Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

Fabiana Braga de Assis

DESENVOLVIMENTO DE SOFTWARE DIRIGIDO POR TESTE DE ACEITAÇÃO

Belo Horizonte
2012

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de Software

DESENVOLVIMENTO DE SOFTWARE DIRIGIDO POR TESTE DE ACEITAÇÃO

por

Fabiana Braga de Assis

Monografia de Final de Curso

Prof. Rodolfo Ferreira Resende

Belo Horizonte
2012

Fabiana Braga de Assis

DESENVOLVIMENTO DE SOFTWARE DIRIGIDO POR TESTE DE ACEITAÇÃO

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Especialista em Informática.

Área de concentração: Engenharia de Software

Orientador(a): Prof. Rodolfo Ferreira Resende

Belo Horizonte
2012

À Deus,
aos professores,
aos colegas de curso e
aos meus familiares,
dedico este trabalho.

AGRADECIMENTOS

Inicialmente quero agradeço a Deus, por esta oportunidade.

Agradeço aos meus pais, pelo amor incondicional.

Aos meus professores, pelos conhecimentos adquiridos.

E finalmente aos colegas de curso pela convivência e trocas.

RESUMO

Algumas novas técnicas de desenvolvimento de software têm sido cada vez mais utilizadas nos últimos anos, em função de alcançarem uma boa relação entre custo e benefício na produção de software. Portanto, elas podem substituir os processos tradicionais, particularmente quando estes últimos não são muito adequados à realidade da organização. Este trabalho apresenta um estudo de uma das técnicas utilizadas em métodos “ágeis”: desenvolvimento de software dirigido por histórias de usuários transformadas em testes de aceitação. Essa técnica foca no desenvolvimento onde os testes de aceitação desempenham um papel importante no desenvolvimento de software e contribuem para a equipe encontrar respostas logo no início do desenvolvimento. Ter conhecimento de tecnologias e ferramentas atualmente é apenas parte do que é preciso para se construir bons softwares. Sabe-se que de nada adianta toda tecnologia disponível, se o software não for construído de forma coerente com as necessidades dos usuários.

Palavras-chave: Desenvolvimento Dirigido por Teste de Aceitação, Engenharia de Software, Metodologia Ágil.

ABSTRACT

Some new software development techniques have been increasingly used in recent years since they provide a good compromise between cost and benefit in the development of software. Therefore, they can replace the traditional methods, particularly when the latter are not very suitable to the reality of the organization. This work presents a study of the technical methods used in "agile" software development driven by user stories turned into acceptance tests. This technique focuses on the development where the acceptance tests play an important role in software development, and help find answers for the team early in development. Knowledge of current technologies and tools is only part of what it takes to build good software. It is known that of all the technology available is useless if the software is not built in a manner consistent with the needs of users.

Keywords*:* Acceptance Test Driven Development, Software Engineering, Agile Methodology.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ATDD	Acceptance Test Driven Development
CMMI	Capability Maturity Model Integration
ISO	International Organization for Standardization
MpsBr	Melhoria de Processos do Software Brasileiro
TDD	Test Driven Development
XP	Extreme Programming

SUMÁRIO

1 INTRODUÇÃO

Todo sistema tem um tempo de vida útil e uma série de fatores podem contribuir para aumentar ou diminuir este tempo, tais como, por exemplo, arquitetura, modelagem, tecnologia utilizada e aceitação do mercado. Neste tempo ou ciclo de vida, o sistema sofre várias alterações. (SOARES, 2011)

De acordo com Sommerville (2007), os negócios atualmente operam em um ambiente global sujeito a rápidas mudanças. Eles têm de responder às novas oportunidades e mercados, mudanças de condições econômicas e ao surgimento de produtos e serviços concorrentes.

Diante desse cenário, pequenas e médias organizações que desenvolvem software, e que não conseguem adotar nenhum processo tradicional devido ao custo dos recursos e falta de sistematização, desenvolvem software de baixa qualidade. Esses aspectos também contribuíram para os profissionais pensarem em novas técnicas para o desenvolvimento de software.

Segundo Soares (2011), foi pensando neste e em outros problemas, que Kent Beck apresentou em 2003, através do seu livro "*Test Driven Development*", uma técnica para desenvolver sistemas, baseada em testes que visa garantir a qualidade e a funcionalidade do software durante seu ciclo de vida. Beck et al. (2003, apud Neto, 2007), relata que o desenvolvimento de software dirigido por testes de aceitação (*Acceptance Test Driven Development* – ATDD) é uma técnica que surgiu recentemente.

Desenvolvimento de software dirigido por histórias de usuário transformadas em teste de aceitação é uma abordagem diferente das tradicionais geralmente adotadas por algumas empresas, pois nela temos os testes de aceitação presentes logo no início das atividades de um projeto com o objetivo de orientar o desenvolvimento do software.

Na Engenharia de Software que prescreve a adoção de métodos e práticas dos "agilistas" (Engenharia de Software Ágil), a forma dominante de representação inicial dos requisitos são relatos do usuário (PARK; FRANK, 2010). Esses relatos, também denominados de histórias de usuários, normalmente são frases escritas em uma linguagem comum que descrevem uma funcionalidade do software, e que o cliente usa no seu dia-a-dia. Em métodos ágeis, essas histórias são utilizadas em substituição aos casos de uso e das especificações de requisitos de software das metodologias clássicas.

Sob a perspectiva de modelos de desenvolvimento, já é comum o uso de histórias de usuários como um aspecto fundamental quando se trata de adotar o método *eXtreme Programming* (XP). A ideia consiste em fornecer uma visão de alto nível dos requisitos de um sistema, usadas como principal entrada de informações para estimativas e cronogramas, além de guiar a identificação de tarefas de desenvolvimento e conjunto de testes de aceitação (AMBLER, 2004; apud SANTOS, 2008).

Os testes de aceitação devem garantir o correto funcionamento de cada requisito a ser desenvolvido, e eles estão intimamente ligados com as histórias de usuário. Para cada história de usuário deve existir pelo menos um teste de aceitação, sendo ela declarada como terminada somente após ser executada por completo (o cliente sabe como o software deve se comportar para que a história seja dada por finalizada).

O ATDD é uma abordagem de desenvolvimento proposta pelos “agilistas”, onde a construção do software é guiada pelos testes de aceitação. Nesta abordagem, os testes de aceitação são criados de maneira colaborativa (todos os envolvidos contribuem com seu conhecimento) e descritos em uma linguagem comum que pode ser entendida por todos os membros da equipe. Assim, todos compartilham o mesmo entendimento do que deve ser feito, as restrições e as definições de "pronto". Aqui os testes são descritos em uma linguagem natural similar a linguagem do negócio e deve ser facilmente entendida pelo cliente, e os testes (critérios de aceitação) devem ser executáveis. (CAETANO, 2011)

Uma das práticas adotadas pelo ATDD é a participação efetiva do cliente na equipe do projeto, que juntos trabalham com a intenção de produzir um produto que traga o máximo de valor para a organização.

Esse trabalho é baseado principalmente no livro escrito pelos autores Grigori Melnik, Gerard Meszaros e Jon Bach. Quando não houver citação de referências, a fonte utilizada é o livro “*Acceptance Test Engineering Guide,Volume I: Thinking about Acceptance. How to Decide if Software is Ready for You and Your Customers*”, que ainda está em acesso antecipado (early access).

1.1 OBJETIVOS

O presente trabalho tem como objetivo geral apresentar um estudo sobre Desenvolvimento de Software Dirigido por Teste de Aceitação. São apresentados vários conceitos de técnicas de desenvolvimento que podem ser aplicadas em qualquer projeto de software, mas que utilizadas em conjunto nessa abordagem servem como forma de melhorar a prática de desenvolvimento no ATDD, e a qualidade do produto.

Dentre os objetivos específicos, é feita uma comparação entre o ATDD e as técnicas tradicionais (Desenvolvimento de Software Dirigido por Teste de Aceitação *versus* Desenvolvimento de Software Tradicional).

São feitas também algumas observações sobre a utilização do ATDD em relação às técnicas tradicionais usadas no processo de desenvolvimento de software. Descrevo ainda algumas razões que podem levar equipes a falharem com a adoção do ATDD.

1.2 JUSTIFICATIVA

A escolha do tema do presente trabalho é devido ao interesse cada vez maior nos métodos ágeis de desenvolvimento de software por parte das organizações.

Como as mudanças de requisitos e a falha de comunicação entre o cliente e o desenvolvedor de software são muito frequentes, torna-se muito difícil escrever, com exatidão e clareza, todas as características de um software.

O estudo será baseado em uma das possíveis soluções para esses problemas – desenvolvimento de software orientado por histórias de usuários transformadas em testes de aceitação.

Este estudo é relevante para promover a divulgação dessa prática de desenvolvimento de software por parte dos desenvolvedores.

1.3 ESTRUTURA DO TRABALHO

Os aspectos gerais do Desenvolvimento de Software Dirigido por Testes de Aceitação são apresentados na Seção 2. Nesta seção, algumas técnicas de desenvolvimento de software também foram pesquisadas, para que a abordagem das práticas utilizadas pudesse ser compreendida.

A Seção 3 contextualiza a técnica do desenvolvimento de software dirigido por teste de aceitação de forma mais detalhada. Nela são abordados conceitos do ATDD, e sucintamente apresenta suas etapas.

A Seção 4 apresenta uma breve comparação entre as técnicas tradicionais de desenvolvimento de software e as práticas usadas no desenvolvimento de software dirigido por histórias de usuários. Esta avaliação, além da literatura leva em conta a experiência profissional da autora.

A Seção 5 apresenta algumas observações finais sobre o ATDD, bem como algumas razões que podem levar as equipes a falharem na utilização da técnica apresentada como tema desse trabalho.

Finalmente, a Seção 6, conclui o trabalho, discutindo como algumas técnicas de desenvolvimento podem ser aplicadas juntamente com o ATDD.

2 DESENVOLVIMENTO DE SOFTWARE DIRIGIDO POR TESTES DE ACEITAÇÃO

A prática de desenvolvimento dirigido através dos testes de aceitação possui princípios simples, mas importantes o suficiente para que o software seja desenvolvido de forma simples, organizada, e que evolua.

Para tal, é necessário entender, num primeiro momento, o que o cliente realmente necessita e espera que o software faça, para depois então construí-lo. Adotando os testes para guiar o desenvolvimento, eles ajudam muito no entendimento do contexto, e evoluem na medida em que as necessidades e expectativas do cliente são esclarecidas.

Considerando que a comunicação entre o cliente e o desenvolvedor de software frequentemente é muito fraca, caso haja algum “mal entendido” (diferentes pessoas interpretam de maneira diferente os requisitos), os testes podem indicar que algo não ocorreu conforme esperado. Isso pode contribuir para a descoberta de novas funcionalidades que não haviam sido identificadas e com qualidade do produto (defeitos e não conformidade com requisitos tiram a confiança do cliente sobre o produto) e diminuir seu custo.

Sommerville (2007) diz que geralmente é benéfico investir esforço no projeto e na implementação para reduzir custos, já que adicionar novas funcionalidades depois da entrega é muito oneroso. Assim, todo trabalho executado durante o desenvolvimento para tornar o software mais fácil de ser compreendido e alterado, contribui para reduzir custos de manutenção.

Abaixo serão descritas algumas técnicas adotadas no desenvolvimento de software baseado em histórias de usuários transformadas em testes de aceitação.

2.1 HISTÓRIAS DE USUÁRIOS

Sabe-se que, na maioria das vezes, o cliente não consegue ou tem dificuldade de expor suas reais necessidades de forma clara e objetiva. Como consequência, o software desenvolvido não atenderá com qualidade suas exigências, pois este será desenvolvido baseado nas informações fornecidas inicialmente pelo cliente, e quase que

frequentemente, quando o software é entregue, o cliente diz que não era bem aquilo o esperado. A partir daí surgem novas ideias para melhorias ou “remendos” no software. (SOARES; FERREIRA, 2011)

Segundo Pádua Filho (2001), o ciclo de vida mais caótico do software é aquele que pode ser chamado de “codifica-remenda”. Partindo apenas de uma especificação (ou nem isso), os desenvolvedores começam imediatamente a codificar, remendando è medida em que os erros vão sendo descobertos.

Esses problemas de falhas de comunicação podem ser resolvidos através das histórias de usuário, que usadas de maneira eficiente e eficaz, criam uma linguagem comum entre todos os envolvidos no processo de desenvolvimento do software.

As histórias de usuário devem ser escritas de maneira simples o suficiente para serem compreendidas pelos clientes e desenvolvedores (normalmente elas são escritas pelos stakeholders e não pelos desenvolvedores). Elas também são usadas para estimar o esforço de sua implementação, para apoiar e incentivar o desenvolvimento.

Além das histórias de usuário definir um conjunto de testes de aceitação, elas também devem ser utilizadas para indicar a prioridade dos requisitos.

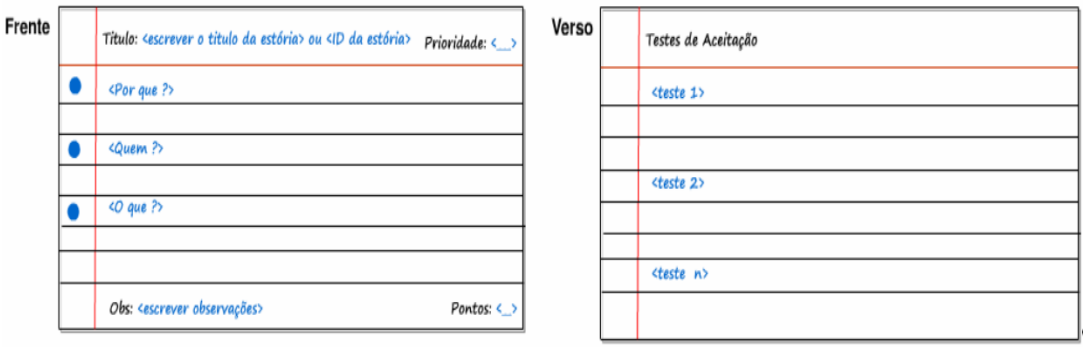

Figura 1 – Modelo para uma história de usuário

Fonte: Adaptada de http://www.slideshare.net/Ridlo/escrevendo-estrias-do-usurio-eficazes

Ao escrever histórias, alguns aspectos críticos devem ser lembrados, como os três C’s. De acordo, com Soares e Ferreira (2011), os três C’s são:

- Cartão (*Card*): o cartão possui frases que direcionarão o desenvolvimento de uma funcionalidade no sistema, nele podem conter notas, estimativas, observações e até comentários úteis para facilitar a estimativa.

- Conversa (*Conversation*): as conversas entre o cliente e desenvolvedores são essenciais, nelas surgem detalhes que muitas vezes ficam implícitos nas histórias.
- Confirmação (*Confirmation*): para ter certeza que o software desenvolvido está de acordo com as necessidades do cliente, é preciso executar os testes de aceitação descritos no cartão. Por isso, os testes merecem tanta atenção quanto o desenvolvimento.

Figura 2 – Os três C's de uma história de usuário
Fonte: Adaptada de http://www.slideshare.net/rkmael/estrias-do-usurio-parte-2

Soares e Ferreira (2011) relatam ainda que Bill Wake, autor do livro *Extreme Programming Explored and Refactoring Workbook*, em 2003 criou um conjunto de seis atributos que definem a eficácia de uma história, chamando de *INVEST*:

- Independente (***I****ndependent*): histórias devem ser independentes uma das outras; as duplicações e dependências entre as histórias devem ser removidas.
- Negociável (***N****egotiable*): histórias não são contratos, mas lembretes para discussões; elas não são imutáveis, pois representam uma conversa e não um requisito.
- Valiosa (***V****aluable*): histórias devem agregar valor para o cliente.
- Estimável (***E****stimatable*): os desenvolvedores devem ser capazes de estimar o tamanho das histórias; que se for muito complexa, deve ser dividida em histórias menores.
- Tamanho apropriado (Pequena) (***S****ized appropriatelly* (*Small*)): histórias grandes ou muito pequenas dificultam as estimativas. Devem ser quebradas ou agrupadas dependendo do caso.
- Testável (***T****estable*): histórias devem ser possíveis de serem testadas

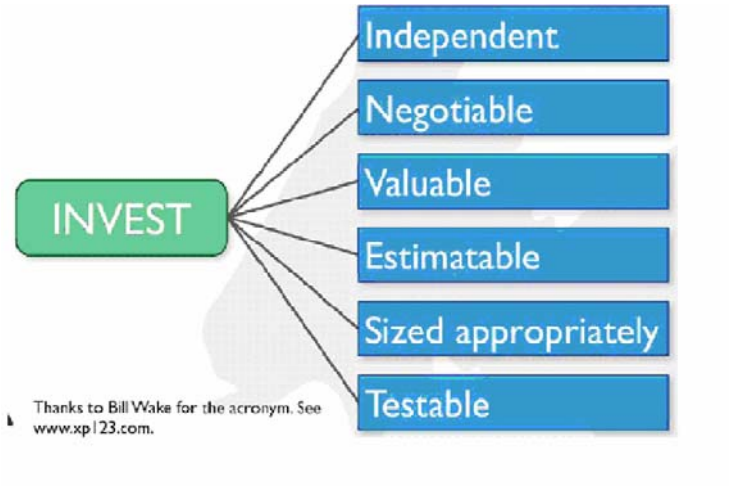

Figura 3 – Modelo *INVEST*

Fonte: http://www.mountaingoatsoftware.com/system/presentation/file/97/Cohn_SDWest2009_EUS.pdf

2.2 TESTES DE ACEITAÇÃO

O custo do software poderia ser minimizado se existissem mapeadas em testes, todas as expectativas do cliente, para orientar o desenvolvimento. Dessa forma, teríamos os melhores critérios para dizer que o software foi desenvolvido atendendo a todas as funcionalidades solicitadas pelo cliente.

De acordo com Sommerville (2007) o teste de aceitação é o estágio final do processo de teste, antes que o sistema seja aceito para uso operacional, e o sistema é testado com dados fornecidos pelo cliente, em vez de dados simulados de testes.

No desenvolvimento de software, temos o modelo V que enfatiza a importância de iniciar as atividades de teste desde o início do seu ciclo de vida.

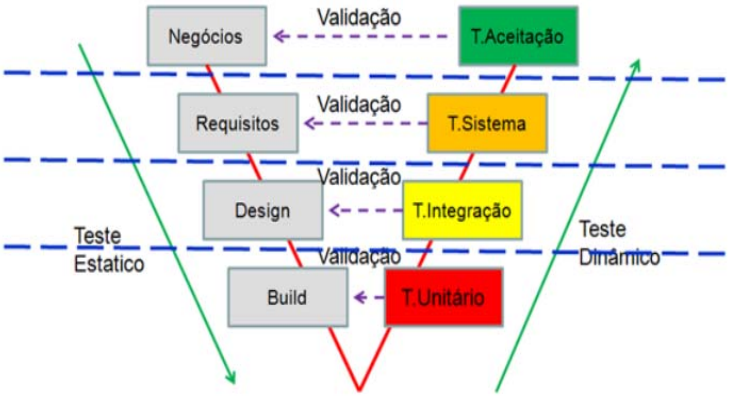

Figura 4 – Modelo V de Desenvolvimento de Software

Fonte: *http://www.devmedia.com.br/testes-de-software-niveis-de-testes/22282*

Esse modelo sugere que o teste de aceitação seja executado no final pelo cliente e usuário para validar se os requisitos foram implementados corretamente no sistema. Nas práticas usadas no ATDD, os testes de aceitação estão presentes no início das atividades de um projeto para orientar o desenvolvimento, servir como importante ferramenta para refinar a análise, e contribuir para que a equipe esteja sempre trabalhando naquilo que deverá ser entregue para o cliente.

Para Melnik (et al. 2009), teste de aceitação é parte do desenvolvimento eficaz – mudanças legítimas de requisitos geram um grande efeito sobre o tempo, papel e técnicas de testes de aceitação. Uma das tendências de desenvolvimento na última década é que o teste tem ocupado um lugar mais cedo, e mais freqüente no desenvolvimento - desenvolvimento dirigido por testes de aceitação.

Segundo Ferreira (2005), teste de aceitação é uma ferramenta poderosa de análise. Ao escrever os testes, é preciso ser bastante rigoroso e tentar prever todas as ações corretas e incorretas do usuário. Ao fazer isso, muitas vezes deparamos com situações que não haviam sido previstas pela análise inicial feita pela equipe. Nestes casos, é possível refinar esta análise inicial, e trazer novos elementos que a equipe de desenvolvimento deve passar a levar em consideração.

Os desenvolvedores escrevem testes para cada funcionalidade antes de codificá-las. Fazendo isso, eles aprofundam o entendimento das necessidades do cliente (o que aprimora a análise). Sabem até onde codificar cada funcionalidade (o que ajuda a manter o sistema simples) e passam a contar com uma massa de testes que pode ser usada a qualquer momento para validar todo o sistema (TELES, 2004).

Para koskela (2008), testes de aceitação são especificações para o comportamento desejado e funcionalidade de um sistema. Eles nos dizem, para uma determinada história de usuário, como o sistema lida com certas condições e insumos, e com quais tipos de resultados. Há uma série de propriedades que um teste de aceitação deve apresentar, segue algumas delas:

- Propriedade do cliente: porque o seu principal objetivo é especificar os critérios de aceitação para a história do usuário. Com o cliente escrevendo os testes de aceitação evita-se problemas comuns com os testes de aceitação escritos pelos desenvolvedores, já que estes geralmente especificam os aspectos técnicos da aplicação em vez de especificar o recurso em si.
- Sobre “o que” e não “como”: uma das características fundamentais que fazem histórias de usuários tão apropriadas para a entrega de valor no início é que

muitas vezes elas se concentram na descrição da fonte de valor para o cliente em vez da mecânica de como esse valor é entregue. Histórias de usuários se esforçam para transmitir as necessidades e desejos - o quê e porquê, e dar pouca atenção à implementação.

- Expresso na linguagem do domínio do problema: uma propriedade importante dos testes de aceitação é que eles usam a linguagem do domínio e do cliente em vez de linguagens que apenas o programador entende. Este é o requisito fundamental para ter o cliente envolvido na criação de testes de aceitação e ajuda muito com o trabalho de validação correta dos testes. Espalhar muito jargão técnico torna os testes mais vulneráveis (os desenvolvedores são atraídos para a tecnologia, em vez de a verdadeira questão de especificar a coisa certa).
- Conciso, preciso e inequívoco: usando uma linguagem própria do domínio, os testes de aceitação devem ser simples e concisos. Cada teste de aceitação deve ser escrito para verificar um único aspecto ou cenário relevante para a história do usuário. Devem ser testes organizados, fáceis de entender (menos ambíguos) e de traduzir para testes executáveis.
- Os testes de aceitação servem para validar se a história de usuário foi implementada corretamente: o ideal é que cada história de usuário esteja associada à um conjunto de testes de aceitação, pois são eles que indicam os resultados esperados e indesejados.

Teste de aceitação não deveria ser visto como algo novo. Desde o primeiro usuário ao primeiro programador já se dizia “Isso não está certo”. Testes de aceitação é parte do fluxo de trabalho de desenvolvimento, e sempre trazem diferentes valores, expectativas e vocabulário, já que é inevitável algum “mal-entendido” entre as pessoas.

3 ATDD

O ATDD é uma forma de comunicar requisitos usando testes. O propósito do ATDD é facilitar uma melhor comunicação entre os clientes e a equipe de desenvolvimento, reduzindo as ambiguidades dos requisitos. (PARK; FRANK, 2010)

De acordo com Koskela (2007, apud SANTOS, 2010), ATDD é a prática que estende o uso de TDD ao nível de testes de aceitação.

Embora o nome sugira que TDD (Desenvolvimento Dirigido por Testes) esteja relacionado apenas a testes, é uma técnica de projeto de software. TDD traz os testes para o primeiro passo, e divide o ciclo em várias pequenas iterações. (SANTOS, 2010)

Figura 5 – Relação entre TDD e ATDD

Fonte: Adaptada de http://www.methodsandtools.com/archive/archive.php?id=72p9

Apesar do TDD e ATDD estarem relacionados a desenvolvimento dirigido por testes e terem como requisito a implementação dos testes antes do código, são duas abordagens diferentes.

No TDD o foco está em testar a funcionalidade de baixo nível, garantindo padrões de design e princípios (como se refere à qualidade interna do código, têm-se um código mais flexível e de fácil refatoração) e são voltados para desenvolvedores.

No ATDD o foco está relacionado com as histórias de usuários que descrevem a funcionalidade a partir da perspectiva do cliente, sendo destinado principalmente para as

partes interessadas não-programadores (não possui uma linguagem padrão para a descrição das funcionalidades).

O desenvolvimento dirigido por testes de aceitação (ATDD) envolve criar testes antes do código, e tais testes representam expectativas quanto ao funcionamento do software. Em ATDD, a equipe cria um ou mais testes de aceitação para uma funcionalidade que está sendo atualmente trabalhada. Normalmente estes testes são discutidos e capturados quando a equipe estiver trabalhando com o cliente para compreender uma história. (HENDRICKSON, 2008b)

A forma tradicional de adicionar funcionalidades ao sistema é: primeiro escrever os documentos de requisito, proceder com a implementação, e somente depois é que o cliente realiza testes de aceitação. ATDD move a fase de testes para antes da implementação. É preciso que o requisito seja descrito de forma explícita e não ambígua, pois ele será a forma concreta de verificar se as necessidades do cliente serão atendidas. (KOSKELA, 2007; apud SANTOS, 2010)

Segundo Park et al. (2010), a idéia do ATDD já está circulando na comunidade de engenharia de software ágil há uma década, tendo começado com a publicação do livro de Beck em 1999 – relatam ainda que têm visto a indústria aceitar e praticar muitos dos conceitos ágeis, em diferentes níveis, apesar de suas objeções iniciais.

Para Larman et al. (2010), o ATDD consiste em três fases: discutir os requisitos; desenvolvê-los em concordância durante a iteração; e entregar os resultados as partes interessadas para aceitação.

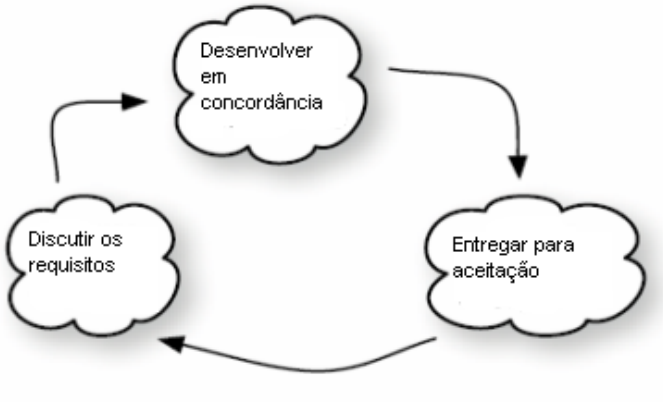

Figura 6 – ATDD: Visão Geral

Fonte: Adaptada de http://www.slideshare.net/pekkaklarck/atdd-using-robot-framework

ATDD não deve ser considerado como sendo uma técnica de teste, mas sim como uma abordagem voltada para ajudar no processo de desenvolvimento do software.

3.1 FASES DO ATDD

ATDD é uma prática em que toda a equipe de forma colaborativa discute critérios de aceitação, com exemplos, e em seguida, os refina em um conjunto de testes de aceitação concretos antes de começar o desenvolvimento. É uma boa maneira de garantir que todos possuam um entendimento compartilhado do que está sendo construindo. É também uma boa maneira de garantir a definição comum de concluído. (HENDRICKSON, 2008a)

3.1.1 DISCUTIR

Essa fase consiste em discutir com a equipe as restrições, premissas, expectativas, etc. para definir os critérios de aceitação. (CAETANO, 2011)

Os participantes são o cliente ou representante, e pessoas interessadas que possuem informações importantes sobre os requisitos.

Ferramentas como FITnesse pode ser facilmente utilizada pelos participantes para definir as histórias de usuário de uma maneira precisa, e certificar que os desenvolvedores irão implementar exatamente o que elas significavam. Eles também podem acompanhar o estado e a evolução do projeto, pois através da execução dos testes podem ver exatamente qual funcionalidade já está implementada.

Nesta fase o foco deve ser na discussão e descoberta de requisitos, mais do que sobre os testes reais, ou seja, o propósito das discussões deve ser a descoberta dos requisitos.

3.1.2 REFINAR

Refinar os critérios de aceitação em um conjunto concreto de cenários/exemplos de uso descrevendo o comportamento esperado da aplicação em uma linguagem comum a todos os membros da equipe (CAETANO, 2011)

Segundo Larman et al. (2010), Elisabeth Hendrickson considerada essa etapa separada em ATDD.

3.1.3 DESENVOLVER

Em geral, uma história dada pelo cliente representa uma parte da funcionalidade do software.

Os exemplos especificados orientam o trabalho de implementação, e a automação dos exemplos pode ser feita em paralelo com a implementação do recurso. (LARMAN et al., 2010)

Das histórias de usuários, são extraídos os testes de aceitação.Quando capturados em um formato suportado por algum framework de automação de testes funcionais, como o *FIT* ou o *FITnesse*, os desenvolvedores podem automatizar os testes escrevendo código de suporte ou instalação (“*fixture*”) de como eles implementam a funcionalidade. (HENDRICKSON, 2008b). Uma instalação (“*fixture”*), é o código de suporte que implementa nos métodos das classes do software as ações descritas em linguagem natural. (CAETANO, 2011)

Essa fase consiste em transformar os testes de aceitação em testes/ especificação. Os testes de aceitação quando automatizados podem ser usados como medida de progresso e indicador dos níveis de qualidade (CAETANO, 2011)

3.1.4 ENTREGAR

As funcionalidades implementadas são demonstradas a todos os interessados. Os exemplos originais são executados, dando um feedback que serve como entrada para a próxima fase de discussão. (KLARCK; HARKONEN, 2010)

4 DESENVOLVIMENTO DE SOFTWARE DIRIGIDO POR TESTES DE ACEITAÇÃO X DESENVOLVIMENTO TRADICIONAL DE SOFTWARE

O Desenvolvimento Dirigido por Teste de Aceitação se diferencia em diversos pontos das técnicas tradicionais normalmente utilizadas no processo de desenvolvimento de software. Apesar da técnica ATDD ser mais voltada para metodologias propostas pelos “agilistas”, ela pode também ser aplicada em qualquer projeto de desenvolvimento de software. A prática do ATDD também não exclui o uso de ferramentas e outros elementos vindos do desenvolvimento tradicional de software.

Nesta seção, algumas diferenças são mostradas fazendo uma breve comparação entre diferentes técnicas de desenvolvimento, mostrando as suas diferenças e ressaltando possíveis vantagens de se utilizar o ATDD.

Uma prática comum da metodologia de desenvolvimento tradicional é utilizar documentação para Especificação dos Requisitos para descrever as funcionalidades do software e Casos de Uso para mostrar a iteração entre o usuário e o sistema. Na técnica ATDD, usa-se o conceito de Histórias de Usuários, que foca nas necessidades dos clientes e não contêm detalhes técnicos como tecnologia, arquitetura, etc. Ou seja, contêm apenas os detalhes suficientes para que haja entendimento entre o cliente e os desenvolvedores, criando uma linguagem comum que facilita o entendimento, e melhora a comunicação entre os envolvidos no processo do desenvolvimento do produto.

Nas técnicas de desenvolvimento tradicional, a criação e execução dos testes de aceitação são realizadas por um grupo de profissionais distintos dos que realizam a implementação das funcionalidades do software – desenvolvedores x testadores. Em contrapartida, o ATDD é conduzido através dos testes de aceitação associados a cada história de usuário criada, ou seja, a funcionalidade é testada pelo próprio desenvolvedor, e essa tarefa deve ser repetida até que todos os testes passem e a história de usuário possa ser dada como concluída.

O desenvolvimento tradicional normalmente sugere que o ciclo de execução dos testes seja feito em partes, envolvendo tanto os testes de caixa branca (testes de unidades que testam a estrutura interna do código), quanto os de caixa preta (testes realizados no final do desenvolvimento, e que não requerem conhecimento de programação ou lógica). Aqui ainda temos os testes de aceitação, que são realizados

pelos usuários finais com o objetivo de verificar se o software atende aos requisitos especificados. O ciclo de execução dos testes no ATDD envolve os testes de aceitação que são realizados no início do desenvolvimento, com o intuito de orientar o próprio desenvolvimento do software.

Tradicionalmente, os testes são realizados após a implementação do software. Por sua vez, o ATDD tem como premissa guiar o desenvolvimento através dos testes de aceitação, ou seja, os testes são escritos antes da implementação e dizem exatamente o que deve ser implementado (garante que a etapa de teste participe ativamente de todo o ciclo de desenvolvimento).

Podemos ainda citar como diferença, a forma como o código é organizado. Na tradicional, como dito anteriormente, os testes são realizados ao final do ciclo de desenvolvimento. Nesse momento, as falhas encontradas no software são retornadas ao desenvolvedor. Como o mesmo possui um tempo menor para acertar a demanda desses problemas, ele não se preocupa tanto com a organização do código, tornando a manutenibilidade do produto final prejudicada. Utilizando o ATDD, o código se torna mais organizado em função do desenvolvimento iterativo e incremental, com a implementação apenas do que foi solicitado.

Outro fator a ser observado é que no ATDD temos a participação direta do cliente junto à equipe de desenvolvimento, para que ambos possam trabalhar em conjunto com o objetivo de produzir um software de qualidade. Mesmo com o objetivo de também produzir um produto de valor, sabemos que tradicionalmente não existe a participação efetiva do cliente no processo de desenvolvimento.

Apesar de existir uma clara diferença entre ATDD e as técnicas tradicionais, nos dois casos, para que o produto final seja entregue com menos defeitos, é importante priorizar a qualidade dos cenários de teste criados, e garantir que as solicitações de funcionalidades por parte do cliente sejam atendidas.

Veja a seguir, um esquema simplificado da comparação apresentada:

Desenvolvimento de Software (Técnicas Tradicionais)	**Desenvolvimento de Software Dirigido por Testes de Aceitação**
Utiliza documentação para Especificação dos Requisitos e Casos de Uso	Usa o conceito de Histórias de Usuários, que foca nas necessidades dos clientes
Requisitos são especificações abstratas	Requisitos podem ser vistos como histórias de usuários testáveis
Documentos de ⇨ requisito implementação ⇨ cliente realiza testes de aceitação	Envolve os testes de aceitação desde o início do desenvolvimento
A criação e execução dos testes de aceitação são realizadas por profissionais que não fazem a implementação das funcionalidades do software	A funcionalidade dos testes de aceitação, associadas a cada história de usuário é testada pelo próprio desenvolvedor
Ciclo de execução dos testes é feito em diferentes etapas - testes caixa-branca e caixa-preta, e testes de aceitação	Os testes de aceitação guiam o desenvolvimento (escritos antes da implementação e dizem o que deve ser implementado)
Organização do código prejudicada por causa dos "remendos"	Código mais organizado em função do desenvolvimento iterativo e incremental
Papéis definidos na execução das tarefas (desenvolverdor x testador)	O sucesso do projeto depende de toda a equipe, pois todos trabalham juntos
Não existe a participação efetiva do cliente no processo de desenvolvimento	Participação direta do cliente durante todo o processo de desenvolvimento

Tabela 1 – Esquema simplificado da comparação

5 ALGUMAS OBSERVAÇÕES IMPORTANTES NO ATDD

Um dos maiores problemas do ATDD é que, por ser uma técnica ainda não muito conhecida e aplicada nas organizações, as equipes nem sempre acreditam na sua eficiência e eficácia, e não reconhecem que se aplicada de forma certa e organizada podem gerar alguns benefícios.

Normalmente quando se fala em software com qualidade pensa-se em modelos de maturidade de software (CMMI e Mps-Br) ou certificações (ISO). No cenário atual, muitas organizações não querem apenas ver o produto pronto e implantado, elas querem seu software funcionando e sem defeitos, com um custo menor e que atendam as demandas do mercado.

Hendrickson (2008b) relata que as equipes que experimentam ATDD normalmente concluem que apenas o ato de se definir testes de aceitação ao discutir requisitos resulta numa melhor compreensão destes requisitos. Porém, utilizar o ATDD realmente não é uma tarefa tão simples quanto parece. Num primeiro momento é preciso quebrar, no mundo do desenvolvimento de software, o paradigma de que os testes somente devem ser realizados no final da implementação, e ainda fazer com que os desenvolvedores assimilem a ideia de implementar por partes, (cada história de usuário deve ser codificada e finalizada antes da próxima ser executada). Sem essas premissas, a implantação do ATDD pode trazer mais perdas do que benefícios.

Outra questão que deve ser analisada antes da implantação da técnica é se existe tempo suficiente para o conhecimento e aprendizado por parte da equipe. É importante avaliar com cuidado os riscos, o custo, o prazo de entrega do projeto para que seja realizado um planejamento dentro da realidade prevista.

Koskela (2010), aponta para algumas razões que levam as equipes a falharem com o ATDD:

- Não colaborar: testes de aceitação precisam ser produzidos de forma colaborativa e acordados. Clientes, desenvolvedores e testadores todos devem contribuir, cada um com sua própria área de especialização.
- Focar em “como” ao invés “do que”: especificações do negócio devem ser claras e compreensíveis, e permitir que a equipe possa implementar a melhor solução possível sem impor a concepção ou implementação. Muitas vezes os testes descrevem como algo deve ser implementado e não o que deve ser

implementado. Testes como estes são de nível muito baixo, de difícil entendimento.

- Focar em ferramentas: algumas equipes se concentram demais em ferramentas e recursos de ferramentas específicas, ignorando coisas que naturalmente não se encaixam em seu conjunto de ferramentas escolhido. Isso faz com que a especificação fique vaga nas partes que não são cobertas por uma ferramenta particular. Em vez disso, as equipes devem se concentrar sobre as especificações de negócios e depois escolher a ferramenta certa para trabalhar. Em particular, as ferramentas não devem desempenhar um papel importante na oficina especificação.

6 CONCLUSÃO

O desenvolvimento de softwares exige técnicas eficientes e eficazes. Alguns aspectos como incompreensão dos requisitos, dificuldade em responder as mudanças frequente dos requisitos, utilização de processos pesados em documentação, dentre outros, levaram os profissionais a pensar em novas técnicas de desenvolvimento de software.

No desenvolvimento de software dirigido por testes de aceitação, os testes são usados para guiar o desenvolvimento e dizer exatamente aquilo que deve ser codificado – o que muda a maneira de escrever um código.

A adoção do ATDD pode ser interessante pelo fato dos testes fazerem parte do ciclo de vida do projeto antes mesmo da etapa do desenvolvimento. Escrever um teste para algo que ainda não foi codificado pode ser uma técnica que ajuda o desenvolvedor a focar na implementação do que realmente é necessário.

Aparentemente são regras simples, mas existe uma mudança de paradigma para aqueles profissionais acostumados com a utilização dos processos tradicionais, e da inversão de algumas tarefas. E relacionado a definição de requisitos, também deve existir uma mudança na mentalidade dos clientes já que os requisitos não são mais especificações abstratas, e sim histórias de usuários que se transformam em testes.

No ATDD podemos dizer que o sucesso do projeto depende de toda a equipe, pois todos trabalham juntos. Todos são responsáveis pelos riscos, ou seja, o desenvolvedor não é o único responsável pela implementação da funcionalidade, nem o testador o único responsável pelos testes do software – práticas que não muito vistas na metodologia tradicional.

REFERÊNCIAS

CAETANO, Cristiano. *The Developer Conference*. ***Automação de Testes de Aceitação com BDD (Behavior Driven Development) e ATDD (Acceptance Test Driven Development)*** (2011). Disponível em: < http://www.slideshare.net/cristianocaetano > Acesso em: 16 nov. 2011.

FERREIRA, Rafael de Faria. Palestra: **XP: Teste Aceitação x Teste de Unidade** Universidade Federal de Santa Catarina Maio 2005. Disponível em: < http://xp.edugraf.ufsc.br/bin/view/XP/TesteAceitacaoXtesteUnidade > Acesso em: 14 jan. 2012.

PÁDUA FILHO, Wilson. Engenharia de Software.Fundamentos Métodos e Padrões. 2. Ed. LTC: Livros Técnicos e Científicos, 2001.

HENDRICKSON, Elisabeth. ***Acceptance Test Driven Development (ATDD): an Overview*** *(2008a)*. Disponível em: < http://testobsessed.com/blog/2008/12/08/acceptance-test-driven-development-atdd-an-overview/ > Acesso em: 22 mai. 2012

HENDRICKSON, Elisabeth. ***Driving Development with Tests: ATDD and TDD***. *An updated version of the materials submitted for my presentations at STANZ 2008 and STARWest 2008* (2008b). Disponível em: < http://testobsessed.com/wp-content/uploads/2011/04/atddexample.pdf > Acesso em: 09 dez. 2011.

KLARCK, Pekka; HARKONEN, Janne. ***Acceptance Test-Driven Developmente using Robot Framework*** (2010). Disponível em: < http://www.slideshare.net/pekkaklarck/atdd-using-robot-framework >. Acessado em: 21 abr. 2012.

KOSKELA, Lasse.Artigo: Teste de Software, Gerenciamento de Projetos, Empregos. Baseado em um capítulo do livro "*Practical TDD and Acceptance TDD*" **Revista de Desenvolvimento de Software - Programação** (2008). Disponível em : <

http://www.methodsandtools.com/archive/archive.php?id=72 >. Acessado em: 23 abr. 2012

KOSKELA, Lasse. *Acceptance* ***Test Driven Development.*** *Scrum Gathering* Orlando (2010). Disponível em: < http://pt.scribd.com/doc/28792339/Acceptance-Test-Driven-Development-Lasse-Koskela-at-Scrum-Gathering-Orlando-2010 > 29 abr. 2012.

LARMAN, Craig; VODDE Bas. Artigo: ***Acceptance Test-Driven Developmente with Robot Framework*** (2010) p. 1 - 15. Disponível em: < http://code.google.com/p/robotframework/wiki/ATDDWithRobotFrameworkArticle >. Acessado em: 21 abr. 2012.

MELNIK, Grigori; MESZAROS, Gerard; BACH, Jon. *Acceptance Test Engineering Guide*. Volume I: *Thinking about Acceptance. How to Decide if Software is Ready for You and Your Customers*. *Microsoft Patterns & Practices*, 2009. (early access)

NETO, Osório Lopes Abath. **Desenvolvimento de Software Guiado por Testes de Aceitação usando *EasyAccept*** . 2007. 125 f. Dissertação (Mestrado em Ciência da Computação) - Centro de Engenharia Elétrica e Informática, Universidade Federal de Campina Grande, Campina Grande, 2007. Disponível em: < http://docs.computacao.ufcg.edu.br/posgraduacao/dissertacoes/2007/Dissertacao_OsorioLopesAbathNeto.pdf > Acesso em: 21 mar. 2012.

PARK, Shelly; FRANK, Maurer.Artigo: ***An Extended Review on Story Test Driven Development***. Departamento de Ciência da Computação, Universidade de Calgary, Calgary, Fevereiro 2010. Disponível em: < http://dspace.ucalgary.ca/bitstream/1880/47762/1/2010-953-02.pdf > Acesso em: 01 mai. 2011.

SANTOS, Rafael Liu. **Emprego de *Test Driven Development* no Desenvolvimento de Aplicações.** 2010.124 f. Monografia (Bacharelado em Ciência da Computação) – Instituto de Ciências Exatas, Universidade de Brasília – UnB, Brasília, 2010. Disponível em: < http://monografias.cic.unb.br/dspace/bitstream/123456789/258/1/monografia.pdf > Acesso em: 21 mar. 2012.

SANTOS, Venícios Gustavo. Artigo: Utilização de *Storytelling* como Ferramenta de Aquisição de Requisitos em Processos de Desenvolvimento de Software apoiados em Modelos Ágeis: o uso apoiado no *Extreme Programming* **Revista e-Tec**. Belo Horizonte, v. 1, n. 1, p. 1-14. Novembro 2008. Disponível em: < http://revistas2.unibh.br/index.php/dtec/article/view/440 > Acesso em: 16 jan. 2012.

SOARES, Ismael; FERREIRA, Luiz. Artigo: Melhorando a Comunicação com Estórias de Usuários. **Revista Java Magazine**. Edição 86, p. 81 - 91, Junho 2011. Disponível em: < http://www.devmedia.com.br/melhorando-a-comunicacao-com-estorias-do-usuario-java-magazine-86/18785 > Acessado em: 24 abr. 2012.

SOARES, Ismael. Artigo: Desenvolvimento orientado por comportamento (BDD). Um novo olhar sobre TDD **Revista Java Magazine**. Edição 91, p. 73 - 84, Junho 2011. Disponível em: < http://www.devmedia.com.br/desenvolvimento-orientado-por-comportamento-bdd-artigo-java-magazine-91/21127 > Acessado em: 24 abr. 2012.

SOMMERVILLE, Ian. Engenharia de Software. 8. Ed. São Paulo: Addison Wesley, 2007.

TELES, Vinícius Manhães. *Extreme Programming*. Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. Desenvolvimento Guiado pelos Testes (2004). Disponível em: < http://www.novateceditora.com.br/livros/extreme/capitulo8575220470.pdf > Acesso em: 21 nov. 2011.