormente neste syllabus, a compilação diária e implantação de software é uma prática fundamental nas equipes ágeis. Isso requer o uso de ferramentas de integração contínua, compilação e de distribuição. Os usos, benefícios e os riscos dessas ferramentas foram descritos anteriormente no capítulo 1.2.4.

3.4.4 Ferramentas de gerenciamento de configuração

Nas equipes ágeis, as ferramentas de gerenciamento de configuração podem ser utilizadas não só para armazenar o código fonte e testes automatizados, como também os testes manuais e outros produtos de trabalho de teste, que são frequentemente armazenados no mesmo repositório do código-fonte do produto. Isto permite a rastreabilidade entre os quais versões do software foram testadas com quais versões específicas de testes, e permite a mudança rápida sem perda de informações históricas. Os principais tipos de sistemas de controle de versão incluem sistemas centralizados de controle de fonte e sistemas de controle de versão distribuída. O tamanho da equipe, estrutura, localização e necessidades de integração com outras ferramentas irá determinar qual sistema de controle de versão é ideal para um projeto específico do Ágil.

3.4.5 Projeto de teste, ferramentas de implementação e execução

Algumas ferramentas são úteis para os testadores ágeis em pontos específicos do processo de teste de software. Embora a maioria destas ferramentas não sejam novas ou específicas para o Ágil, elas fornecem funcionalidades importantes devido à rápida mudança dos projetos.

- *Ferramentas de projeto de teste*: O uso de ferramentas, tais como mapas mentais, se tornaram mais populares para projetar e definir rapidamente os testes para uma nova funcionalidade.

- *Ferramentas de gestão de casos de teste*: O tipo de ferramentas de gestão de caso de teste utilizado no Ágil pode ser parte da ferramenta de gestão do ciclo de vida do aplicativo ou gestão de tarefa de toda a equipe.
- *Ferramentas para preparação e geração de dados de teste*: Ferramentas que geram dados para preencher um banco de dados de um aplicativo são muito benéficas quando um grande volume de dados e combinações de dados é necessário para testar o aplicativo. Essas ferramentas também podem ajudar a redefinir a estrutura de banco de dados conforme o produto sofre alterações durante um projeto do Ágil e refatora os scripts para gerar os dados. Isso permite a atualização rápida de dados de teste conforme as mudanças ocorrem. Algumas ferramentas de preparação de dados de teste utilizam fontes de dados de produção como matéria-prima e utilizam scripts para remover os dados sigilosos ou torná-los anônimos. Outras ferramentas de preparação de dados de teste podem ajudar a validar grandes entradas ou saídas de dados.
- *Ferramentas de carga de dados de teste*: Depois que os dados foram gerados para teste, eles precisam ser carregados no aplicativo. A entrada manual de dados é muitas vezes demorada e sujeita a erros, mas as ferramentas de carga de dados estão disponíveis para tornar o processo confiável e eficiente. Na verdade, muitas das ferramentas geradoras de dados incluem um componente de carga de dados integrados. Em outros casos também é possível grandes volumes de carga, utilizando os sistemas de gestão de base de dados.
- *Ferramentas de execução de testes automatizados*: Existem ferramentas de execução de teste que são mais alinhadas com o teste do Ágil. Ferramentas específicas estão disponíveis em ambas as vias comerciais e *opensource* para apoiar as abordagens do primeiro teste, tais como desenvolvimento orientado para comportamento, desenvolvimento orientado para testes e desenvolvimento orientado para teste de aceite. Essas ferramentas permitem que os testadores e pessoal de negócios expressem o comportamento do sistema esperado em tabelas ou linguagem natural usando palavras-chave.
- *Ferramentas de teste exploratório*: Ferramentas que captam e registram atividades realizadas em um aplicativo durante uma sessão de teste exploratório são benéficas para o testador e desenvolvedor, pois registram as ações tomadas. Isso é útil quando um defeito é encontrado, pois as ações tomadas antes da falha ocorrida foram captadas e podem ser utilizadas para relatar o defeito para os desenvolvedores. O registro das etapas realizadas em uma sessão de teste exploratório pode vir a ser benéfico se o teste for, em última análise, incluído no conjunto de testes automatizados de regressão.

3.4.6 Ferramentas de Computação Nuvem e Virtualização

A virtualização permite que um único recurso físico (servidor) opere como muitos recursos separados, menores. Quando as máquinas virtuais ou instâncias de nuvem são utilizadas, as equipes têm um maior número de servidores disponíveis para desenvolvimento e testes. Isso pode ajudar a evitar atrasos associados à espera de servidores físicos. O provisionamento de um novo servidor ou restauração de um servidor é mais eficiente com recursos de *snapshot* incorporado na maioria das ferramentas de virtualização. Algumas ferramentas de gerenciamento de teste atualmente utilizam tecnologias de virtualização para fotografar servidores quando uma falha for detectada, permitindo que os testadores compartilhem a foto com os desenvolvedores que estão investigando a falha.

Referências

Normas

- [DO-178B] RTCA/FAA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992
- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE), 2005

Documentos ISTQB

- [ISTQB_ALTA_SYL] ISTQB Advanced Level Test Analyst Syllabus, Version 2012
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 2012
- [ISTQB_FA_OVIEW] ISTQB Foundation Level Agile Tester Overview, Version 1.0
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 2011

Livros

- [Aalst13] Leo van der Aalst and Cecile Davis, "TMap NEXT® in Scrum," ICT-Books.com, 2013.
- [Adzic09] Gojko Adzic, "Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing," Neuri Limited, 2009.
- [Anderson13] David Anderson, "Kanban: Successful Evolutionary Change for Your Technology Business," Blue Hole Press, 2010.
- [Beck02] Kent Beck, "Test-driven Development: By Example," Addison-Wesley Professional, 2002.
- [Beck04] Kent Beck and Cynthia Andres, "Extreme Programming Explained: Embrace Change, 2e" Addison-Wesley Professional, 2004.
- [Black07] Rex Black, "Pragmatic Software Testing," John Wiley and Sons, 2007.
- [Black09] Rex Black, "Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 3e," Wiley, 2009.
- [Chelimsky10] David Chelimsky et al, "The RSpec Book: Behavior Driven Development with Rspec, Cucumber, and Friends," Pragmatic Bookshelf, 2010.
- [Cohn04] Mike Cohn, "User Stories Applied: For Agile Software Development," Addison-Wesley Professional, 2004.
- [Crispin08] Lisa Crispin and Janet Gregory, "Agile Testing: A Practical Guide for Testers and Agile Teams," Addison-Wesley Professional, 2008.
- [Goucher09] Adam Goucher and Tim Reilly, editors, "Beautiful Testing: Leading Professionals Reveal How They Improve Software," O'Reilly Media, 2009.
- [Jeffries00] Ron Jeffries, Ann Anderson, and Chet Hendrickson, "Extreme Programming Installed," Addison-Wesley Professional, 2000.
- [Jonesll] Capers Jones and Olivier Bonsignour, "The Economics of Software Quality," Addison- Wesley Professional, 2011.
- [Linz14] Tilo Linz, "Testing in Scrum: A Guide for Software Quality Assurance in the Agile World," Rocky Nook, 2014.

- [Schwaber01] Ken Schwaber and Mike Beedle, "Agile Software Development with Scrum," Prentice Hall, 2001.
- [vanVeenendaal12] Erik van Veenendaal, "The PRISMA approach", Uitgeverij Tutein Nolthenius, 2012.
- [Wiegers13] Karl Wiegers and Joy Beatty, "Software Requirements, 3e," Microsoft Press, 2013.

Terminologia do Ágil

Palavras-chave que são encontradas no Glossário ISTQB ão identificadas no início de cada capítulo. Para termos comuns do Ágil, contamos com os seguintes recursos da Internet que fornecem definições.

http://guide.agilealliance.org/

http://whatis.techtargetcom/glossary

http://www.scrumalliance.org/

Convidamos os leitores a verificar esses sites caso eles não estejam familiarizados com os termos relacionados ao Ágil neste documento. Estes links estavam ativos no momento de lançamento deste documento.

Outras Referências

As referências a seguir apontam para informações disponíveis na internet e em outros locais. Mesmo que essas referências tenham sido verificadas no momento da publicação deste programa, o ISTQB não se responsabiliza se as referências não estiverem mais disponíveis.

- [Guia Agile Alliance] Vários contribuintes, HYPERLINK "http://guide.Agilealliance.org/".
- [Agilemanifesto] Vários contribuintes, HYPERLINK "http://www.agilemanifesto.org".
- [Hendrickson]: Elisabeth Hendrickson, "Desenvolvimento orientado para teste de aceite", testobsessed.com/2008/12/acceptance-test-driven-development-atdd-an-overview.
- [INVEST] Bill Wake, "Investir em boas estórias e tarefas inteligentes" xp123.com/articles/invest- in-good-stories-and-smart-tasks.
- [Kubaczkowski] Greg Kubaczkowski e Rex Black, "Missão a possibilidade de," www.rbcs-us.com/images/documents/Mission Made-Possible.pdf