GABRIELA DE OLIVEIRA PATUCI

Test Backlog:
Nova abordagem incremental de planejamento e execução de testes para Scrum

CAMPINAS

2013

UNIVERSIDADE ESTADUAL DE CAMPINAS

FT – Faculdade de Tecnologia

GABRIELA DE OLIVEIRA PATUCI

Test Backlog: Nova abordagem incremental de planejamento e execução de testes para Scrum

Dissertação apresentada à Faculdade de Tecnologia da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestra, na área de Tecnologia e Inovação.

Orientadora: Profa. Dra. Regina Lúcia de Oliveira Moraes

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DISSERTAÇÃO DEFENDIDA PELA ALUNA GABRIELA DE OLIVEIRA PATUCI, E ORIENTADA PELA PROFA. DRA. REGINA LÚCIA DE OLIVEIRA MORAES

CAMPINAS

2013

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Faculdade de Tecnologia
Vanessa Evelyn Costa - CRB 8/8295

P278t

Patuci, Gabriela de Oliveira, 1985-
Test Backlog: Nova abordagem incremental de planejamento e execução de testes para Scrum / Gabriela de Oliveira Patuci. – Limeira, SP : [s.n.], 2013.

Orientador: Regina Lúcia de Oliveira Moraes.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Tecnologia.

1. Software - testes. 2. Metodologia. 3. Planejamento. I. Moraes, Regina Lúcia de Oliveira. II. Universidade Estadual de Campinas. Faculdade de Tecnologia. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: *Test Backlog*: New incremental approach for planning and execution of tests in Scrum
Palavras-chave em inglês:
Software - testing
Methodology
Planning
Área de concentração: Tecnologia e Inovação
Titulação: Mestra em Tecnologia
Banca examinadora:
Regina Lúcia de Oliveira Moraes [Orientador]
Ana Estela Antunes da Silva
Alexandre Álvaro
Data de defesa: 06-12-2013
Programa de Pós-Graduação: Tecnologia

DISSERTAÇÃO DE MESTRADO EM TECNOLOGIA

ÁREA DE CONCENTRAÇÃO: TECNOLOGIA E INOVAÇÃO

Test Backlog: Nova abordagem incremental de planejamento e execução de testes para Scrum

Gabriela de Oliveira Patuci

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:

Profa. Dra. Regina Lúcia de Oliveira Moraes
FT / UNICAMP
Presidente

Profa. Dra. Ana Estela Antunes da Silva
FT / UNICAMP

Prof. Dr. Alexandre Álvaro
UFSCar - Sorocaba

RESUMO

O uso crescente das metodologias ágeis em todo mercado nacional e internacional tem exigido uma atenção especial às necessidades de adequação das metodologias tradicionais e suas respectivas implementações, para atender um novo ambiente de desenvolvimento de software.

Pelo fato de ser incerta a distribuição de tarefas nesse novo ambiente multifuncional, este trabalho apresenta o papel do analista de testes em ambientes ágeis e, principalmente, identifica como as atividades de testes poderão ser executadas de maneira a obter resultados de qualidade e total adequação aos "*times ágeis*".

A abordagem proposta neste trabalho consiste em realizar todas as tarefas de teste no dia-a-dia do time de maneira ainda mais incremental se comparada com a que está sendo feita hoje. Além disso, é proposto um novo formato para as *user stories* e o *backlog* do projeto: o *Test Backlog*. Este formato é proposto como um formato mais completo, utilizado para documentar um requisito, que engloba não só a visão de negócio do cliente relacionado ao que o sistema deve atender, mas também a visão técnica do profissional de teste relacionado ao que deve ser verificado antes da estória ser considerada feita, ou no caso do Scrum, "*pronta*".

O problema, para o qual está sendo apresentada uma nova abordagem, faz parte da realidade de muitas empresas de desenvolvimento de *software*, que estão fazendo a migração de um modelo de desenvolvimento tradicional, onde todos os papéis e o próprio processo são bem definidos, para o Scrum, a metodologia ágil que foi estudada.

Os resultados encontrados nos estudos de caso apresentam um forte indicativo de que existem maneiras de executar testes de maneira mais incremental, mantendo a qualidade sempre buscada em todas as metodologias anteriores e, ainda sim, ser ágil.

ABSTRACT

The increasing use of agile methodologies across national and international markets has required special attention to traditional methodologies needs of adequacy and their respective implementations to meet a new software development environment.

Due to the unclear distribution of tasks (that were previously delegated to specific roles) in this new multi-functional environment, this work presents the test analyst role in agile environments and also, identifies through research, how testing activities could be performed in order to obtain high quality results and overall suitability to "agile teams".

The approach proposed in this work is to perform all the day by day test tasks in a more incremental way if compared with what is being done today. It also consists on creating the user stories and project backlog in a new format: the Test Backlog. This format is proposed as a more complete requirement format, which includes not only client's business vision of what the system must do, but also the tester technical point of view of what should be checked before considering an user story as done.

The problem to which is being presented a new approach is part of the reality of many software development companies, which are making the migration from a traditional development model, where all the roles and the process are well defined, to Scrum, the agile methodology that was studied.

Finally, the results found in the case studies show a strong indication that there are ways to run tests in a more incremental mode, always keeping the quality sought in all previous methods and still being agile.

SUMÁRIO

AGRADECIMENTOS

Agradeço primeiramente a minha família, que é, e sempre foi, a minha base para tudo. Sem vocês nada seria possível. Agradeço principalmente a minha mãe, a mulher mais forte que já conheci, o meu exemplo para a vida toda.

Agradeço ao meu pai, tenho certeza que até hoje tenho seu apoio, como sempre tive, em todas as minhas decisões. Obrigada por me ensinar que tomar a decisão certa sempre vale a pena.

Agradeço ao meu noivo, meu companheiro nos momentos mais difíceis e nos mais felizes também. Eu não poderia ser maior ou melhor, com qualquer outra pessoa ao meu lado.

Agradeço a minha orientadora, que foi uma luz durante estes últimos anos. Ela foi a responsável por deixar as minhas ideias mais claras, mais objetivas. Ela sempre enxergou além. Este trabalho só foi possível porque ela acreditou na minha proposta e me ajudou a seguir em frente sempre.

Agradeço aos meus colegas de trabalho de ambos os times do estudo de caso. Eles tiveram toda a paciência e receptividade desde o início. Sem sua ajuda, eu não teria obtido os resultados necessários para a validação do meu trabalho. Obrigada pelo seu *feedback* constante ao longo deste ano, ele foi essencial para a melhoria da proposta como um todo.

“A persistência é o menor caminho do êxito”.

(Charles Chaplin)

“Quanto mais aumenta nosso conhecimento, mais evidente fica nossa ignorância”.

(John F. Kennedy)

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ATDD	*Acceptance Test Driven Development* (Desenvolvimento Dirigido a Testes de Aceitação)
Backlog	Conjunto de estórias
Block	Impedimento que pode bloquear o trabalho do time Scrum
Done	Status da tarefa que está pronta (desenvolvida e testada)
Estória	Formato de requisito
Planning Poker	Método utilizado para estimativas de esforço
PO	*Product Owner* (Dono do Produto)
Release	Entrega de um pacote (parte do produto em código)
SM	*Scrum Master*
Sprint	Iteração de desenvolvimento
Task	Tarefa - pequeno pedaço de requisito que será desenvolvido e testado
TDD	*Test Driven Development* (Desenvolvimento Dirigido a Testes)
UAT	*User Aceptance Testing* (Testes de Aceitação de Usuário)

1. INTRODUÇÃO

Mesmo com a evolução dos computadores, das técnicas e ferramentas nos últimos anos, a produção de *software* confiável, correto e entregue dentro dos prazos e custos estipulados ainda é um evento muito raro (Soares, 2004). Por esse motivo, diversas empresas de tecnologia estão migrando seu desenvolvimento tradicional para modelos de desenvolvimento que permitam gestão e planejamento mais leves, tais como SCRUM (Scrum Alliance, 2013), XP (Beck and Andress, 2004) entre outros. Estas metodologias de desenvolvimento são denominadas ágeis.

As metodologias ágeis para desenvolvimento de software são alternativas às chamadas metodologias tradicionais que, pelas exigências em termos de documentação e atendimento às normas, requerem um tempo maior de desenvolvimento. Este tempo alongado que se faz necessário para cumprir todas as etapas das metodologias tradicionais entra em conflito com a exigência de redução do *time-to-marketing* nos mercados atuais.

Pode-se ter uma ideia da ampla utilização do Scrum e do crescimento de procura por profissionais experientes nessa área se levarmos em conta dados como o número de usuários da Scrum Alliance (Comunidade Mundial e Oficial de Agile). Segundo o site da Scrum Alliance (2013), são mais de 190 mil membros registrados e certificados oficialmente. Dentre este número, podemos considerar os papéis oficiais como CSM (*Certified Scrum Master*), CSPO (*Certified Scrum Product Owner*), CSD (*Certified Scrum Developer*) e CSP (*Certified Scrum Professional*). Também devem ser levados em consideração os líderes da comunidade como *Trainners* e *Coaches*.

Outra maneira de se medir o aumento da utilização desta metodologia nos últimos anos é através do surgimento de um grande número de Congressos, Eventos e Treinamentos Oficiais. Dados retirados do site oficial (Scrum Alliance, 2013), contam mais de 1137 eventos relacionados ao Scrum apenas em 6 meses (3 últimos meses de 2012 e 3 primeiros de 2013).

Este grande aumento do uso do Scrum, também trouxe consigo um grande número de profissionais utilizando a metodologia de forma equivocada, aumentando, consequentemente, o número de *software* produzido com qualidade insuficiente por falta de

validação de requisitos e verificação da implementação efetuada. Nesse contexto é que se justifica a importância deste trabalho. As metodologias ágeis estão sendo amplamente utilizadas, mas muitas vezes, a busca por redução de tempo e custo tem feito com que as empresas desprezem fases essenciais do desenvolvimento, que mais tarde se configuram em um baixo nível de qualidade do produto desenvolvido. Logo, a definição de uma metodologia mais adequada e a correta aplicação das atividades de teste se faz necessária para a garantia da qualidade dos produtos desenvolvidos em ambientes ágeis. A atividade de teste, embora já exista nos ambientes de desenvolvimento ágeis, pode ser mais eficiente e eficaz se adequações forem feitas nas práticas atualmente utilizadas.

Um levantamento informal, feito em vários ambientes de desenvolvimento que utilizam a metodologia Scrum (empresas da região, projetos nos quais a aluna já trabalhou e trabalha atualmente, congressos e debates do assunto), identificou que as atividades de teste vêm sendo desenvolvidas de maneira muito díspar entre as empresas. Mesmo diferentes times, dentro de uma mesma empresa, não utilizam um padrão dessa atividade, muitas vezes, por não ter o conhecimento exato de como a atividade de teste deve ser corretamente desenvolvida em equipes que empregam metodologias ágeis.

Os trabalhos em andamento, relacionados às metodologias ágeis de desenvolvimento, em sua maioria, ainda estão preocupados em refinar a metodologia e resolver lacunas no processo básico de desenvolvimento e esquecem que uma das fases mais importantes para a entrega de um produto de qualidade é a fase de teste.

Existe uma lacuna nas pesquisas e publicações da área que precisa ser preenchida para auxiliar as empresas a melhor adequar as atividades de teste, já conhecidas e utilizadas nas metodologias mais tradicionais, à nova realidade do ambiente ágil. Durante a pesquisa sistemática da literatura (que será melhor detalhada no Capítulo 6, Trabalhos Relacionados), pouco material foi encontrado (se comparado com o material relacionado a outras áreas) que abordasse especificamente o teste de software no ambiente ágil de desenvolvimento. Grande parte do material existente refere-se ao trabalho liderado por uma pesquisadora, Liza Crispin (Crispin and Gregory, 2009) que compõe a bibliografia principal desta pesquisa. Mesmo pesquisando em sites e publicações da comunidade ágil, ainda existe uma dificuldade muito grande para se encontrar pesquisas que tenham como foco a atividade de teste, seja ela em artigos ou livros.

O Scrum é uma metodologia e não limita os tipos de testes que podem ser realizados pela equipe. Segundo Schwaber (2009), Scrum não é um processo prescribente, ou seja, ele não descreve o que fazer em cada situação. Ele é usado para trabalhos complexos nos quais é impossível predizer tudo o que irá ocorrer. Durante os últimos projetos pesquisados, foram verificados diversos tipos de testes funcionais em produção, incluindo regressão, *smoke*, *walkthrough*, como também, não funcionais incluindo testes de usabilidade. Estes conceitos serão melhor descritos no capítulo de Embasamento, mais especificamente no subcapítulo 2.2 (Testes de *Software*).

1.1. Objetivo

Considerando o contexto apresentado, o principal objetivo deste trabalho foi realizar uma pesquisa relacionada ao teste de software e uma melhor definição do papel do analista de testes em ambientes ágeis, sugerindo formas mais adequadas de executar as atividades diárias.

Especificamente, a proposta revê o formato utilizado para se escrever as estórias e inclui um novo padrão, em que casos de testes funcionais (e não funcionais quando pertinentes) são nelas integrados. Destaca ainda, a criação do papel do analista de teste e suas atividades como elementos essenciais na interação entre o dono do produto e o time de desenvolvimento. As atividades executadas pelos analistas de teste visam contribuir, de maneira diferenciada, para a equipe como um todo, já que agora este papel está inserido em times multidisciplinares, e não em times de testes como no passado. Pretende-se assim, manter os resultados de qualidade e total adequação às necessidades dos chamados “times ágeis”.

Não é objetivo do trabalho proposto descobrir ou indicar, exatamente, quais os tipos de testes serão executados pelas equipes que utilizam esta metodologia, uma vez que isso é uma decisão do time, baseado no contexto do sistema que está sendo desenvolvido.

O fato da área de testes não ter ainda uma documentação das boas práticas no desenvolvimento ágil, motivou a criação da proposta de implantação das atividades de testes e melhoria das práticas cotidianas. Espera-se que com a nova abordagem e as práticas sugeridas tomadas como guia, as atividades de teste possam colaborar com os times ágeis

para que se possa obter o equilíbrio entre a qualidade e a confiabilidade do trabalho das antigas equipes de testes e os princípios estabelecidos para a metodologia Scrum.

1.2. Contribuições

Esse trabalho de mestrado traz contribuições que impactam diversas partes do Scrum e que estão relacionadas a seguir:

- Estórias de Usuário (especificação de um requisito) – o formato das estórias foi redesenhado para incluir casos de testes funcionais e não funcionais (quando pertinentes), de forma que esta complementação auxilie os desenvolvedores a fazer uma implementação mais aderente aos requisitos do cliente. Permite que o analista de teste documente os casos de testes e amplie a cobertura da estória por ocasião dos testes do produto; este formato foi chamado *Test Backlog*;
- Pontos de Estórias (estimativa de desenvolvimento de uma estória) – o formato das estórias amplia o conhecimento da equipe do Scrum e auxilia na estimativa dos pontos de estória, fazendo com que o esforço necessário para completar cada estória seja mais realisticamente estimado;
- *Backlog* do Produto (conjunto de requisitos) - na fase de definição do *Backlog* do Produto os requisitos serão definidos no novo formato, ou seja, incluindo os casos de testes. Dessa forma, é essencial a participação do analista de teste para auxiliar o Dono do Produto (*Product Owner* – PO) na preparação dessa documentação. Sendo assim, o analista de teste estará discutindo os pormenores dos requisitos do sistema com o PO e trará uma visão mais completa desses requisitos para o time Scrum, sendo um elo importante de ligação entre cliente e time do Scrum;
- *Backlog* do *Sprint* (conjunto de requisitos que compõe a nova *release*) – como a definição do *backlog* do *Sprint* depende da estimativa das estórias, o mesmo impacto positivo que se destaca para os pontos de estórias estará presente no *backlog* do *Sprint*. O tamanho de cada *sprint* será mais realisticamente definido e o comprometimento do time em entregar um número de estórias será mais confiável;
- *Sprint* (período da iteração) - a atuação do analista de testes volta a ser essencial para a validação das *estórias* do *sprint*. Visto que ele acompanhou toda a definição juntamente com o PO, ele estará melhor preparado para os testes de validação dos

requisitos e de regressão do produto, que deverão ser constantemente aplicados durante o *Sprint*;

- Atividades de Testes (Teste Incremental) – ao invés de trabalhar com *tasks* (tarefas) de teste que são iniciadas ao final de cada estória, como um dos requisitos da definição de "*pronto*" (do Scrum, o conceito de *done* que é estabelecido pelo time) desta estória, o testador responsável pela estória diluí o trabalho de teste ao longo de todas as *tasks* da estória. Logo, é feito um *trabalho conjunto* entre o testador e o desenvolvedor da estória para que o testador acompanhe o desenvolvimento desde o início e possa antecipar os defeitos. O teste feito após o desenvolvimento também é mantido, mas é apenas mais um passo da validação, e não a validação completa.

- Atividades de Testes (Testes de Regressão) – Muitos times, atualmente, têm dificuldades em realizar os testes de regressão, pois estes testes exigem que os casos de testes mais importantes sejam refeitos, e estes casos de testes planejados e executados durante o desenvolvimento da estória, não estavam sendo armazenados. Muitos times consideravam importante armazenar apenas os defeitos encontrados e não o planejamento de testes e quais testes foram executados. Nesta proposta, pretende-se atrelar o planejamento de testes à estória, e facilitar, desta forma, a reexecução dos casos selecionados para os testes de regressão de *software*.

Para se entender a viabilidade e aplicabilidade da proposta foi realizado um experimento controlado comparando o uso conhecido da metodologia Scrum com a proposta do Scrum que inclui o *Test Backlog*. Os experimentos foram realizados por 2 times que estavam desenvolvendo projetos com conjuntos de requisitos muito similares (*hotsites* de medicamentos de uma grande empresa farmacêutica) e mesma tecnologia (Drupal). Os resultados obtidos nestes experimentos trazem fortes indícios de que a qualidade dos projetos pode ter uma considerável melhoria se utilizada a metodologia proposta.

1.3. Organização do Trabalho

Este trabalho de mestrado foi dividido em capítulos. Este primeiro capítulo foi composto por uma introdução que contextualiza o trabalho executado, a motivação da escolha deste tema, o objetivo da pesquisa e ressalta as contribuições. O segundo capítulo trata-se do

Embasamento Teórico utilizado para desenvolver todo o trabalho de mestrado. As propostas defendidas neste trabalho como Testes Incrementais e um novo formato de *backlog* chamado aqui de *Test Backlog* são melhor descritos ao longo do capítulo 3. A Metodologia e todos os recursos utilizados para obter os resultados necessários estão descritos em detalhes no capítulo 4. Todos os Resultados e a Discussão se concentram no capítulo 5. O Capítulo 6 traz os trabalhos relacionados ao tema proposto. As Conclusões baseadas nestes resultados e os Trabalhos Futuros são apresentados no capítulo 7.

Complementarmente, o Anexo A traz exemplos dos artefatos utilizados pelos Times durante os *Sprints* que compõem os experimentos reportados neste trabalho, o Apêndice A apresenta uma cópia do artigo publicado no 3rd Brazilian Workshop on Agile Methods (WBMA'2012) que ocorreu em conjunto com a conferência Agile Brazil 2012 em São Paulo, Brasil (Patuci e Moraes, 2012) e o Apêndice B traz um conjunto de exemplos de Estórias para ilustração do método proposto e comparação com outros métodos (Scrum padrão e BDD – Behavior Driven Development).

2. EMBASAMENTO TEÓRICO

Esta seção apresenta o conteúdo que foi usado como base para as pesquisas deste projeto. O capítulo foi dividido entre os conceitos básicos da metodologia Scrum (utilizada amplamente neste trabalho de mestrado), dos testes em geral e por fim, de dois tipos de desenvolvimento: o orientado a testes (TDD, do inglês *Test Driven Development*) e o orientado a testes de aceitação (ATDD, *Acceptance Test Driven Development*).

2.1. Metodologias Ágeis

As metodologias utilizadas para a produção de software antes do surgimento das metodologias ágeis eram mais voltadas a sistemas maiores e mais complexos do que a produção em si. Com o objetivo de produzir este tipo de sistema, o desenvolvimento envolvia equipes enormes, o que justificava uma metodologia com muitos detalhes e documentação mais pesada, como o antigo RUP (Rational Unified Process), mais conhecido nos dias atuais como IRUP (IBM Rational Unified Process) (IBM, 2013).

Quando o desenvolvimento passou a ser voltado também, para projetos menores e equipes mais reduzidas, o trabalho com a documentação passou então a ser considerado pesado e demandar, muitas vezes, mais tempo do que o desenvolvimento do produto em si. O fato dos requisitos mudarem com certa frequência também acabava por exigir grande retrabalho na documentação já feita, o que acontecia não só pelo dinamismo das regras de negócio, mas também pelo fato do tempo demandado para a preparação da documentação ser grande o suficiente para ser considerado responsável por grande parte do tempo total do projeto.

Nesse contexto da transformação das necessidades do mercado, começam a surgir propostas de metodologias mais leves. Uma das propostas, senão a mais conhecida de todas, foi desenvolvida por alguns estudiosos e profissionais com reconhecida competência no mercado. Esses profissionais se uniram para apresentar uma ideia que revolucionaria o mercado de *software*, até então, baseado no *Rational Unified Process*, o RUP (IBM, 2013). O Manifesto Ágil (2013) veio para coroar a apresentação de todos os conceitos

considerados vitais para o sucesso de projetos que quisessem considerar a utilização de metodologias ágeis. Os conceitos considerados pelo Manifesto (Manifesto Ágil, 2013) são:

"Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar:

Indivíduos e interação entre eles *mais que processos e ferramentas*

Software em funcionamento *mais que documentação abrangente*

Colaboração com o cliente *mais que negociação de contratos*

Responder a mudanças *mais que seguir um plano."*

Além dos conceitos citados anteriormente, o manifesto também possui 12 princípios que foram a base para a construção do manifesto em si. Estes princípios, citados no manifesto (Manifesto Ágil, 2013) são:

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.

2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.

3. Entregar *software* funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.

4. Pessoas relacionadas aos negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.

5. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.

6. O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.

7. *Software* funcional é a medida primária de progresso.

8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.

9. Contínua atenção à excelência técnica e bom *design*, aumenta a agilidade.

10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.

11. As melhores arquiteturas, requisitos e *designs* emergem de times auto-organizáveis.

12. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajusta e otimiza seu comportamento de acordo.

Deve-se atentar ao fato de que muitas vezes, os times pensam que documentar os requisitos, definir uma boa arquitetura, fazer a análise do projeto e planejar os casos de teste não são obrigatórios e principalmente, que não precisam de rigor para a execução destas tarefas. Podemos verificar nos princípios acima, que um dos pontos citados é que se tenha atenção contínua e excelência técnica. Logo, um projeto só pode estar completo com um mínimo de rigor.

2.1.1. Estórias

Segundo Cohn (2004) "uma Estória descreve funcionalidades que vão ser valiosas para o usuário ou comprador de um sistema". Estórias são compostas por três aspectos:

- Uma descrição da estória para ser usada no planejamento e também como um lembrete;
- Discussões sobre a estória que vão servir para levantar todos os seus detalhes;
- Testes que transmitem e documentam os detalhes e poderão ser usados para determinar quando a estória estará completa.

O terceiro aspecto citado por Cohn é uma das bases para este trabalho, visto que já deixa clara a importância dos testes como base para o desenvolvimento e para ajudar no detalhamento das estórias. Porém, os testes aos quais se refere esse aspecto foram interpretados como sendo os testes de aceitação e isso faz parte hoje do estado da prática no Scrum.

Estórias são uma das principais bases para as práticas ágeis em geral. De certa forma, elas substituem o formato de requisitos, como Casos de Uso, utilizado anteriormente por metodologias tradicionais como o RUP (IBM, 2013).

Mais do que apenas saber do que se tratam as estórias, o que se espera dos profissionais que vão trabalhar com elas, principalmente os *Product Owners* e os Analistas de Testes, é a prática de escrever boas estórias. A maneira mais eficiente de se fazer isso, segundo Cohn (2004) e Crispin (2003) seria trabalhar com testes de aceitação.

Em seu livro, Mike Cohn (2004) exemplifica o padrão de criação de estórias desta forma: "Como <**tipo de usuário**>, eu quero <**objetivo da estória**> para que <**razão**>". O exemplo mais conhecido desde então é o da reserva de hotel (Figura 1), descrito por Cohn em seu livro.

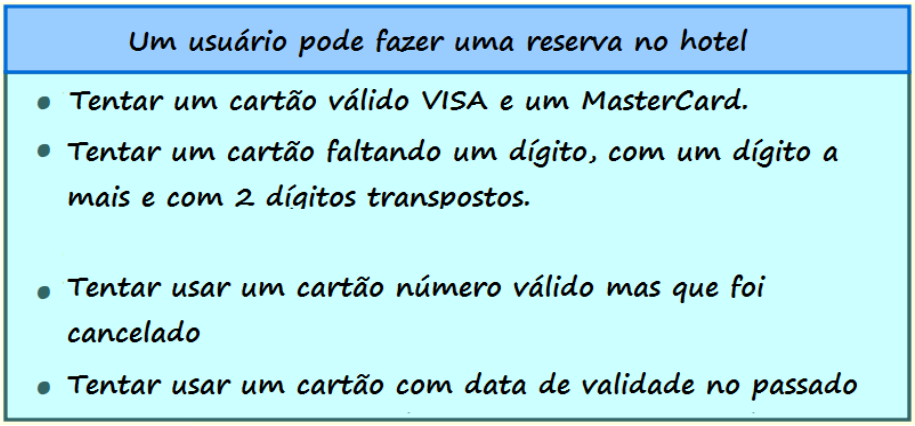


Figura 1: Cartão de Critérios de Aceitação (traduzido de Cohn, 2004)

Este exemplo demonstra a utilização de testes de aceitação como critérios de aceitação das *user stories*. Este formato ainda não é tão completo quanto o proposto neste trabalho, e sim o inicial proposto por Cohn e utilizado desde então na grande maioria dos times Scrum.

No capítulo "*Acceptance Testing User Stories*", Cohn defende a ideia de que uma estória pode ser escrita de várias maneiras, inclusive utilizando-se cartões que tragam o nome da estória e os casos de testes em forma de critérios de aceitação, como já citado anteriormente na Figura 1 (Cohn, 2004).

Outro fator muito importante citado por Cohn nesse mesmo livro é o fato de que "estórias devem ser escritas de forma a serem testáveis" (Cohn, 2004). Um exemplo citado é uma estória não testável, tal como:

"Um usuário não deve esperar muito tempo para visualizar uma das páginas". Neste caso, pode-se perceber que não se pode pensar em casos de teste sem conhecer o que

está sendo considerado como “muito tempo”. Esta mesma estória, se escrita de outra maneira, poderia ser testável:

“Telas devem aparecer dentro de dois segundos em 95% dos casos”. Valores definidos corretamente e estórias elaboradas de maneira a serem testáveis no futuro, bem como a escrita de estórias que são acompanhadas de casos de testes são fatores de extrema importância para a abordagem que estamos propondo nesta dissertação.

Atualmente, as estórias são escritas pelo *Product Owner* (PO) e utilizadas como base para a reunião de planejamento do *sprint.* Elas possuem uma visão do PO, logo uma visão de *business*, ressaltando o que o produto precisa ter. É também na reunião de planejamento que o time analisa a estória e realiza a estimativa de tempo necessário para o desenvolvimento da funcionalidade a ela relacionada.

2.1.2. Scrum

O surgimento do Scrum e sua primeira citação deram-se no artigo de Takeuchi e Nonaka (1986), "*The New Product Development Game*". Neste período, a metodologia surgiu muito mais como um estilo de gerenciamento de projeto, utilizando pequenas equipes multidisciplinares. O fato dessas equipes serem altamente eficazes e atacarem seus objetivos de maneira agregada e de uma só vez, fez surgir o nome da formação do jogo de Rugby, o Scrum. Assim, a função primária desta metodologia seria a de gerenciar projetos de *software*, assim como outras metodologias ágeis. Atualmente, o Scrum é a representação mais difundida dos métodos ágeis (Scrum Alliance, 2013).

Segundo Schwaber (2004), o Scrum tem as seguintes fases: Pré-jogo (Concepção ou Iniciação), Jogo (Desenvolvimento) e Pós-jogo (Fechamento ou Entrega). A Figura 2 apresenta um esquema dessas fases.

Pré-Jogo

• Planejamento: Definição de uma nova release baseada no *backlog* conhecido, juntamente com uma estimativa do cronograma e do custo. Se um novo sistema está sendo desenvolvido, esta fase consiste na concepção e análise. Se um sistema existente está sendo melhorado, esta fase consiste em uma análise limitada com foco na parte do sistema envolvida na manutenção.

Depois de feito o trabalho de criação do *Backlog* de Estórias pelo PO, os próximos pontos a serem considerados, segundo a Metodologia Scrum são: (Scrum Methodology, 2013)

- ✓ Data de entrega;
- ✓ O número da *release* e as funcionalidades;
- ✓ As funcionalidades mais importantes (seleção);
- ✓ A lista de objetos do *Product Backlog*;
- ✓ A estrutura do time;
- ✓ As ferramentas necessárias;
- ✓ Controle de risco;
- ✓ Os custos da *release* (não apenas o desenvolvimento mas também o custo de treinamento do time e de *marketing*).

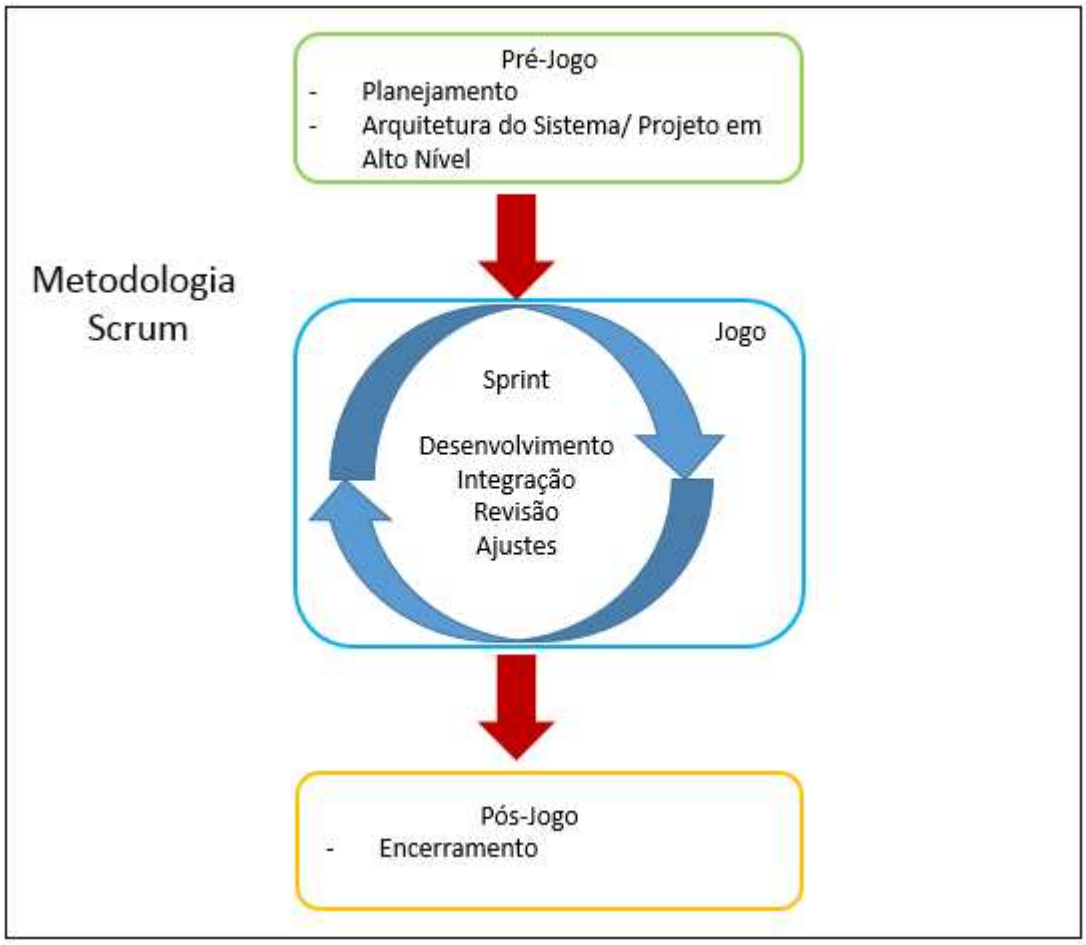


Figura 2. Fases do Desenvolvimento Scrum traduzido e adaptado (Schwaber, 2009)

• Arquitetura: Projeto de como os itens do *backlog* serão implementados. Esta fase inclui a modificação da arquitetura do sistema e *design* de alto nível. Nesta fase algumas atividades podem ser executadas, dependendo das estórias selecionadas durante a reunião de planejamento (Scrum Methodology, 2013):

- ✓ O time deve revisar o *backlog*;
- ✓ O time deve pensar em quais mudanças devem ser feitas, nas novas funcionalidades e no *design* do processo de desenvolvimento;
- ✓ Refinar o produto antigo (se for o caso);
- ✓ Aprender novas tecnologias, fazer análise;
- ✓ Resolver problemas que apareçam durante o processo;
- ✓ Realizar reuniões para revisar o *design* no final do processo, onde o time troca as informações e delega as mudanças ainda necessárias.

Jogo

• *Sprints* de desenvolvimento: Desenvolvimento de funcionalidade nova, com relação constante com as variáveis de tempo, os requisitos de qualidade, custo e concorrência. A interação com estas variáveis define o fim desta fase.

Cada *sprint*, ou seja, cada fase de Jogo, pode ser composto por 4 passos (Schuwaber, 2004):

1. Desenvolvimento (análise, *design* e desenvolvimento) – o time realiza uma análise da situação atual do produto, pensa em quais alterações devem ser feitas na implementação dos requisitos retirados do *backlog*, desenha o processo e finalmente, desenvolve o código: implementando, testando e documentando o produto;
2. Abertura e fechamento de pacotes (*releases*);
3. Reuniões de revisão do *sprint* onde o time apresenta o trabalho desenvolvido e o progresso, resolvendo problemas, adicionando novos itens do *backlog*, fazendo a revisão dos riscos;
4. Ajuste – informação colhida durante a revisão é consolidada em pacotes.

Pós-Jogo

- Encerramento: Preparação para o lançamento de uma nova release, incluindo a documentação final, pré-lançamento, testes e lançamento.

2.1.2.1. Estimativa

O Scrum, como uma metodologia ágil, documenta os requisitos do produto como Estórias que, usualmente, seguem o modelo apresentado por Mike Cohn (2004), como apresentado na Figura 1. Além de documentar os requisitos, as Estórias são tomadas como base para estimar o esforço necessário para o desenvolvimento de cada Estória e do sistema como um todo.

A estimativa destas Estórias durante a reunião de planejamento é feita através de uma técnica chamada de "*Planning Poker*" ou "*Scrum Poker*", no caso específico da metodologia Scrum. O método foi descrito pela primeira vez por James Grenning (2002), e mais tarde popularizada por Mike Cohn em seu livro "*Agile Estimating and Planning*" (Cohn, 2005). A Figura 3 ilustra a prática utilizada pelo Scrum para estimar esforço.

Figura 3. Planning Poker (Transcrito de Cohn, 2005)

O cliente explica a Estória. O time pondera, discute e depois, todos juntos, mostram suas cartas com os pontos que consideram adequados para expressar o esforço da Estória em questão. Os membros do time, que tiveram as cartas com valores extremos, iniciam uma discussão para buscar um consenso e convergir para uma estimativa comum. Ao final, cada

jogador termina com um conjunto de cartas na mão, que parece a mão do jogo de *poker,* sendo esse o motivo da denominação "*Planning Poker*" (Grenning, 2002). Sendo assim, todos os participantes são jogadores e participam igualmente para estimar o esforço do conjunto de Estórias que formam um incremento do sistema até que este seja considerado "*Pronto*" (*Done*). A estimativa é obtida pelo consenso do time, não ficando restrita à opinião dos mais comunicativos ou mais experientes.

Scrum exige que os times desenvolvam um incremento de funcionalidades do produto a cada *sprint*. Segundo Schwaber (2009), esse incremento deve ser potencialmente "entregável", pois o *Product Owner* pode optar por implantar a funcionalidade imediatamente. Para isso ser possível, o incremento deve ser um pedaço completo do produto. Ele deve estar "Pronto". Cada incremento deve ser adicionado a todos os incrementos anteriores e exaustivamente testado, garantindo que todos os incrementos funcionem juntos.

Schwaber (2009) também constata que, no desenvolvimento de produtos, afirmar que o incremento do sistema está "*Pronto*" pode levar alguém a presumir que o subproduto está pelo menos bem codificado, refatorado, que tenha passado por testes unitários, que tenha sido preparado o *build* e que tenha passado por testes de aceitação. Outros podem presumir que apenas o código tenha sido desenvolvido. Se ninguém sabe qual a definição de "*Pronto*", os outros dois pilares (Fazer ou "*To Do*" e Em Progresso ou "*In Progress*") do controle de processos empíricos não podem cumprir seus respectivos papéis. Quando alguém descreve algo como "*Pronto*", todos devem entender o que "*Pronto*" significa.

Cada time geralmente cria sua definição de "*Pronto*" na primeira reunião de planejamento (*planning)*. Esta definição pode incluir alguns pontos como: análise, projeto, refatoração, programação, documentação e testes (funcional, unitário, entre outros). O mais importante é esta definição estar alinhada dentro de todo o time e também com o PO.

2.1.2.2. Artefatos do Scrum

Os conceitos que serão apresentados nos próximos subcapítulos são relacionados aos principais artefatos utilizados pela metodologia Scrum, metodologia esta utilizada amplamente neste projeto de pesquisa.

Backlog do Produto

O *backlog* do produto é uma relação organizada de tudo o que deverá ser produzido e entregue durante o projeto. Aqui se concentram todos os requisitos funcionais e não funcionais, que serão identificados unicamente, descritos e estimados quanto ao esforço. Mudanças nas regras de negócio, de legislação e de tecnologias se refletirão em mudanças no *backlog* do produto que, por esse motivo é dinâmico.

O principal responsável pelo *backlog* do produto é o PO que tem a responsabilidade sobre o conteúdo e disponibilidade, bem como, pela priorização de seus itens. A prioridade deve ser estabelecida levando-se em consideração o valor para o negócio, riscos e necessidades da empresa.

O tamanho de um item do *backlog* do produto (esforço estimado) não pode exceder o esforço que o time consegue alcançar em um único *sprint*. Caso isso ocorra, o item deverá ser decomposto em subitens.

Sprint

O *Sprint* é uma iteração, eventos com duração fixa. Durante o *sprint*, o *Scrum Master* garante que não será feita nenhuma mudança que possa afetar a Meta do *Sprint*. Tanto a composição do time quanto as metas de qualidade devem permanecer constantes durante o *sprint*. Os *sprints* contêm e consistem na reunião de Planejamento de *Sprint*, o trabalho de desenvolvimento, a Revisão do *Sprint* e a Retrospectiva do *Sprint*. Os *sprint*s ocorrem um após o outro, sem intervalos entre eles.

Backlog do *Sprint*

Durante a Reunião de Planejamento do *Sprint*, o time define como transformará o *Backlog* do Produto selecionado em um incremento pronto. O time geralmente começa projetando o trabalho. Enquanto projeta, o time identifica tarefas. Essas tarefas são pedaços detalhados do trabalho necessário para converter o *Backlog* do Produto em *software* funcional. As tarefas devem ser decompostas para que possam ser feitas em menos de um dia. Essa lista de tarefas é chamada de *Backlog* do *Sprint*. O time se auto-organiza para se

encarregar e se responsabilizar pelo trabalho contido no *Backlog* do *Sprint*, tanto durante a Reunião de Planejamento do *Sprint* quanto no próprio momento da execução do *sprint*.

Além dos conceitos apresentados no Manifesto Ágil (2013), a comunicação é considerada imprescindível para as atividades do dia-a-dia, como por exemplo, as reuniões diárias previstas na metodologia. Todo período de desenvolvimento (*sprint*) deve ser iniciado com uma reunião de planejamento chamada *Planning Meeting*. Nesta reunião serão discutidos todos os requisitos que, normalmente, são documentados no formato de estórias (*user stories*), que se transformarão em itens de produto e serão contabilizados para uma visão do projeto. Com os requisitos e a visão do projeto estabelecidos, inicia-se a segunda parte desta reunião, onde tais estórias serão subdividas em atividades menores chamadas tarefas (*tasks*).

Realizadas as duas etapas de planejamento, existem reuniões diárias durante todo o *sprint*, para que sejam discutidas as atividades que estão sendo desenvolvidas por cada membro do time, visando identificar impedimentos para se completar as atividades e definir o que se pretende fazer como próximo passo. As reuniões diárias, feitas em pé e com duração máxima de 15 minutos, são denominadas *Stand Up* ou *Daily Meeeting*.

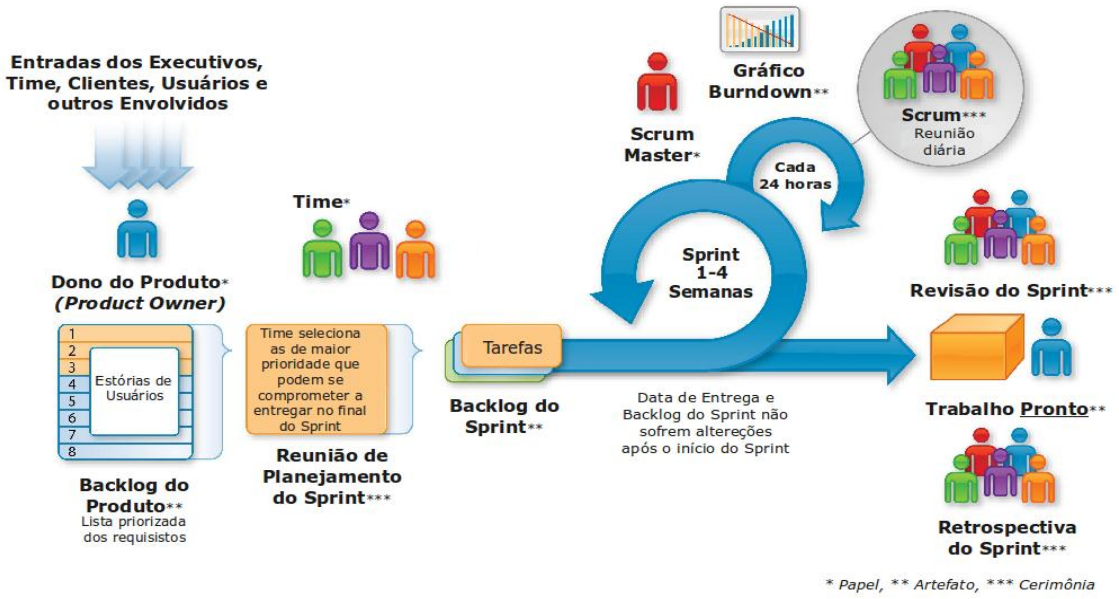


Figura 4: Desenvolvimento ao longo do *sprint* (Traduzido e adaptado de Scrum Alliance, 2013)

Chegado o final de cada *sprint*, são realizadas mais duas reuniões, com duração de uma hora cada: a reunião de Revisão (*Sprint Review)*, onde todos os membros do time exibem os resultados do trabalho do *sprint* para o cliente (representado pelo *Product Owner* - PO*);* e a Retrospectiva (*Sprint Retrospective)*, onde estas mesmas pessoas discutem quais foram as lições aprendidas durante este tempo de desenvolvimento e o que pode ser melhorado para o próximo *sprint*. A Figura 4 representa todo o processo descrito anteriormente, desde a seleção do *Product Backlog* até a entrega do produto final, destacando os participantes de cada uma das etapas do processo.

2.1.2.3. Equipe do Scrum

Times de desenvolvedores transformam o *Backlog* do Produto em incrementos de funcionalidades potencialmente entregáveis em cada *sprint*. Segundo Ken Schwaber (2009), times também são interdisciplinares: membros do time devem possuir todo o conhecimento necessário para criar um incremento no trabalho. Membros do time frequentemente possuem conhecimentos especializados, como programação, controle de qualidade, análise de negócios, arquitetura, projeto de interface de usuário ou projeto de banco de dados. No entanto, os conhecimentos que os membros do time devem compartilhar - isto é, a habilidade de pegar um requisito e transformá-lo em um produto utilizável - tendem a ser mais importantes que aqueles que são mais específicos.

Times também são auto-organizáveis. Ninguém - nem mesmo o *Scrum Master* - diz ao time como transformar o *Backlog* do Produto em incrementos de funcionalidades entregáveis. O time descobre por si só. Cada membro do time aplica sua especialidade a todos os problemas. A sinergia que resulta disso melhora a eficiência e eficácia geral do time como um todo.

Ken Schwaber (2009) também considera o tamanho ótimo para um time como sendo de sete pessoas mais ou menos duas pessoas. Quando há menos do que cinco membros em um time, há menor interação e, como resultado, há menor ganho de produtividade. Mais do que isso, o time poderá encontrar limitações de conhecimento durante partes do *sprint* e não ser capaz de entregar uma parte pronta do produto. Se há mais do que nove 9 membros, há simplesmente a necessidade de muita coordenação. Times grandes geram muita

complexidade para que um processo empírico consiga gerenciá-los. No entanto, temos encontrado alguns times bem-sucedidos que excederam os limites superior e inferior dessa faixa de tamanhos. O Product Owner e o *Scrum Master* não estão incluídos nessa conta, a menos que também sejam "porcos", o que na linguagem Scrum quer dizer: a menos que também vão trabalhar no desenvolvimento das tasks do *sprint*.

Scrum Master

Pode-se adicionar a todos os conceitos citados, um dos principais, que é o papel do *Scrum Master*. O *Scrum Master* deverá ser um facilitador, tanto no decorrer de todas as reuniões citadas, quanto na solução dos impedimentos. Durante as reuniões, o *Scrum Master* é responsável pelo bom andamento da comunicação do time e pelo cumprimento do limite de tempo combinado anteriormente. Se houver impedimentos que possam desviar o foco do time no cumprimento de suas tarefas, é o *Scrum Master* o responsável por manter o time sempre desbloqueado e apto a finalizar as tarefas iniciadas.

Product Owner

Schwaber (2009) também define o *Product Owner* como a única pessoa responsável pelo gerenciamento do *Backlog* do Produto e por garantir o valor do trabalho realizado pelo time. Essa pessoa mantém o *Backlog* do Produto e garante que ele está visível para todos. Todos sabem quais itens têm a maior prioridade, de forma que todos sabem em que se irá trabalhar. O *Product Owner* é uma pessoa, e não um comitê. Podem existir comitês que aconselhem ou influenciem essa pessoa, mas quem quiser mudar a prioridade de um item, terá que convencer a pessoa "*Product Owner"*. Empresas que adotam Scrum podem perceber que isso influencia seus métodos para definir prioridades e requisitos ao longo do tempo.

Para que o *Product Owner* obtenha sucesso, todos na organização precisam respeitar suas decisões. Ninguém tem permissão de dizer ao time para trabalhar em outro conjunto de prioridades, e os times não podem acatar outras decisões. As decisões do *Product Owner* são visíveis no conteúdo e na priorização do *Backlog* do Produto. Essa visibilidade requer que o *Product Owner* faça seu melhor, o que faz do papel de *Product Owner* um papel exigente e recompensador ao mesmo tempo.

Analista de Testes

Apesar do analista de testes fazer parte do time, faz-se necessário dar uma atenção especial a este papel visto que a metodologia que será proposta tem grande foco em seu trabalho. Podemos utilizar como uma das principais definições do papel de um testador nos *times ágeis*, uma citação de Liza Crispin em seu livro "*Agile Testing" (*Crispin e Gregory, 2009):

> "Nós definimos um testador ágil desta forma: um testador profissional que promove mudanças, colabora também com pessoas técnicas e de negócios e entende o conceito de utilização de testes para documentar requisitos e promover o desenvolvimento dirigido a testes. Testadores ágeis tendem a ter boas habilidades técnicas, saber como podem colaborar com o time para automatizar os testes, e também são experientes em testes exploratórios. Eles estão dispostos a aprender o que os clientes fazem para que possam compreender melhor as necessidades desses clientes".

Analisando esta citação e comparando com o trabalho de testes em metodologias tradicionais, pode-se perceber que muito da diferença entre os papéis se concentra em dois fatores: trabalho mais próximo do cliente para entender suas necessidades e também receber do cliente o *feedback* sobre o nível de qualidade exigido para os *sprints* que se seguirão. Sendo assim, a criação do planejamento de testes deve ser feita tendo em vista a sua possível utilização como requisitos de projeto. Nesse caso, uma adaptação do desenvolvimento dirigido a testes (TDD), parece adequada.

2.2. Testes de Software

Segundo Crispin e Gregory (2009), o mais interessante é discutir aspectos novos sobre as metodologias ágeis, mas admite-se que um projeto que usa esta metodologia ainda precisa ter maneiras de medir o progresso, de rastrear defeitos e planejar testes. Porém, estas tarefas precisam ser adaptadas para que, concomitantemente, os princípios ágeis sejam respeitados.

Em seu artigo, Jussi Auvinen (2005) deixa claro que:

"A grande diversidade de fases de testes necessárias coloca limitações sobre o processo de entrega em geral. Como consequência, é muito difícil ter um processo

verdadeiramente ágil no desenvolvimento de sistemas. Logo, ao invés de tentar implementar um processo completamente ágil, tem se utilizado um conjunto de práticas ágeis selecionadas como um piloto.".

Com esta observação o autor mostra que, até nos estudos mais reconhecidos, a dificuldade de se utilizar o processo ágil por completo, mantendo todo o processo de testes com qualidade, ainda existe.

Dando continuidade a estas pesquisas também descobriu-se que a preocupação com o processo de testes dentro das metodologias ágeis surgiu desde meados de 1999, juntamente com o início da utilização do Scrum no mundo todo. Essa preocupação pode ser vista em pesquisas de profissionais como Rex Black (2009), que já afirmava em seus estudos que os testes seriam um dos grandes desafios da ambientação com o pensamento ágil. Alguns dos desafios citados são: trabalhar com o grande volume de mudanças em alta velocidade, continuar sendo efetivo mesmo com *sprint*s de curta duração e fazer acontecer com menos: testar com a mesma qualidade mesmo com menor volume de dados de teste.

Discorrer sobre testes de software se faz necessário para o melhor entendimento desta pesquisa. Segundo Myers (2004), não se pode garantir que todo *software* funcione corretamente, sem a presença de defeitos. Logo, faz-se necessário que todo *software* passe pelo maior e mais seguro processo de verificação e validação possível, onde os Testes de *Software* estão inseridos. Myers também cita o teste de *software* como um processo que tem a intenção de encontrar defeitos nos artefatos de *software* (Myers,2004).

Como é quase sempre muito difícil executar uma grande quantidade de testes, acaba-se relacionando os resultados dos testes à qualidade do processo e dos profissionais envolvidos.

Testes Fumaça ou *Smoke* são utilizados com muita frequência para garantir, pelo menos, a existência de todas as funcionalidades projetadas para a tela de um sistema. Estes testes muitas vezes são automatizados e auxiliam a validação de baterias de testes inteiras, antes mesmo de todos os outros testes funcionais serem executados pelo time. Segundo Guimarães (2008), o *smoke test* pode ser caracterizado como uma estratégia de integração constante. Este deve ser suficientemente criterioso para que, se a construção passar, a equipe de teste possa assumir que ela é suficientemente estável para prosseguir com testes mais rigorosos. Como benefício, expõe antecipadamente problemas nas fases de

integração, dos testes sistêmicos e de aceitação, trazendo uma confiança de que as mudanças sofridas pelo software não afetaram negativamente as áreas abordadas pelo *smoke test.*

Outro tipo de teste, considerado funcional, é o Exploratório, muitas vezes chamado de *Walkthrough Testing*. Rubin (1994) considera relevante fazer testes exploratórios e testes de avaliação numa fase inicial de desenvolvimento, enquanto que a validação deve ser realizada numa fase mais adiantada do processo. O teste exploratório deve ser realizado quando se está definindo e concebendo um serviço ou recurso. Realizar este tipo de teste, para garantir que um *checklist* mínimo seja cumprido quando um produto é lançado em ambiente de produção ou homologação, é muito importante para garantir a qualidade da entrega. O teste exploratório não exige um planejamento acurado e dá liberdade ao profissional exercer a tarefa de teste segundo o seu conhecimento do sistema.

Segundo Sommerville (2007), o teste funcional envolve dois passos principais: identificar as funções que o *software* deve realizar e criar casos de teste capazes de checar se essas funções estão sendo corretamente realizadas pelo *software*. As funções que compõem o *software* são identificadas a partir de sua especificação. Assim, uma especificação bem elaborada e de acordo com os requisitos do usuário é essencial para esse tipo de teste.

O teste de sistema se inicia com o teste funcional que tem foco nas funcionalidades do *software*. Para o teste de sistema, a estrutura do *software* é ignorada e, devido a essa característica, os testes funcionais de sistemas são chamados Teste Caixa Preta, baseando-se em entradas controladas e suas respectivas saídas. Existem várias técnicas para selecionar os melhores valores e deixar o teste mais rico. Duas das principais técnicas utilizadas em teste caixa preta são: Partição de Equivalência e Valores Limites (Sommerville, 2007).

Segundo Sommerville (2007), os dados de entrada e os resultados de saída de um *software* pertencem a classes diferentes, cada uma delas com características comuns, tais como números positivos, números negativos, intervalos bem definidos. Nesses casos, o software se comporta de maneira comparável para todos os membros de uma mesma classe, definindo então uma partição de equivalência ou domínio. Identificando-se todas as partições de equivalência de um *software* e projetando-se casos de teste de modo que as

entradas e saídas se acomodem nessas partições, temos uma abordagem sistemática para o projeto de casos de teste. Para que se obtenha uma boa cobertura, deve-se selecionar pelo menos um caso de teste de cada partição, escolhendo entradas válidas e inválidas.

Para a escolha das entradas, a técnica de Valores Limite se configura em uma boa opção. Nesta técnica, cada valor limite válido ou inválido, deve ser representado em ao menos um caso de teste. Justifica-se o uso da técnica de Valores Limite, pelo fato que esses valores são, frequentemente, valores atípicos (por exemplo, valores nulos, zero, valores extremos de um domínio, entre outros) ignorados pelos desenvolvedores. Muitas vezes ocorrem falhas no *software* quando esses valores atípicos são processados (Sommerville, 2007).

Durante o desenvolvimento de um *software*, muitas modificações são realizadas por diversas razões, tais como, correção de erros, inclusão de novos requisitos, mudança de ambiente e melhoria de desempenho. As regras de negócio são dinâmicas e é inevitável que novos requisitos surjam, que requisitos existentes sejam alterados ou abandonados.

Após as mudanças terem sido realizadas, o sistema precisa ser testado para garantir que ele está se comportando como desejado e que as modificações não causaram nenhum impacto adverso nas partes não modificadas do software. Estes testes realizados após as mudanças são chamados de Testes de Regressão (Binder, 1999).

Teste de Regressão são sempre muito utilizados durante um desenvolvimento incremental e não é diferente para o Scrum, onde esse tipo de teste é constante quando um novo *Sprint* termina e um novo incremento é preparado para a entrega. Um teste de regressão é aplicado à nova versão, com o objetivo de verificar se esta ainda realiza, da mesma maneira, as funções que já estavam inseridas nas versões anteriores (Pfleeger, 2004).

A principal diferença entre os testes de regressão e as demais fases de testes realizadas durante o desenvolvimento do *software* é que se espera que já exista um conjunto de casos de testes disponível, que tenha sido aplicado à versão anterior do *software*. Um conjunto de testes de regressão é formado por testes da versão anterior do *software* que já “passaram”, isto é, cuja execução está de acordo com o esperado. Portanto, esse conjunto só se torna eficaz para revelar a presença de falhas quando modificações são realizadas. As falhas podem ser devidas a correções malfeitas ou que deveriam ter sido feitas, mas não o foram,

bem como incompatibilidades ou efeitos colaterais entre a parte modificada e a parte não modificada. Essas são as chamadas falhas de regressão (Binder, 1999).

A forma mais simples de realizar testes de regressão consiste em reaplicar todos os testes já aplicados ao programa. É a chamada abordagem *retestar tudo* (*retest all*). Sua aplicação é simples, pois não requer tempo de análise para escolha dos testes a serem reaplicados, uma vez que todos serão reaplicados. É também fácil de automatizar, pois muitas ferramentas de automação da execução dos testes são capazes de reexecutar os casos de testes tantas vezes quantas forem necessárias. O problema é que o número de testes pode se tornar muito grande, fazendo com que o tempo gasto para reaplicá-los se torne inaceitável na prática.

Uma alternativa consiste em usar uma *abordagem seletiva*, na qual um subconjunto dos testes é reaplicado. A dificuldade está na escolha desse subconjunto. Essa escolha se dá através da identificação de partes do código ou da especificação de um *software* que foram modificados e que precisam ser retestados. Uma abordagem, que permite a seleção e preserva a segurança de se testar as partes modificadas e por elas impactadas, é baseada nas dependências e foi denominada como "*firewall*" (Kung et al., 1996). Para se determinar um *firewall* utiliza-se um grafo de chamada que representa a interação entre os módulos (ou classes na orientação a objetos). O *firewall* limita os módulos do sistema que podem ser impactados por uma mudança no código ou na especificação de outro módulo com o qual os módulos que pertencem ao *firewall* têm uma interação que traga algum tipo de dependência.

Técnicas seletivas auxiliam na concisão dos testes de regressão, permitindo que estes sejam aplicados com maior frequência para que cumpram o papel de preservar a qualidade das novas versões a serem entregues ao final de cada *Sprint*.

Os testes de aceitação estão inseridos como parte do Scrum na grande maioria das publicações sobre o tema. O exemplo de Collins e Lobão (2010) descreve que considerou-se um resumo da importância dos testes de aceitação em Scrum. Colins e Lobão citam que os passos necessários são: planejar os casos de teste para as estórias do *Sprint* atual, baseando-se nos critérios de aceitação e dando ênfase para a criação de casos de teste dirigidos a falhas.

Podemos considerar como testes de aceitação toda e qualquer validação que foi planejada para ser levada em consideração ao definir se uma estória está pronta ou não. O conceito de testes de aceitação fica claro quando Cohn (2004) considera que, "Teste de Aceitação é o processo de verificar se as estórias foram desenvolvidas para que cada uma delas funcione exatamente da forma como o cliente esperava que funcionassem".

O teste de aceitação, levando em consideração os processos de testes tradicionais de um sistema de *software*, é a última etapa de teste a ser exercitada, exatamente antes do seu uso operacional. Quando os testes funcionais e as validações de requisitos não funcionais foram concluídos, os clientes devem avaliar o resultado para determinar se o sistema que foi construído realmente satisfaz as suas necessidades e expectativas (Pfleeger, 2004). Normalmente, o sistema é testado com dados fornecidos pelo cliente e não mais com os dados simulados de teste. São escritos, conduzidos e avaliados pelos clientes com a assistência dos desenvolvedores somente para esclarecimento de questões técnicas. O teste de aceitação é uma importante ferramenta para revelar omissões na definição de requisitos do sistema, como também revelar problemas de requisitos que façam com que o sistema não atenda às necessidades do usuário ou apresente um desempenho inaceitável (Sommerville, 2007).

Segundo Pfleeger (2004), três tipos de teste de aceitação podem ser utilizados para avaliar o sistema. O **teste de benchmark** avalia o sistema com base em padrões especificados. Os casos de teste representam as condições típicas sob as quais o sistema irá operar quando instalado e os usuários reais avaliam o desempenho para cada um dos casos de teste planejados. Nesse caso, os usuários responsáveis pelos testes devem estar familiarizados com os requisitos e devem conhecer o verdadeiro desempenho do sistema. O **teste piloto** instala o sistema e permite que os usuários façam uso do sistema como se ele estivesse realmente instalado no ambiente operacional. Toma-se como base o trabalho diário do sistema e, frequentemente, o cliente prepara uma lista onde são incorporadas funcionalidades do dia-a-dia de cada usuário. O **teste em paralelo** pode ser utilizado quando um novo sistema estiver substituindo um sistema já existente. Nesse caso, o novo sistema opera em paralelo à versão anterior e, gradualmente, os usuários vão se acostumando com o novo sistema. Essa transição gradual possibilita a comparação entre o

novo sistema e o antigo e permite que usuários mais cépticos adquiram confiança no novo sistema.

Depois dos testes de aceitação, normalmente, o cliente relata à equipe de desenvolvimento quais requisitos não foram satisfeitos, quais devem ser excluídos, revisados ou incluídos devido à necessidade de modificações. Essas modificações devem ser analisadas e suas consequências no projeto, na implementação e nos testes devem ser registradas. Normalmente, um acordo com o cliente é feito para resolver as pendências e possibilitar a entrega do sistema no prazo mais razoável possível.

2.3. Desenvolvimento Dirigido por Testes

Técnica amplamente utilizada nos últimos tempos, o TDD (do inglês "*Test Driven Development*" e em português "Desenvolvimento Dirigido por Testes"), segundo Beck (2002) pode ser dividida em 2 partes:

1. Nunca escrever uma única linha de código a menos que você tenha um caso de testes automatizado para ela;
2. Eliminar a duplicação.

Kent Beck resume os passos do TDD na repetição de um ciclo de desenvolvimento muito curto da seguinte forma: primeiro o desenvolvedor escreve um (inicialmente falho) caso de teste automatizado que define uma melhoria desejada ou funcionalidade nova, então produz a quantidade mínima de código para passar no teste e, finalmente, refatora o novo código para padrões aceitáveis (Beck, 2002).

Um estudo de 2005 descobriu que o uso de TDD significava escrever mais testes e, por sua vez, os programadores que escreveram mais testes tendiam a ser mais produtivos (Erdogmus e Morisio, 2005).

Programadores usando TDD puro em projetos relataram que, raramente, sentiram a necessidade de utilizar um depurador. Quando os testes falham inesperadamente, a reversão do código para a última versão em que todos os testes passaram pode, muitas vezes, ser mais produtivo que a depuração. Nesse caso, a utilização do sistema em conjunto com um sistema de controle de versão é essencial (Llopis, 2007).

Durante o tempo que o trabalho nos times (estudos de caso) foi acompanhado na empresa onde a aluna trabalha, verificou-se que o papel do analista de testes, nos times Scrum, foi muito relevante, visto que ele pôde ajudar de forma relevante na criação dos casos de teste, aproveitando seu conhecimento e visão crítica. Verificou-se também que muitos times utilizam esta visão para trabalhar com *pair programming,* quando desenvolvedor e testador trabalham em conjunto para a criação dos scripts automatizados.

Em muitos casos, a técnica pode ser inicialmente considerada mais demorada ou até mesmo mais cara, mas com o passar do tempo, pode-se constatar a criação de produtos de software com os seguintes benefícios:

- Menos uso de um depurador;
- Menor quantidade de defeitos;
- *Software* feito de maneira mais rápida e com melhor qualidade.

Apesar dos benefícios, o TDD apresenta limitações. Um exemplo de tais limitações é o fato da qualidade do código ser muito dependente da qualidade dos testes em si. Logo, se os testes não forem criados de forma criteriosa, o código também não será bom o suficiente.

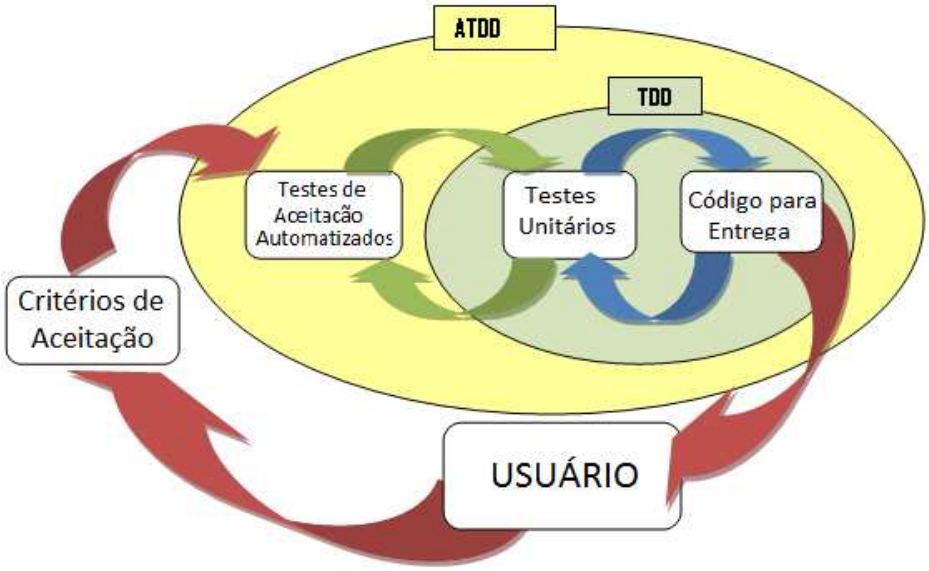


Figura 5 - Ciclo de TDD e execução das tarefas de desenvolvimento (adaptado de Bohl).

A Figura 5 mostra como deve ser feita a execução das tarefas dentro do ciclo do TDD. Considerando-se este ciclo, o papel do testador seria elaborar o plano de testes com todos

os testes que serão automatizados e integrá-los utilizando uma ferramenta de teste de código.

Segundo Bohl (2008), o processo de TDD e seu ciclo, representados na Figura 5, podem ser divididos em 7 partes:

1) O critério de aceitação nos dá um objetivo a ser alcançado quando escrevermos os casos de teste automatizados e assim, esses critérios devem ser escritos na forma de um conjunto de testes automatizados. Tais testes devem ser capazes de demonstrar ao cliente que o seu requisito vai ser alcançado. Idealmente, eles devem incluir o maior número de detalhes possível, sendo assim, deve ter o maior número possível de testes e de validações, para assegurar que cada critério seja coberto.
2) Os casos de testes nos dão a razão para criar o código (neste momento eles ainda falham). Estes casos podem ser escritos (todos) antes do código ou em paralelo, seguindo o processo.
3) O próximo passo é construir um *framework* vazio para incluir o código que será escrito. Este passo, em específico, desvia um pouco do TDD tradicional.
4) Para fazer com que todos os testes passem, precisa-se começar a escrever o código dentro do *framework* vazio. Nesta etapa é necessário aplicar os conceitos padrão do TDD e também escrever um caso de teste de unidade que vai falhar (inicialmente).
5) Neste ponto, tem-se um caso de teste de unidade criado, logo pode-se criar código o suficiente para atender aos testes planejados.
6) Repete-se o passo 4, refatora-se o código, e continua o processo descrito até o momento.
7) Quando todos os casos de testes automatizados passarem e o caso de teste criado para a unidade passar, a estória é finalizada. O ciclo continua a partir da próxima estória.

Considera-se como principais fatores positivos do TDD:

i. O fato do código ser mais enxuto, pois se implementa apenas o que é necessário, sendo mais fácil de atingir as metas de tempo e custo.

ii. Produzir um conjunto de casos de teste mais completo e que pode ser utilizado a qualquer tempo como casos de testes de regressão e identificar mais rapidamente as falhas que surgirem.
iii. Ter uma cobertura dos testes maximizada, pois todo código está relacionado a um caso de teste.

Alguns pontos negativos também devem ser considerados segundo Barros (2009):

i. O desenvolvimento é mais lento.
ii. Inicialmente se exige uma mudança de hábito por parte dos times.
iii. Quem cria os testes é o desenvolvedor, portanto ambos os testes e o código de produção podem ter os mesmos erros conceituais.

2.4. Desenvolvimento Dirigido a Testes de Aceitação

Segundo Hendrickson (2011), assim como o TDD, o *Acceptance Test Driven Development* (ATDD), também se baseia em criar testes antes do código e estes testes representam as expectativas do comportamento que o *software* deverá ter, segundo a visão do cliente (no caso deste estudo, do *Product Owner*). No ATDD, o time cria um ou mais testes de aceitação para a funcionalidade antes de começar a desenvolvê-la. Tipicamente, estes testes são discutidos e criados quando o time está trabalhando com o *Product Owner* para entender as estórias do *backlog*. O ATDD também é citado na Figura 5 e pode ser melhor descrito através do fluxo da Figura 6.

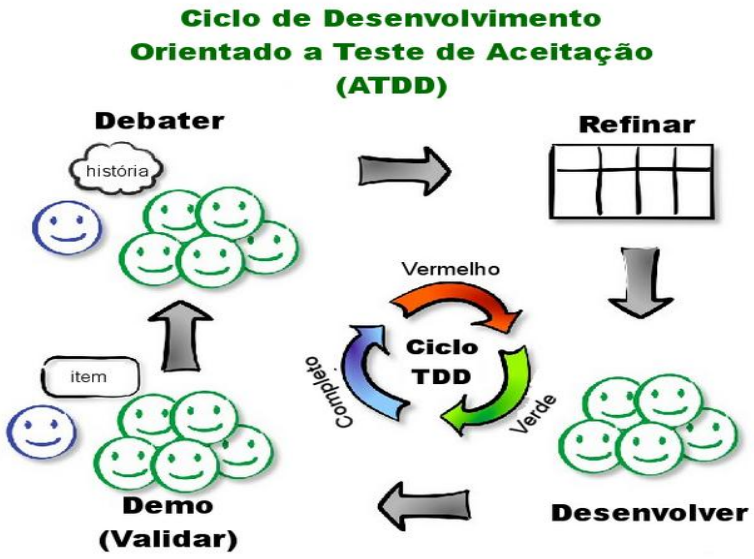


Figura 6 - Ciclo de ATDD (adaptado de Hendrickson, 2011)

O ciclo se inicia quando os requisitos (ou estórias) são analisadas e debate-se com os envolvidos os critérios para que os requisitos sejam considerados atendidos. Constrói-se, então, exemplos concretos que realmente traduzam as expectativas do cliente. Nesse momento, inicia o ciclo TDD, ou seja, cria-se os casos de testes baseados nos critérios de aceitação, cria-se o código para satisfazer esses critérios, aplica-se os testes unitários, liberando o código para a produção.

Depois de terem sido apresentados todos os principais conceitos relevantes e relacionados a esta pesquisa, tais como alguns conceitos relacionados a testes de *software*, metodologias ágeis e aos tipos de desenvolvimento dirigidos a testes e a testes de aceitação, tem-se o embasamento teórico suficiente para apresentar as propostas deste trabalho, como será feito no próximo capítulo (Capítulo 3).

3. PROPOSTA

A primeira preocupação deste trabalho foi de rever o formato utilizado na escrita das estórias (requisitos ágeis) e incluir um novo padrão em que casos de testes funcionais (e não funcionais) se integrassem a estes requisitos. Este novo formato difere dos formatos já propostos, que continham exclusivamente testes de aceitação. A este novo formato chamamos, neste trabalho, de *Test Backlog*.

Segundo Liza Crispin (2003), uma das contribuições mais importantes dos testadores ágeis hoje é ajudar o *Product Owner* na criação dos requisitos e articular cada estória em formato de casos de teste de aceitação que serão utilizadas durante a homologação contínua dos *sprint*s de desenvolvimento. Com base nestes casos de teste, o testador então, trabalha em conjunto com seus colegas de equipe para transformar tais requisitos em código executável.

Esta foi a base para a proposta de um novo formato de estórias que teria, além dos casos de teste de aceitação, também os casos de testes funcionais (e não funcionais quando pertinentes). Esse novo formato de estórias seria utilizado pelos testadores e por todo o time durante o desenvolvimento de código em si, (principalmente para times que utilizam o desenvolvimento dirigido a testes - TDD). Nossa hipótese é que esse novo formato facilitaria os testes funcionais no decorrer de um *sprint*, mas mais que isso, traria um entendimento mais claro a respeito de detalhes das funcionalidades a serem implementadas pelos desenvolvedores. Ao ler o plano de teste previsto, um desenvolvedor implementa o código para que cada caso de teste seja sucesso e isso traria um entendimento mais amplo sobre detalhes das funcionalidades do produto.

Esta proposta se assemelha ao TDD (*Test Driven Development*), o que faz muito sentido, visto que alguns conceitos foram mesmo baseados nesta técnica. Desenvolver o código com base em um requisito que foi criado já tendo os testes que serão aplicados em vista, tem muita relação com o TDD e por isso, acredita-se que uma adaptação da técnica deve se encaixar muito bem para metodologias ágeis em geral (incluindo o Scrum).

A Figura 7 ilustra o *design* criado para o exemplo que vamos utilizar afim de demonstrar a criação de uma *user story* no formato sugerido por este trabalho. A estória que

será criada terá como base a necessidade de criação de um *banner* (também chamado de *marquee* ou *carousel*) de conteúdo na página principal de um site de uma grande empresa da indústria farmacêutica norte-americana. Na imagem, o trabalho a ser desenvolvido e a funcionalidade a ser criada estão destacados por uma linha vermelha.

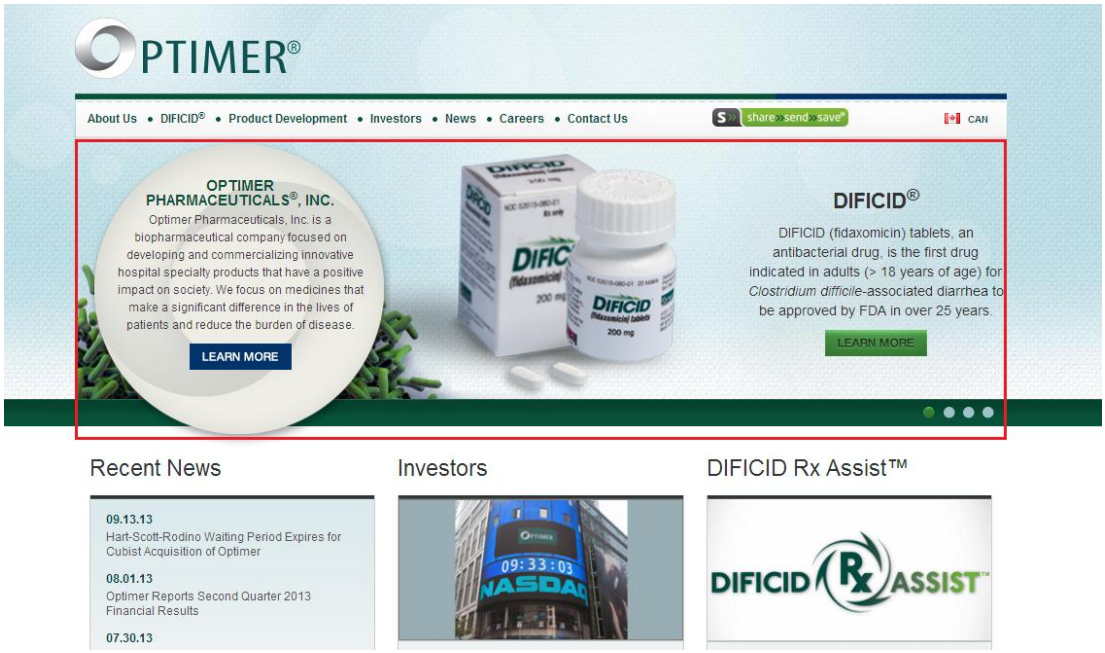


Figura 7 – *Banner* implementado na *Homepage* da empresa

Apresentado o que deveria ser construído nesta estória, temos a estória criada utilizando o modelo proposto neste projeto. Este formato de estória foi utilizado pelo time experimental durante o desenvolvimento dos *sprints* na construção de um hotsite. A Tabela 1 exemplifica o formato de *user story* proposto.

Tabela 1 – Exemplo de User Story

Como *product owner*, eu gostaria de ter um *banner* na página principal do meu site, que exibisse as informações principais, para que meu usuário final possa ter fácil acesso às informações e consequentemente, às principais páginas do site.
O círculo com um texto deve ser fixo e não deve ser afetado por nenhuma ação. - *Verificar todos os tamanhos possíveis fazendo a ação de reestimativa (resize) desde o menor tamanho de tela até o maior, o formato, o*

tamanho do círculo e o texto não devem se alterar - PASS

- *Verificar através de zoom se o texto ou layout é afetado. O layout deve se manter o mesmo. - PASS*
- *Verificar o layout e texto do círculo do site nos browsers em versões desktop e mobile. – FAIL (O texto e o layout para mobile são diferentes, logo para mobile o círculo some)*

O círculo deverá ter um botão que redirecione o usuário a página "About Us".

- *Verificar se clicando no botão o usuário é redirecionado a url correta: /contact-us. - PASS*
- *Verificar se ao clicar no botão a página é aberta em uma nova aba. – PASS*
- *Verificar se ao clicar no botão a página é aberta na mesma tela - FAIL*
- *Verificar se clicando com o botão direito do mouse e pedindo para que o conteúdo seja aberto em uma nova aba, a url correta é exibida. - PASS*

Todos os itens do banner devem redirecionar o usuário para uma página específica através de um botão "Learn More".

- *Verificar se clicando no botão "Learn More" do primeiro item do banner, o usuário é redirecionado para a página "/dificid". - PASS*
- *Verificar se clicando no botão "Learn More" do segundo item do banner, o usuário é redirecionado para a página "/careers". - PASS*
- *Verificar se clicando no botão "Learn More" do segundo item do banner, o usuário é redirecionado para a página "/about-us/investigator-initiated-studies". - PASS*
- *Verificar se clicando no botão "Learn More" do segundo item do banner, o usuário é redirecionado para a página /dificid/dificid-rx-assist". - PASS*
- *Verificar todos os casos de teste anteriores garantindo que os links corretos são abertos também ao clicar com o botão direito do mouse. - PASS*

O texto de cada banner deve ser editável e pode conter no máximo 120 caracteres.

- *Editar o texto do banner 1 e utilizar 119 caracteres. - PASS*

- *Editar o texto do banner 1 e utilizar 121 caracteres. - FAIL*
- *Editar o texto do banner 1 e utilizar 120 caracteres. - PASS*
- *Editar o texto utilizando caracteres especiais. - FAIL*
- *Editar o texto utilizando apenas caracteres alfanuméricos. - PASS*
- *Refazer os testes acima em todos os banners (2, 3 e 4).*

A imagem de background dos banners deve ser editável e pode ser do tipo .jpg ou .png.

- *Editar a imagem do banner 1 e utilizar um arquivo .jpg. – PASS*
- *Editar a imagem do banner 1 e utilizar um arquivo .png. - PASS*
- *Editar a imagem do banner 1 e utilizar um arquivo .gif. – FAIL*
- *Editar a imagem do banner 1 e utilizar um arquivo de um formato diferente – FAIL*
- *Refazer os testes acima em todos os banners (2, 3 e 4).*

Refazer todos os textos nos browsers IE8, IE9, FF (última versão), Chrome (última versão) e Safari 5.

A Tabela 1 mostra o formato proposto e que foi aplicado nos experimentos. Nota-se que a primeira linha da estória, seu título, segue o formato padrão do *Scrum* ("Como um [usuário papel], quero [meta], para que eu possa [motivo]"). A segunda parte da tabela trata-se dos requisitos da estória, estes sim, adaptados ao formato da proposta. O padrão da estória foi mantido no formato original no que se refere aos testes de aceitação e os casos de teste funcionais (ou não funcionais), parte adicionada pela proposta, foram destacados em itálico por motivos meramente ilustrativos. As linhas que detalham os requisitos de aceitação da estória *(acceptance criteria*), sublinhados na Tabela 4, foram criados basicamente pelo PO com ajuda do Analista de Testes. Já as linhas dos casos de testes, foram criadas pelo Analista de Testes e revisadas pelo PO.

Este formato em muito se assemelha com o BDD (do inglês *behavior-driven development*, traduzido livremente para "desenvolvimento guiado por comportamento") (North, 2006), que também foi concebido com base no TDD, este último anteriormente descrito no Capítulo de Embasamento Teórico. O diferencial desta proposta é basicamente combinar as necessidades dos profissionais do time (e até mesmo do cliente) em um único

documento. Este documento pode ser utilizado futuramente como: (i) base para o requisito da estória, (ii) plano de testes funcionais e (iii) base para o planejamento de testes de regressão (se necessário, por exemplo, em futuros times que recebem o produto como legado).

Outro fator muito relevante é o fato de se ter apenas um documento que necessita de manutenção ao longo do ciclo de vida do produto, diminuindo assim as chances de documentos desatualizados por falta de tempo e recursos livres para sua manutenção.

O formato proposto visa não só facilitar e melhorar a qualidade dos testes de software, mas também melhorar a previsibilidade do tamanho das estórias e da estimativa de esforço (size) dado pelo time durante o planejamento, visto que acredita-se ter melhores estimativas quando se tem maior detalhamento das estórias.

Tabela 2 – Modelo Padrão de Estória

Como um [usuário papel], quero [meta], para que eu possa [motivo]
Critério de Aceitação 1 *Caso de Teste 1 a n (referentes ao critério de aceitação número 1) – PASS/FAIL* Critério de Aceitação 2 • *Caso de Teste 1 a n (referentes ao critério de aceitação número 2) – PASS/FAIL* ... Critério de Aceitação N • *Caso de Teste 1 a n (referentes ao critério de aceitação número N) – PASS/FAIL*

A Tabela 2 apresenta um modelo padrão que exemplifica como deve ser a criação de uma estória, baseada no formato utilizado nos experimentos e no formato apresentado no exemplo da Tabela 1. Este formato pode então ser aplicado em qualquer estória, sendo apenas necessária a inclusão dos dados relevantes a cada estória. Vale salientar que, sempre que pertinente, técnicas clássicas de teste caixa preta devem ser utilizadas como, por exemplo, Partição de Equivalência, Análise de Valores Limite, Tabelas de Decisão entre

outras, cujas abordagens devem ser incorporadas nas linhas de casos de testes funcionais do modelo proposto.

Após a criação do *backlog* agora chamado de *Test Backlog*, o passo que se segue é o trabalho durante a reunião de planejamento de *sprint* (ou *Planning Meeting*).

Durante reuniões de planejamento e estimativa, testadores devem possuir uma visão que englobe necessidades de negócios (visão do cliente) e do usuário final, desenvolvimento e suporte. Eles são os profissionais mais indicados para levantar possíveis problemas com o máximo de antecedência e propor maneiras mais adequadas para que a equipe de desenvolvimento trate tais dificuldades. Esta proatividade pode ser demonstrada através de perguntas durante a reunião que levem a solução do número máximo de dúvidas técnicas da estória, visando tanto um melhor trabalho de estimativa da estória quanto um melhor entendimento do produto a ser construído, que vai colaborar para alcançar maior fluidez no desenvolvimento da estória ao longo do *Sprint*.

Outra preocupação relacionada ao período de desenvolvimento do *Sprint* e também ao período de execução das atividades de *software* foi a de adequar as atividades diárias de testes através de uma forma de testar mais incremental e que ainda assim combinasse com o formato da metodologia estudada (Scrum). A esta forma incremental de testar chamaremos de Testes Incrementais.

O papel do analista de testes sempre existiu desde muito antes das metodologias ágeis e do Scrum propriamente dito, mas fez-se necessário revisar as principais responsabilidades do analista de testes e todas as novas possibilidades que podem ser atingidas quando existe maior interação, desde o início do processo de desenvolvimento, entre o dono do produto (PO) e este profissional do time. Atividades como: (i) discussão sobre as necessidades de produto e criação de *backlog,* em conjunto com o PO, com base nestas necessidades já considerando possíveis casos de testes funcionais (e não funcionais); (ii) ajuda direta ao time durante o planejamento e execução do *sprint* e (iii) auxílio ao PO nos testes de validação são atividades atribuídas ao analista de teste com base no que foi proposto neste trabalho.

Os conceitos relacionados a profissionais de testes em metodologias ágeis foram adequados para o que acredita ser o papel de um testador em equipes ágeis.

Atualmente, os princípios das equipes de testes segundo Liza Crispin (2009) são:

- Fornecer *feedback* contínuo;
- Entregar valor ao cliente;
- Facilitar a comunicação cara a cara;
- Ter coragem;
- Manter as coisas simples;
- Praticar a melhoria continua;
- Responder rapidamente a mudanças;
- Ter auto-organização e foco nas pessoas.

Entregar valor ao cliente continuamente é algo que deve ser buscado (e atingido) nos projetos que envolvem desenvolvimento de *software*. Duas das maneiras de agregar valor ao negócio, relacionadas diretamente ao papel do analista de testes em times ágeis são: a revisão de estórias e a participação ativa na reunião de planejamento. Estes podem ser considerados os dois passos iniciais para transformar a atividade de testes em algo ainda mais incremental, deixando de considerar que os testes funcionais se iniciam alguns dias antes do término do *sprint*. Pela proposta deste trabalho, o profissional de qualidade começa sua atuação já no início do *sprint*, antecipando assim a detecção dos problemas (e dos defeitos) muito antes da etapa em que eram previstos os testes funcionais, isto é, trazendo a expertise do profissional de teste para apoiar a fase de criação do *product backlog*.

A experiência adquirida como *Scrum Master* em projetos ágeis também contribuiu para que se identificasse a necessidade desta adaptação, visto que em muitos casos, o cliente utiliza boa parte do seu tempo validando os testes de aceitação planejados como requisitos e acaba encontrando falhas ou até mesmo solicitando modificações no produto inicial, devido ao entendimento parcial adquirido pelo time sobre os requisitos. A ideia de conseguir adequação a real necessidade do cliente no menor tempo possível deve-se ao fato das metodologias ágeis acreditarem ser de extrema importância o valor gerado ao cliente. Isso vem ao encontro da proposta, uma vez que a participação do profissional de teste na definição das estórias acelera essa adequação, fazendo desse profissional um elo importante para um entendimento mais completo dessas necessidades.

Facilitar a comunicação entre a equipe também é um ponto de suma importância em projetos e times ágeis. Todas as discussões iniciadas relativas à maneira com que se

desenvolverá um código deve ter um mínimo de integrantes: o programador, o testador e o *product owner* (PO). O PO, idealmente, será representado pelo requisito (estórias) e até mesmo pelo testador em si, sempre que o requisito não esteja claro entre todas as partes. Por esse motivo, faz-se necessário que o testador seja sempre envolvido ao longo do *sprint* nestas discussões. Este pode ser considerado como uma terceira meta para tornar a atividade de testes diluída ao longo do *sprint*, enfim, incremental.

O membro da equipe responsável pelas atividades de teste deve fazer o *follow-up* sobre o desenvolvimento da estória e, sempre que possível, fornecer um *feedback* completo para os outros membros da equipe sobre a forma como o trabalho está sendo desenvolvido a partir do ponto de vista da qualidade e, em particular, prevendo quaisquer possíveis defeitos. Todas as discrepâncias relacionadas com a interpretação do requisito devem ser apontadas para o Scrum Master e para o PO e resolvidas o mais rápido possível, uma vez que qualquer dúvida não elucidada pode levar a defeitos.

A forma de comunicação mais participativa já auxilia naturalmente a levar o testador para um próximo nível. Neste novo nível, atividades como prever, diagnosticar e prevenir possíveis defeitos o mais cedo possível seriam as mais relevantes, visto que tais defeitos, se reportados apenas no momento de testes funcionais (isto é, no final do *sprint*), podem aportar mais custos para o projeto como um todo. A equipe deve sempre ter em mente que os testes constituem um processo incremental e que defeitos caros são aqueles que demoram mais tempo para ser encontrados. Após este passo, os testes funcionais executados após a finalização de cada estória se iniciam e as atividades usuais já vivenciadas em times ágeis são realizadas.

A Figura 8 foi adaptada com a proposta deste trabalho (que inclui testes mais incrementais e o novo formato de *Test Backlog*). O especialista em testes (um membro do time) foi envolvido juntamente ao PO na elicitação de requisitos e na criação das *user stories*. A lista priorizada dos requisitos (ao qual normalmente chamamos de *Product Backlog*), nesta versão, inclui casos de testes e foi renomeada para *Test Backlog*.

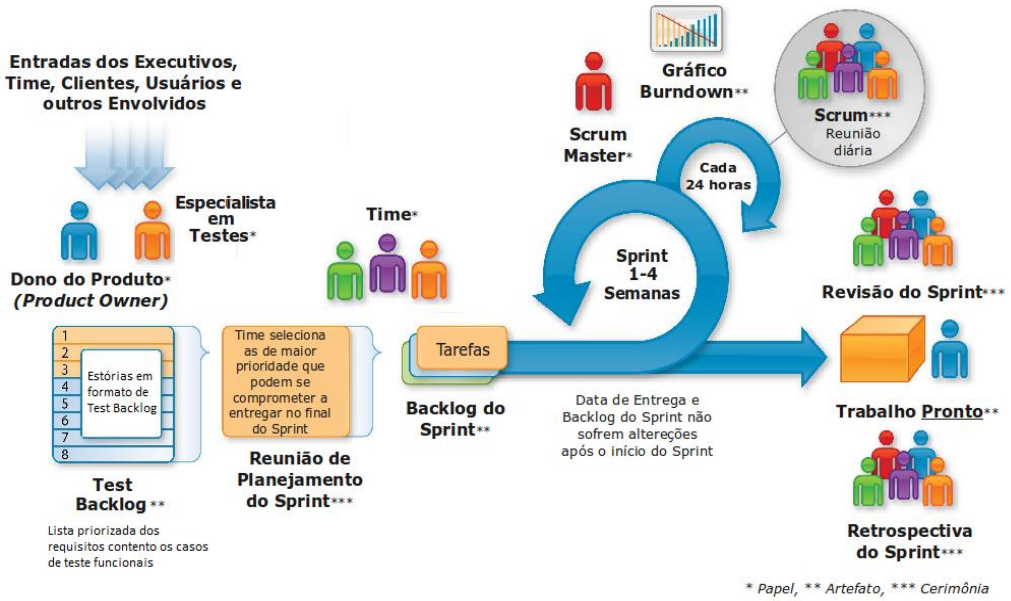


Figura 8: Desenvolvimento ao longo do *sprint* (adaptado com as propostas deste trabalho)

É importante citar que o trabalho dos testes das últimas tarefas do Sprint e do reteste dos último defeitos é feito ao mesmo tempo que o trabalho de revisão das estórias do próximo Sprint juntamente ao *Product Owner*. Isso pode configurar um gargalo nas atividades do profissional de teste que, por outro lado, é minimizado pela antecipação das tarefas de teste que ocorreram durante todo o *sprint*. Acredita-se que o tempo poupado com a antecipação das tarefas de teste, em contraponto ao tempo necessário para cumprir todas as etapas do teste ao final do *sprint*, como é aplicado atualmente, possa equivaler ao tempo necessário para a revisão das estórias que precede o início do *sprint*, eliminando ou minimizando o impacto de um possível gargalo relacionado ao tempo exigido do profissional de teste.

Tabela 3 – Atividades de Testes ao longo do *Sprint*

Atividade	**Passos**
Planejamento de Testes	Antes da reunião de planejamento do *sprint*, na escrita das estórias, o analista de testes deverá: • **Analisar / Validar os critérios de aceitação juntamente com o PO;**

	• **Planejar os cenários de teste e incluí-los na escrita das estórias com a revisão do PO;** • Produzir/Atualizar scripts para testes automatizados (quando aplicável);
Preparação para a execução dos testes	Antes de iniciar a execução dos testes, o analista de testes deve: • Criar/melhorar a base de dados de teste para atender às exigências da estória a ser testada; • **Criar dados de teste para os cenários previstos (estes também deverão ser incluídos na estória);**
Testes Incrementais	**Durante todo o desenvolvimento da estória:** • **O membro da equipe responsável pelas atividades de teste deve fazer o *follow-up* sobre o desenvolvimento da estória e, sempre que possível, fornecer um *feedback* completo para os outros membros da equipe sobre a forma como o trabalho está sendo desenvolvido a partir do ponto de vista da qualidade e, em particular, prevendo quaisquer possíveis defeitos;** • **Todas as discrepâncias relacionadas com a interpretação do requisito devem ser apontadas para o Scrum Master e para o PO e resolvidas o mais rápido possível, uma vez que dúvida não esclarecida pode levar a defeitos;** • **A equipe deve sempre ter em mente que os testes constituem um processo incremental e que os defeitos caros são aqueles que demoram mais tempo para ser encontrados.**
Executar os testes	A execução dos testes compreende os passos abaixo: • Obter a versão de compilação com a estória liberada para testes; • Instalar a versão lançada no ambiente de teste; • Executar casos de teste previamente planejados e incluídos nas estórias; • Verificar se os resultados obtidos satisfazem os critérios de aceitação e os testes funcionais;

	• Abrir defeitos para os casos em que os resultados não satisfazem os casos de testes funcionais.
Reportar Defeitos	• Defeitos devem ser abertos na ferramenta padrão, usada para fazer o *follow-up* de tarefas; • Informação que é relevante para a reprodução de defeito deve ser anexada ao defeito;
Monitorar a correção de defeitos	Os defeitos abertos pela equipe de desenvolvimento devem ser monitorados para que: • Novos defeitos sejam abertos/reabertos para sanar inconsistências em defeitos que não foram resolvidos ou que foram apenas parcialmente resolvidos; • Defeitos corrigidos sejam alterados para o status concluído (*done*); • Tarefas (*tasks*) que estão livres de defeitos possam ser concluídas (*done*).
Fornecer *feedback*	O membro da equipe de desenvolvimento responsável pela execução dos testes deve: • Proporcionar à equipe de desenvolvimento *feedback* sobre a qualidade da estória; • Fornecer opinião sobre incluir ou não a estória na reunião de demonstração para o cliente (*Demo meeting*);

Este capítulo apresentou os conceitos da proposta deste trabalho de mestrado. O próximo capítulo exibirá a metodologia utilizada nos experimentos conduzidos neste trabalho de mestrado.

4. METODOLOGIA DO EXPERIMENTO

O Experimento, o Estudo de caso e o S*urvey* são as três principais formas primárias de se conduzir uma investigação experimental (Kerlinger e Taylor, 1979). Este trabalho utilizou o Experimento. Segundo Kerlinger e Taylor (1979), um experimento é um tipo de pesquisa científica no qual o pesquisador manipula e controla uma ou mais variáveis independentes e observa a variação nas variáveis dependentes, concomitantemente à manipulação das variáveis independentes. Em geral, essa forma traz um maior controle sobre a execução, medição, investigação e facilidade de repetição. Porém, o custo e o risco são altos (Travassos et al., 2002).

O pesquisador tem dois objetivos: extrair conclusões válidas sobre os efeitos de variáveis independentes sobre o grupo em estudo e também fazer generalizações para uma população maior. Quando há constatação de que o tratamento foi a causa real do efeito, o experimento tem validade interna. Quando a relação de causa e efeito encontrada no experimento pode ser generalizada para toda a população, então o mesmo tem validade externa (Malhotra, 2011).

Dadas as fases das quais pode se compor um experimento, tais experimentos poderão também ser classificados entre si. Segundo Malhotra (2011), pode-se classificar os projetos experimentais como:

Projetos Pré-experimentais: Os projetos pré-experimentais são estudos exploratórios e não têm quase nenhum controle sobre as variáveis estranhas. Além da seleção das unidades sob teste não ser aleatória, são projetos mais simples e com pouca confiabilidade. Os planejamentos desse tipo utilizam geralmente amostras pequenas.

Projetos Experimentais Verdadeiros: Os projetos experimentais verdadeiros são aqueles que utilizam o recurso de aleatorização para selecionar as unidades de teste, e por isso são mais confiáveis que os pré-experimentais para fazer inferências causais. Entretanto, a aleatorização é difícil de ser aplicada quando a amostra é pequena. Outra característica importante dos projetos experimentais verdadeiros, que os tornam mais válidos, é a presença de um ou mais grupos de controle além dos grupos experimentais.

Projetos Quase Experimentais: Os projetos quase experimentais não utilizam a atribuição aleatória como os experimentais verdadeiros, mas por outro lado, oferecem mais informações que os projetos pré-experimentais. Geralmente, são projetos com várias medições ao longo do tempo.

Projetos estatísticos: Os projetos estatísticos diferem dos demais, pois permitem controles estatísticos e análise de variáveis externas, mesmo sendo afetados pelas variáveis estranhas. Embora exijam uma elaboração mais complexa, os projetos estatísticos são mais ricos em informações e permitem medição de efeitos de mais de uma variável simultaneamente.

Este trabalho pode ser classificado como um Projeto Quase Experimental, isso porque tanto a distribuição dos recursos dentro dos times quanto o trabalho que cada time deveria executar dentro dos *sprints* não foram aleatórios.

Independentemente dessa classificação, cada experimento é composto por várias fases. Segundo Mattar (2005), estas fases são:

1. Formulação

Formulação, definição ou constatação de um problema de pesquisa, que requeira experimentação.

- Definição dos objetivos da pesquisa e formulação da hipótese causal.
- Definição do tratamento e das variáveis da pesquisa (dependentes e independentes).
- Determinação de dados necessários e suas fontes.

2. Metodologia

- Escolha do tipo de projeto experimental e descrição da metodologia do experimento.
- Identificar quais variáveis estranhas não serão controladas com a configuração do projeto experimental.
- Determinação da população, do tamanho de amostra e processo de amostragem.
- Ajuste de orçamento e duração do projeto (geralmente são os limitadores do teste).

3. Execução

- Monitoramento do processo com cuidado para garantir o funcionamento do planejamento e metodologia. Erros nesta fase podem destruir a validade experimental.

4. Conclusão

- Análise dos dados, através de métodos estatísticos para que as conclusões não sejam subjetivas.
- A partir da análise dos dados o pesquisador precisa chegar a conclusões práticas e recomendar uma ação.
- A comunicação dos resultados é feita, geralmente, através de recursos gráficos.

Analisando o contexto deste trabalho, a primeira etapa desenvolvida foi a Formulação. Durante este período foram definidos os objetivos da pesquisa apresentados no Capítulo de Introdução. Também foram selecionados os dados que seriam capturados e quais projetos seriam objetos de estudo. Foram escolhidos dois times ágeis de uma mesma empresa que estavam, no momento da pesquisa em campo, trabalhando no desenvolvimento de produtos semelhantes. Mais detalhes serão descritos no subcapítulo 4.1.

A etapa seguinte foi a de Metodologia. Durante esta etapa foram pesquisados e definidos os dados apresentados neste capítulo, especialmente os dados apresentados na Tabela 6, relativos às possíveis ameaças ao tipo de experimento que seria desenvolvido.

Logo após a fase de Metodologia, iniciou-se a fase de Execução. Esta fase é composta, como o próprio nome diz, pela execução do experimento em si. O desenvolvimento dos *product backlogs* através de vários *sprints* compõe o período de execução. Durante este período foram recolhidos todos os dados pré-definidos durante a Formulação.

A última fase foi a de Conclusão. Todas as definições e conclusões efetuadas nesta fase poderão ser melhor analisadas nos capítulos finais: Capítulo 5: Resultados e Discussões e Capítulo 7: Conclusões.

Em um experimento o pesquisador possui controle sobre a quantidade e o tipo de grupos de participantes, qual grupo fará o quê, em que momento haverá a intervenção e, em

alguns casos, quem participará de cada grupo (Wainer, 2007). Na etapa de Definição deste projeto foram escolhidos os grupos que seriam utilizados como experimento - que são apresentados na subseção 4.1 - e os produtos que seriam desenvolvidos – que são apresentados na subseção 4.2.

4.1.Descrição dos Times

Os times utilizados como estudo de caso neste trabalho fazem parte da empresa onde a aluna trabalha. Todos estes times são montados considerando uma distribuição igualitária quanto a quantidade de recursos e a *expertise* de seus membros. Este nivelamento é pensado tanto para o conhecimento técnico (1 desenvolvedor, 1 web design, 1 testador, 1 líder técnico, 1 *Scrum Master*) quanto na senioridade (1 sênior, 1 pleno, 3 *juniors*).

Além disso, destaca-se que todos os membros tinham, na época da execução do experimento, tempo de empresa maior do que 1 ano, treinamento pela própria empresa na metodologia Scrum e, a maioria já tinha alcançado certificação na área.

Outros cuidados específicos foram tomados para que a validade do experimento (mais detalhes na subseção 4.3) não fosse afetada. Dois profissionais (Scrum Master e Líder Técnico) foram mantidos em ambos os times, para que o controle das atividades, segundo as propostas deste trabalho, fosse mantido e também para que a qualidade técnica e expertise fosse exatamente a mesma (mesmos recursos). Todos os profissionais foram previamente treinados tanto com relação ao conhecimento da metodologia Scrum quanto na tecnologia de desenvolvimento, Drupal. Os times também tinham prévia experiência em outros projetos muito similares (formato de produto parecido e mesmo cliente).

4.2. Descrição dos Produtos

Além das escolhas de times, outro fato não aleatório foi a escolha dos *product backlogs* que seriam desenvolvidos por tais times. Para que a comparação fosse possível, escolheu-se projetos muito similares. Ambos os times construíram websites para produtos farmacêuticos da mesma empresa, com *layout* muito similar e número de páginas semelhantes. Um exemplo desse *layout* pode ser observado na Figura 7. A Tabela 4 mostra a estrutura que foi criada para cada site e a similaridade de ambos os projetos. O Time 1, time experimental que trabalhou com base na proposta, teve que desenvolver apenas um

website, enquanto o segundo time, trabalhou inicialmente em um site (*Website* 1) como grupo de controle (sem utilizar as diretrizes da proposta) e depois, trabalhou em um segundo *website* (*Website* 2), agora com base na proposta deste trabalho, para fins de confirmação de resultados. O objetivo desse último *Sprint* era observar se o mesmo time, desenvolvendo um produto bastante similar e com alteração significativa apenas na metodologia, teria resultados diferentes e mais aderentes aos requisitos do cliente.

Tabela 4 – Estrutura e Macro Estórias (Times 1 e 2)

Website 1 – Time 1	*Website* 1 – Time 2	*Website* 2 – Time 2
1 Formulário (Contato)	1 Formulário (Contato)	1 Formulário (Contato)
14 Páginas conteúdo básico	15 Páginas conteúdo básico	13 Páginas conteúdo básico
1 Home (Banner e *Vídeo)*	*1 Home (Banner* e *Vídeo)*	*1 Home (Banner* e *Vídeo)*
1 Sitemap	*1 Sitemap*	*1 Sitemap*
3 páginas simples (Termos de Uso, Política Privada e Aviso Legal)	2 páginas simples (Termos de Uso e Política Privada)	3 páginas simples (Termos de Uso, Política Privada e ISI - *Important Safety Information*)
Área Administração + Inserção de Conteúdo	Área Administração + Inserção de Conteúdo	Área Administração + Inserção de Conteúdo
Criação de 2 papéis: *Admin* + *Content Admin*	Criação de 2 papéis: *Admin* + *Content Admin*	Criação de 2 papéis: *Admin* + *Content Admin*

Ressalta-se o fato de ambos os produtos terem sido desenvolvidos utilizando a mesma tecnologia, *Drupal* (tecnologia baseada em *PHP*), com alguns recursos de *JavaScript*. Não houve diferenças e/ou utilização de outras tecnologias em nenhum dos projetos. Este fator também colaborou para o equilíbrio do experimento.

4.3. Definição do Experimento

O objetivo deste experimento é de analisar a efetividade das propostas para melhor definição do papel do analista de testes e da escrita das *user stories* (*Test Backlog*) em times que trabalham com desenvolvimento Scrum.

O experimento foi executado em uma empresa multinacional brasileira, cuja matriz é situada em Campinas, onde a aluna trabalha há pouco mais de 5 anos, sendo os primeiros 2 anos como especialista em testes e os últimos 3 anos como *Scrum Master* em projetos internacionais de grande porte.

A duração dos *sprint*s foi de 15 dias, todos com a mesma duração. Os times utilizaram uma ferramenta que já era padrão da empresa tanto para apontamento de tarefas como para defeitos e validação de *software*. A ferramenta utilizada foi o JiraCloud, da Atlassian (2013). Através do uso desta ferramenta foi possível coletar os dados utilizados e analisados.

Tabela 5: Objetivos do Experimento

Objetivo	Analisar a viabilidade da proposta apresentada nesta pesquisa: Testes Incrementais e *Test Backlog* em times Scrum.
Contexto	Objeto: Criação de sites para produtos de uma grande empresa do ramo farmacêutico utilizando a tecnologia Php/Drupal. Sujeitos: Desenvolvedores especializados em Drupal com experiência em projetos similares dentro da empresa.
Hipótese Nula	Uma maior quantidade de defeitos em homologação. O mesmo número ou uma maior quantidade de defeitos em produção. Um maior número de *block*s do Time. Um maior número de horas extras.
Fator Principal	Propostas: Teste Incremental e Test *Backlog*.

Outros Fatores	Experiência dos times em Scrum e se possível, com mais de 3 anos de experiência em PHP.
Variáveis Dependentes	Registro de Defeitos. Registro de Comentários de *Retrospective*.

4.4. Validação do Experimento

Outra preocupação importante com trabalhos que utilizam pesquisa experimental é a análise da validade do experimento. Nesta análise, segundo Mafra e Travassos (2006), especificamente na etapa de planejamento da investigação experimental, o pesquisador deve analisar e definir as ameaças à validade do seu experimento segundo quatro tipos de validade: interna, externa, de construção e de conclusão.

1. Validade Interna

A validade interna, para Travassos et al. (2002), define se o relacionamento observado entre o tratamento e o resultado é causal, e não é resultado da influência de outro fator – não controlado ou medido.

Para Wainer (2007), as ameaças mais comuns a este tipo de validade são:

- Instrumentação: a diferença nos resultados é consequência de uma medição incorreta ou instrumento inadequado;
- Testagem: o desenho faz com que os participantes aprendam com seus próprios erros; está relacionada à sequência da coleta dos resultados versus o momento da intervenção, por exemplo;
- Maturação: os sujeitos podem tornar-se mais capazes ou ficarem desmotivados com o passar do tempo;
- História: possibilidade de que o resultado do processo é consequência de um evento externo ao experimento.

2. Validade Externa

A validade externa, conforme Travassos et al. (2002), define condições que limitam a habilidade de generalizar os resultados de um experimento para a prática industrial; as ameaças mais comuns a este tipo de validade são:

a) Participantes: selecionar participantes que não possuem relação ou não refletem o comportamento da população, ou, trabalhar com uma amostra não representativa quantitativa e qualitativamente da população;

b) Tempo: por exemplo, impor ou suprimir restrições de tempo;

c) Configuração do experimento: realizar o experimento em um ambiente muito diferente do ideal, ou adotar matérias ou instrumentos distantes da realidade.

3. Validade de Construção

A validade de construção, para Travassos et al. (2002), considera os relacionamentos entre a teoria e a observação, ou seja, se o tratamento reflete bem a causa e o resultado reflete bem o efeito. As ameaças mais comuns a este tipo de validade estão relacionadas com:

a) Projeto do experimento: em geral, má definição da base teórica ou da definição do processo de experimentação;

b) Fatores humanos (ou sociais): por exemplo, os participantes podem basear seus comportamentos nas hipóteses de pesquisa ou eles podem estar envolvidos em outros experimentos.

4. Validade de Conclusão

A validade de conclusão está, segundo Travassos et al. (2002), relacionada à habilidade de chegar a uma conclusão correta a respeito dos relacionamentos entre o tratamento e o resultado do experimento.

A eventual existência de uma determinada ameaça que não foi previamente definida e devidamente tratada pode comprometer os resultados do experimento, e com isso invalidar o estudo como um todo (Mafra e Travassos, 2006). Logo, decidiu-se analisar o que se considerou possíveis ameaças para a esta experimentação. A Tabela 6 cita as situações das ameaças levantadas durante o planejamento do experimento realizado ao longo deste projeto.

Tabela 6: Ameaças à pesquisa experimental

Ameaça (Descrição)	Time Susceptível	Plano de Contingência
Validade Interna		
Maturação	Times 1 e 2	Os times foram escolhidos por terem perfis parecidos. Têm a mesma data de início e final de projeto e receberam o requisito inicial no mesmo dia. Considera-se as chances de adquirirem experiência, tanto técnica quanto de negócio, muito parecidas.
Validade Externa		
Configuração do experimento	Times 1 e 2	Por mais parecidos que os requisitos sejam, mesmo assim têm algumas diferenças. O plano foi escolher os 2 times com o requisito inicial de projeto o mais semelhante possível para minimizar este risco.
Validade de Conclusão		
Confiabilidade das medidas	Times 1 e 2	Todas as medidas foram feitas pelos times e inseridas dentro da ferramenta utilizada na empresa (Jira). Para minimizar o risco dos times esquecerem-se de inserir as medidas corretas, a aluna fez o acompanhamento deste times (visto que ela era a *Scrum Master* responsável por ambos os times). Porém, a interação da aluna se limitou a conferir se os dados tinham sido registrados e alertar o time caso não tivesse sido.

Este capítulo descreveu a metodologia utilizada durante o projeto nos dois experimentos selecionados. O próximo capítulo (Capítulo 5) detalha os resultados obtidos através destes experimentos.

5. RESULTADOS e DISCUSSÕES

Baseado no estudo descrito no capítulo de Metodologia (Capítulo 4), foram desenvolvidos experimentos envolvendo dois times distintos. O Time 2, considerado de controle, fez o desenvolvimento usando os padrões já utilizados pela empresa e com os quais o time já vinha trabalhando há mais de um ano. O Time 1, considerado o experimental, recebeu um treinamento de 4 horas para o uso dos novos padrões propostos por este trabalho. A mesma quantidade de horas de treinamento foi feita com o Time 2 antes da aplicação do *sprint* 5, onde a metodologia proposta foi aplicada pela primeira vez nesse time.

Devido a complexidade e demanda de recursos humanos, o experimento só foi realizado uma vez, portanto não foi possível fazer uma análise estatística dos dados coletados. Desta forma, foi efetuada uma análise quantitativa comparativa entre o grupo de controle (Time 2) e o grupo experimental (Time 1).

A empresa onde a aluna trabalha já possuía sua forma particular de balancear os times para que todos tenham um nível de conhecimento parecido e eficiência (*velocity)* similar. Isso permitiu a comparação dos resultados de forma mais segura. Outro fator importante foi o formato do produto a ser construído. Todos os projetos eram relacionados a sites simples de conteúdo estático com 15 páginas internas, que tinham como objetivo promover produtos de uma grande empresa farmacêutica internacional.

A etapa de Formulação consistiu na escolha das informações que seriam relevantes ao método proposto, estas são:

- Número de Defeitos (gerados em todas as fases do processo);
- Tipos de Defeitos;
- Número de linhas de Código;
- Tempo de *sprint* (desenvolvimento);
- Número de *blocks*;
- Número de horas de teste;
- Comentários das reuniões de *Retrospective* e;
- Número de horas extras realizadas por ambos os times.

As informações selecionadas para serem coletadas, utilizando-se da Avaliação, na fase de conclusão, foram utilizadas como base para a criação de tabelas e gráficos para análise de resultados. A etapa de Execução durou o tempo de 4 *sprints* para o Time 1 (que utilizou a metodologia proposta) e 5 *sprints* para o Time 2, já que foi conduzido um último *Sprint* para confirmação dos resultados utilizando-se da metodologia proposta, até então não implementada neste time.

Por motivo de sigilo empresarial, as estórias não puderam ser incluídas nesse trabalho. Estórias que não traziam muitos detalhes sigilosos, foram incluídas no Anexo 1 como exemplo dos documentos que foram utilizados durante o experimento.

A Tabela 7 mostra o número de defeitos abertos durante todas as etapas: desenvolvimento, UAT (do inglês *User Aceptance Testing*, ou seja, Teste de Aceitação do Usuário) e Produção. Estes dados foram coletados para verificar se o fato dos testes terem sido mais incrementais faria com que o número de defeitos fosse cada vez menores em UAT. O número esperado para defeitos em produção é sempre o mais perto de zero possível.

Tabela 7 – Número de Defeitos ao longo do projeto nos Times 1 e 2

Time 1	***Sprint* 1 *Test Backlog***	***Sprint* 2 *Test Backlog***	***Sprint* 3 *Test Backlog***	***Sprint* 4 *Test Backlog***	
N° defeitos em desenvolvimento	25	32	33	35	
N° defeitos em UAT	5	2	1	0	
N° defeitos em produção	0	0	0	0	
Time 2	***Sprint* 1**	***Sprint* 2**	***Sprint* 3**	***Sprint* 4**	***Sprint* 5 *Test Backlog***
N° defeitos em desenvolvimento	17	23	20	18	31
N° de defeitos em UAT	14	8	10	7	2
N° de defeitos em produção	4	2	0	1	0

Nota-se, nos resultados da Tabela 7, que durante o desenvolvimento dos *sprints* do Time 1, foram encontrados 125 defeitos distribuídos nos quatro *Sprints,* contra 78 defeitos nos primeiros quatro *Sprints* do Time2. Isso representa uma taxa de aumento de detecção de falhas de 60% no Time1, com forte indício de que a metodologia colaborou para a descoberta de defeitos o mais cedo possível e que a qualidade dos testes foi maior quando utilizou-se um planejamento de testes mais completo. O Time 2, utilizou apenas os testes exploratórios com base nos testes de aceitação definidos pelo PO.

Analisando a última coluna da Tabela 7, observa-se um aumento expressivo de detecção de falhas na etapa do desenvolvimento pelo Time 2. O aumento, considerando o melhor caso dos *sprints* anteriores do mesmo time, foi de aproximadamente 35%. Vale ressaltar que a única mudança em relação aos sprints anteriores foi a aplicação da metodologia (os membros do time foram todos conservados), o que traz fortes indícios de que o impacto positivo foi em decorrência da metodologia proposta.

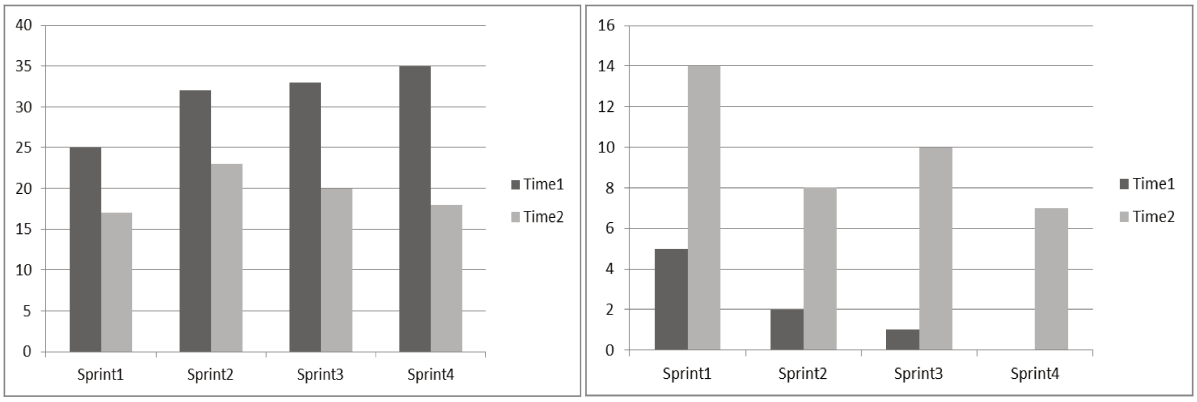
Já na fase dos testes de aceitação (UAT) o PO revelou 39 defeitos no produto entregue pelo Time 2, contra 8 defeitos encontrados no produto entregue pelo Time1, representando um número de defeitos quase cinco vezes maior no time de controle. Dessa forma, a primeira hipótese nula relacionada na Tabela 5 foi rejeitada. Nesse caso, há indícios de que defeitos que poderiam ter sido encontrados durante o desenvolvimento foram encontrados pelo cliente durante UAT.

Novamente analisando a última coluna da Tabela 7, observa-se uma diminuição de aproximadamente 70%, considerando o pior caso, do número de defeitos em UAT no Time 2. Como a única mudança foi a metodologia, há fortes indícios de que essa diminuição foi em consequência da aplicação da metodologia.

Sobre os defeitos de produção, ambos os times tiveram um número bem baixo, mas enquanto o produto desenvolvido pelo Time 2 apresentou 7 defeitos, o Time 1 não apresentou nenhum defeito em produção até o fechamento desta pesquisa, demonstrando maior qualidade atingida no desenvolvimento. Dessa forma, a segunda hipótese nula relacionada na Tabela 5 foi rejeitada.

A Figura 9 apresenta os defeitos revelados em fase de desenvolvimento (a) e em fase dos testes de aceitação (b). Pode-se observar o mesmo comportamento em todos os *Sprints*, tanto no desenvolvimento, em que o Time 1 sempre revela mais defeitos, quanto no teste de

aceitação, em que o produto do Time 1 sempre apresenta um menor número de defeitos revelados.



a) Desenvolvimento b) Teste de Aceitação

Figura 9: Defeitos Revelados

A Tabela 8 mostra os tipos de defeitos encontrados. A motivação para a escolha destes dados foi verificar se o tipo de defeito é mais complexo ou mais simples de acordo com a qualidade do requisito. Nesta tabela não foram incluídos os defeitos do *Sprint* 5 (Time 2) para que a comparação fosse mais direta e não houvesse dificuldade em distinguir os dados com e sem a utilização da proposta de *Test Backlog*.

Analisando os dados da Tabela 8 nota-se que os defeitos de funcionalidades que foram revelados nos artefatos produzidos pelo Time 2 foram três vezes e meia maior que os revelados nos artefatos produzidos pelo Time 1. No Time 1 os defeitos se concentraram mais em defeitos causados por erros no *copy* (arquivo vindo das agências com o conteúdo a ser inserido no site) que representaram o dobro do mesmo tipo de defeitos revelados no Time2, como também, defeitos de layout que contabilizam duas vezes e meia mais defeitos desse tipo se comparado com o mesmo tipo de defeito no Time 2. Em resumo, o Time 1 teve muito mais defeitos relacionados aos documentos que foram fornecidos pelas agências. Já o Time 2, este sim teve muito mais defeitos relacionados a funcionalidade, que poderiam ter sido evitados se o requisito fosse mais completo e deixasse bem claro o que deveria ser testado (e desenvolvido), como proposto neste trabalho.

Tabela 8 – Tipos de Defeitos

TIME 1	Tipos de Defeitos			
Etapas	Defeito de Funcionalidade	Defeito de *copy* enviado pela agência	Defeito Simples de *Layout*	Defeito Complexo de *Layout*
Desenvolvimento	21	46	15	43
UAT	0	4	1	3
Produção	0	0	0	0
TIME 2	Tipos de Defeitos			
Etapas	Defeito de Funcionalidade	Defeito de *copy* enviado pela agência	Defeito Simples de *Layout*	Defeito Complexo de *Layout*
Desenvolvimento	46	23	1	8
UAT	22	2	0	15
Produção	5	1	0	1

Outro importante dado a ser analisado é o número de *blocks* criado durante o desenvolvimento de ambos os projetos. Sabe-se que quanto maior o número de *blocks*, menor o fluxo contínuo de trabalho dos times. Um grande número de *blocks* é algo que deve ser altamente evitado em Scrum. Nota-se uma considerável diferença entre o número de *blocks* gerado pelo Time 1 durante os 4 *sprints* e o número de *blocks* do Time 2 também nos 4 primeiros *sprints* o que pode ser observado na Tabela 9. O Time 2 teve duas vezes e meia mais *blocks* que o Time1, uma vez que apresentou 41 *block*s durante o desenvolvimento, contra 16 do Time1. Dessa forma, a terceira hipótese nula relacionada na Tabela 5 foi rejeitada, havendo indícios de que a aplicação da metodologia fez com que o número de *blocks* durante o *sprint* fosse reduzido.

Outro fator relevante foi a diminuição dos *blocks* no último *Sprint* (quinto *Sprint*) de desenvolvimento do Time 2, no qual este time também utilizou a metodologia proposta. Nota-se que houve uma redução de pelo menos 50% no número de *block*s (se compararmos o *Sprint4* com o *Sprint5*) alcançando a média de *blocks* observados no Time1. Confirmando o indício de que a metodologia auxiliou na redução do número de *blocks*.

Tabela 9 – Número de *blocks* gerados ao longo dos *sprints*

Número de *Blocks* abertos durante o *sprint*	*Sprint* 1	*Sprint* 2	*Sprint* 3	*Sprint* 4	*Sprint* 5
Time 1	6	4	3	3	N/A
Time 2	14	11	9	7	4

Ao analisar o número de horas extras realizados pelos times percebeu-se uma diferença significativa nos dados dos dois times. O Time2, que não utilizou a proposta, apresentou um número alto de horas extras, contabilizando 39 horas extras nos primeiros 4 *sprints*, enquanto que o Time 1, que utilizou a metodologia só apresentou duas horas extras no último *sprint*. Estas informações podem ser visualizadas na Tabela 10. Ao estudar a causa das horas extras necessárias para o término do *Sprint* no prazo constatou-se que as horas se concentraram no final do *sprint* e foram majoritariamente alocadas para a correção de defeitos que foram reportados nos testes funcionais. As correções que o Time 1 precisou fazer foram diluídas ao longo do *sprint*, pois os bugs eram reportados continuamente. Dessa forma, a quarta hipótese nula relatada na Tabela 5 foi rejeitada, havendo fortes indícios de que a metodologia auxiliou na redução das horas extras para que o *sprint* fosse completado. Também ao observar o número de horas extras do quinto *sprint* do time 2, verifica-se que houve uma redução de 75 % no número de horas extras quando a metodologia foi aplicada no mesmo Time 2.

Tabela 10 – Número de Horas Extras realizadas

	Sprint 1	*Sprint* 2	*Sprint* 3	*Sprint* 4	*Sprint* 5
Time 1	0	0	0	2	N/A
Time 2	8	10	12	9	2

Outra informação relevante foram os Comentários da *Retrospective Meeting*. Todos os comentários de reuniões de *retrospective* são sempre colhidos e incluídos nos sites de cada time da empresa. Notou-se ao longo do desenvolvimento dos *sprint*s que o Time 2, que não utilizou a proposta, citava alguns pontos que necessitavam de melhorias e entre estes incluem-se pontos que a presente proposta visa melhorar (estes pontos foram detalhados na

coluna “Time 2 (O que podemos melhorar?)” da Tabela 11. Também foram citados pontos positivos pelo Time 1 que eram relacionados a metodologia que estava sendo utilizada, estes foram descritos na coluna “Time 1 (O que foi bom?)” da Tabela 11. A Tabela 11 mostra parte dos comentários das reuniões, selecionados como os mais relevantes. Os nomes dos membros dos times e algumas informações confidenciais foram suprimidas.

Tabela 11 – Comentários da *Retrospective Meeting*

Time 1	O que foi bom?	O que podemos melhorar?
Sprint 1	- Não passamos tanto tempo corrigindo defeitos. - Não fizemos hora extra como nos *sprints* anteriores.	- Qualidade do *copydeck* que veio da agência podia ser melhor.
Sprint 2	- O número de defeitos encontrados pelo PO diminuiu.	- Planning meeting atrasou 1 hora por atraso de um dos membros do time.
Sprint 3	- Tivemos menos *blocks* do que nos últimos *sprints*.	
Sprint 4	- Não tivemos nenhum defeito em produção.	- 2 membros do time faltaram da *daily* no último dia de *Sprint*.
Time 2	O que foi bom?	O que podemos melhorar?
Sprint 1		- Tivemos que fazer horas extras para atingir a meta. - Foi encontrado um número grande de defeitos em produção.
Sprint 2		- Erramos nas estimativas das estórias. - Tivemos um número grande de defeitos em UAT e ainda tivemos alguns em produção.
Sprint 3		- Estamos com um número grande de *blocks* que está sendo descoberto após a *planning*.
Sprint 4	- Feedback positivo do cliente pelo time ser muito comprometido.	
Sprint 5	- Mudança no formato dos requisitos facilitou os testes; - Mudança dos requisitos fez com que o desenvolvimento fosse mais “certeiro”; - Menor número de *blocks* ao longo do	- Pouco tempo para o planejamento de testes em conjunto com o PO, se ele tivesse mais tempo livre, talvez o trabalho tivesse ficado ainda melhor. - 2 horas extras do *tester* com o PO para

	Sprint, tivemos menos dúvidas já que a maioria dos detalhes estava no *Test Backlog*.	revisar as estórias fora do horário de trabalho porque o PO estava com a agenda cheia.

Neste capítulo foram apresentados os resultados colhidos durante todo o período de execução e todas as observações pertinentes a tais resultados. O próximo capítulo apresenta todos os trabalhos relacionados ao assunto desta pesquisa.

6. TRABALHOS RELACIONADOS

Inicialmente, para encontrar trabalhos relacionados ao tema desta pesquisa, foi realizada uma busca na literatura através do site para pesquisa Free Search (2013), que traz assuntos específicos de ciência da computação. Esta busca foi utilizada através do formato de pesquisa sistemática e foi adaptada do trabalho de Jaqueira et al (2012). Além dessa base especifica, foram utilizadas as bases da ACM, IEEE e Google Schoolar, por serem reconhecidas no meio cientifico como as mais relevantes. Várias sentenças lógicas foram utilizadas para a busca de trabalhos relacionados, combinando os termos "test cases", "user story", "incremental test" e "agile test".

Obter uma resposta sobre a confiabilidade e cobertura das funcionalidades solicitadas pelo cliente é um dos principais objetivos de se realizar testes de *software*. Para demonstrar se o código satisfaz a especificação, a técnica de teste de *software* é um dos métodos mais utilizados (Voas, 1997). O processo de execução dos testes tem como foco minimizar possíveis riscos causados pelas falhas inseridas durante o processo de desenvolvimento do *software*, uma vez que não é possível garantir que o software estará isento de falhas. Além disso, testes exaustivos dos sistemas computacionais não são possíveis para a grande maioria dos sistemas (Voas, 1997) e qualquer que seja a metodologia utilizada para o desenvolvimento do software é importante que muitas técnicas sejam aplicadas para diminuir a quantidade de defeitos no produto final. Dessa forma, trabalhos relacionados ao teste de software são numerosos na literatura e nessa seção daremos foco aos testes incrementais e trabalhos que relacionam os testes como apoio ao desenvolvimento incremental e ágil.

O trabalho de Uzuncaova et al.(2008), cita uma abordagem para o refinamento incremental de suítes de testes para cada produto de uma linha de produtos, gerando automaticamente as entradas de teste para cada um destes produtos. Segundo os autores, os resultados experimentais utilizando diferentes estruturas de dados (diferentes tipos de sistemas e de casos de teste) demonstram que a abordagem incremental teve como resultado um aumento da velocidade da geração de casos de teste em relação às técnicas convencionais. O uso incremental da especificação para a geração parcial de casos de teste vem ao encontro da proposta do presente trabalho, uma vez que ao utilizar a metodologia

Scrum, na grande maioria dos casos não se tem, a priori, a especificação completa e a geração incremental dos casos de teste é a única abordagem possível.

Outro trabalho que também cita os testes como sendo executados de maneira incremental é o de Jaaskelain(2010), onde o foco são os testes baseados em modelos. Como nestes casos é preciso utilizar apenas a parte implementada como base dos testes, sugere-se uma solução na qual as partes do modelo que se referem a implementações já realizadas possam ser filtradas e utilizadas para gerar os testes. Esta abordagem incremental pode ser de grande valia para times que se inserem em contextos em que os testes baseados em modelos sejam mais adequados (Jaaskelain, 2010).

A ideia de gerar suites de testes, com base na especificação de forma incremental, surge também no trabalho de Soundarajan (2000). Segundo o autor, cujo foco é voltado para sistemas orientados a objeto, para verificar se todo o potencial da herança foi atingido, também precisamos ser capazes de gerar os casos de testes dos sistemas de forma incremental, considerando as classes bases e as derivadas. Uma classe derivada pode simplesmente herdar um método da classe base como pode redefinir um método herdado ou criar um novo método. A abordagem incremental permite testar as classes derivadas reutilizando as suítes de teste da classe base quando um método é herdado, adaptando a suíte de teste quando há polimorfismo e criando novas suítes de teste quando se trata de um novo método.

O trabalho de Melnik e Maurer (2007) valida a eficácia dos testes de aceitação para a comunicação, clareza e validação dos requisitos de negócio de um projeto de software. A validação é feita no âmbito da metodologia de desenvolvimento XP (eXtreme Programming) utilizando o método Desenvolvimento Dirigido a Testes de Aceitação (do inglês, *Acceptance Test-Driven Development* - ATDD). Os resultados obtidos permitiram concluir que os testes de aceitação são eficazes no contexto onde foram aplicados e envolvendo vários papeis: o cliente, o desenvolvedor e o profissional de teste. Ainda relata a melhora na qualidade do produto, quando a abordagem foi aplicada em times reais. Os preceitos do ATDD apresentados no trabalho de Melnik e Maurer (2007) embasaram a proposta defendida neste trabalho. A adaptação permite que os testes sejam utilizados tanto como base para o desenvolvimento quanto no detalhamento dos requisitos visando atingir níveis melhores de qualidade e maior adequação às necessidades do cliente.

Os desafios e as diferenças entre os testes aplicados em metodologias de desenvolvimento tradicionais e a maneira incremental de testar na metodologia Scrum são muito bem relacionados por Sumrell (2007) em seu trabalho apresentado na Agile Conference (2007). Segundo o autor, ao fazer a transição para o uso do Scrum, surgiram grandes dificuldades, tanto ao incorporar os testes quanto ao incorporar as estratégias de automação de testes. Vários dos trabalhos citados até o momento neste capítulo já apresentavam o início de ideias que sugerem manter o modo de testar mais incremental de uma forma geral e menos parecido com o antigo modelo cascata (*waterfall*), mas não propunham maneiras de executar tal tarefa.

O trabalho de Jaqueira et al (2012) analisam, utilizando-se de uma revisão sistemática, o desafio da construção de requisitos em metodologias ágeis. O foco deste trabalho foi voltado ao desafio de trabalhar com constantes mudanças e refleti-las no requisito. Jaqueira et al. citam e resumem trabalhos que apontam alguns desafios para os quais o presente trabalho traz propostas de solução. Uma constatação do trabalho de Cao e Ramesh (2008), citado no trabalho de Jaqueira et al (2012), se refere à dificuldade do cliente escrever os testes de aceitação, dizendo que os clientes têm limitações para cumprir essa tarefa e assim muitas empresas que trabalham com testes de aceitação em ambientes ágeis têm um profissional destinado a ajudar esses clientes a criar os testes. Essa constatação embasa a proposta de colocar um membro do time, que chamamos de *tester*, para participar da definição das estórias junto ao cliente e apoiar a equipe durante o desenvolvimento.

O trabalho de Gallardo-Valencia e Sim (2009), também citado no trabalho de Jaqueira et al (2012), fala da falta de documentação em projetos ágeis, relacionando esta omissão à qualidade das estórias de usuários. O trabalho evidencia os resultados de um experimento controlado que visava aumentar a eficiência da construção de requisitos ágeis. Gallardo utilizou-se de casos de uso (ao invés de apenas estórias) para melhorar a qualidade e o detalhamento das estórias. Percebe-se que neste trabalho o autor também almejava a melhoria da qualidade dos requisitos ágeis, mas acredita-se que voltar ao formato utilizado nos métodos tradicionais pode não ser a melhor solução.

A mesma preocupação foi destacada no trabalho de Espinoza e Garbajosa (2005), onde destaca-se a deficiência do rastreamento dos requisitos no desenvolvimento ágil e atribui-se

essa dificuldade à ausência de uma documentação adequada. Para tal desafio, esse trabalho apresenta uma proposta visando melhorar e estender tais estórias.

Nenhum dos trabalhos citados por Jaqueira et al. (2012) mostrou a preocupação de tornar os requisitos testáveis, como o presente trabalho propõe. O próprio autor ainda cita que poucos trabalhos têm foco no contexto da criação de requisitos ágeis e que, apesar de ocorrerem muitas alterações nos requisitos, pouca ênfase é dada à atividade de especificação de requisitos no contexto de métodos ágeis.

Um dos trabalhos que mais se aproxima da presente proposta é o apresentado por Repert (2004). Voltado para o desenvolvimento ágil usando a metodologia *Extreme Programming* (XP), este artigo propõe o uso de Story Test Driven Development (STDD), que se baseia nas estórias e em ATDD para dirigir o desenvolvimento com base em casos de testes de aceitação, gerados a partir das estórias. O STDD utiliza os testes de aceitação tradicionais e reúne clientes, desenvolvedores e profissionais de teste, mesmo antes de qualquer código ter sido escrito. Esses profissionais colaboram entre si para identificar uma funcionalidade a ser desenvolvida. Particularmente, o cliente e o profissional de teste trabalham para definir critérios para validar as estórias. Isso se assemelha com a proposta do corrente trabalho que propõe que o cliente (ao qual chamamos de Product Owner) e o profissional de teste escrevam as *user stories* conjuntamente, de forma a definir não somente os testes de aceitação, mas também, testes funcionais mais amplos e maiores detalhes das estórias. Ainda considerando esta proposta, considera-se que os detalhes adicionais poderão melhorar o entendimento da estória, beneficiando desenvolvedores, profissionais de teste e os próprios clientes que, ao discutir e escrever os detalhes conseguem definir melhor suas necessidades e adquirem maior confiança de que o time fará o sistema solicitado, gerando mais valor agregado. Além disso, a proposta deste trabalho é voltada para o Scrum e não para a metodologia XP. Alguns resultados apontados por Repert (2004) indicam que STDD integrou melhor o profissional de teste na equipe XP, que deixa de ser visto como um "vilão" e passa a ser um importante colaborador e facilitador do time. Os participantes do experimento de Repert consideraram que a aplicação do STDD trouxe uma melhora considerável para o processo de desenvolvimento e que a definição dos resultados esperados, sendo feita por um profissional de garantia de qualidade (Quality Assurance – QA), pode ajudar o time a descobrir cálculos incorretos,

interpretações inexatas e problemas de comunicação. Outro resultado positivo apontado foi a simplicidade do código resultante, que foi considerado muito mais simples do que aquele obtido com TDD, por exemplo.

Outro trabalho fortemente relacionado à presente proposta foi apresentado por Dan North (2006) ao qual foi dado o nome de *Behaviour-Driven Development* - BDD. Neste trabalho, North apresenta o formato onde um expert em business trabalha juntamente com um analista para identificar as necessidades dentro da estória. A ideia foi baseada no entendimento de que o comportamento de uma estória está definido pelos critérios de aceitação que são escritos para essa estória – se o sistema cumpre todos os critérios de aceitação ele está se comportando corretamente. O modelo proposto pelos autores capturam os critérios de aceitação e desenvolvem cenários completos para valida-los. A equipe à qual North pertencia já utilizava um formato para as estória detalhada como segue: "As a **Role,** I request a **Feature** To gain a **Benefit**"**,** que tinha como objetivo ressaltar o valor que a estória trazia para o negócio. Eles então propuseram que os critérios de aceitação dessas estórias fossem capturados e analisados para que um documento registrasse os cenários que poderiam se encaixar em cada um desses critérios de aceitação e escreveram um formato para a validação das estórias com o seguinte formato: "Given **contexto inicial ou conjecturas** When **ocorrência de um evento** Then **garante determinadas saídas",** padronizando assim a criação dos cenários de teste para cada estória que são registradas em um documento de plano de teste. A proposta de North (2006), como a nossa, complementam as estórias com detalhes que precisam ser validados, mas ainda assim, não focam obrigatoriamente nos testes funcionais. Além disso, a complementação é feita em um documento à parte das estórias o que requer um número maior de documentação e manutenções futuras desse documento complementar.

Este capítulo apresentou os trabalhos mais diretamente relacionados à proposta dessa dissertação, apontando os pontos de similaridade e discrepâncias. O levantamento bibliográfico reuniu trabalhos com foco em testes de aceitação, testes incrementais e trabalhos que relacionam os testes como apoio ao desenvolvimento incremental e ágil. O próximo capítulo apresenta as conclusões e os trabalhos futuros.

7. CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou o *Test Backlog*, um conjunto de práticas ágeis baseadas na metodologia Scrum considerando a complementação das estórias com testes funcionais (e não funcionais quando necessário). Ainda utilizou-se a mesma estrutura de fases, eventos e artefatos do Scrum, destacando um membro do time responsável pelos testes. Este membro do time tem um papel central na execução do *Test Backlog,* pois será o responsável por melhorar a comunicação entre o time e o PO. A proposta é que ele acompanhe o PO na criação das estórias, discuta os detalhes destas estórias no momento de sua criação e escreva, neste mesmo momento, os casos de testes funcionais e não funcionais que deverão ser exercitados durante o *Sprint*. Com esta interação mais forte com o PO, ele adquire um maior conhecimento das estórias, estando apto a esclarecer boa parte das dúvidas que surgem no time durante o planejamento e execução do *Sprint*.

Além disso, os testes mais incrementais são sugeridos e cabe a este profissional a contínua tarefa de aplicação destes testes, fazendo com que o final do *Sprint* não se configure em uma pressão sobre o time devido as correções de defeitos que são reportados nos momentos finais.

Para avaliar a eficácia da proposta, o *Test Backlog* foi aplicado em um time de uma empresa que desenvolveu um software real, enquanto que, em paralelo, um time de controle desenvolvia um software muito similar, também real e destinado ao mesmo cliente, usando a metodologia que vinha sendo empregada na empresa no último ano.

Através dos resultados encontrados ao longo do desenvolvimento dos experimentos pode-se notar fortes indícios da validade da proposta desta dissertação de mestrado.

O primeiro resultado analisado foi o número de defeitos encontrados ao longo dos *sprints*. O formato das estórias foi redesenhado para incluir casos de testes e assim auxiliar os desenvolvedores a fazer uma implementação mais aderente aos requisitos do cliente. O número de defeitos encontrado durante os 4 *sprints* do Time 1 (time que utilizou a proposta) e também durante o último *sprint* do Time 2 (*sprint* utilizado para confirmação dos resultados da proposta) indica fortemente a melhora da qualidade da implementação e maior adequação aos requisitos do cliente. Os resultados permitem concluir que há um forte

indício de que a metodologia colaborou para a descoberta de defeitos o mais cedo possível e que a qualidade dos testes foi maior quando foi utilizado um planejamento de testes mais completo. Os pontos que foram observados que podem comprovar tais indícios são:

- menor número de defeitos durante o período de homologação;
- nenhum defeito em produção (até o fechamento desta pesquisa);
- maior número de defeitos encontrados ainda em tempo de desenvolvimento.

Outro fator relativo ao formato proposto e ao conhecimento do requisito que ele proporciona ao time é o tipo de defeito gerado pelos dois times. Observou-se a tendência de encontrar uma grande quantidade de defeitos mais simples (relacionados a conceitos básicos que poderiam ter sido elencados na descrição do requisito), chegando a três vezes e meia mais defeitos, ao longo dos *sprint*s do Time 2. Já os defeitos do Time 1 eram mais relacionados à questões que eram de responsabilidade da agência contratada pelo cliente, tais como, *layout* e *copy*. Um menor número de defeitos foi encontrado em períodos finais do desenvolvimento (quase cinco vezes menor do que o time de controle) como consequência do maior número de defeitos encontrados o mais cedo possível, ainda ao longo do *sprint*, indicando maior cobertura dos testes quando utilizou-se um planejamento de testes mais abrangente.

Por fim, o último fator analisado que está diretamente relacionado ao formato de estórias proposto foi a quantidade de *blocks* elencados pelo time durante o *sprint*, resultado das deficiências de entendimento das estórias e que deveriam ter sido sanadas nas reuniões de planejamento, como indicado pelo Scrum. Notou-se nos dados apresentados na Tabela 9 um número de *blocks* consideravelmente maior no time 2, chegando no melhor caso ao dobro dos *block*s que ocorreram no Time 1, indicando fortes indícios de que a qualidade do requisito não foi alta o suficiente para levantar discussões sobre as necessidades do cliente durante o planejamento e que o detalhamento com que as estórias foram escritas também não foram suficientes para sanar essas dúvidas ao longo do desenvolvimento do *Sprint*. Ao contrário do Time 1, que teve poucos *blocks*, o Time 2 teve um número grande de *blocks* ao longo dos 4 *sprint*s e consequentemente, um grande tempo onde o time esteve bloqueado, causando atrasos. Notou-se ainda que o número de *blocks* diminuiu e se alinhou ao número observado no Time 1, quando o Time 2 usou a metodologia no *Sprint* 5. Como a variação, nesse caso, foi significativa apenas em relação à metodologia (conservando-se os mesmos

elementos no time e produto similar em desenvolvimento), entende-se que houve um impacto positivo da metodologia para diminuir o número de *blocks*. Esse fato leva ao próximo dado analisado, o número de horas extras.

Outro ponto importante desta proposta é a mudança no papel do testador e em suas atividades diárias. Ao invés de iniciar as atividades de teste ao final de cada estória, o testador responsável pela estória (no Time 1) diluía o trabalho de testes ao longo de todas as *tasks* da estória. Isso não aconteceu no Time 2, enquanto esse time não utilizou a proposta. Dados combinados das horas extras (Tabela 10) e dos comentários da reunião de retrospectiva (Tabela 11) mostram o número maior de horas extras e uma pior distribuição de trabalho no Time 2. Um número maior de testes ocorreu apenas no final do *sprint* que culminaram em horas extras nesse período, para cobrir os atrasos causados pelos *blocks* e finalizar todas as horas de testes necessárias.

Ao final de cada *sprint*, testes de regressão foram executados para garantir a qualidade do produto entregue. O Time 2, teve algumas dificuldades para executar estes testes que exigiam que os casos de testes mais importantes fossem reexecutados, mas nem todos eles haviam sido armazenados ou bem detalhados. Em contrapartida, ao final de cada *sprint* do Time 1, observou-se uma maior qualidade nos testes de regressão, o que também contribuiu para um menor número de defeitos em homologação e em produção.

Concluiu-se ao longo deste trabalho, mas principalmente após a análise dos resultados dos times, que o modelo proposto poderá ajudar tanto na execução quanto no planejamento de testes de software em equipes ágeis que utilizam o Scrum. Os resultados mostraram uma melhoria tanto na qualidade do desenvolvimento e maior adequação do produto às reais necessidades do negócio, quanto na qualidade dos testes de software, encontrando defeitos cada vez mais cedo e diminuindo a quantidade de defeitos em homologação e produção.

Acredita-se que a automação dos testes agregaria ainda mais valor a esta proposta e deve ser considerada como um próximo passo. Infelizmente a automação não pôde ser aplicada no escopo dos projetos estudados pelo perfil do produto desenvolvido (sites pequenos e com páginas, em sua maioria, de conteúdo simples), mas a automação é fortemente recomendada para ser executada juntamente com a metodologia Scrum. Trabalhos futuros relacionados a esta proposta poderão analisar os benefícios que a

automação poderia trazer para aumentar a eficiência dos times e a qualidade dos produtos desenvolvidos.

Também seria interessante fazer experimentos com um maior número de times (não apenas um time seguindo a proposta e um time de controle). Experimentos com software mais complexos que exigissem definição de testes não funcionais também poderiam ser considerados, assim o comportamento deste tipo de teste na definição das estórias seria melhor validado.

Outro ponto muito interessante a ser avaliado em trabalhos futuros é uma melhor definição de métricas das hipóteses. Definir métricas específicas facilitaria muito o trabalho e auxiliaria na medição do que é um número excelente de *blocks*, defeitos e até mesmo hora extras (ou a sua inexistência).

Este trabalho de mestrado gerou uma publicação no Congresso WBMA 2012, 3rd Brazilian Workshop on Agile Methods, com o título: “Incremental Tests: An Approach to Improve Software Tests in Agile Teams”. Uma cópia do artigo foi anexado no Apêndice A deste trabalho de mestrado.

REFERÊNCIAS BIBLIOGRÁFICAS

Agile Conference. Washington D.C., 2007. Disponível em: http://agile2007.agilealliance.org/ , Último acesso: 15/10/2013.

Atlassian Software. JIRA Attlassian. Sydney HQ, 2013. Disponível em: https://www.atlassian.com/software/jira. Último acesso: 17/10/2013.

Auvinen, J., Back, R., Heidenberg, J., Hirkman, P., Milovanov, L. Improving the Engineering Process Area at Ericsson with Agile Practices. A Case Study. Turku Centre for Computer Science, 2005. Disponível em: http://tucs.fi/publications/view/?pub_id=tAuBaHeHiMi05a, Último acesso: 15/10/2013.

Barros, L. Test-Driven Development: uma visão prática - LES. Puc RIo, 2009. Disponível em: https://www.google.com.br/#q=Test-Driven+Development%3A+uma+vis%C3%A3o+pr%C3%A1tica Último acesso em: 17/10/2013

Bakalova, Z., Daneva, M., Herrmann, A., Hieringa, R. Agile requirements prioritization: what happens in practice and what is described in literature? Berlin, 2011. Disponível em: http://dl.acm.org/citation.cfm?id=1987386, Último acesso: 08/10/2013.

Beck, K. Test Driven Development: By Example. 1ª Edição, Addison-Wesley Professional, pp. 240, 2002.

Beck, K and Andress, C. Extreme Programming Explained: Embrace Change. 2 ª Edição, Addison-Wesley Professional, pp.224, 2004.

Binder, R. V. “Testing OO Systems: Models, Patterns and Tools”. Addison-Wesley, c.15, 1999.

Black, R. Managing the Testing Process. 3ª Edição, John Wiley Consumer, pp. 672, 2009.

Bohl, E.J. Acceptance Test Driven Development, Explained. Test Driven Developer, 2008. Disponível em: http://testdrivendeveloper.com/2008/04/02/AcceptanceTestDrivenDevelopmentExplained.aspx. Último acesso: 07/11/2013.

Braz, A. Método Ágil aplicado ao Desenvolvimento de Software Confiável baseado em Componentes. Disponível em: http://alanbraz.files.wordpress.com/2011/05/wtd.pdf. Último acesso: 15/10/2013.

Cao, L. and Ramesh, B. Agile Requirements Engineering Practices: An Empirical Study. IEEE Software, Vol. 25, Issue 1, pp. 60-67, 2008.

Cohn, M. User Stories Applied. 1ª Edição, Addison Wesley, 2004.
Cohn, M. Agile Estimating and Planning. 1ª Edição, Prentice Hall, pp.368, 2005.

Collins, E. and Lobão, L.M.A. Experiência em Automação do Processo de Testes em Ambiente Ágil com SCRUM e ferramentas OpenSource. In: Proceedings of IX Simpósio Brasileiro de Qualidade de Software, 2010. Disponível em: http://www.lbd.dcc.ufmg.br/colecoes/sbqs/2010/RL4_eliane_collins.pdf Último acesso: 15/10/2013.

Crispin, L. and Gregory, J. Agile Testing: A Practical Guide for Testers and Agile Teams. 1ª Edição, Addison-Wesley, pp. 533, 2009.

Crispin, L. XP Testing without XP: Taking advantage of Agile Testing Practices. Methods & Tools, Summer 2003. Disponível em http://www.methodsandtools.com/archive/archive.php?id=2. Último acesso: 01/05/2013.

Erdogmus, H. and Morisio, T., Torchiano, M. On the Effectiveness of Test-first Approach to Programming. IEEE Transactions on Software Engineering, Vol. 31, Issue 3, pp. 226-237, 2005.

Espinoza, A., Garbajosa, J. Study to Support Agile Methods More Effectively through Traceability. Innovations in Systems and Software Engineering, Vol. 7, Issue 1, pp. 53-69, Springer-Verlag, 2005.

Free Search – Toward Computer Science Ideas, 2013. Disponível em: http://dblp.kbs.uni-hannover.de/dblp/. Último acesso: 01/05/2013.

Gallardo-Valencia, R.E., Sim, S.E. Continuous and Collaborative Validation: A Field Study of Requirements Knowledge in Agile. In: Proceedings of the Second International Workshop on Managing Requirements Knowledge, pp. 65-74, Atlanta, USA, 2009.

Gallardo-Valencia, R.E., Oliveira, V., Sim, S.E. Are Use Cases Beneficial for Developers Using Agile Requirements? In: Proceedings of the Fifth International Workshop on Comparative Evaluation in Requirements Engineering, pp. 11-22, New Delhi, India, 2007.

Grenning, J. Planning Poker or How to avoid analysis paralysis while release planning. Renaissance Software Consulting. 2002. Disponível em: http://www.renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf. Último acesso em 15/10/2013.

Guimaraes Jr., A. A. Ferramenta para geração de testes de unidade em linguagem C. Universidade de Brasília, 2008. Disponível em: http://monografias.cic.unb.br/dspace/bitstream/123456789/166/1/monografia.pdf. Último acesso: 06/05/2013.

Hendrickson, E. Driving Development with Tests: ATDD and TDD. Quality Tree Software, Inc. Disponível em: http://testobsessed.com/wp-content/uploads/2011/04/atddexample.pdf. Último acesso: 23/09/2013.

IBM. IBM Rational Unified Process, 2013. Disponível em http://www-01.ibm.com/software/rational/rup/. Último acesso: 15/10/2013.

Jääskeläin, A. Filtering Test Models to Support Incremental Testing. In: Proceedings of 5th International Academic and Industrial Conference, TAIC PART. Windsor, UK, Lecture Notes in Computer Science, Spring-Verlag, pp. 72-87, 2010.

Jaqueira, A.; Andreotti, E; Lucena, M.; Aranha E. Desafios de Requisitos em Métodos Ágeis: Uma Revisão Sistemática. In: Proceedings of the 3rd Brazilian Workshop on Agile Methods, 2012. Disponível em: http://www.ime.usp.br/~kon/wbma2012.pdf. Último acesso: 01/04/2013.

Kayes, I. Agile Testing: Introducing PRAT as a Metric of Testing Quality in Scrum
ACM SIGSOFT Software Engineering Notes, Vol. 36, Issue 2, pp. 1-5, ACM New York, NY, USA, 2011.

Kerlinger, T. C., Taylor, J. R. Marketing research: an applied approach. Tóquio: McGraw-Hill Kogakusha, 1979.

Kung, D., Gao, J., Hsia, P., Wen, F., Toyoshima, Y., Chen, C. On regression testing of object-oriented programs. The Journal of Systems and Software, 32(1):21--40, January 1996.

Llopis, N. Stepping Through the Looking Glass: Test-Driven Game Development (Part 1). Games from Within, 2007. Disponível em: http://gamesfromwithin.com/stepping-through-the-looking-glass-test-driven-game-development-part-1. Último acesso: 15/10/2013.

Mafra, S. N. and Travassos, G. H. Estudos Primários e Secundários apoiando a busca por Evidência em Engenharia de Software. RT-ES 687/06. COPPE/UFRJ, Rio de Janeiro, 2006.

Malhotra, N. Pesquisa de marketing: uma orientação aplicada. 6ª edição. Bookman Companhia, pp. 768, 2011.

Manifesto Ágil. Disponível em: http://agilemanifesto.org/. Último acesso: 07/05/2013.

Mattar, F. N. Pesquisa de Marketing: metodologia e planejamento. 6ª. Edição. São Paulo: Atlas, 2005.

Melnik, G., Maurer, F. Multiple Perspectives on Executable Acceptance Test-Driven Development. In: Proceedings of 8th International Conference, XP 2007, Como, Italy, Springer Berlin Heidelberg, pp. 245-249, 2007.

Myers, G. J. The Art of Software Testing. 2ª Edição, John Wiley & Sons, Inc., Hoboken, New Jersey, 2004.

North, D. Introducing BDD. Dan North & Associates. Better Software, 2006. Disponível em: http://dannorth.net/introducing-bdd/. Último acesso: 15/10/2013.

Patuci, G., Moraes, R. Incremental Tests: An Approach to Improve Software Tests in Agile Teams. In: Proceedings of 3rd Brazilian Workshop on Agile Methods (WBMA'2012), Agile Brazil, São Paulo, Brasil, pp. 61-71,2012.

Pfleeger, S. L. Engenharia de Software: teoria e prática. 2ª Edição. Prentice Hall, São Paulo, 2004.

Repert T. Don´t Just Break Software, Make Software. Better Software, July / August 2004. Disponível em http://www.industriallogic.com/wp-content/uploads/2005/09/storytest.pdf. Último acesso: Agosto 2013.

Rubin, J. Handbook of Usability Testing. 1ª Edição, John Wiley Consumer, New York, USA, pp. 352, 1994.

Schawber, K. Agile Project Management With Scrum. 1ª Edição. Microsoft Press, pp. 163, 2004.

Schawber, K. Guia do Scrum. Scrum Alliance, pp. 22, 2009.
Disponível em: http://www.trainning.com.br/download/GUIA_DO_SCRUM.pdf
Último acesso: 15/10/2013.

Schwaber, K. SCRUM Development Process, OOPSLA'95. In: Proceedings of the Workshop on Business Object Design and Implementation. Austin, 1995. Disponível em: http://www.jeffsutherland.org/oopsla/schwapub.pdf Último acesso em: 22/08/2013.

Scrum Alliance, 2013. Disponível em: http://scrumalliance.org/. Último acesso: 01/05/2013.

Scrum Methodology. 3 Scrum Phases, 2013. Disponível em: http://www.scrummethodology.org/scrum-phases.html Último acesso: 15/10/2013.

Soares, M. S. Metodologias Ágeis: Extreme Programming e Scrum para o Desenvolvimento de Software. Revista Eletrônica de Sistemas de Informação, Vol. 3, No 1, 2004. Disponível em: http://revistas.facecla.com.br/index.php/reinfo/article/view/146.
Último acesso: 07/10/2013.

Sommerville, I. Engenharia de Software. 8ª Edição. Pearson Addison-Wesley BRA. São Paulo, pp. 568, 2007.

Soundarajan, N. Specification-Based Incremental Testing of Object Oriented Systems. 2000. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.1270. Ultimo acesso: 07/05/2013.

Sumrell, M. From Waterfall to Agile - How does a QA Team Transition? Agile Conference (AGILE), 2007. Disponível em: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4293611 Último acesso: 07/05/2013.

Takeuchi, H. and Nonaka, I. The New New Product Development Game. Harvard Business Review. The Magazine, 1986.

Travassos, G. H.; Gurov, D.; Amaral, E. A. G. Introdução à Engenharia de Software Experimental. RT–ES–590/02. COPPE/UFRJ, Rio de Janeiro, 2002. Disponível em: http://www.ufpa.br/cdesouza/teaching/topes/4-ES-Experimental.pdf. Último acesso: 07/05/2013.

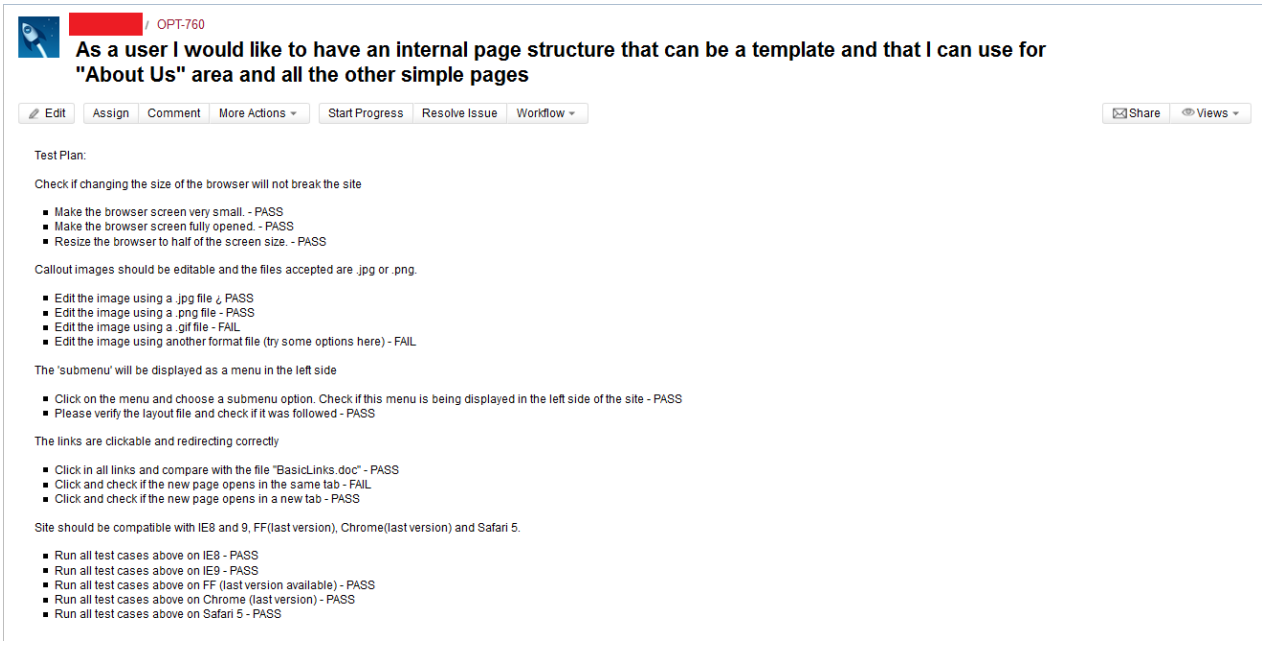
Uzuncaova, E.; Garcia, D.; Khurshid, S.; Batory, D. Testing software product lines using incremental test generation. In Proc. of the 19th International Symposium on Software Reliability Engineering. Seattle, Redmond, USA, 2008.

Voas, J; Mcgraw, G. Software Fault Injections: Inoculating programs against errors. John Wiley & Sons, Inc., New York, NY, USA, 1997.

Wainer, J. Métodos de pesquisa quantitativa e qualitativa para a Ciência da Computação. Sociedade Brasileira de Computação e Editora PUC-Rio, 2007. Disponível em: http://www.ic.unicamp.br/~wainer/papers/metod07.pdf. Ultimo acesso: 07/09/2013.

ANEXO A

Exemplos de Artefatos utilizados no Experimento.



[redacted] / OPT-760

As a user I would like to have an internal page structure that can be a template and that I can use for "About Us" area and all the other simple pages

Edit | Assign | Comment | More Actions | Start Progress | Resolve Issue | Workflow | Share | Views

Test Plan:

Check if changing the size of the browser will not break the site

- Make the browser screen very small. - PASS
- Make the browser screen fully opened. - PASS
- Resize the browser to half of the screen size. - PASS

Callout images should be editable and the files accepted are .jpg or .png.

- Edit the image using a .jpg file ¿ PASS
- Edit the image using a .png file - PASS
- Edit the image using a .gif file - FAIL
- Edit the image using another format file (try some options here) - FAIL

The 'submenu' will be displayed as a menu in the left side

- Click on the menu and choose a submenu option. Check if this menu is being displayed in the left side of the site - PASS
- Please verify the layout file and check if it was followed - PASS

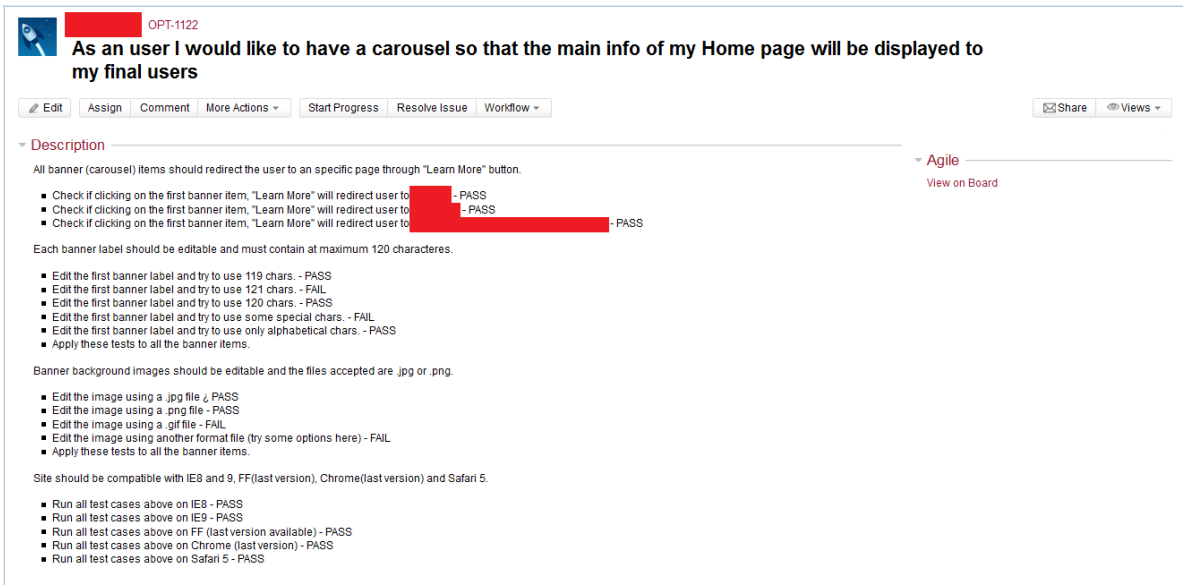
The links are clickable and redirecting correctly

- Click in all links and compare with the file "BasicLinks.doc" - PASS
- Click and check if the new page opens in the same tab - FAIL
- Click and check if the new page opens in a new tab - PASS

Site should be compatible with IE8 and 9, FF(last version), Chrome(last version) and Safari 5.

- Run all test cases above on IE8 - PASS
- Run all test cases above on IE9 - PASS
- Run all test cases above on FF (last version available) - PASS
- Run all test cases above on Chrome (last version) - PASS
- Run all test cases above on Safari 5 - PASS

Figura A.1: Exemplo de Estória utilizada pelo Time



[redacted] OPT-1122

As an user I would like to have a carousel so that the main info of my Home page will be displayed to my final users

Edit | Assign | Comment | More Actions | Start Progress | Resolve Issue | Workflow | Share | Views

Description

All banner (carousel) items should redirect the user to an specific page through "Learn More" button.

- Check if clicking on the first banner item, "Learn More" will redirect user to [redacted] - PASS
- Check if clicking on the first banner item, "Learn More" will redirect user to [redacted] - PASS
- Check if clicking on the first banner item, "Learn More" will redirect user to [redacted] - PASS

Each banner label should be editable and must contain at maximum 120 characteres.

- Edit the first banner label and try to use 119 chars. - PASS
- Edit the first banner label and try to use 121 chars. - FAIL
- Edit the first banner label and try to use 120 chars. - PASS
- Edit the first banner label and try to use some special chars. - FAIL
- Edit the first banner label and try to use only alphabetical chars. - PASS
- Apply these tests to all the banner items.

Banner background images should be editable and the files accepted are .jpg or .png.

- Edit the image using a .jpg file ¿ PASS
- Edit the image using a .png file - PASS
- Edit the image using a .gif file - FAIL
- Edit the image using another format file (try some options here) - FAIL
- Apply these tests to all the banner items.

Site should be compatible with IE8 and 9, FF(last version), Chrome(last version) and Safari 5.

- Run all test cases above on IE8 - PASS
- Run all test cases above on IE9 - PASS
- Run all test cases above on FF (last version available) - PASS
- Run all test cases above on Chrome (last version) - PASS
- Run all test cases above on Safari 5 - PASS

Agile

View on Board

Figura A.2: Exemplo de Estória utilizada pelo Time

Figura A.3: Exemplo de gráfico de Tasks Burndown utilizado pelo Time 1 - *Sprint* 1.

Conforme apresenta a Figura A.3, houve um desenvolvimento (“queima”) mais lento do que o esperado devido a adaptação do Time ao novo formato de escrita das estórias, uma vez que nesse dia foi feita a reunião de planejamento. Vencido essa etapa, houve um período onde o desenvolvimento foi mais rápido do que o esperado devido ao menor número de impedimentos (*blocks*). Observa-se que durante todo do restante do tempo, o *sprint* acompanhou a curva de previsão (que é o desejável) não tendo ocorrido pressão sobre o time durante todo o tempo. Ao final, o sprint terminou conforme o previsto.

APÊNDICE A – ARTIGO PUBLICADO

Incremental Tests: An Approach to Improve Software Tests in Agile Teams

Gabriela de Oliveira Patuci, Regina Lucia de Oliveira Moraes

School of Technology – University of Campinas (UNICAMP)
Limeira – SP – Brazil

opgabi@gmail.com, regina@ft.unicamp.br

Abstract. *Most agile and testing practices primarily make use of user stories for validation. This paper presents how to enhance user stories that normally only register system requirements to also be useful for test cases. This test plan should be as complete as possible so that it could be used in the sprint functional tests. Testers should have an essential role all throughout the process, starting from reviewing the requirements in conjunction with the Product Owner and building the backlog in a test case format. Incremental tests could be done in pairs and combining new functional testing, each day towards the sprint ending. Regression tests complement previous test whenever integration is done.*

1. Introduction

The agile software development methodologies have emerged as alternatives to the traditional methodologies, where the need of large documentation and compliance with standards, requires a longer development cycle. This lengthened time that is necessary to complete all steps of traditional methods, conflicts with the demand to reduce time-to-marketing in today's markets.

Even with the evolution of computers, techniques and tools in recent years, the production of reliable software, correct and delivered within the stipulated time and cost is

still a very rare event [Soares, 2004]. For this reason, agile methodologies have been used by several Information Technology (IT) companies that are migrating from their traditional development models to models that allow lighter planning and management, such as Scrum.

Agile methodologies are being used widely, but often, as they seek for reduced cost and time, has caused many companies to despise or get rid of key stages, that later will be configured as low-quality in the developed product. There has been a wide acceptance of Scrum and the growth of the demand for experienced professionals in the field, as can be seen from the number of ScrumAlliance members (Agile Official and Global Community) that represents over 100,000 registered members and officially certified [ScrumAlliance, 2012].

Another way to measure an increase in the use of Scrum in recent years is through the appearance of a large number of conferences, official events and trainings in general. Data taken from the official website [ScrumAlliance, 2012] report the creation of more than 726 events related to Scrum in only 6 months (the last three months of 2011 and the first three of 2012).

Software testing is an activity that is part of the development as a whole and that is performed regardless of the methodology adopted. In Scrum, testing activities are not always completely defined and, by the fact that this activity is not the focus of most researches and case studies in agile environment, a better definition has been left open.

Articles and texts that refer to the agile test reality were researched during the last months. Much of the existing material in the area refers to the work of the same researchers, Crispin and Gregory (2009). Even searching on sites and publications of the agile community, there is still a great difficulty on finding works that focus on the testing activity in agile context, whether in articles or books. Researches in progress that are related to agile development methodologies still reflect the concern of the authors in refining the methodology and resolve gaps in the basic coding process and still didn´t have time enough to get in the validation steps, that are also a very important part of the product development itself.

The main objective of this paper is to show the results of a study created to cover this gap: adapting user stories to be also used as test cases. Also the daily activities are

detailed in a way that this expertise can be a differential contribution to the team as a whole. Another objective is to identify how these activities will be carried out thus maintaining quality results and total adaptation to the needs of the agile teams.

The organization of the paper is as follows: after the first section that presents an introduction and motivation, Section 2 presents backgrounds about user stories, acceptance test and acceptance test driven development; the proposal and discussion is presented in Section 3 and Section 4 presents our conclusions.

2. Background

Important concepts such as User Stories, Acceptance Tests and Acceptance Test Driven Development are essential for a better understanding of this paper and its general scope. These will be presented in next subsections.

2.1. User Stories

According to Cohn (2004), "A user story describes functionality that will be valuable to either a user or purchaser of a system or software. Stories are composed of three aspects:

- A written description of the story to be used in planning and also as a reminder;
- Discussion about the story that will serve to raise all the details of the story;
- Tests that convey and document the details and can be used to determine when the story is complete."

The third factor cited by Cohn is the basis for this proposal since it already makes clear the importance of testing as the development basis and to assist in user stories detailing.

More than just knowing what the user stories are about, what is expected of professionals who will work with that, especially the Product Owners and Test Analysts, is to write good stories. The most efficient way of doing this, according to Cohn (2004) and Crispin and Gregory (2009) would be to work with acceptance tests, which would be considered as the acceptance criteria of this user story.

2.2. Acceptance Tests

Acceptance testing can be considered as any validation that was planned to be taken into consideration to define if a user story is ready or not. The concept of acceptance testing becomes clear when Cohn (2004) considers that "Acceptance Testing is the process of verifying that the stories were developed so each of them works exactly the way the customer expects".

The acceptance test, taking into account the processes of traditional tests, is the last test step to be exercised before its operational use. Normally, the system is tested with data provided by the client and not with the simulated test data. Acceptance testing is an important tool to reveal gaps in the system requirement definition, but also to reveal problems of requirements that make the system does not meet user needs or provide an unacceptable performance [Sommerville, 2010].

According to Atlee and Pfleeger (2009), three types of acceptance test can be used to evaluate the system. The benchmark test evaluates the system based on specified standards. The test cases represent the typical conditions under which the system will operate when installed and are used to evaluate the actual performance for each of the planned test cases. The pilot test allows the user to install the system and enables users to make use of the system as if it was actually installed in the operating environment. It is based on the daily work of the system and often the client prepares a list which incorporates features of the day to day for each user. And last, the parallel test can be used when a new system is replacing an existing one. In this case, the new system operates in parallel to the previous version, and gradually, users will get used to the new system. The gradual transition allows the comparison between the new and the old system and also allows more skeptical users to gain confidence using the new system.

After acceptance testing, the client reports to the development team which requirements were not achieved which should be deleted, revised or included, due to the need for modifications. These changes should be analyzed and the consequences in the design, implementation and testing should be recorded.

2.3. Acceptance Test Driven Development (ATDD)

Acceptance Test Driven Development (ATDD) is increasing its use in software development team in general. The technique consists on using the steps presented in Figure 1 to assure the team build very good user stories and, more than that, a very good final product. The steps to follow are [Hendrickson, 2008]:

- Discuss the requirements;
- Distill tests in a framework-friendly format;
- Develop the code (and hook up the tests);
- tests);

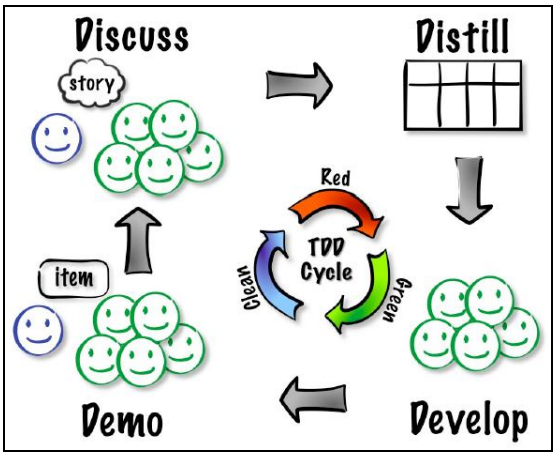


Figure 1. ATDD cycle model [Hendrickson, 2008]

The first step, to Discuss, can be described by the idea of having the right questions on the planning meeting and from them, to have also a better understanding of what has to be done. Once we have this understanding, the acceptance tests can be written in collaboration with the PO. Hendrickson (2008) says that "By asking the right questions we are prompting the business stakeholder to think carefully about his expectations for the feature and gleaning insight into the acceptance criteria for the feature".

After discussing and writing the User Stories, the next step would be to Distill. The idea is that having the tests cases sketched out, the team would be able to capture the tests in an automation framework format such as FIT, Fitnesse and Concordian. The next two steps, team has to Code and Demo the product, so the cycle would be completed.

The two first steps are the main idea of the ATDD model and also, the ones this proposal was most based.

3. The Proposal

When creating the requirements in a new format, it is possible to have a boost on the project quality assurance improving the day by day activities that should be executed, so the tests can be done in the most productive way. The most important concept here is to take advantage of all the sprint days and not having any professional "*blocked*" waiting for some user story to be ready for functional testing phases. The details of this will be presented in the next two subsections.

3.1. Test *Backlog*

User Stories construction and how they are applied nowadays were already described on the previous section (refer to Section 2.1). The proposal of this work is to present a new format and also a new function for User Stories. It is a fact that the User Stories have to be well constructed to be the base to a really good software development but since Scrum methodology claims less work on writing documentation in general, to write down good user stories and also to create detailed test plans seems not to be a so agile approach and a rework, since both have to be the expected behavior of the system very well detailed. Figure 2 presents a User Story as it is used currently in most of agile teams.

As a user I would like to have a carousel so that the main info of my Home page will be displayed to my final users.
Browsers supported: IE8, IE9, FF (last version), Chrome (last version) and Safari 5. Marquee should be friendly to browser screen resizes. Standalone text box should not be affected to browser screen resizes. Time is EST (hh:mm). Marquee should have the 3 banners options redirecting to different places. Banner text and image should be editable and have a limit of 120 characters. Slider must be 3 bullets (one for each banner).

Figure 2. A User Story in the current agile team's format

One role of the tester role is to follow up with the Product Owner on construction of the user stories to help ensure that they best fit into the team necessities, It should be noted that

by both working together provided a good match to build really good testable user stories that would benefit both the project requirement and also the test plan.

Another advantage of this approach is the approval of the test plan by the client and its use to a really constructive acceptance tests in every build delivery and validation that would match the value generation expectation. Figure 3 presents the same User Story presented in Figure 2 but now complemented with test cases. The User Story new format can guide tester to validate the system, developers to best understand the requirement and user to make easier to perform acceptance tester Stories construction.

As it can be visualized on Figure 2, the way user stories are written in some of current agile projects is to have all the details that the Product Owner wants the system to have on the User Story cards. It is nice to have these details on the card, but they have to be rewritten in test case format after the planning meeting and, when this time comes, a lot of questions and possible *block*s can arise.

The format proposed on Figure 3 uses the idea that is being already started in some projects, but in the simplest way possible to document it. It shows that all the requirements have to be testable so they can be used for both development bases during the whole sprint and test plan for the functional tests execution. All the lines show the possible test cases and the expected results (that would be to pass or to fail). The example shows a very small user story but this template can be followed for all the possible user stories on the sprint planning and for the whole sprint *backlog*, that in this new format, we are calling Test *Backlog*.

3.3. Preliminary Results

The approach proposed on this paper for user stories construction was effectively tested in one of the projects that the author has being working as Scrum Master. The format of the team to be select as study case was chosen exactly for being the most similar to what is found usually in the default agile teams (small team, multifunctional professionals and presence of a Product Owner and a Scrum Master). It is important to point out some of the main details and the structure of this team used as the study case:

- 6 people scrum team (3 developers, 2 web designers and 1 tester);

- Product Owner not fully allocated for the project;
- Scrum Master fully allocated for this team.

Table 2 shows the results that were obtained on this project and the number of opened tickets (during sprint development) was chosen as metric.

Sprint	Methodology	Defects/Issues opened during sprint development
1	**1**	**221**
2	**2**	**76**
3	**3**	**28**
4	**3**	**35**
5	**3**	**19**

Table 2. Project Results using the proposed approach

During the Sprint 1, a usual methodology was used to write the user stories (and that was called methodology 1). This one was more focused on the specification made in a very similar way to the old Use Cases model (from Rational Unified Process). The relevant factors to be cited are: the creation of the requirements only by the client (without scrum team help), many scope changes requested during the sprint (due to the lack on the prior specification) and user stories very low quality.

The migration from this model to the one it´s being proposed was actually started with an intermediate methodology (called here as 2). It was used during Sprint 2. The documents began to be performed with the team´s help. They´re still shaped like use cases but the long descriptions began to be replaced by a step by step testing.

The last three sprints (3 to 5) were performed using the methodology proposed on this paper (which we call 3). In this methodology, documentation and creation of user stories were made by the Product Owner with the help of the Test Analyst. The format adopted was to describe user stories as executable test cases, with steps and expected results as presented in Figure 3. A clear difference between the total of defects found in these sprints and the relevance of the proposal can be validated by comparing the results of sprints 1 to 5 on Table 2.

Finally, it is important to emphasize the fact that the only variable is the methodology used to create the user stories. Other factors were not changed, even the size of the sprints and the team members were kept the same so that the proposal could be validated with more consistence.

As a user I would like to have a carousel so that the main info of my Home page will be displayed to my final users.

- Check layout attached on IE8, IE9, FF(last version), Chrome(last version) and Safari 5 (pass)
- Resize the browser screen: Layout should adapt, marquee should keep the same (pass)
- Resize the browser screen: Should not affect the standalone text box (pass)
- Check if the time is displaying in hh:mm format (pass).
- Change the time manually (fail).
- The 3 banners in marquee are redirecting correctly
- Add text in the banner with 120 characters (pass).
- Add text in the banner with 118 characters (pass).
- Add text in the banner with 121 characters (fail).
- Add text in the banner with 121 characters (fail).
- Change the image from the banner for a psd image (fail).
- Change the image from the banner for a jpg image (fail).
- Check if the slider has 3 bullets (pass)
- Check if the sliders are linking to the correct 3 banners in each bullet (pass).

Figure 3. A User Story in the proposed format

It is possible to see on the Figure 4 the same image that is very famous on agile scenarios, but now, with the change to call the Product *Backlog* with a new nomenclature, "Test *Backlog*" (and Sprint Test *Backlog* in consequence). No other changes and no new step were added on the image (on purpose), since the idea is not to add new steps but to take advantage of the time for the *backlog* creation to do both works: requirement and test plan. This figure was done based on the original one.

Figure 4. Test *Backlog* in the Scrum Development
(adapted from MontainGoat, 2012)

3.2. Testing work – Day by day activities proposal

As previously stated, the reason to name this phase "Incremental Test" is to make it clear that every piece of code can be tested and there is no need to wait until the entire user story is developed to start the test phase. Also, the testers should not be blocked in the beginning of the sprint anymore, waiting for the final code to be delivered for tests. This idea of unblocking the tester is even clearer when proposing a new task of helping the PO on writing good and testable user stories.

These activities are described in the Table 3. The first column presents the data evolution during the Sprint, in the second column the activities planned in current agile projects are shown and the last column presents the proposed activities allowing to compare the changes in the test team activities during the Sprint with the way they are being run currently on the projects.

Date in the Sprint	Current Agile Projects	Activities Proposal
Day 0 (before the planning)	There is no tester participation on this phase.	Like it was described on 3.1, the tester should work together with the PO on the user stories so the backlog will have a test format, what can be called as "Test *Backlog*".

Day 1 (Planning Meeting)	Tests responsible is always giving the size for the test related tasks and just involved in quality discussions.	There is a gap in the planning meeting for the tester role. Tester can be the more suitable professional to understand what is the customer necessities (customer point of view) and also, can explain it technically to the development team (technical point of view). The tester can filter team questions and raise new blocks to the scrum master.
Day 2 (first sprint development day)	Tester is creating the test plans and not fully involved on the other team members work.	Testing from the very beginning is involved in all tasks. Since the test plan was already created (the requirement will be used for that now), the tester can have his time to work in pairs with developers to make sure the entire team is on the same page and all the bugs that can be avoided will be discussed at this point. Here is the perfect time for the tester to work to help improving the unit test, having a stronger unit test is also the best way to avoid defects and issues in the future (and it´s also a very good way to run what we are calling a incremental test).
Day 3 to N (sprint development period)	Functional tests are executed following the test plan created in the Day 2 of the sprint. When there is no user story fully	From the functional tests perspective, anything has to be changed: all the tests will be executed normally by the tester and also by the team, if it is necessary. From the regression tests

	ready to be tested (which means with no other task open but the test execution), the tester(s) is usually blocked.	perspective, this activity should be executed every time a new integration is done, not only on the last day of the sprint (also thinking on an incremental perspective).
Day N + 1 (last sprint development day)	Testers are still running functional tests and have no time to dedicate to the regression tests	Focus on the last Regression Tests execution. All the issues found on this phase should be prioritized and addressed to the next sprint. It´s not recommended fixing any issue during regression (all the work would need to be restarted in this case).

Table 3. Day by day tasks execution.

All these changes can make the day by day more productive and also, should keep the high quality in the deliverables. All the testers and team tasks and their day by day can be visualized on the Figure 5.

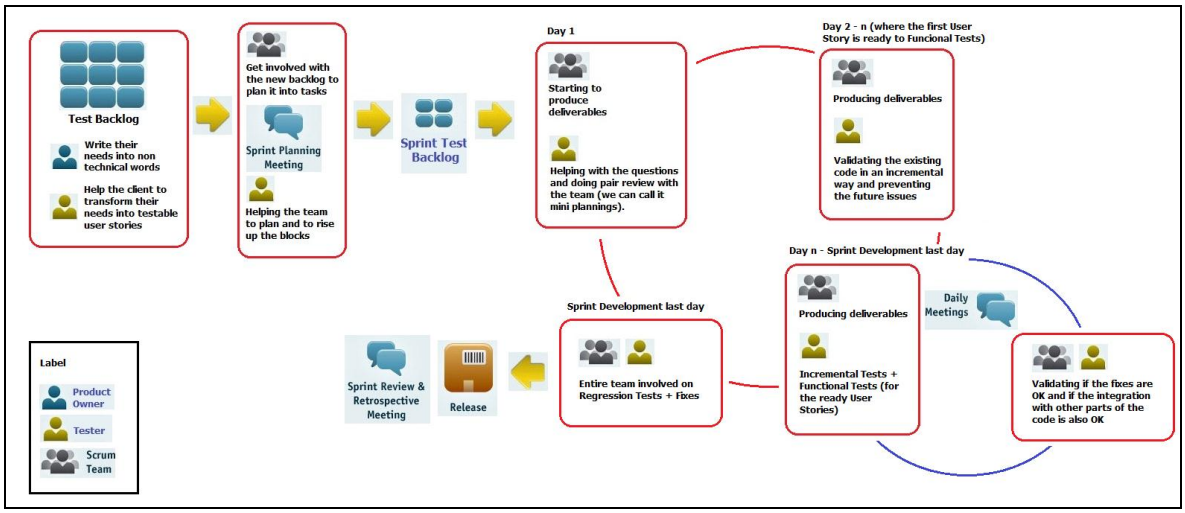


Figure 5. Proposed cycle with tasks and new *backlog* format.

3.3. Preliminary Results

The approach proposed on this paper for user stories construction was effectively tested in one of the projects that the author has being working as Scrum Master. The format of the team to be select as study case was chosen exactly for being the most similar to what is found usually in the default agile teams (small team, multifunctional professionals and presence of a Product Owner and a Scrum Master). It is important to point out some of the main details and the structure of this team used as the study case:

- 6 people scrum team (3 developers, 2 web designers and 1 tester);
- Product Owner not fully allocated for the project;
- Scrum Master fully allocated for this team.

Table 4 shows the results that were obtained on this project and the number of opened tickets (during sprint development) was chosen as metric.

Sprint	Methodology	Defects/Issues opened during sprint development
1	**1**	**221**
2	**2**	**76**
3	**3**	**28**
4	**3**	**35**
5	**3**	**19**

Table 4. Project Results using the proposed approach

During the Sprint 1, a usual methodology was used to write the user stories (and that was called methodology 1). This one was more focused on the specification made in a very similar way to the old Use Cases model (from Rational Unified Process). The relevant factors to be cited are: the creation of the requirements only by the client (without scrum team help), many scope changes requested during the sprint (due to the lack on the prior specification) and user stories very low quality.

The migration from this model to the one it´s being proposed was actually started with an intermediate methodology (called here as 2). It was used during Sprint 2. The

documents began to be performed with the team´s help. They´re still shaped like use cases but the long descriptions began to be replaced by a step by step testing.

The last three sprints (3 to 5) were performed using the methodology proposed on this paper (which we call 3). In this methodology, documentation and creation of user stories were made by the Product Owner with the help of the Test Analyst. The format adopted was to describe user stories as executable test cases, with steps and expected results as presented in Figure 3. A clear difference between the total of defects found in these sprints and the relevance of the proposal can be validated by comparing the results of sprints 1 to 5 on Table 2.

Finally, it is important to emphasize the fact that the only variable is the methodology used to create the user stories. Other factors were not changed, even the size of the sprints and the team members were kept the same so that the proposal could be validated with more consistence.

4. Conclusions

This work proposes a new format to prepare the User Stories including functional testing. Also, this new format proposal suggested the collaboration between the PO and the tester. They will help each other during the User Stories creation and both will help the team to better understand the requirements and user needs.

Most part of the issues that could be seen on scrum team work, related to test and product validation could be addressed if they work with less documentation and more alignment (communication) with the customer expectations. Creating the requirement together could be a very good way to address this issue and to have someone inside the development team with a very good vision of what is the real value to be achieved.

Another important issue that was found and could be better treated at this time is that some test teams spend a long period blocked during the sprint development waiting for the user stories to be totally ready for testing. Incremental tests idea and all the activities described on the subsection 3.2 can be a very good way to make the tester responsible to contribute for the project´s quality.

Results showed in Table 2 points to conclude that collaborations of the team during the user stories writing drastically decrease the number of defects/issues, since we observed almost 66% less defects/issues (from 221 down to 76) when only this variant was changed. Another important reduction of defects/issues was observed when the user stories were written using the new format proposed by this work. Again, 64% less defects/issues were observed (from 76 down to 28), culminating in a reduction of 75% (from 76 down to 19) when beyond the format changes the team acquires maturity using the new format.

This proposal can resemble to ATDD (Acceptance Test Driven Development), which makes sense, since some concepts were based on this technique. Developing the code based on a condition that has been created with a test view has much to do with the ATDD and TDD (Test Driven Development) and therefore, it is believed that an adaptation of the technique is very well fit to the Scrum.

The experience acquired as a Scrum Master in agile projects also helped in identifying the need for this adjustment, since in many cases, the client uses this time validating the planned tests as requirements and ends up finding gaps or even asking for changes in the starting . This idea can fit the real needs of the customer in the shortest time possible due to the fact that agile believe to be extremely important to generate the customer values.

If these teams start to use the activities proposed on this paper, most part of this blocks could be resolved and the resources would be more productive for the project as a whole. The key is a whole new mindset for the entire team working together to improve the quality work since the very first day, not only one test responsible can do that, but the scrum team together: improving requirement, unit tests, functional tests and acceptances tests.

References

Cohn, Mike. 2004. User Stories Applied. Addison Wesley, ISBN 0-321-20568-5.

Crispin, L. and Gregory, J. 2009. Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley. ISBN 0-321-53446-8.

Hendrickson, Elisabeth. 2008. Driving Development with Tests: ATDD and TDD, http://testobsessed.com/wp-content/uploads/2011/04/atddexample.pdf, June.

MountainGoat. 2012. Introduction to Scrum, http://www.mountaingoatsoftware.com, June.

Pfleeger, S. L. and Atlee, J.M. 2009. Software Engineering: Theory and Practice. Prentince Hall. ISBN 978-0136061694.

ScrumAlliance (2012) Official Website, https://http://www.scrumalliance.org, June.

Soares, Michel dos Santos. 2004. Agile Methodologies: Extreme Programming and Scrum for Software Development. System Information Eletronic Magazine, Vol. 3, Number 1.

Sommerville, Ian. 2010. Software Engineering, Pearson Addison-Wesley, 9th Edition.

APÊNDICE B – EXEMPLOS DE ESTÓRIAS

SCRUM (PRODUCT BACKLOG)

As a user I want to sign up the application (Smartphone)
Required Fields: Username, Password, Name, E-mail, User's photo field * User's photo field (this field will receive Facebook user's photo, if the user doesn't connect with FB, this photo will be an default placeholder) A - The username and the email are unique B - The required fields are: Username, Password and E-mail C - The username field must have at maximum 25 characters D - The username field can only contain letters, numbers and underscore E - The password field must have at maximum 64 characters F - The password field must have at minimum 5 characters G - The email field must have at maximum 64 characters H - The field email must follow the email format (e.g. text@text.text) I - The field name must have at maximum 50 characters (not allowed just spaces or spaces in sequence) J - The field password can't contain any spaces The keyboard must open as soon as the user enters in this screen When the user taps outside the keyboard and the cursor is not inside the field, the keyboard must close. There are two links below the fields that point to Privacy Policy and Terms of Services.

BDD (BACKLOG + TEST PLAN)

As a user I want to sign up the application (Smartphone)
Required Fields: Username, Password, Name, E-mail, User's photo field * User's photo field (this field will receive Facebook user's photo, if the user doesn't connect with FB, this photo will be an default placeholder) A - The username and the email are unique B - The required fields are: Username, Password and E-mail C - The username field must have at maximum 25 characters D - The username field can only contain letters, numbers and underscore E - The password field must have at maximum 64 characters F - The password field must have at minimum 5 characters G - The email field must have at maximum 64 characters H - The field email must follow the email format (e.g. text@text.text) I - The field name must have at maximum 50 characters (not allowed just spaces or spaces in sequence) J - The field password can't contain any spaces The keyboard must open as soon as the user enters in this screen When the user taps outside the keyboard and the cursor is not inside the field, the keyboard must close. There are two links below the fields that point to Privacy Policy and Terms of Services.

Test Plan – User Story 1

+Scenario 1: Create new account+

Given the user wants to create a new account

And his username and email are unique

And the data is valid

When user fills in all the fields

Then create a new account

+Scenario 2: Create new account+

Given the user wants to create a new account

And his username already exists but the email is unique

And the data is valid

When user fills in all the fields

Then shows an error message

+Scenario 3: Create new account+

Given the user wants to create a new account

And his username is unique but the email is already registered

And the data is valid

When user fills in all the fields

Then displays an error message

+Scenario 4: Create new account+

Given the user wants to create a new account

And his username and email are unique

But the e-mail is invalid

When user fills in all the fields

Then displays an error message

+Scenario 5: Create new account+

Given the user wants to create a new account

And his username and email are unique

But the username is invalid

When user fills in all the fields

Then displays an error message

+Scenario 6: Use FB photo+

Given the user wants to use his FB profile photo

And he has a FB account

When user logs into FB

Then upload his FB profile photo and insert into his new account

+Scenario 6: Use FB photo+

Given the user wants to use his FB profile photo

And but he has no FB account

When user tries to log into FB

Then the system won´t return the needed data and the user will see an error message

…

TEST BACKLOG (PROPOSAL)

As a user I want to sign up the application (Smartphone)

This screen must contains the following fields: Username, Password, Name, E-mail, User's photo field

* User's photo field (this field will receive Facebook user's photo, if the user doesn't connect with FB, this photo will be an default placeholder)

Also must contain a button that retrieve the Facebook information (Name, email and photo).

Consider the following rules:

A - The username and the email are unique

Enter a username that doesn't exist [PASS]

Enter a user that exists [FAIL]

B - The required fields are: Username, Password and E-mail

Tap to "Use your Facebook Info" (The application must return and the following fields must be filled with information: Email and Name. Also the photo must come from Facebook. Click on "Done.") [PASS]

Check that the user has received one activation email [PASS]

Check if user received any other e-mail [FAIL]

Check if the user didn´t receive an e-mail [FAIL]

C - The username field must have at maximum 25 characters

Enter a username with 24 chars [PASS]

Enter a username with 25 chars [PASS]

Enter a username with 26 chars [FAIL]

D - The username field can only contain letters, numbers and underscore

Enter a username with only letters [PASS]

Enter a username with only numbers [PASS]

Enter a username with letters, numbers and underscore [PASS]

Enter a user with one special char as ($) [FAIL]

Enter a username with space char in between [FAIL]

E - The password field must have at maximum 64 characters

F - The password field must have at minimum 5 characters

J - The field password can't contain any spaces

G - The email field must have at maximum 64 characters

Enter a password (more than 5 chars) [PASS]

Enter a password with 5 chars and no space [PASS]

Enter a password with 64 chars and no space [PASS]

Enter a password with 4 chars and no space [FAIL]

Enter a password with 65 chars and no space [FAIL]

Enter a password with 5 chars with space [FAIL]

H - The field email must follow the email format (e.g. text@text.text)

Enter an e-mail with no char [FAIL]

Enter an e-mail with 64 chars and like test@test.com [PASS]

Enter an e-mail with 65 chars and like test@test.com [FAIL]

Enter an e-mail with 64 chars and like test.test.com [FAIL]

Enter an e-mail with 64 chars and like www.test.com [FAIL]

I - The field name must have at maximum 50 characters (not allowed just spaces or spaces in sequence)

Enter a name with no char [FAIL]

Enter a name with 49 chars [PASS]

Enter a name with 50 chars [PASS]

Enter a name with 51 chars [FAIL]

Enter a name with space char only [FAIL]

Enter a name with two spaces in between [FAIL]

Consider the following message regarding the rules above:

A: Username already exists.

A: A Stash account with this email address already exists

B: Username, email and password required

B: Please enter a username.

C: Max. 25 characters

D: Username can only contain letters, numbers and underscore.

E: Max. 64 characters.

F: Min. 5 characters.

G: Max. 64 characters.

H: Not a valid email address.

I: Max 50 characters.

J: Password cannot contain any spaces.

The keyboard must open as soon as the user enters in this screen.

When the user taps outside the keyboard and the cursor is not inside the field, the keyboard must close.

There are two links below the fields that point to Privacy Policy and Terms of Services.