<file:///C:/Users/raphael.silva/Downloads/testes-ageis.pdf>

<http://pt.slideshare.net/Qualister/teste-software-metodos-metodologia-ageis-agil-agile-testing?related=2>

<http://pt.slideshare.net/cristianocaetano/mini-curso-testes-ageis?qid=52a04028-0741-4624-a15e-24e7321ba4e6&v=default&b=&from_search=1>

<http://blog.adaptworks.com.br/2013/11/introducao-do-quadrante-de-teste-agil/>

**Desenvolvimento ágil de software**

O desenvolvimento ágil é um fruto da constatação feita, de forma independente, por diversos profissionais renomados na área de engenharia de software, de que, apesar de terem aprendido segundo a cartilha tradicional, só conseguiam minimizar os riscos associados ao desenvolvimento de software, pensando e agindo de forma muito diferente do que tradicionalmente está nos livros. Estes profissionais, a maioria veteranos consultores para projetos de softwares, decidiram reunir-se no início de 2001 durante um workshop realizado em Snowbird, Utah, EUA.

Embora cada envolvido tivesse suas próprias práticas e teorias preferidas, todos concordavam que, em suas experiências prévias, os projetos de sucesso tinham em comum um pequeno conjunto de princípios. Com base nisso eles criaram o Manifesto para o Desenvolvimento Ágil de Software, freqüentemente chamado apenas de Manifesto Ágil. Os valores descritos no Manifesto Ágil priorizavam:

* Indivíduos e interação entre eles ao invés de processos e ferramentas;
* Software em funcionamento ao invés de documentação abrangente;
* Colaboração com o cliente ao invés de negociação de contratos;
* Responder a mudanças ao invés de seguir estritamente um plano.

Adicionalmente, o Manifesto ágil, preconizava os seguintes princípios fundamentais:

* Nossa maior prioridade é satisfazer o cliente, através da entrega contínua de software de valor;
* Aceitar mudanças de requisitos (processos ágeis se adaptam a mudanças, para que o cliente possa tirar vantagens competitivas);
* Entregar software funcionando com freqüência, na escala de semanas até meses, com preferência aos períodos mais curtos;
* Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto;
* Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho;
* O método mais eficiente e eficaz de transmitir informações para um time de desenvolvimento é através de uma conversa cara a cara;
* Software funcional é a medida primária de progresso;
* Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e clientes, devem ser capazes de manter indefinidamente, passos constantes;
* Contínua atenção à excelência técnica e bom design, aumenta a agilidade;
* Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito;
* As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis, motivados e confiantes;
* Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

O termo “Método Ágil” identifica metodologias de desenvolvimento que adotam os valores e princípios do manifesto ágil. Dentre os inúmeros métodos ágeis existentes, podemos destacar: Extreme Programming (XP), SCRUM, Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), Crystal Clear, entre outros.

Desenvolvimento guiado pelos testes

Independente da metodologia de desenvolvimento, um dos problemas mais recorrentes no desenvolvimento de software é a alta incidência de defeitos. Software defeituosos causam:

* Lentidão no desenvolvimento: Quando é relatado uma falha/problema, o desenvolvedor gasta muito tempo depurando (debugging) o código para descobrir onde está o defeito e depois para corrigí-lo. Além disso, nem sempre o desenvolvedor que escreveu a funcionalidade é responsável pela correção dos seus defeitos. E para piorar, muitas vezes as empresas atribuem desenvolvedores inexperientes para a corrigir os defeitos;
* Efeitos colaterais: Frequentemente, a correção de um defeito introduz como efeito colateral outros defeitos. Isso acontece em função de vários fatores, como por exemplo: falta de experiência do desenvolvedor, falta de arquitetura no software, falta de organização nos processos de trabalho da empresa, prazos muito apertados, falta de atenção do desenvolvedor em função de exaustão por ter que trabalhar muitas noites e finais de semana seguidos, entre outros;
* Custos altos: Defeitos encontrados em produção tem um custo de correção cem vezes (às vezes mil vezes) mais alto do que os defeitos encontrados durante o ciclo de desenvolvimento. Isso sem contar os custos associados a insatisfação dos clientes.

Os métodos ágeis, por sua vez, mitigam os riscos e desafios associados a uma alta incidência de defeitos por meio de testes. Teste de software é mais do que uma simples tarefa, na verdade, é o pilar de sustentação que permite a implementação de muitos princípios de desenvolvimento ágil, como por exemplo:

* Entrega contínua de software de valor: Testes são a única forma de demonstrar se o software atende as necessidades do cliente (software de valor);
* Aceitar mudanças: Testes dão confiança ao time para realizar mudanças sem medo de causar efeitos colaterais e instabilidade no sofware;
* Envolvimento de pessoas relacionadas à negócios e desenvolvimento: Testes são descritos em uma linguagem comum a todos os membros do time. Dessa forma, todo o time compartilha o mesmo entendimento do que deve ser feito, as restrições e as definições de "Pronto";
* Software funcional é a medida de progresso: Testes executados frequentemente demonstram o progresso no desenvolvimento de novas funcionalidades, assim como, se o software ainda funciona (não foram introduzidos defeitos);
* Excelência técnica, bom design e simplicidade: Testes escritos antes do código induzem o desenvolvedor a pensar com mais profundidade na implementação da funcionalidade, tornando dessa forma, o código mais simples e com melhor design;
* Times motivados e confiantes: Testes executados com sucesso aumentam a motivação (estamos realizando nosso trabalho corretamente) e confiança do time (estamos tomando as decisões certas);
* O time reflete em como ficar mais efetivo: Testes demonstram falhas tanto no código quando no processo de trabalho (e atitudes do time), com base nas lições aprendidas durante a correção da falha, o time ajusta e otimiza seu comportamento de acordo.

Na perspectiva ágil, o teste é uma tarefa paralela e intrínseca a todas as etapas do desenvolvimento, inclusive, é uma tarefa intimamente ligada ao ato de programar. Na metodologia ágil conhecida como Extreme Programming (XP), por exemplo, o teste guia o desenvolvimento. Por meio da prática chamada TDD (Test-Driven Development), o desenvolvedor escreve o código de teste antes de escrever o código que implementa a funcionalidade. Nesta filosofia, o desenvolvedor é induzido a primeiro pensar nas regras e restrições da funcionalidade para então mapear essas regras e restrições em testes (unitários). Depois, o desenvolvedor escreve o código da funcionalidade, de forma a atender o comportamento esperado que é determinado pelos testes. Segundo Kent Beck, criador e disseminador do Extreme Programming (XP), quando o programador atinge o nível de adorar escrever testes, significa que o programador tornou-se Test Infected.

No entanto, diferente do que muitas pessoas acreditam, as técnicas de testes preconizados pelas metodologias ágeis, não são limitadas a aferir apenas o código. Sem sombra de dúvida, técnicas como o TDD (Test-Driven Development), ajudam o desenvolvedor a eliminar defeitos no código, no entanto, o código não é a única fonte de defeitos. Em função disso, é importante destacar que os métodos ágeis oferecem um conjunto bastante sofisticado de técnicas de testes para prevenir e detectar defeitos no código, requisitos, arquitetura, assim como, defeitos originados por falta de entendimento, assunções erradas, etc.

**Testes ágeis**

O teste ágil é caracterizado por ser uma atividade: desempenhada por todos os membros do time; que ocorre em todas etapas do ciclo de vida de desenvolvimento; através de mecanismos automatizados (quando possível); e que o ocorre frequentemente em ciclos contínuos. Nas metodologias ágeis, todos são responsáveis pelos testes. A qualidade do software é responsabilidade de todos os membros do time. Cada membro do time contribui para a qualidade do software realizando testes sob a sua perspectiva. Dessa forma, os testes ocorrem de maneira colaborativa e complementar, da seguinte maneira:

* Os clientes testam sob a perspectiva da funcionalidade (estória por estória).
* Os desenvolvedores testam sob a perspectiva do código (método por método);

**Testes sob a perspectiva do cliente**

Grande parte das metodologias ágeis expressa os requisitos dos clientes em termos de estórias informais e leves. As funcionalidades são representadas através dessas estórias que refletem as necessidades do cliente e são suficientemente pequenas para permitir que os desenvolvedores consigam implementar um conjunto delas em cada iteração (ou Sprint). Uma estória deve ser compreensível por todos os membros do time, testável e valiosa para o cliente. Testável, neste contexto, indica que toda estória deve incluir os critérios de aceitação para assegurar que a funcionalidade foi implementada corretamente e atende as necessidades do cliente.

Para atingir este objetivo, o cliente é responsável por escrever testes de aceitação (Acceptance Testing) para cada estória. Os testes de aceitação são elencados pelos clientes em função de que eles conhecem profundamente o negócio e, portanto, também conhecem os objetivos que devem ser alcançados. Os testes de aceitação descrevem de forma leve e na linguagem do negócio um conjunto de exemplos concretos de uso que servem para determinar se a funcionalidade foi implementada corretamente e atende as expectativas explícitas e implícitas do cliente.

É importante destacar que o teste de aceitação descrito pelo cliente, é um dos critérios usados para determinar se a implementação da estória foi concluída. Ou seja, a estória está “pronta“ apenas quando os testes de aceitação forem executados com sucesso. Kent Beck, criador e disseminador do Extreme Programming (XP) afirma categoricamente: Estórias que não podem ser demonstradas por meio de testes, simplesmente não existem.

No que se refere a execução dos testes de aceitação, não existe um consenso absoluto, mas eles podem ser executados pelo próprio desenvolvedor manualmente, por um outro desenvolvedor, por um testador ou até mesmo pelo próprio cliente. Por outro lado, as metodologias ágeis aconselham que os testes sejam executados sempre que possível por meios automáticos para viabilizar ciclos curtos entrega e feedback contínuo. Em função disso, frequentemente os testes de aceitação são automatizados pelos desenvolvedores (ou pelos testadores) usando ferramentas especializadas, tais como:

* Fitnesse: (http://fitnesse.org/);
* Green Pepper: (http://www.greenpeppersoftware.com);
* StoryTestIQ: (http://storytestiq.solutionsiq.com);
* Robot Framework: (http://code.google.com/p/robotframework/);
* Entre outros.

O teste sob a perspectiva do cliente complementa a descrição da estória, oferecendo definições adicionais sobre o comportamento esperado da funcionalidade. Além disso, quando os testes de aceitação são automatizados, ele agem como um guardião da qualidade, alertando o time quando a estabilidade do software é violada em função de efeitos colaterais ou defeitos.

**Testes sob a perspectiva do desenvolvedor**

Enquanto nas metodologias tradicionais o desenvolvedor apenas escreve código, nas metodologias ágeis o desenvolvedor também é responsável pelos testes. No entanto, os testes do desenvolvedor tem o objetivo de prevenir e detectar defeitos na perspectiva do código. Ou seja, o desenvolvedor deve garantir a qualidade de cada unidade do código individualmente. Unidade, neste contexto, deve ser entendida como o menor trecho de código de um software que pode ser testado, podendo ser uma função ou procedimento em linguagens de programação procedurais ou métodos de classes em linguagens orientadas a objetos.

Para demonstrar se uma unidade atende os seus requisitos, o desenvolvedor escreve testes unitários (Unit Tests). Para tal tarefa, o desenvolvedor escreve fragmentos de código com o único objetivo de executar isoladamente uma unidade e observar o seu comportamento e os resultados produzidos. Na prática, estes fragmentos de código, são os testes unitários.

Os testes unitários são normalmente escritos usando o mesmo ambiente e a mesma linguagem de programação utilizada para a implementação da unidade que está sendo testada. Em linhas gerais, os desenvolvedores escrevem testes unitários usando bibliotecas especializadas que fornecem um ambiente padronizado, mecanismos para apresentar os resultados da execução dos testes, etc. Dentre as bibliotecas mais conhecidas, podemos destacar:

* JUnit: (http://www.junit.org/);
* NUnit: (http://www.nunit.org/);
* DUnit: (http://dunit.sourceforge.net/);
* CPPTest: (http://cpptest.sourceforge.net/);
* Entre outros;

Embora os testes unitários não impeçam a incidência de defeitos, eles representam um mecanismo muito eficaz para detectá-los rapidamente, o que reduz o tempo gasto em depuração e correção. A adoção de testes unitários traz muitos benefícios, como por exemplo:

* Testes unitários revelam os defeitos tão logo eles são introduzidos no software, o que evita a replicação do defeito em outras áreas do software;
* Testes unitários apontam onde e como ocorre um defeito, reduzindo o tempo gasto em depuração;
* Testes unitários são uma rede de segurança que dá confiança ao desenvolvedor realizar modificações e otimizações no software (Refactoring) sem medo de efeitos colaterais;
* Testes unitários, ao contrário da documentação escrita, não perde o sincronismo com o código e são considerados uma forma mais leve de documentação;
* Entre outros.

Na metodologia ágil conhecida como Extreme Programming (XP), é recomendada a adoção do estilo de desenvolvimento chamado TDD (Test-Driven Development), onde o desenvolvedor escreve o código de teste antes de escrever o código que implementa a funcionalidade. Apesar desta prática não ser exigida por outros métodos ágeis, ela oferece muitos benefícios, como por exemplo:

* Escrever testes antes da implementação induz o desenvolvedor a buscar mais detalhes sobre as características e regras associadas a estória;
* Escrever testes antes da implementação é uma excelente forma de expressar claramente as intenções do que cada unidade deve fazer, e como consequência, os testes servem como documentação do comportamento esperado de cada unidade;
* Escrever testes antes da implementação exige que o desenvolvedor escreva unidades mais simples e fáceis de testar. Como consequência, o código tem um melhor design.

Em resumo, independente da prática ou biblioteca de testes unitários adotada pelo time, o simples compromisso de criar testes induz o desenvolvedor a pensar mais profundamente na estória e também a escrever códigos mais simples e com o melhor design possível.

**Testes sob a perspectiva do testador**

Na visão teórica de algumas metodologias ágeis, não existe o papel conhecido por “Testador”, uma vez que todos os membros dos times ágeis são auto-suficientes, tecnicamente capacitados e todos são responsáveis pelos testes. Pelo termo “Testador“, queremos destacar um membro do time tecnicamente capacitado, com bastante experiência na área de testes e que dedica grande parte do seu tempo exclusivamente para testes. O testador contribui para o time de várias maneiras, especialmente estreitando a lacuna de comunicação entre o desenvolvedor e o cliente. Afinal, na prática, o desenvolvedor muitas vezes quer discutir somente aspectos técnicos da estória e o cliente, por sua vez, quer discutir apenas aspectos de negócio. Neste cenário, o testador atua como uma ponte permitindo que a informação seja traduzida e possa fluir para ambos os lados sem “ruídos“.

Além disso, os testadores apoiam na definição dos testes de aceitação com base nas idéias vagas levantadas pelos clientes. Afinal, nem todos os clientes tem condições de escrever testes (ou pensar em situações de testes). Por outro lado, os testadores, com base na sua experiência com o uso de técnicas de testes, complementam os testes dos clientes com idéias diversas, situações negativas e testes mais assertivos. Muitas vezes, porém, os testadores também são responsáveis pela automação dos testes de aceitação usando ferramentas especializadas.

Os testadores também apoiam os desenvolvedores no desenvolvimento das estórias e dos testes unitários fornecendo idéias e sugestões ou até mesmo lado a lado desempenhando programação em par (Pair Programming). Afinal, não devemos esquecer, os testadores são técnicamente capacitados tanto quanto qualquer outro desenvolvedor do time.

Todavia, a maior contribuição do testador para a equipe é a sua visão do software como um todo. Enquanto o foco dos testes dos desenvolvedores é método por método e o dos clientes é estória por estória, os testadores focam no sistema como um todo. Sua visão sistêmica ajuda a descobrir assunções incorretas e problemas de integração entre módulos. Para descobrir esses tipos de problemas, o testador executa testes exploratórios, que é um tipo de teste manual baseado na experiência e intuição.

O teste exploratório é, na sua definição mais básica, o aprendizado, a criação e a execução de um teste. Quando se realiza um teste exploratório, normalmente o testador não tem informações muito detalhadas sobre o que vai testar ou como vai testar. O testador se baseia na sua experiência e julgamento, assim como no conhecimento que ele vai adquirindo sobre o software durante a execução do teste exploratório. A partir dessa perspectiva, podemos afirmar que o teste exploratório é uma atividade iterativa e empírica de exploração que exige idas e vindas num processo de investigação contínuo onde a intuição, a criatividade e a experiência do testador são indispensáveis para garantir a eficácia do teste. O teste exploratório, tem as seguintes características:

* Não segue um roteiro rígido (segue guias e diretrizes);
* É baseado em pensamento estruturado e exploração livre;
* É adaptativo e flexível;
* Enfoca o aprendizado em paralelo;
* A execução do teste é guiada/aprimorada com base em execuções anteriores;
* Exige testadores experientes;
* Fornece fluxo imediato de feedback (e correção de curso);
* Amplifica a cobertura dos testes (unitários e de aceitação).

O teste realizado pelo testador é considerado por muitos autores como redundante, em função de que teoricamente os testes unitários e de aceitação seriam capazes de prevenir e detectar todos os tipos de defeitos. Kent Beck, no seu livro "Extreme Programming Explained: Embrace Change", afirma: “Você não pode resolver os defeitos com apenas uma prática. Os defeitos são muito complexos e cheios de facetas e nunca serão resolvidos completamente. Algumas práticas são certamente redundantes, identificando os mesmo tipos de defeitos. Apesar dessas redundâncias serem um desperdício, seja cauteloso ao remover práticas redundantes que sirvam para alguma proposta. O preço da redundância é mais do que pago pela economia de evitar a incidência de um defeito”.

Em resumo, os testadores fornecem uma linha de defesa adicional com objetivo de prevenir e detectar os defeitos precocemente. Além disso, na prática, os testadores realizam testes que normalmente são ignorados pelos usuários e desenvolvedores, tais como: performance, usabilidade, segurança, entre outros. Também devemos destacar a importância do testador em times ágeis na análise da causa raiz dos problemas a fim promover a melhoria e aprendizado contínuo.

**Conclusão**

Qualidade de software não está ligada apenas a encontrar mais defeitos, é uma questão de não introduzir defeitos. Por isso é essencial que as equipes de desenvolvimento sejam capazes de reduzir a incidência de defeitos e os custos associados a depuração e correção dos mesmos.

Neste contexto, a mudança de perspectiva em relação ao teste de software constitui umas das grandes diferenças entre as metodologias ágeis em relação a metodologias tradicionais de desenvolvimento de software. Enquanto nas metodologias tradicionais o teste é uma atividade realizada ao final do desenvolvimento por uma equipe independente, nas metodologias ágeis o teste é uma atividade comunitária feita por todos membros do time. Na visão ágil, a busca pela qualidade não é uma mera formalidade feita por alguém em algum momento, é uma filosofia de trabalho.

Afinal de contas, o teste ágil é o alicerce fundamental que tem por objetivo: guiar o desenvolvimento, induzir a um melhor design, fornecer segurança para sejam feitas mudanças e melhorias, complementar os requisitos, documentar o comportamento esperado, definir uma linguagem comum estreitando o relacionamento entre os membros do time, assegurar a integridade do software, detectar e apontar o local dos defeitos, aumentar a velocidade dos ciclos de feedback, reduzir o tempo gasto com depuração e demonstrar se o software funciona e atende as necessidades do cliente.

<http://blog.adaptworks.com.br/2013/09/o-que-e-agile-testing/>

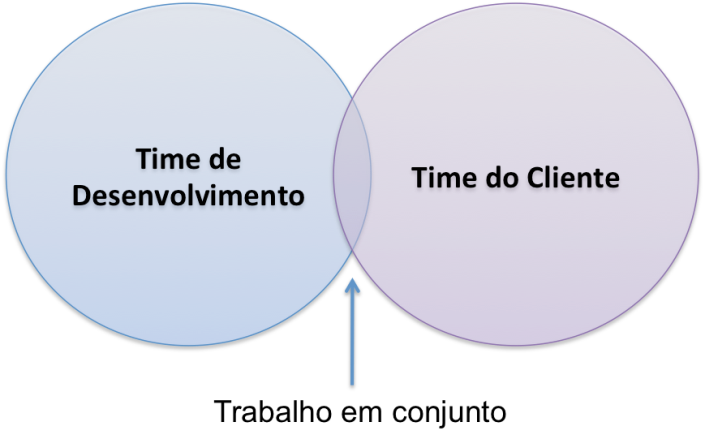
**Times**

Dentro de uma equipe ágil possuímos, basicamente, dois times:

Time do Cliente: todos que estão do lado do negócio no projeto (product owner, business experts, business analyts). Este time escreve histórias e funcionalidades que o time de desenvolvimento entrega.

Time de Desenvolvimento: todos que entregam código são parte integrante deste time, onde o papel de cada um por ser misto ou variado. Eles transformam todas as histórias do time do cliente em software.

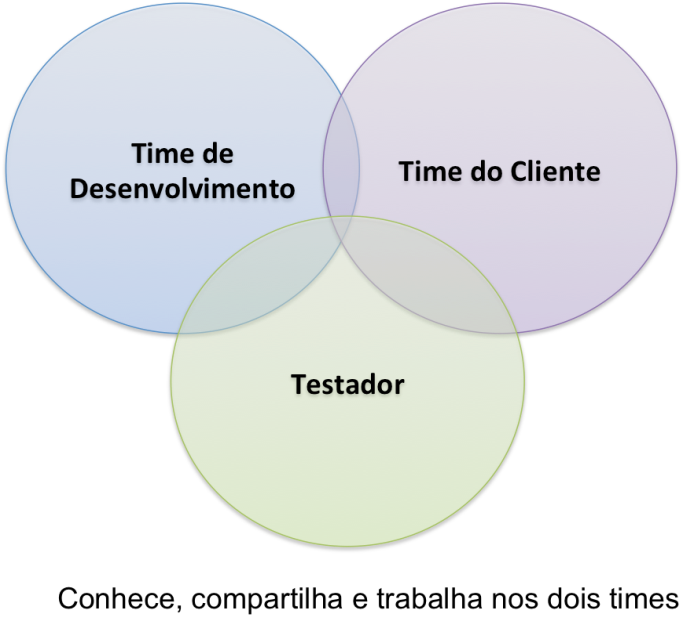
Estes dois times trabalham próximos uma grande parte do tempo e se transformam em um único time com um objetivo em comum: entregar valor para a organização.

[](http://blog.adaptworks.com.br/2013/09/16/o-que-e-agile-testing/times1/)

**E o testador?**

Falando do conceito de time para o Time do Cliente, os testadores são membros parciais do negócio, ajudando-os a descobrir requisitos e exemplos e também ajudando o time a expressar requisitos como testes.  
Testadores também são parte do Time de Desenvolvimento advogando por qualidade à favor do cliente e ajudando o time a entregar o máximo de valor ao negócio.

Muitos times não possuem membros se intitulando testadores. Entretanto o time precisa de alguém que ajude o Time do Cliente a escrever histórias e características, a escrever testes para elas, garantir que elas estão atendendo as necessidades e, automatizar os testes de regressão (para termos um feedback rápido e contínuo sobre a qualidade/saúde do software).

[](http://blog.adaptworks.com.br/2013/09/16/o-que-e-agile-testing/time2/)

**O que é então Agile Testing?**

Agile Testing é um conjunto de práticas, seguindo o Manifesto Ágil, que incorpora todas as técnicas de teste comumente utilizadas por profissionais de teste, tendo um grande foco em automação. A principal função é de criticar o produto, ou seja, constantemente garantir que o que está sendo especificado e desenvolvido realmente atende as necessidades do cliente e irá entregar valor ao negócio.

**O que é um Testador Ágil?**

Um Testador Ágil é aquele profissional que abraça as mudanças, colabora com pessoas técnicas e de negócio e entende o conceito de usar testes para documentar requisitos e guiar o desenvolvimento.  
Ele tende a ter bons conhecimentos técnicos para colaborar com o time de desenvolvimento a automatizar os testes e também para explorar o sistema a procura de comportamentos, testes e problemas.

Ser um testador ágil está ligado muito mais a atitude e comportamento do que conhecimento técnico. Ele olha para o produto como um todo, com uma visão de usuário e/ou cliente, que é o fator mais importante. Assim ele consegue tanto interrogar o Time do Cliente quanto a requisitos e funcionalidades como guiar o desenvolvimento, através de testes, exemplos e ferramentas, para que o Time de Desenvolvimento tenha também a visão do valor que cada um destes itens tem para o produto final.

**BDD – Foco no comportamento do sistema**

Esta nova visão faz com que os testes sejam apenas uma consequência necessária para garantir que o seu sistema esteja de acordo com as user stories e com os critérios de avaliação levantados pelo PO.

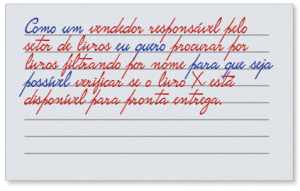
O primeiro benefício de BDD para o desenvolvimento ágil está em direcionar os programadores na criação dos testes. Não existirão mais dúvidas como:

* Como devo chamar o teste criado?
* O que devo testar?
* Até quando devo testar?

O foco agora está em testar o que realmente importa para o usuário e verificar que o que ele espera do sistema esteja de fato acontecendo. Não será mais necessário buscar o que deve ser testado, as user stories e os critérios de avaliação dirão isso para os desenvolvedores.

Outro ponto positivo é que seus testes estarão focados no que realmente tem valor para o usuário e, portanto, será mais fácil convencer o PO de que o tempo utilizado na criação de testes não é tempo perdido. A qualidade do seu sistema melhorará e impactará diretamente na experiência do usuário, o que, sem dúvida, será bastante valorizado pelo PO.

O processo de BDD é simples e se assemelha bastante ao fluxo de TDD. A partir da user story, como a apresentada abaixo, é possível identificar o **papel** que executará uma **funcionalidade** em busca de um **resultado**.



A partir daí, o próximo passo é criar o teste que garantirá que o código implementado está de acordo com o que foi especificado. Lembrem-se que o foco deve estar no comportamento, então nada além do que está previsto na user story e nos critérios de aceitação deve ser levado em consideração. Com os testes criados, é o momento de codificar a funcionalidade de forma que ela passe no teste gerado previamente. Vale também ressaltar que esses testes não devem ser esquecidos em algum lugar do projeto. A cada novo release é necessário rodar todos os testes para verificar se as funcionalidades continuam funcionando corretamente. Em alguns casos, a especificação pode mudar. Caso isso aconteça, o teste também deve ser atualizado.

Esse processo pode ser apoiado por frameworks como o [JBehave](http://jbehave.org/). Focado para desenvolvimento de BDD em Java, ele permite criar os testes a partir da descrição textual do que deve ser testado. No entanto, nada impede que os testes sejam criados com o que já conhecemos, como o JUnit. É apenas uma questão de opção da equipe.

No fim, a pergunta que fica é: BDD resolverá todos os meus problemas? Não. É preciso avaliar se outras técnicas de testes já conhecidas se adequam melhor aos seus projetos. Em alguns casos, será necessário criar testes mais focados na implementação e na verificação de determinada parte do código. No entanto, acredito que BDD pode sim agregar mais valor ao produto e evidenciar para o PO as melhorias que as equipes já sabem que os testes trazem. Além disso, ele pode tornar o desenvolvimento dirigido por testes mais amigável para os desenvolvedores.

**Os Critérios de Aceitação**

Os Critérios de Aceitação são representados por uma lista de itens de negócio que expressam formas de usar a funcionalidade implementada em uma História. O objetivo dessa lista é validar se a História foi implementada de acordo com o que o PO queria, por isso o nome Critério de Aceitação.

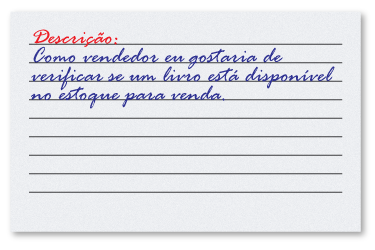
Ao final da sprint, na reunião de Revisão, a equipe apresentará História a História para o PO e com base nos Critérios de Aceitação, que foram definidos para cada História, que será realizada a apresentação e validação quanto ao funcionamento da mesma.

Aceitos os critérios estabelecidos a História já está em parte aprovada, faltando apenas validar se a mesma também atende a definição de História Pronta (mas isso é assunto para um próximo post…).

Agora, como surgem e como escrevemos os Critérios de Aceitação de uma História?

Esses Critérios (itens) surgem de perguntas que a equipe faz ao PO, no momento em que a História está sendo descrita, na busca por obter mais detalhes do que deve ser implementado. 

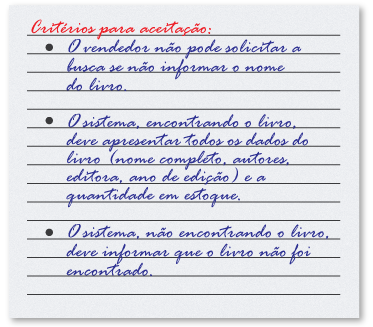
Vejamos a História descrita abaixo:



Algumas perguntas podem ser feitas ao PO, quanto a usabilidade dessa funcionalidade, como por exemplo:

* Como o usuário fará essa busca no sistema?
* Como o usuário deseja que seja apresentada a informação solicitada?
* O que deve ser feito se o livro não for encontrado?

Para o exemplo acima os possíveis critérios de aceitação seriam:



Como base nos Critérios de Aceitação a equipe deve escrever os Testes de aceitação, também abordado no post “[Testes de Aceitação no desenvolvimeno ágil](http://blog.myscrumhalf.com/?p=3194)” semana passada aqui no blog do [ScrumHalf](http://blog.myscrumhalf.com/" \t "_blank).

Ao final do desenvolvimento são esses testes que dirão se a funcionalidade foi implementada de forma a ser aceita ou não pelo PO. Se os testes forem criados com uso de ferramentas específicas para automatização da execução dos mesmos, isso garantirá maior agilidade e qualidade do produto.

Incluir os critérios de aceitação como parte da história só vem a agregar vantagens, como:

1. Prover material para a equipe pensar em como uma funcionalidade será executada pelo ponto de vista do usuário.
2. Eliminar ambiguidades quanto aos requisitos.
3. Confirmar que a história está completa e funcionando.
4. Garantir maior satisfação do usuário.  
   Resumindo… escrever Histórias, é muito mais do que apenas descrevê-las. É também especificar a usabilidade da mesma, com base nos Critérios de Aceitação.

**Testes de aceitação no desenvolvimento ágil**

Testes de aceitação (TDA), um dos tipos de testes usados para validar e verificar se o produto desenvolvido está de acordo com o que foi estabelecido nos requisitos. Esses testes são construídos a partir de situações de uso do sistema, estabelecendo valores de entrada, saída, tempo de resposta, etc. O objetivo do TDA é garantir que o sistema seja capaz de executar a funcionalidade acordada da forma desejada, e, consequentemente, garantir a qualidade do produto em desenvolvimento.

Os testes de aceitação envolvem testar uma funcionalidade do início ao fim, dando entradas para o sistema e observando o comportamento de saída. O resultado deve estar compatível com os requisitos do sistema em termos de tempo de resposta, validade do resultado, facilidade de uso e qualquer outro critério que o cliente tenha estabelecido.

É possível executar testes de aceitação de forma automatizada. Nós, utilizamos o JUnit e o Selenium para automatizar nossos testes. Dessa forma, a reprodução dos testes pode ser feita automaticamente, sem a necessidade de uma pessoa executar todos os passos manualmente a cada vez que tiver que retestar uma funcionalidade. Vale lembrar que, automatizados ou não, os testes de aceitação devem ser baseados em casos de testes elaborados pelo cliente, porque só o dono do negócio pode decidir se a funcionalidade está adequada para seu uso.

O TDA envolve um tipo de teste chamado "Teste de caixa preta", onde a importância do teste se encontra apenas no resultado obtido. Se o sistema apresentar a resposta esperada para uma determinada entrada, aquela parte do software pode ser considerada válida. É importante ressaltar que tão importante quanto testar a operação normal é testar também as situações em que o sistema deverá apresentar erro ou impedir uma ação do usuário. Se o produto responde corretamente a uma entrada válida não significa que ele irá responder negativamente a um valor incorreto. Cabe ao dono do negócio priorizar quais as operações que são mais importantes no sistema e quais as possibilidades de entrada que serão testadas.

Em métodos ágeis, utilizamos esse tipo de teste a cada porção de software funcionando entregue ao cliente. Dessa forma, podemos garantir que todas as funcionalidades entregues estão de acordo com a solicitação do cliente e o retrabalho é evitado. Uma vez que o desenvolvimento de uma User Story é finalizado, ela estará em pleno acordo com as funcionalidades que foram solicitadas.

Testar seu produto permite verificar e validar se o desenvolvimento do mesmo está convergindo para o idealizado pelo cliente. Quanto antes as divergências forem encontradas, mais barato é corrigi-las. O teste de aceitação é particularmente importante porque sua execução é feita na própria interface do software, validando o sistema para o cliente da maneira que ele será usado.

**10 Mandamentos de um Agile Tester**

**Forneça Feedback Contínuo**

Como a ideia é utilizar esses conceitos em projetos ágeis, o conceito de feedback contínuo não é nenhuma novidade. Uma das principais funções do testador é apoiar o Product Owner e o Cliente à escrever os requisitos de cada user story, na forma de exemplos e testes.

**Entregue valor para o cliente**

Desenvolvimento ágil é entregar valor em ciclos curtos para o Cliente. Nesse caso, o testador deve ficar atento às mesmas, se estão exatamente de acordo com o que o Cliente priorizou recentemente. Agile Testers sempre estão focados no projeto como um todo, i.e., as funcionalidades mais importantes de cada release devem estar prontas para serem entregues, mesmo que com isso, outras funcionalidades devam ficam em segundo plano.

**Buscar a comunicação olho no olho**

​Nenhum time funciona bem sem uma boa comunicação. Como hoje em dia existem times distribuídos, em diferentes continentes, trabalhando no mesmo projeto, o Agile Tester deve buscar uma forma única para facilitar a comunicação com todos da equipe.

**Tenha coragem**

​Coragem é um valor importante em projetos ágeis, práticas como testes automatizados e integração contínua permitem ao time praticar esse valor. O time deve ter coragem de realizar mudanças, mas, sem uma suite de testes de regressão automatizados, essa mudança pode ser muito insegura. O Agile Tester deve ter coragem para encontrar os erros de outros e ajudá-los a não cometerem o mesmo erro. Deve ter coragem de pedir ajuda, mesmo quando quem pode ajudá-lo for uma pessoa de difícil acesso.

**Mantenha a simplicidade**

​Agile Testers e seus times não devem apenas produzir o sofware mais simples possível que atenda aos requisitos do cliente, mas também devem encontrar a forma mais simples de medir se o software atende a esses requisitos. Logo, medições são importantes, mas, não devem ser uma barreira para a condução do projeto.

**Pratique a melhoria contínua**

​O Agile Tester sempre deve estar em busca de novas ferramentas, técnicas, habilidades ou práticas que auxiliem em seu trabalho de garantir que os desejos do cliente serão atendidos da forma mais simples possível.

**Responda a mudanças**

​Apesar de esse ser um dos valores mais importantes para times ágeis, é um dos mais difícies conceitos para Agile Testers. Pois com a estabilidade, o testador pode dizer que, após ele testar algo, aquilo está pronto. Entretanto, a adaptação a mudança é necessária, logo, a utilização de ferramentas automatizadas para testes pode auxiliar um Agile Tester a responder a mudanças com maior rapidez e segurança.

**Seja auto-organizado**

​O Agile Tester é parte do time auto-organizado do projeto. Logo, ele deve buscar apoio de todos do time para atingir seus objetivos.

**Foque nas pessoas**

​Projetos de sucesso são aqueles onde boas pessoas conseguem fazer seu melhor trabalho. Como o tester encontra erros, ele deve fazê-lo sempre respeitando a todos da equipe, nunca focando na pessoa que cometeu algum erro, mas sim no erro que foi cometido, para evitar algum mal estar entre os membros da equipe e ajudar a todos a não cometerem os mesmos erros.

**Aproveite**

​Trabalhe em um time onde se sinta confortável, ninguém consegue realizar um bom trabalho em um ambiente ruim.

Finalizando, pode-se perceber que o Agile Tester deve estar alinhado com os princípios e valores ágeis, todos os 10 mandamentos são baseados neles. E você, se considera um Agile Tester??

<http://blog.myscrumhalf.com/2011/08/os-9-principios-de-agile-testing/>

**9 principios de agile testing**

A maior diferença entre Agile Testing e os métodos tradicionais é apenas filosófica: os testes deixam de ser uma forma de evitar a entrega de um sistema ruim para passarem a fazer parte de uma política de desenvolver um bom produto desde o início do projeto.

Neste post, apresentamos os 9 princípios de Agile Testing.

O primeiro deles diz que um processo de desenvolvimento que inclui testes resulta em um produto melhor. Os testes têm que ser vistos como uma forma de criar um produto de qualidade desde o início e não apenas como uma forma de prevenir bugs na hora que o sistema é entregue ao cliente. Visando apoiar esta estratégia, o segundo princípio diz que os testes não podem ser vistos como apenas uma fase do projeto, eles devem ser considerados requisito para a conclusão de cada história da sprint.

O terceiro princípio diz que todos testam, não apenas o especialista em testes. Já o quarto princípio diz que uma história só é considerada pronta se forem realizados os testes necessários. Esses dois princípios se relacionam entre si. Se apenas o especialista em testes é o responsável por testar todos os artefatos gerados em uma sprint, essa atividade se tornará um gargalo e diminuirá a velocidade da equipe. Além disso, os tipos de testes realizados pela equipe e pelo especialista serão distintos e farão uma cobertura melhor do sistema. Por isso, todos devem testar sem esquecer que o teste deve estar presente em cada história.

Outro princípio de Agile Testing é diminuir o tempo entre a implementação de uma história e a validação com o usuário. Para as equipes ágeis este princípio é fácil de ser alcançado já que possuem a vantagem de entregar produtos para o cliente em pequenos ciclos de tempo.

O sexto princípio diz que os erros encontrados devem ser resolvidos assim que possível para que o código permaneça limpo, sem bugs.

O sétimo princípio diz que testes têm o potencial de descobrir requisitos não explicitados. Ao testar um produto, diversas expectativas sobre o seu comportamento diante de situações adversas são levantadas. A pergunta é: seria essa expectativa um requisito implícito ou apenas uma preocupação não justificada? O ideal é ter a resposta para essa pergunta antes de gastar tempo testando cenários que podem ser encarados como dispensáveis pelo product owner.

O oitavo princípio de Agile Testing é a redução da documentação de teste. Não é preciso escrever longos documentos com scripts a serem seguidos durante o teste. O uso de checklists e documentações leves já resolvem o problema. O importante é manter o foco na essência do teste e não nos detalhes envolvidos.

O último princípio de Agile Testing envolve a técnica de Test Driven Development (TDD). Definir os testes junto com a criação dos artefatos de cada história e usá-los como apoio do processo de desenvolvimento auxilia a criação de um projeto limpo e bem feito.

Então, resumindo, os 9 princípios de Agile Testing são:

1. Um processo de desenvolvimento que inclua testes resulta em um produto melhor;

2. Os testes não podem ser vistos apenas como uma fase do projeto;

3. Todos testam, não apenas o especialista em testes;

4. Uma história só é considerada pronta se forem realizados os testes necessários;

5. Diminua o tempo entre a implementação de uma história e a validação com o usuário;

6. Erros encontrados devem ser resolvidos assim que possível;

7. Testes descobrem requisitos não explicitados;

8. Reduza a documentação de teste;

9. Adote, sempre que possível, a técnica de Test Driven Development (TDD).

**A Testing Pyramid Agile**

A Testing Pyramid Agile é uma maneira prática de descrever a diferença entre o teste de software tradicional e testes para o desenvolvimento iterativo. As diferenças são importantes para o sucesso do Agile e necessitam de re-pensar e re-ferramentas para muitas organizações.

Deixe-me começar este post dizendo que não há nada de novo aqui a não ser talvez a minha escolha de palavras e expressões. O conceito do Testing Pyramid Agile foi descrito por muitas pessoas e, eu tenho quase certeza, originada por [Mike Cohn](http://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid). Vou descrevê-lo aqui, usando as brincadeiras que eu uso em minhas aulas, porque incorpora alguns conceitos e ferramentas que eu aprendi ao longo dos anos e parece passar a mensagem muito bem.

Testes tradicionais de software pode ser descrito utilizando a pirâmide no lado esquerdo da Figura 1. A grande maioria de teste é feito através de planos de teste para exercer manualmente o sistema, através da interface. Pode haver alguma automação na camada intermediária para testar serviços. Pode haver alguns testes unitários pelos desenvolvedores. Estou constantemente fico espantado com retorno que recebo quando eu pergunto as pessoas técnicas em meus organizações clientes se eles fazem o teste de unidade. Quando eu soube-o de volta no início de 2000 eu me senti como eu estava atrasado para o jogo. Aparentemente, ele ainda é uma prática bastante raro.

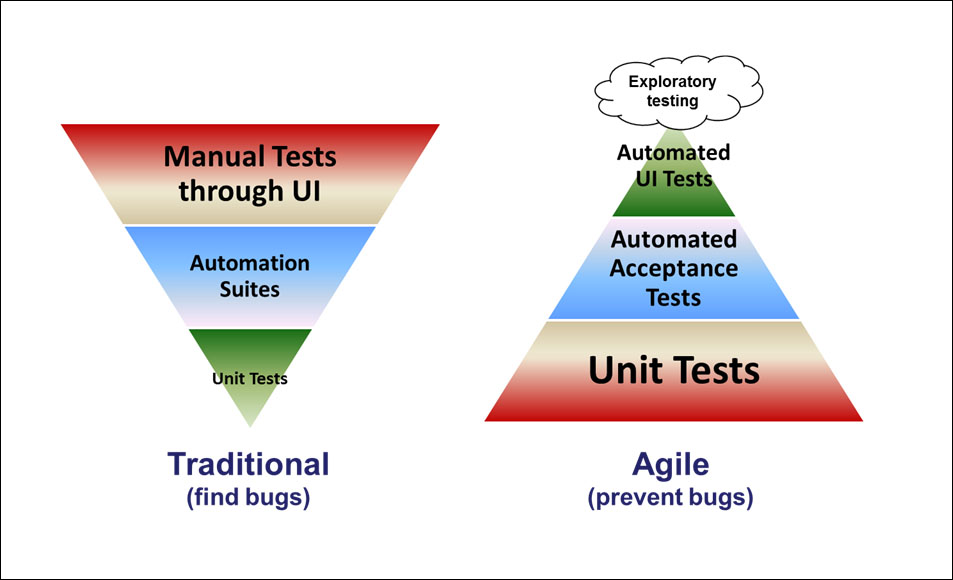
[](http://www.agilecoachjournal.com/index.php/2014-01-28/testing-2/the-agile-testing-pyramid/attachment/agiletestingpyramid-3/)

Figura 1: Testando Pyramids

Podemos pensar em testes tradicionais como projetado para "encontrar bugs". Essa filosofia se alinha com a tradicional faseada "cachoeira" processo algumas pessoas chamam de "código e correção". Os desenvolvedores escrever código para implementar recursos, sempre confiantes de que seu código é perfeito. Em seguida, os coordenadores do QA martelo sobre ele para encontrar os problemas, registrando defeitos para os desenvolvedores de corrigir. A fase de teste é muito mais sobre codificação como teste quando muitos defeitos são encontrados. Como o prazo se aproxima, vem para fora a fita adesiva digital para fazer reparos rápidos; dívida técnica acumula para o reembolso futuro.

Teste Agile tem uma forma diferente na Figura 1. Agora temos uma pirâmide com uma base sólida em vez de o cone de sorvete de testes tradicionais que podem tombar a qualquer momento. Teste Agile confia mais na automação. Ela exige uma maior contribuição dos desenvolvedores. E tem uma filosofia de base diferente - para **evitar** bugs.

Automação requer uma compreensão clara do que um sistema deve cumprir com sucesso. Nos velhos tempos, nós chamamos essas especificações funcionais. Quando sabemos que estes, podemos escrever testes que comprovam que eles foram atingidos. Para reduzir o processo e documentação em cima, equipes ágeis eficientes irá diretamente a partir de critérios de aceitação história do usuário para testes automatizados. Isto significa que a maior parte dos testes é feita usando testes automatizados na unidade e história nível (funcional) - escrever testes que indicam o que deve ser verdade em vez de testar se os requisitos funcionais estão trabalhando. Automação nos dá confiança de que o produto é adequado (faz o que é solicitado pela empresa) e é construído para a direita (oferece a funcionalidade solicitada com alta qualidade). Nós podemos validar a correção em uma base contínua, executando nossos testes automatizados muitas vezes ao dia.

A pirâmide Agile exige novas habilidades e atitudes em muitas organizações. Os desenvolvedores devem validar seu próprio trabalho com testes de unidade. As ferramentas para este típicos são implementações xUnit para a língua em uso. Minha abordagem favorito para isso é Test Driven Development. Profissionais de QA deve aprender ferramentas de automação que trabalham no nível história. Algumas ferramentas populares são Fitnesse, Concordion, pepino, Behat e Framework Robot.

Note-se que o topo da pirâmide, Teste de UI é muito pequena. A razão para isso é simplesmente que UI muda frequentemente. Quando isso acontecer, qualquer automação na camada de interface do usuário vai ter que mudar com ele. A manutenção de teste pode rapidamente superar o valor dos testes na camada de interface do usuário. Por isso, incentivamos as equipes de usar registro e tipo de reprodução testes tão pouco quanto possível. Aqui está mais algumas discussões sobre esta questão: a partir de [Elisabeth Hedrickson](http://testobsessed.com/2008/04/agile-friendly-test-automation-toolsframeworks) e [Martin Fowler](http://martinfowler.com/bliki/TestPyramid.html) . Em Agile ", testes automatizados" bem-sucedida não significa o uso de ferramentas do tipo Selenium para executar scripts gravados. Isso é apenas uma pequena parte de uma estratégia de ensaio eficiente.

Para realizar testes automatizados na camada intermediária, uma mudança filosofia de design pode ser necessária, uma que tem sido conhecida há muito tempo chamado de The Thin UI ou The Humble padrão caixa de diálogo. A essência desse padrão é que toda a lógica é movida para a camada lógica (ou seja, regras de negócios, serviços) abaixo da interface do usuário, deixando apenas os controles de interface do usuário que enviam e recebem mensagens. Ao fazer isso, equipamentos de teste pode acessar a lógica a ser testado sem passar os widgets de interface do usuário. Eles apenas se comunicar com a camada de lógica para testar tudo e abaixo dela.

Ao desenvolver de forma iterativa, testes de regressão é essencial. O sistema muda diariamente. Queremos saber que as coisas que costumavam trabalhar ainda funcionam, por isso tudo precisa ser testado continuamente. Isto é muito difícil de fazer quando o teste é feito manualmente. É repetitivo, chato, e provavelmente passível de erro a ser incompleta. Ele também leva mais tempo e mais tempo para o ponto onde há cada vez menos tempo para o novo trabalho recurso porque o teste leva tempo de todos. Figura 2 mostra como carga de teste de regressão aumenta geometricamente em desenvolvimento iterativo.

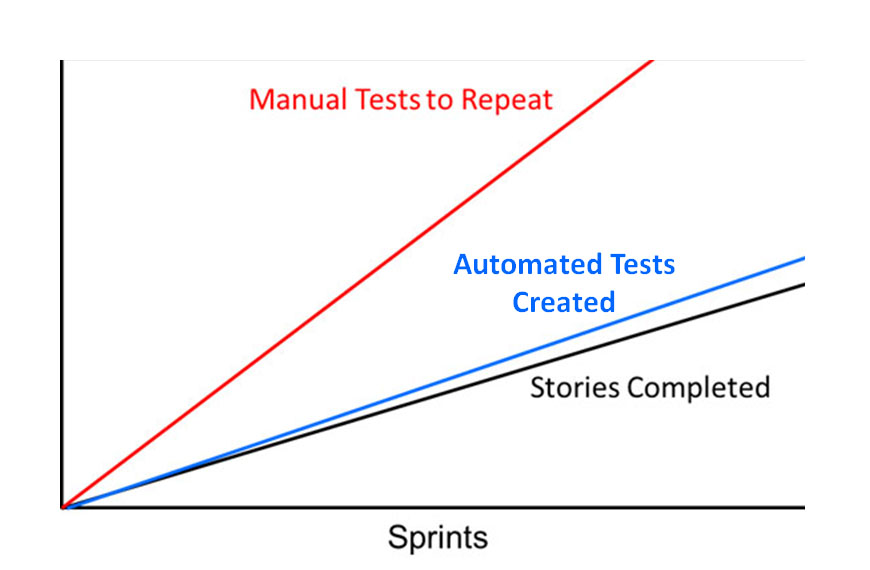
[](http://www.agilecoachjournal.com/wp-content/uploads/2014/01/TestLoad2.jpg)

Figura 2: Test Workload - Manual vs. Automated

Automação nos dá um benefício adicional que melhora a eficiência geral da equipe. Nos velhos tempos, correções de bugs, muitas vezes resultou em novos bugs em outras partes do sistema. Encontrar a causa de um bug pode levar um longo tempo. Com testes automatizados e de cobertura de código alta que recebemos, de graça, um conjunto de indicadores de problemas que nos diz quando uma mudança quebrou outra coisa e uma maneira mais fácil de encontrar, onde a quebra é: vá para o teste que está a falhar, ver o código ele testa e fazer a correção. Os testes automatizados são também uma forma de "rede de segurança" que nos ajuda a refatorar código com maior confiança. Se todos os testes passarem antes e depois da refatoração, sabemos que a nova implementação não quebrou algo que costumava trabalhar. Essa confiança é mapeado para a ampla base, sólida do Agile Testing Pyramid.When a maioria dos testes é automatizado, a quantidade de trabalho de teste pode acompanhar de forma linear com o desenvolvimento da história, também mostrado na Figura 2. Alguns testes de idade podem ter de mudar como novo histórias causar alterações na velhas histórias, mas estes serão bastante infreqüente. Ao automatizar o que sabemos deve ser verdade, temos de validação constante de testes de regressão e algumas surpresas. Eles ainda podem ser alguns comportamentos inesperados da complexidade global dos sistemas que tendem a construir de modo que o tempo economizado pela automação pode ser aplicada para o teste exploratório - alvo testes manuais ou script para encontrar as surpresas.

Teste Agile bem feito vai dar a sua equipe uma base sólida para o desenvolvimento iterativo. Podemos ser claro sobre o que deve estar funcionando. Podemos ter certeza de que eles estão trabalhando. E podemos mais rapidamente atender às coisas que deixar de trabalhar como nós adicionar funcionalidades.