UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

MORGANA CADORIN NAZÁRIO

CONSTRUÇÃO DE UM PROCESSO UTILIZANDO O CONCEITO DE TEST DRIVEN DEVELOPMENT (TDD) COMO APOIO A MELHORIA DA QUALIDADE NO DESENVOLVIMENTO DE SOFTWARE

CRICIÚMA

2014

MORGANA CADORIN NAZÁRIO

CONSTRUÇÃO DE UM PROCESSO UTILIZANDO O CONCEITO DE TEST DRIVEN DEVELOPMENT (TDD) COMO APOIO A MELHORIA DA QUALIDADE NO DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. MSc. Gustavo Bisognin

CRICIÚMA

2014

MORGANA CADORIN NAZÁRIO

CONSTRUÇÃO DE UM PROCESSO UTILIZANDO O CONCEITO DE TEST DRIVEN DEVELOPMENT (TDD) COMO APOIO A MELHORIA DA QUALIDADE NO DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau bacharel no Curso de Ciência da Computação Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Engenharia de Software

Criciúma, 27 de Junho de 2014.

BANCA EXAMINADORA

Prof. Gustavo Bisognin – MSc. - (UNESC) – Orientador

Prof. Gilberto Vieira da Silva – Esp. - (UNESC)

Prof.ª Ana Claudia Garcia Barbosa - MSc. - (UNESC)

Dedico este trabalho à minha família, base de toda a minha existência e valor. Sem seus ensinamentos e mais importante, sem seu carinho nunca teria chegado até aqui.

AGRADECIMENTOS

À Deus acima de tudo, agradeço pela saúde, alegria e força nos momentos mais difíceis. Por me proporcionar uma vida digna e com uma família incrível.

Agradeço a minha querida mãe, Varlini, que amo mais que tudo nesta vida, e por sempre dar seu melhor por mim, não há palavras suficiente para demostrar o grande amor e expressar minha eterna gratidão por ela. Agradeço por compreender a minha ausência, e por segurar a saudade naqueles momentos que o tempo não permite uma breve visita.

E a minha admiração, amor e dedicação é dele, meu namorado, noivo e futuro marido Fabricio, que desde que entrou em minha vida, só me trouxe alegria, seu companheirismo simplesmente é o combustível em minha vida e torna meus dias os mais felizes. Ele merece aquele agradecimento cheio de afeição, porque é meu orgulho, meu espelho, uma pessoa de caráter, que luta por seus sonhos e comigo, celebra cada momento de conquista de sua vida. Muito obrigada amor.

Aos meus irmãos, Roan e Sarita, que são parte de minha vida, que sempre estiveram caminhando junto a mim, nos momentos tristes e felizes, juntos somos mais fortes. Aos meus “cunhados” Edson, Nice, Francieli e ao meu sogro Luis Carlos que fazem parte da família e são pessoas que tenho apreço e carinho. Não posso de deixar de agradecer aos meu pequenos, Dionathan e Emilyn, o amor da tia por vocês é tão grande, vocês são a luz no meu caminho.

Não posso deixar de agradecer meu orientador, prof. MSc. Gustavo Bisognin, pela confiança e por estar presente nos momentos que necessitei de seu apoio, nas orientações, nas conversas informais e pela grande amizade, e por ser um exemplo a seguir.

E finalmente a todos aqueles que participam de minha vida de algum modo, amigos, colegas, professores e conhecidos. A todos muito obrigado!

.

.

“Sempre agradeça a Deus pela vida, pois o resto você corre atrás e conquista.”

Autor desconhecido

RESUMO

Com o crescimento de novas tecnologias, fabricas de softwares atuam fortemente na qualidade de seus produtos desenvolvidos. A busca por soluções é perseverante e o surgimento de novos processos, técnicas e ferramentas auxiliam na concepção de software, no entanto, não garantem a qualidade do software e a entrega dentro do prazo. Para minimizar problemas frequentes e retorno de possível erros antecipados, as metodologias ágeis entram em cena, e propõem a obtenção de resultados práticos em um período menor, assim, tirando o foco do processo e o colocando no produto e nas pessoas que o produzem. Este tipo de metodologia envolve uma quebra de paradigma, que nem sempre é fácil, mas que tem trazido resultados expressivos às empresas que conseguem incorporar estes métodos ágeis. Um desses métodos ágeis, conhecido por Programação extrema (XP) se tornou bastante popular entre desenvolvedores de software, uma vez que, discute práticas focadas no código. Desenvolvimento Guiado por Testes (TDD) foi originado através da XP, herdando diversos aspectos para o desenvolvimento de software, essa metodologia tem foco em pequenos *feedback* sobre o código. TDD é importante no desenvolvimento de software, uma vez que, o programador escreve o teste antes de codificar código do requisito. Neste âmbito, TDD é para o desenvolvedor um auxilio durante o processo de criação do projeto de classes, fazendo-os criar classes menos acopladas e mais coesas. Para realizar os testes é necessário aplicar a prática de testes unitários, que tem como apoio ferramentas automatizadas, com objetivos de obter *feedback*, ajudando a garantir tanto a qualidade externa (aquela perceptível pelo usuário) quanto a qualidade interna (aquela perceptível pelo desenvolvedor). A capacidade de executar continuamente os testes, permite a identificação imediata de possíveis alterações indesejadas no software. O trabalho desenvolvido visa por qualidade e busca por respectivas informações sobre técnicas ágeis para o desenvolvimento de software, após o estudo e compreensão da metodologia TDD, foi realizado a construção de um processo de desenvolvimento que tem como foco a utilização do ciclo TDD para garantir a qualidade do código implementado e, por consequência, minimizar falhas. Com os testes unitários automatizados na ferramenta jUnit, os scripts podem ser executados em qualquer momento, dessa forma, constatar futuras alterações no sistema, com mais segurança e estabilidade aos desenvolvedores.

Palavras-chave: Desenvolvimento guiado por teste. Qualidade externa e interno do software. Testes unitários. Automatização de Testes.

ABSTRACT

With the growth of new technologies, software factories act strongly on the quality of the products developed. The search for solutions is persevering and the emergence of new processes, techniques and tools help design software, however, do not guarantee software quality and delivery on time. To minimize common problems and possible return of anticipated errors, agile methodologies come into play, and propose obtaining practical results in a shorter period, thus taking the focus of the case and putting in the product and the people who produce it. This type of methodology involves a paradigm shift, which is not always easy, but it has brought impressive results for companies that can incorporate these agile methods. One such agile methods, known as eXtreme Programming (XP) has become quite popular among software developers, since the code discusses focused practices. Guided by Tests Development (TDD) was originated through XP, inheriting various aspects of software development, this methodology focuses on small feedback about the code. TDD is important in the development of software, since the programmer write a test code before the code requirement. In this context, TDD is an aid to the developer during the creation process of class design, making them create more cohesive and less coupled classes. To perform the tests is necessary to apply the practice of unit testing, which has to support automated tools, for the purposes of obtaining feedback, helping to ensure both external quality (that noticeable by the user) as the internal quality (that noticeable by the developer). The ability to continuously run the tests, allows immediate identification of possible unwanted software changes. The work is aimed at by quality and their search for information responsive to software development techniques, after studying and understanding the TDD method, the construction of a development process that focuses on the use of TDD cycle was performed to ensure implemented quality code and, therefore, minimize failures. With automated unit tests in JUnit tool, scripts can be run at any time, thereby observe future changes in the system, with more security and stability to developers.

Keywords: test-driven development. External and internal quality of the software. Unit testing. Test Automation.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
BPMN	Business Process Modeling Notation
CMMI	Capability Maturity Model Integration
IBM	International Business Machines
IID	Desenvolvimento Iterativo e Incremental
MPS.br	Melhoria do Processo do Software Brasileiro
RUP	Rational Unified Process
SIT	Testes Integrados de Sistema
SPEM	Software Process Engineering Metamodel
TDD	Test Driven Development
PRD	Processo de Desenvolvimento
XP	Extreme Programming

SUMÁRIO

1 INTRODUÇÃO

Atualmente a tecnologia está evoluindo e seu uso se tornando cada vez mais frequente na vida cotidiana, desta forma, quanto maior for à dependência maior será a probabilidade de enfrentar possíveis problemas. Neste cenário, procura-se incorporar a qualidade nos produtos ofertados pelas empresas de software, sendo que nem sempre e a realidade oferecida. Os problemas derivados das falhas de software demonstram dados estatísticos que comprovam grandes prejuízos financeiros para empresas e também para o consumidor do produto, assim, ocasionando um desgaste muito maior em relação à qualidade.

A engenharia de software empenha-se para minimizar os problemas existentes no processo de desenvolvimento de software, contudo, muitos desses problemas persistiram e se tornam os grandes “vilões”. Com isso, a engenharia de software abrange processos de software, métodos de gerenciamento de desenvolvimento de software, como também apoia ferramentas para auxiliar a qualidade (PRESSMAN, 2011).

Para obter a qualidade e diminuir o índice de defeitos na construção de um produto de software, a engenharia de software constitui e utiliza sólidos princípios de engenharia a fim de atrair softwares econômicos, confiáveis e eficientes. Os requisitos são necessidades explícitas do cliente e, para suprir e conquistar a satisfação do próprio, é essencial o desenvolvimento de um produto com a baixa existência de defeitos, erros ou falhas. (GUERRA; COLOMBO, 2009).

No desenvolvimento de software é preocupante a falta de qualidade, recorrente atrasos na entrega, e em alguns casos, a descontinuidade dos projetos. Para lidar com estes problemas, as empresas de software buscam por métodos eficientes, que tem como objetivo melhorar a qualidade dos produtos de software e aumentar a produtividade no processo de desenvolvimento.

Nesse cenário, houve a necessidade, então, de se abordar a construção de software de um modo que permita criar e evoluir produtos de software com qualidade e que agrega valor ao cliente. Neste contexto, surgiram as metodologias ágeis, que encaram o desenvolvimento de software como sendo uma atividade não caracterizada por ser um processo definido, ou seja, onde os resultados não podem ser mensurados de forma sistemática e que visam permitir ao projeto se adequar a

ambientes de desenvolvimento, especialmente onde requisitos são voláteis ou onde está se adotando uma nova tecnologia (HIGHSMITH, 2002).

Este trabalho se propõe demonstrar metodologia ágeis de desenvolvimento de software guiado por testes, uma prática de desenvolvimento para progredir na qualidade do software, e antecipando a identificação de erros, dessa forma contribuir para o aumento da qualidade e reduzir custos.

O Desenvolvimento Guiado por Testes, do inglês Test Driven Development (TDD) é uma prática que se popularizou com metodologias ágeis, e sua origem foi através da Extreme Programming (XP), cujo objetivo é acelerar o desenvolvimento do software visando a melhoria contínua do processo (SANTOS, 2010).

Diante do cenário apresentado, esta pesquisa propõe o levantamento de dados referente a metodologia ágil XP que apoia a metodologia TDD, para a construção de um processo de software. Como também a utilização de testes automatizados na cobertura de testes unitários, empregando-se critérios ágeis. Com isso, objetiva-se reduzir o número de defeitos, código refatorado e o tempo total reduzido na execução de testes, assim, garantindo maior qualidade e redução dos custos associados ao projeto de desenvolvimento de software.

1.1 OBJETIVO GERAL

Construir um processo de desenvolvimento orientado a testes, utilizando o conceito de Test Driven Development (TDD) como apoio a melhoria da qualidade no desenvolvimento de software.

1.2 OBJETIVO ESPECÍFICO

O presente trabalho possui os seguintes objetivos específicos:

a) Diminuir o tempo entre a inserção de um defeito no código, a sua detecção e, posteriormente, sua correção.
b) Relacionar técnica com metodologias e a atividades envolvidas em um projeto de desenvolvimento guiado por testes. Apresentar a utilização de TDD como apoio a melhoria da qualidade do desenvolvimento.

c) Reduzir o tempo gasto com depuração durante e depois de o código estar implementado.
d) Produzir os testes unitários antes de implementar o código do sistema, assim, diminuindo a quantidade de defeitos.
e) Definir os requisitos funcionais de forma a guiar o desenvolvimento da solução.
f) Destacar ferramentas como JUnit, e técnicas de metodologias como XP e TDD.

1.3 JUSTIFICATIVA

É notório que, no cenário atual, a velocidade das mudanças e a disponibilidade de informações crescem de forma exponencial e globalizada. A crescente facilidade de acesso à Internet vem permitindo que, cada vez mais, empresas e pessoas tenham acesso a esse veículo informacional, resultando em uma distribuição mais democrática dos conhecimentos da humanidade, oportunizando mercados e negócios a quem tiver competência.

A questão, entretanto, não é aceitar ou não a evolução e a mudança. CASTELLS (1999). Mais sim garantir qualidade aos produtos ofertados. Segundo Pressman (1995, p. 724), qualidade de software é a "conformidade a requisitos funcionais e de desempenho explicitamente declarados, a padrões de desenvolvimento claramente documentados e a características implícitas que são esperadas de todo software profissionalmente desenvolvido".

A pesquisa Chaos publicada periodicamente pela Standish Group, mostra que em 2012 houve um aumento nas taxas de sucesso nos projetos de software. A pesquisa mostra que 39% dos projetos foram entregues no prazo, dentro do orçamento, com características e funções necessárias; 43% não atingiram as metas estabelecidas, como atraso na data de entrega e requisitos incompletos; e 18% não entregaram o software, cancelaram antes da conclusão do mesmo (CHAOS, 2013, tradução nossa). Comparando estes números com o ano de 2004, que somente 29% dos softwares obtiveram sucesso, a demanda por qualidade está crescendo, porém ainda não é um número razoável.

Em circunstância, é óptico que qualidade de software tem sido uma preocupação crescente no desenvolvimento e manutenção de software. Para melhorar a qualidade do software, é necessário gerenciar o projeto da maneira adequada e produzir código eficaz. Entretanto, processos de software podem auxiliar os gestores a obter maior visibilidade, mas não 100% de qualidade sobre o projeto. Neste contexto, a construção de um processo de desenvolvimento de software que tem como base metodologias ágeis, com XP e TDD, ambos obtém rapidamente uma primeira versão do software e feedback. Assim, é possível associar a qualidade das soluções desenvolvidas num tempo menor.

As alterações realizadas na estrutura interna do software devem ser feitas de maneira fácil de ser compreendido e menos custosos de ser modificados, e no mais, sem alterar o comportamento do software (FOWLER, 2004). Um software sempre sofrerá alterações, e em alguns casos, podem desencadear muitas outras modificações no sistema, devido à dependência e inter-relacionamento entre as classes. Esse fato justifica a importância de analisar o impacto das ações.

As empresas vivem em desafios constantes, o principal é produzir software com melhoria contínua (MANIFESTO, 2008, tradução nossa). Devido à rápida expansão do mercado de software, a concorrência ocorre muito mais em custo do que em diferenciação. Para obter vantagem competitiva, as empresas devem atualizar-se continuamente na tecnologia, buscar a maturidade nos processos e eliminar a ineficiência operacional. Um ponto bastante importante é a utilização de uma infra-estrutura adequada.

O presente trabalho, almeja demonstrar a filosofia dos métodos ágeis de desenvolvimento de software e em práticas recomendadas pela Programação eXtrema (XP), testes automatizados, testes unitários, em técnica voltada principalmente para a melhoria da qualidade do código do software. É perceptível que ao aplicar um processo que cultiva métodos ágeis no processo de desenvolvimento possibilita bons resultados para empresa, e por consequência, clientes satisfeitos.

1.4 ESTRUTURA DO TRABALHO

Para alcançar os objetivos desta pesquisa, o trabalho foi dividido em 5 capítulos, sendo o primeiro composto pela introdução, objetivo principal, objetivos específicos, justificativa e a própria estrutura do trabalho, aqui relatada. Quanto aos demais capítulos, os assuntos abordados foram:

O capítulo 2 trata da Engenharia de Software, como também os processos, modelos e técnicas que visam melhorar a qualidade. Apresentação dos testes de software, juntamente com seus tipos e técnicas existentes.

O capítulo 3 apresenta a metodologia de desenvolvimento ágil, expondo características do eXtreme Programming e TDD. Apresenta o ciclo de vida do desenvolvimento guiado por testes (TDD) e seus benefícios. Há considerações sobre testes de software dentro de metodologia de desenvolvimento ágil, neste caso específico, o desenvolvimento guiado por testes, metodologia conhecida como desenvolvimento de testes antes da implementação do código.

O capítulo 4 da ênfase nos testes automatizados como apoio ao processo desenvolvido, apresentando os testes unitários para auxiliar a melhoria do software em construção. Como apoio a implementação dos testes automatizados será usado o *framework* jUnit para realizar os testes e obter os resultados.

A apresentação dos trabalhos correlatos está descrita capítulo 5, e por fim, no capítulo 6 é apresentado a metodologia e a definição do processo proposto, juntamente com toda a criação do processo desenvolvido.

2 ENGENHARIA DE SOFTWARE

As empresas vivem desafios constantes de construir software de alta qualidade, com ótimo custo benefício e, acima de tudo, em curto prazo. A engenharia de software estabelece a importância de considerar a estratégia da organização e o planejamento na implantação dos seus projetos. Para isso é necessário escolher uma das opções de modelos de desenvolvimento e manutenção de sistemas, classificados como ágeis ou rigorosos.

O modelo de desenvolvimento de software deve assegurar as características pretendidas ou pressupostas pelo cliente final do produto de software para que sejam atendidas e incorporadas no produto de software resultante desse processo. A engenharia de software contribui para esse propósito seja atingido, desse modo é uma abordagem sistemática e disciplinada para desenvolvimento de software (PRESSMAN, 2006). Até meados de 1990, a engenharia de software encontrava-se em um estágio imaturo, sem abordagens de práticas para a resolução de problemas comumente alocada nas empresas que desenvolvem software.

Muito embora, o cenário nas organizações ainda seja de prazos estourados, gastos exorbitantes com a manutenção do software, baixa qualidade e o principal, usuários insatisfeitos, é possível observar um crescimento de melhoria no requisito de maturidade da engenharia de software, através da proposição de novos métodos, ferramentas e metodologias avançadas que possibilitem o desenvolvimento de software mais confiáveis, de melhor qualidade, e com custos reduzidos. As empresas estão em busca de tecnologias, metodologias, que possam ajudar na construção de softwares com alto nível de qualidade.

A engenharia de software é apresentada em sua definição, como sendo uma disciplina que pode ser vista, de forma objetiva, com estabelecimento e o uso dos princípios básicos da engenharia com a finalidade de desenvolver softwares de maneira sistemática e econômica, resultando em um produtos confiável e eficiente (PRESSMAN, 2001). Assim, propõe métodos que visam a construção de sistemas organizados.

Conforme Pressman, a Engenharia de Software é uma tecnologia dividida em três camadas (figura 1): processo, método e ferramenta e tendo como a camada qualidade.

Figura 1 – Estrutura do processo de software.

Fonte: Adaptado de Pressman (2006).

Considerando a figura 1, cada etapa é preenchida por um conjunto de ações relacionadas:

a) foco na qualidade: comprometimento com a qualidade do software;
b) processos: desenvolvimento racional e dentro do prazo. É o alicerce da engenharia de software;
c) métodos: informações técnicas, proporciona os detalhes de “como fazer”;
d) ferramentas**:** suporte automatizado.

Estas ações fornecem as técnicas necessárias para a construção de um software, pois contempla um conjunto de tarefas que incluem comunicação, análise de requisitos, modelagem de projeto, construção de programas, teste e manutenção (PRESSMAN, 2001).

A engenharia de software está relacionada com todos os aspectos de produção de software, ele se inicia a partir do estágio inicial de especificação do sistema até a sua manutenção, ou seja, mesmo depois que este entrar em operação (SOMMERVILLE, 2001).

É essencial o papel da engenharia de software dentro do ciclo de vida do projeto e do produto de software, pois é através dos métodos usados que o software pode garantir uma melhor qualidade e produtividade, baseando-se nos estudos e padrões de engenharia mais recomendados pela área de desenvolvimento de software.

2.1 PROCESSO DE SOFTWARE

Um processo de software define um conjunto de atividades e resultados associados que levam à produção de um produto de software (SOMMERVILLE, 2003). Para Pressman (2006) o processo de software define como uma estrutura para as tarefas que são necessárias para construir software de alta qualidade.

Na engenharia de software, um processo é definido como um conjunto de passos parcialmente ordenados, cujo objetivo é atingir uma meta para entregar um produto de software de maneira eficiente, previsível, e que vá ao encontro das necessidades de negócios (MOLINARI, 2008).

Há diversas formas de representar um processo de software. Seja através de técnicas específicas de desenho de processo, tal como o BPMN ou SPEM. É necessário conhecer literaturas específicas nesta área e não ignorar esses detalhes importantes. As atividades fundamentais comuns entre elas são (SOMMERVILLE, 2007):

a) especificação de software: é preciso definir a funcionalidade do software e as restrições em sua operação;
b) projeto e implementação de software: deve ser produzido o software de modo que cumpra sua especificação;
c) validação de software: o software precisa ser validado para garantir que ele faz o que o cliente deseja;
d) evolução de software: o software precisa evoluir para atender às necessidades mutáveis do cliente.

Ainda que, não exista um processo de software ideal, há um espaço para o aprimoramento (SOMMERVILLE, 2007). Para um processo de software ser eficaz e conduzir à construção de produtos de boa qualidade, o processo deve ser adequado ao domínio da aplicação e ao projeto específico. Deste modo, processos devem ser definidos caso a caso, considerando-se as especificidades da aplicação, a tecnologia a ser adotada na sua construção, e a organização onde o produto será desenvolvido. As organizações devem criar, verificar, validar e aperfeiçoar seus próprios métodos (CMMI, 2006). Sendo que, muitas organizações desenvolvem

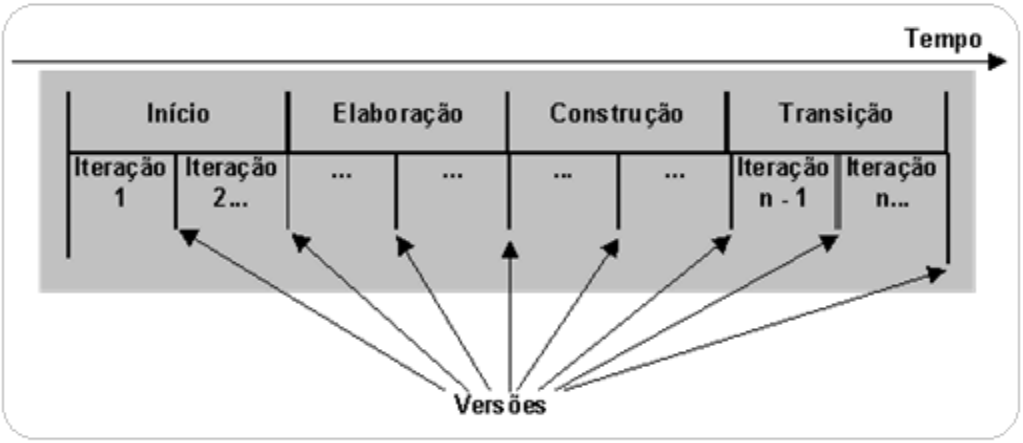
abordagens diferentes, adequadas à sua realidade, para o desenvolvimento de software.

2.1.1 Processo Unificado - RUP (Rational Unified Process)

Processo Unificado é uma técnica que emprega algumas características e recursos dos modelos de desenvolvimento de software convencionais com o objetivo de construir um software com princípios de desenvolvimento ágil (PRESSMAN, 2006).

O ciclo é composto pelas fases de início, elaboração, construção e transição e são subdivididas em iterações, conforme figura 2.

Figura 2 - Ciclo com suas fases e iterações.



Fonte: Martins (2009).

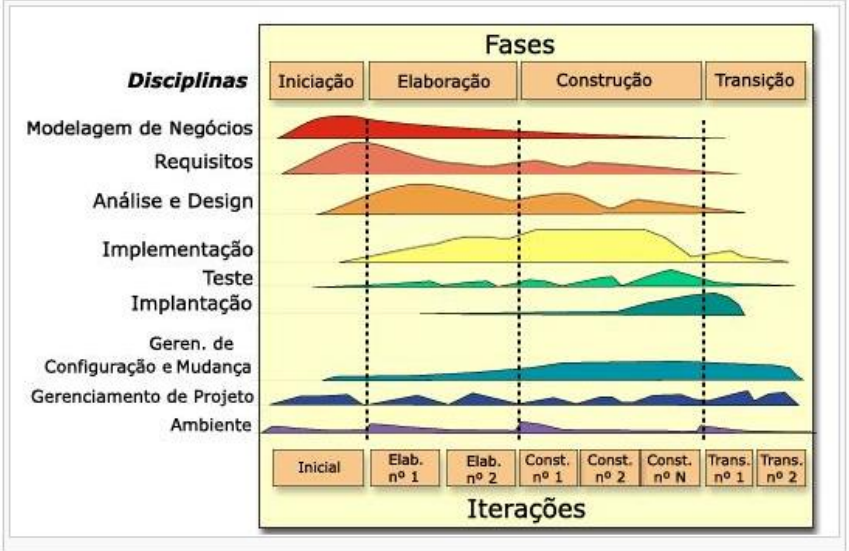
O processo unificado é formado por ciclos ao longo da vida do sistema que são repetidos. A cada conclusão dos ciclos, uma versão do sistema é gerada e distribuída interna, ou externamente (MARTINS, 2009).

O processo de desenvolvimento deverá servir como orientação às diversas atividades, recursos e elementos. Isto se dá através da definição ou especificação de artefatos que devem ser desenvolvidos, através do planejamento e atribuição de tarefas a elementos individuais ou a equipes (CARVALHO, 2002).

O RUP (Rational Unified Process), um framework para configuração e adaptação de processos de desenvolvimento de software proposto pela Rational. O

objetivo é uma abordagem disciplinar, que ao atribuir tarefas e responsabilidades para todos dentro de uma empresa de software, aumentam a produtividade da equipe, aprimorando o processo de desenvolvimento, facilitando a administração da complexidade do projeto e a manutenção da integridade do sistema. As fases (vide figura 3) indicam a ênfase que é dada no projeto em um dado instante. Para capturar a dimensão do tempo de um projeto, o RUP divide o projeto em quatro fases diferentes (figura 3):

Figura 3 - Fases RUP



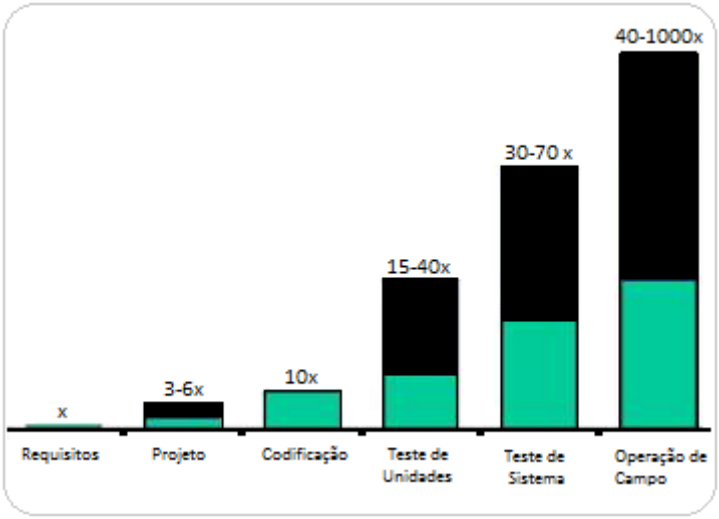
Fonte: Adaptado de OMG (2008).

O Processo Unificado pode ser definido como um conjunto de atividades utilizadas para transformar requisitos do usuário em um sistema de software iterativo e incremental. O sucesso da aplicação de um processo de desenvolvimento, principalmente do conhecimento da equipe com relação a tecnologia, método e práticas impostas e principalmente é necessário que se conheça muito bem a organização onde está sendo implantado o processo para que seja possível evitar ao máximo a imposição de atividades burocráticas tornando assim o processo útil a organização.

2.2 TESTE DE SOFTWARE

Existe uma grande insatisfações na qualidade de software, e também grandes perdas financeiras, que são ocasionadas por falhas de software. Muitas fabricas de software estudam e utilizam práticas de métodos ágeis que visam aumentar a qualidade. Os custos podem ser reduzidos com uma infraestrutura adequada e com possibilidade de identificar erros com antecedência, sendo que, o custo de correção de um erro aumenta exponencialmente com o desenvolvimento do software. O custo de defeitos na fase de projeto é cerca de três a seis vezes mais alto do que na definição de requisitos e o custo é ainda maior em fases finais (figura 4), ou seja, na fase de testes (PRESSMAN, 2006).

Figura 4 – Custo de defeitos



Fonte: Adaptado de Pressman (2006).

Os problemas de software são de 100 a 1000 vezes mais difíceis de identificar e consertar após a implantação. Verificar e gerenciar a qualidade durante o ciclo de desenvolvimento é essencial para atingir os objetivos do sistema.

Teste de software se define como uma investigação conduzida para gerar informações sobre a qualidade de um produto ou serviço. Testar o software é uma forma que temos para encontrar erros (PRESSMAN, 2006). Ao executar as atividades relacionadas ao teste é possível encontrar diferenças entre os resultados obtidos e os resultados esperados (GUERRA; COLOMBO, 2009), desta forma, notar os pontos que alcançarão as especificações e se preencheram os requisitos ao qual foi projetado.

Teste não está somente relacionado ao processo de executar um sistema com intuito de encontrar falhas (MYERS, 1979), mas evidenciar para prover informações para tomada de decisões. O teste é parte integrante do ciclo de desenvolvimento de sistemas e pode ser aplicado em qualquer fase do ciclo de desenvolvimento (PAN, 1999).

Pesquisas mostram que as falhas de software podem causar grandes perdas já que sistemas são usados nas mais diversas áreas. Qualidade significa que o sistema irá se comportar conforme o requisito estabelecido. O software que funciona corretamente, o requisito mínimo de qualidade, significa que este irá se comportar conforme o esperado nas circunstâncias previstas (PAN, 1999). Ocasionalmente os projetos possuem um tempo muito reduzido para a realização dos testes os números de testes a serem realizados nas aplicações são inúmeros. É necessário possuir um planejamento bem elaborado para que todo o projeto de testes, consiga entregar um produto com o mínimo da qualidade esperada pelo cliente e no prazo que foi acordado. É importante frisar que é impossível testar todo o sistema e garantir que ele não irá apresentar nenhum defeito, já que, sistemas são geralmente muito complexos e, sua complexidade não pode ser controlada já que os clientes sempre querem que uma nova funcionalidade seja implementada.

2.2.1 Métodos de Teste

Nas correções de software encontra-se métodos de testes específicos para cada iteração. E possível dividir os métodos de testes em caixa-preta e caixa-branca. As estratégias de caixa, como são mais conhecidas, compreendem os requisitos mínimos para que o sistema funcione, ou seja, é o propósito essencial para os testes (PAN, 1999). O responsável pelo teste não precisa necessariamente conhecer os detalhes internos do sistema, portanto precisa adotar os pontos de vista de caixa branca e o ponto de vista de caixa preta para executar cada teste.

2.2.1.1 Teste Caixa-Preta

A técnica de caixa-preta avalia o comportamento externo do componente de software, sem considerar o comportamento interno do mesmo. Também conhecido como teste funcional, já que, é baseado nos requisitos funcionais do software. Neste método dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um resultado esperado. Uns dos problemas enfrentados por este método é a complexidade em saber quando a especificação, critério usado para criar um caso de teste, está correta, em função das restrições da linguagem utilizada para escrevê-la (geralmente linguagem natural) devido a ambiguidade é que frequentemente inevitável além de ser responsável por aproximadamente 30% de todas as falhas do sistema (PAN, 1999).

2.2.1.2 Teste Caixa-Branca

O objetivo de teste de caixa-branca são testes criados para testar o código de um software, é avaliar os possíveis caminhos lógicos que podem ser executados pelo software. Também conhecido como teste estrutural, é projetado em função da estrutura do componente e permite uma averiguação mais precisa do comportamento dessa estrutura. “Caminhos lógicos internos ao software e colaborações entre comportamentos são testados, definindo-se casos de testes que exercitam conjuntos de condições e/ou ciclos.” (PRESSMAN, 2006, p 340). Este método é realizado após o código ser codificado, pois o teste é realizado no código escrito. Através desse método é possível testar exaustivamente o sistema, pois é possível criar casos de teste para cada tomada de decisão.

2.3 PRINCIPAIS TIPOS DE TESTE DE SOFTWARE

Os testes de software visam aumentar a estabilidade e confiabilidade analisando diversos aspectos do sistema, assim, identificando desvios contra o comportamento previsto. Para aumentar a confiabilidade e estabilidade há diversos tipos de testes que podem ser aplicados, dependendo da situação que o software necessita. Os testes não iniciam apenas após o desenvolvimento, deve-se começar a estruturar os testes logo no início do projeto, no levantamento de requisitos. Neste momento o planejamento e definição apropriada do que será testado e como será

testado é fundamental para o sucesso do projeto e os seguintes tipos de testes podem ajudar.

2.3.1 Teste Unitários

Teste de unidade ou teste unitário avalia a menor parte de um sistema, ou melhor, uma parte do código, um componente ou uma classe (MOLINARI, 2008). Esses testes podem ser escritos pelo analista de testes ou pelo próprio desenvolvedor. Este tipo de teste é executado no início da implementação do software, uma vez que, é nesta etapa que é mais propício encontrar os erros no software. Para isso, é preciso testar o software separadamente, sem preocupações em validar as funcionalidades como um todo, ou seja, validar as funcionalidades de cada parte do software para garantir que durante os testes integrados de sistema (SIT na sigla em inglês) o conjunto responda adequadamente ao que foi proposto a fazer.

2.3.2 Teste de Aceitação

O teste de aceitação é efetuado com o propósito de avaliar a qualidade externa do produto, ou seja, quando o produto já está pronto para ser colocado em produção. Desta forma, só é possível realizar os testes quando o software está concluído e pronto para ser implantado. Evidentemente, é um teste com forte relação com o cliente, que participa do planejamento e realização dessa atividade. O teste de aceitação é geralmente denominado de “alfa” quando realizado no ambiente de desenvolvimento (qualidade externa) e “beta” quando no ambiente do cliente (qualidade de uso). Uma forma de paliar a impossibilidade de utilizar a plataforma definitiva de execução é simulá-la, por exemplo, utilizando dados reais.

2.3.3 Teste de Integração

Visa garantir que um ou mais componentes combinados funcionam corretamente (MOLINARI, 2008). É executado em uma combinação de componentes para verificar se, integrados, eles funcionam corretamente. Deste modo, assegura a

correção de componentes individuais, mas não é capaz de garantir que as dependências funcionais entre componentes estejam perfeitamente implementadas.

2.3.4 Teste de Regressão

É preciso realizar testes sempre que houver mudanças no software, novas características inseridas, isso é necessário para assegurar que defeitos não tenham sido introduzidos. Dessa forma, aumentando a integridade da aplicação após a adição de novos componentes. Em alguns caso são usadas ferramentas específicas para o teste de regressão, as ferramentas de automação.

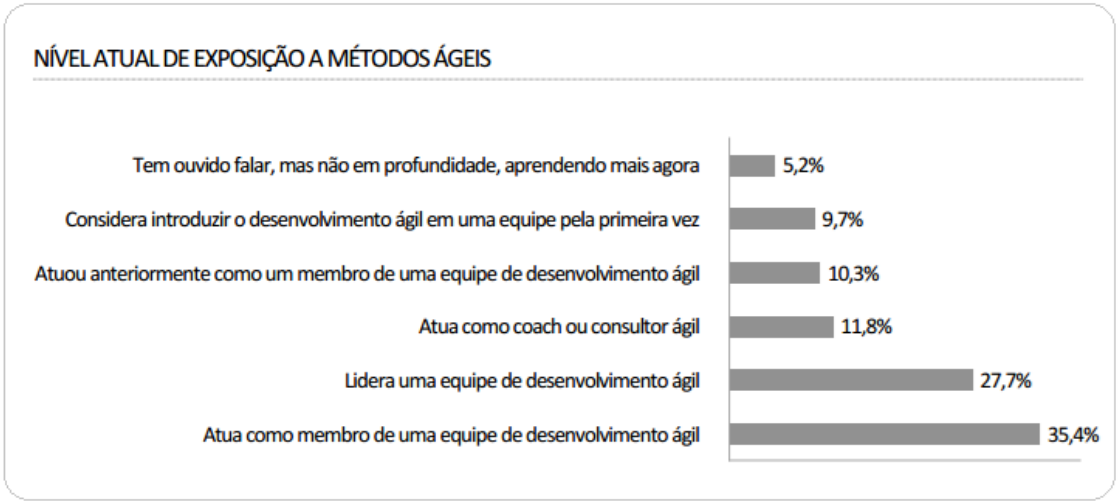
3 MÉTODOS ÁGEIS DE DESENVOLVIMENTO DE SOFTWARE

Um software é caracterizado por tarefas e requisitos que tendem a mudar constantemente, deste modo, desenvolver um software se torna algo complexo (TURNER; BOEHM, 2005). Na busca por processos de desenvolvimento de software mais flexíveis e capazes de suportar ambientes de negócios altamente voláteis e imprevisíveis, várias organizações optam pela adoção de metodologias ágeis. Os métodos ágeis se propõe resolver diversos desses problemas através de práticas que favorecem o produto final. O “Manifesto Ágil” baseia-se em valores como comunicação, *feedback* constantes, colaboração com clientes e constantes adaptação.

Segundo Erdogmus (2005, tradução nossa), na última década o paradigma mais importante que surgiu no desenvolvimento de software foi o desenvolvimento ágil de software. Mesmo que não represente o mais popular deles, com certeza é o mais comentado. Atualmente existem várias metodologias ágeis, ou seja, metodologias que seguem os princípios do manifesto ágil e entre elas, uma que se destaca entre desenvolvedores é Extreme Programming, em português, programação extrema, ou XP como é comumente conhecida (SOARES, 2010).

No Brasil o crescimento de métodos ágeis está se tornando imprescindível dentro das organizações. O crescimento à adesão aos métodos ágeis se justifica. Uma vantagem notória do desenvolvimento ágil é a adaptabilidade às mudanças. Metodologias tradicionais tentam prever todas as possibilidades sem deixar espaço para descobertas durante o desenvolvimento (BASTOS, 2013). Uma pesquisa de métodos ágeis realizada pelo grupo de pesquisa da Universidade de São Paulo (USP) mostra uma relação do nível atual referente ao conhecimento de métodos ágeis. Na figura 5 é possível observar o atual cenário:

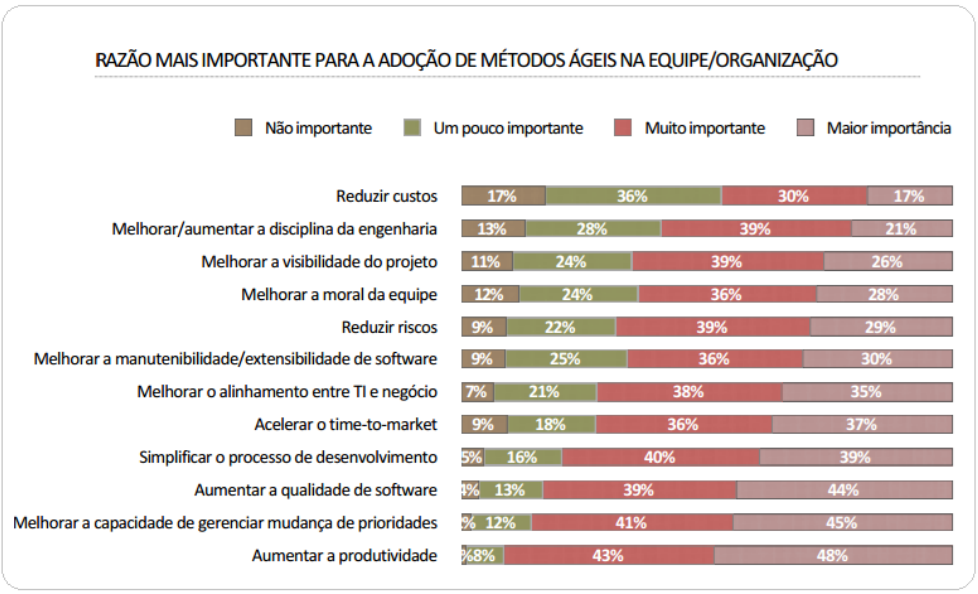
Figura 5 – Nível atual de exposição a métodos ágeis.



Fonte: Melo et al (2012).

Ao aplicar os métodos ágeis as empresas tiveram um relativo aumento na efetividade das equipes de desenvolvimento de produto, criando sistemas que agregam valores ao negócio do cliente. Como também, o índice de projetos concluídos dentro do prazo e de acordo com os requisitos aumentaram. Existem várias razões pelas quais as empresas adotam os métodos ágeis. Entre ela estão representadas na figura 6:

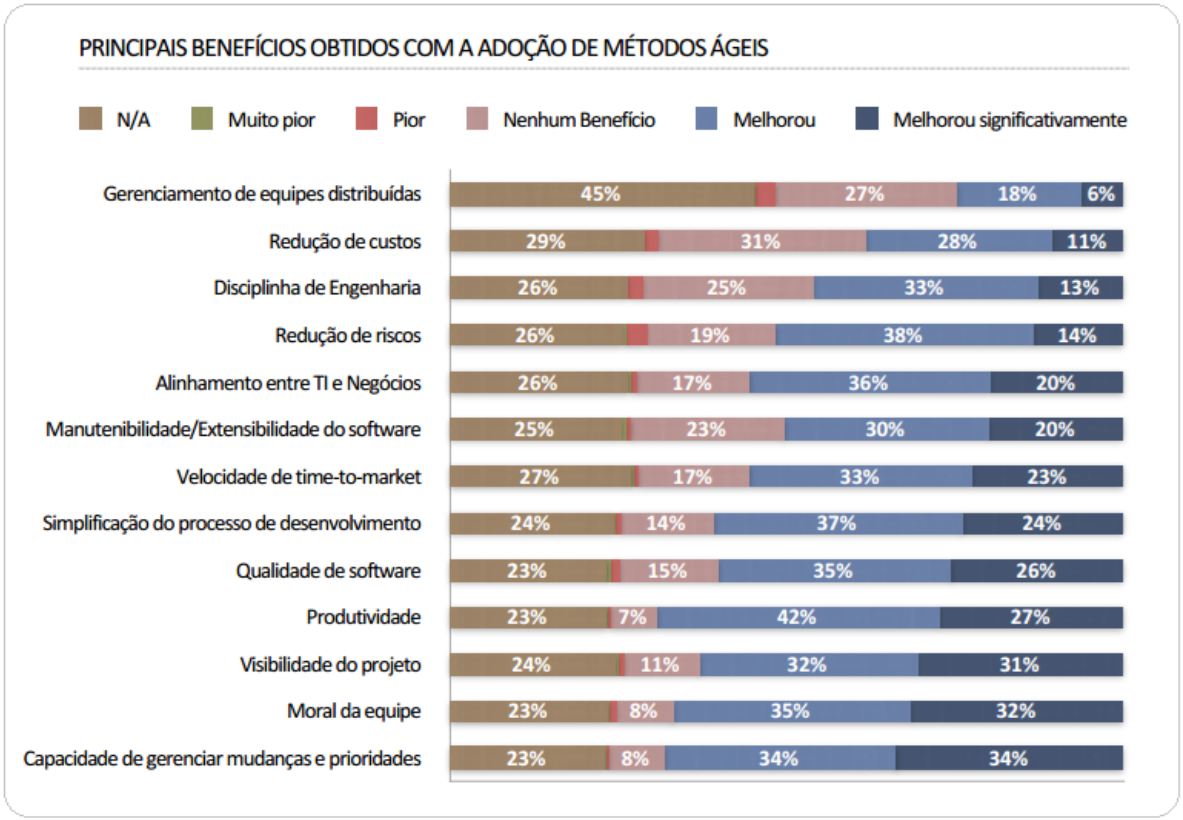
Figura 6 – Razão mais importante para a adoção de métodos ágeis



Fonte: Melo et al (2012).

O uso de metodologias ágeis é fundamental no atual cenário, com demandas crescentes e fracassos recorrentes. Fomentar a motivação dos desenvolvedores e a remuneração por produtividade é importante na implementação de metodologias ágeis já que, é mais apropriada para aumentar à efetividade a motivação das equipes. Na figura 7 é possível observar os conceitos que levam a utilização de metodologia ágil e os principais benefícios, dentre diversos benefícios que melhoraram ou melhoraram muito, os mais frequentes foram (MELO et al, 2012): Habilidade de Gerenciar mudanças de prioridades (76%), Aumento da Produtividade (76%) e da Qualidade (75%), Aumento da Moral do time (75%) e Simplificação do processo de desenvolvimento (75%). Em relação à velocidade dos projetos, 67,1% dos respondentes indicaram que projetos ágeis terminam mais rápido que projetos tradicionais.

Figura 7 – Principais benefícios obtidos com a adoção de métodos ágeis.



Fonte: Melo et al (2012).

Os objetivos dessas metodologias é deixar o processo de desenvolvimento de software mais rápido, simples e com qualidade. A sua popularidade cresce devido a velocidade do mercado atual, onde respostas rápidas são cada vez mais exigidas. As empresas devem se aperfeiçoar constantemente para encontrar novas soluções que tragam ganhos para a qualidade do software e o aprendizado de novos conceitos deve ser adequada a realidade em que a empresa está inserida.

3.1 DESENVOLVIMENTO GUIADO POR TESTES

Desenvolvimento Guiado por Testes ou Test Driven Development (TDD), é um conjunto de técnicas associadas com Extreme Programming (XP). Essa técnica de desenvolvimento de sistemas tem sido usada esporadicamente por décadas. A referência mais antiga do seu uso foi na data de 1960 no Projeto Mercury da NASA (LARMAN; BASILI, 2003). Mais somente se popularizou em 2001, por meio do livro TDD: By Example do autor Kent Beck.

A prática de TDD guia o desenvolvedor durante o processo de criação do projeto de classes por meio de constante feedback sobre a qualidade do projeto. Ela se baseia na repetição de um pequeno ciclo de atividades. No processo tradicional de desenvolvimento de um sistema, o primeiro passo é projetar, em seguida implementar o projeto e por último testar a implementação do projeto (KOSKELA, 2007). Neste cenário o desenvolvedor só descobrirá alguma falha na última etapa do ciclo, ou seja, depois de já ter implementado tudo que o projeto contemplava, a quantidade de código já é grande tornando a tarefa de corrigir a falha mais complexa.

TDD repete pequenos ciclos de Desenvolvimento Iterativo e Incremental (IID). O desenvolvimento incremental constrói um sistema requisito por requisito, ao invés de construir todas as camadas e componentes e os integrando ao final do desenvolvimento (FREEMAN et al, 2004, tradução nossa). Portanto a implementação do requisito é melhorada de maneira progressiva até que a mesma esteja boa o bastante.

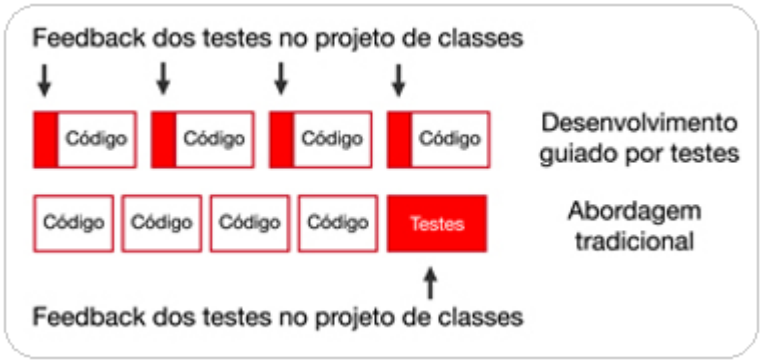
É comum alguns desenvolvedores relacionarem TDD a práticas de testes de software. Embora a criação de testes seja algo intrínseco ao processo, é dito que

TDD também auxilia o desenvolvedor a criar classes mais legíveis, mais coesas e menos acopladas. O teste é a ferramenta que o programador utiliza para avaliar o projeto da classe que está sendo criada.

É importante que os testes sejam *baby steps*, ou seja, avance em passos bem curtos, para ter um bom controle do código. Com o passar do tempo, o programador começa a adquirir segurança nessa técnica, e pode ir um pouco além, mas é sempre prudente e recomendado que não se avance muito na hora de escrever um teste.

Quem faz TDD trabalha da seguinte forma, escreve um pouco de teste, um pouco de implementação e recebe *feedback* sobre o código. Isso acontece ao longo do desenvolvimento de maneira frequente. O desenvolvedor que não pratica TDD espera um tempo (às vezes longo demais) para obter o mesmo *feedback* do software (figura 8).

Figura 8 - Feedback provido pela prática de TDD



Fonte: Aniche, 2012.

Quanto mais cedo o desenvolvedor receber *feedback*, mais fácil será para melhorar o código, corrigir problemas, e reduzindo custo, já o desenvolvedor que recebeu a mesma mensagem muito tempo depois não receberá este mesmo *feedback*. De fato é muito mais fácil e rápido alterar o código que ele implementado recentemente do que o código escrito 3 dias atrás (ANICHE, 2012).

Segundo a International Business Machines (IBM) é possível observar uma maior qualidade de software quando é utilizada a prática TDD. A adoção da prática teve um resultado de diminuição de 50% de defeitos encontrados em um

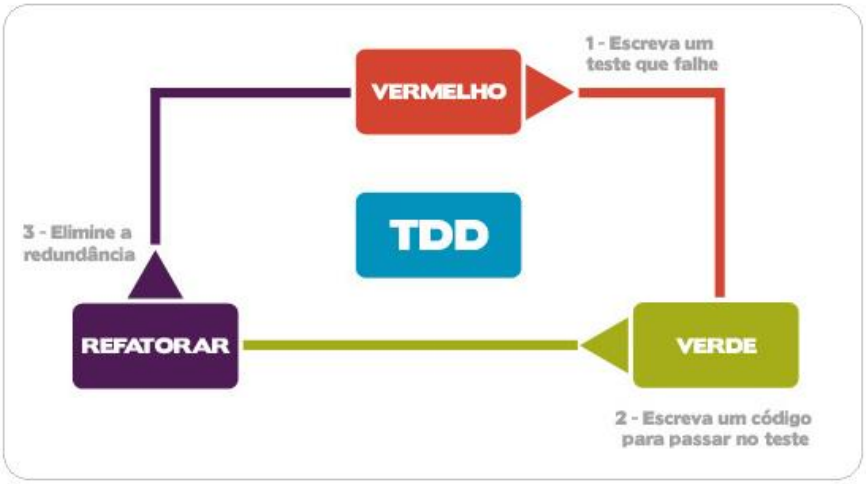
projeto comparando com um sistema semelhante com um impacto mínimo sobre o tempo e produtividade (WILLIAMS; MAXIMILIEN, 2003).

TDD pode ser definido como uma prática de projeto de classes. De acordo com Beck, Astels e Martins (2010) a mudança na ordem do ciclo de desenvolvimento tradicional, apesar de simples, agrega diversos outros benefícios ao código codificado, entre os benefícios estão: maior simplicidade, menor acoplamento e maior coesão das classes criadas.

3.2 CICLO TDD

O ciclo de desenvolvimento do TDD (figura 9) é constituído de iterações. Cada iteração corresponde à implementação de uma nova funcionalidade ou correção de alguma falha e é composta dos seguintes passos (BECK, 2002).

Figura 9 - Ciclo TDD



Fonte: Devmedia (2013).

O ciclo TDD é representando pelos seguintes passos:

a) primeiro passo: o primeiro passo será escrever um caso de teste. Para escrever o caso de teste, o desenvolvedor necessita entender o que deve ser produzido e qual o caminho a chegar a esse resultado. É preciso apontar as funcionalidades esperada neste momento;
b) segundo passo: o caso de teste deve ser executado assim que for definido, mesmo que não exista implementação para que este passe. Este passo serve como uma calibragem para o caso de teste. Se o

novo caso de teste passar, sem qualquer modificação no código do programa, então o teste pode estar errado ou ser desnecessário;

c) terceiro passo: alterar o mínimo de código para que o teste passe Após verificar que existe um teste que falha, é necessário escrever o código para que este passe. Neste ponto, deve ser escrito o mínimo de código para o teste ser executado com sucesso. Esta característica do TDD faz com que o desenvolvimento seja feito através de pequenos incrementos, formados por poucos métodos e pouca lógica. Ao final, esses incrementos juntos produzem uma solução completa;

d) quarto passo: o próximo passo é executar os casos de teste automatizados e observar se eles passam ou não. Em caso de sucesso, o programador terá certeza de que o código implementado por ele atende aos requisitos testados. Caso contrário, o código ainda precisa ser modificado;

e) quinto passo: o último passo é o *refactoring* e a limpeza do código duplicado. Nesta etapa nenhuma modificação relacionada à lógica do sistema deve ser feita. Apenas modificações estruturais são permitidas. Depois disso, os casos de teste devem ser executados mais uma vez para garantir que o *refactoring* não danificou alguma parte do sistema que já estava funcionando anteriormente.

O ciclo TDD auxilia o desenvolvedor a construir um software de maneira incremental, os riscos são reduzidos uma vez que, a quantidade de trabalho não terminado continua pequena e garantindo a entrega final do software (KOSKELA, 2007).

A propósito escrever o teste antes faz com que seja obrigatório implementar um código testável, desta forma, implica um código com baixo acoplamento, aspectos extremamente importante com bom design de software (GAZOLA, 2012).

4 AUTOMAÇÃO DE TESTES UNITÁRIOS

No mundo atual o modo convencional de desenvolver um requisito é estudar o problema, pensar em uma solução e, em seguida, implementá-la. Em seguida, o desenvolvedor faz testes manuais para verificar se está tudo funcionando como o esperado. Alguns erros são descobertos ao longo do processo de desenvolvimento, então os desenvolvedores precisaM corrigi-lo e refazer o conjunto de testes manuais (BERNARDO; KON, 2008).

Além disso, é comum submeter o software a um processo de avaliação de qualidade, antes de colocar o sistema em produção. Esse controle de qualidade geralmente é realizado com o auxílio de testes manuais executados por desenvolvedores, usuários ou mesmo por equipes especializadas em testes.

Este cenário, é frequente em empresas que utilizam metodologias rígidas que possuem fases bem definidas, geralmente derivadas do modelo de cascata (BERNARDO, 2011). Na maioria das vezes este tipo de metodologia leva à aparição de diversos problemas recorrentes, tais como atrasos nas entregas, criação de produtos com grande quantidade de erros e dificuldade de manutenção e evolução.

A execução manual de um caso de teste é rápida e efetiva, mas a execução e repetição de um vasto conjunto de testes manualmente é uma tarefa muito dispendiosa e cansativa. Desta forma, grande parte de testadores não validam novamente todos os casos de testes a cada mudança significativa do código; é deste cenário que surgem os erros de software, trazendo prejuízo para as equipes de desenvolvimento que perdem muito tempo para identificar e corrigir os erros e também prejuízo para o cliente que, entre outros problemas, sofre com constantes atrasos nos prazos combinados e com a entrega de software de qualidade duvidosa.

Como já exposto no presente trabalho, a metodologia TDD recomenda que toda a equipe da fábrica de software trabalhe controlando a qualidade do produto todos os dias e a todo momento, pois acredita que prevenir o defeitos é mais fácil e barato que identificá-los e corrigi-los um tempo depois. A Programação Extrema, em particular, recomenda explicitamente testes automatizados para ajudar a garantir a qualidade dos sistemas de software. A automação de testes compreende a execução das mesmas rotinas, porém de forma automatizada, com um computador

e um software que simulam as entradas válidas ou inválidas que um usuário faria de forma manual, para assim avaliar o produto de software. A grande vantagem desta abordagem, é que todos os casos de teste podem ser facilmente e rapidamente repetidos a qualquer momento e com pouco esforço (MOLINARI, 2010).

A reprodutibilidade dos testes permite simular inúmeras vezes situações específicas, garantindo que passos importantes não serão ignorados por falha humana e facilitando a identificação de um possível comportamento não desejado.

Além disso, como os casos para verificação são descritos através de um código interpretado por um computador, é possível criar situações de testes bem mais elaboradas e complexas do que as realizadas manualmente, possibilitando qualquer combinação de comandos e operações. Assim, soluciona os problemas encontrados nos testes manuais, diminuindo a quantidade de erros e aumentando a qualidade do software. Como é relativamente fácil executar todos os testes a qualquer momento, mudanças no sistema podem ser feitas com segurança, o que aumenta a vida útil do produto.

4.1 FERRAMENTA PARA AUXILIAR OS TESTES UNITÁRIOS

A construção de um processo de software requer o emprego de ferramentas que apoiem o processo. Existem diversas ferramentas que possibilitam esta prática, entre elas então:

a) JUnit: é um framework de teste para Java, que permite a criação de testes unitários. Além disso, está disponível como plug-in para os mais diversos IDE'S como Eclipse, Netbeans, e entre outas tecnologias;
b) TesteNG: outra ferramenta de teste unitária, disponível para Java;
c) PHPUnit: framework XUnit para teste unitário em PHP, também é possível integrar aos IDE's assim como o JUnit;
d) SimpleTest: ferramenta para realização de teste para PHP. Além de possibilitar os testes unitários, é possível realizar MOCKS e outros testes;
e) NUnit: framework de teste no molde XUnit para a plataforma .NET;
f) Jasmine**:** framework para teste unitário de JavaScript;

g) CUnit: ferramenta para os testes unitários disponível para Linguagem C;

h) PyUnit: framework Xunit para testes na linguagem Python.

O subcapítulo seguinte abordará características da ferramenta jUnit, a qual será empregada na elaboração do trabalho atual.

4.2 A FERRAMENTA JUNIT

JUnit é um Framework open-source, criado por Eric Gamma e Kent Beck, com suporte à criação de testes automatizados na linguagem de programação Java. Para usar o JUnit é necessário instalar um plugin no eclipse, que ao ser executado mostra uma lista de todos os teste, destacando de verde nos casos de sucesso, ou vermelho para caso de falha.

O Objetivo deste framework é facilitar a criação de código para a automação de testes, assim, apresentar os resultados obtidos. Com ele, pode ser verificado se cada método de uma classe funciona da forma esperada, exibindo possíveis erros ou falhas. Essa verificação é chamada de teste unitário ou teste de unidade, pois é a fase do processo de teste em que se testam as menores unidades de software desenvolvidas e tem como objetivo prevenir o aparecimento de *bugs* oriundo de códigos mal escritos e garantir um nível de qualidade de produto durante o processo de desenvolvimento de software.

Um teste automatizado na ferramenta JUnit para uma aplicação Java é organizado em casos de teste definidos através de anotações Java *@Test* antes da definição de cada método de teste. O arcabouço também possui uma gama de métodos auxiliares para comparar os efeitos colaterais esperados com os obtidos, tais como *assertEquals* que compara a igualdade do conteúdo de objetos, *assertSame* que compara as duas referências se referem ao mesmo objeto, *assertNull* que verifica se uma dada referência é nula, entre outros (vide figura 10).

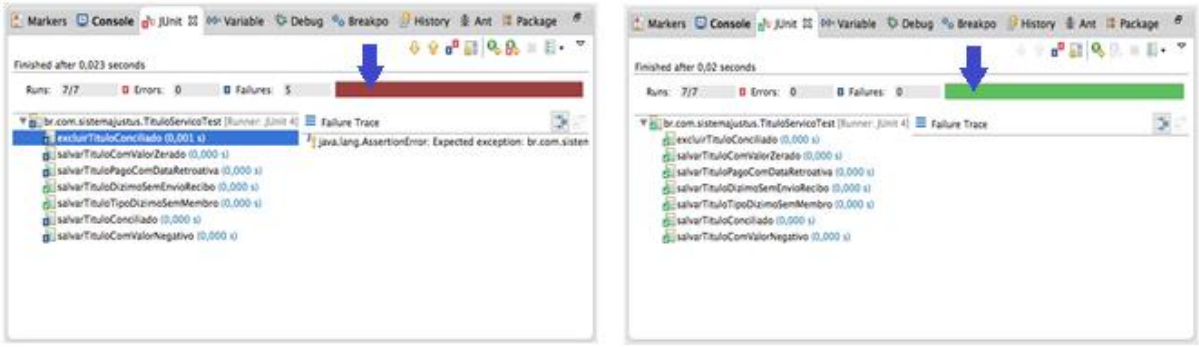
Figura 10 - Exemplo de teste automatizado com o JUnit

```java
package br.com.sistemajustus;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.junit.Test;

public class TituloServicoTest {

    private SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

    @Test(expected = AplicacaoException.class)
    public void salvarTituloTipoDizimoSemMembro() {
        Titulo titulo = new Titulo();
        titulo.setCodigo(1);
        titulo.setConciliado(Boolean.FALSE);
        titulo.setData(new Date());
        titulo.setReciboEmail(Boolean.TRUE);
        titulo.setTipo(TipoTitulo.DIZIMO);
        titulo.setValor(10D);
        titulo.setPago(Boolean.FALSE);

        TituloServico servico = new TituloServico();
        servico.verificarDizimoSemMembro(titulo);
    }
```

Fonte: Do autor.

Para auxiliar os testes a um conjunto de ferramentas auxiliares para a integração da ferramenta com ambientes de programação, dessa forma, facilitando a execução da bateria de testes e a leitura dos resultados obtidos. Com esta facilidade é possível retirar um relatório para verificar os resultado dos testes executados dentro da IDE Eclipse. O relatório contém uma barra que fica verde quando todos os testes passam com sucesso, ou vermelha quando pelo menos um caso de teste falha (figura 11). Para facilitar a localização de erros, é impressa a pilha de execução apenas dos testes que falharam.

Figura 11 – Relatório do testes do plug-in do Junit para IDE Eclipse



Fonte: Do autor.

O código do teste, assim como o código do sistema, pode conter erros; por isso, é imprescindível que o código do teste seja o mais simples possível. O desenvolvedor deve evitar testes longos, código repleto de condições (ifs), testes responsáveis por muitas verificações e também nomes obscuros.

É importante ressaltar que a legibilidade dos testes precisa ser boa, dessa forma, facilita a manutenção do software e para obter pistas explícitas de qual funcionalidade está quebrada quando um teste falhar. Além disso, testes claros e legíveis podem ser utilizados como documentação do sistema ou como base para gerar a documentação através dos relatórios dos resultados.

5 TRABALHOS CORRELATOS

Durante o desenvolvimento desta pesquisa, foram analisados diversos trabalhos com abordagem no desenvolvimento e aplicação de processos, semelhantes ao trabalho apresentado, mas com foco em outras áreas específicas.

5.1 MINIMIZANDO AS FALHAS NA CONCEPÇÃO DOS SISTEMAS COM O DESENVOLVIMENTO GUIADO POR TESTES

Trabalho de Conclusão de Curso de Nelson Senna Do Amaral para obtenção do grau de Tecnólogo em Processamento de Dados, concluída em 2013, pela Faculdade de Tecnologia de São Paulo – TFATEC.

O trabalho apresenta um estudo referente ao Desenvolvimento Guiado por Testes como alternativa para redução de falhas nos sistemas de informação. O objetivo foi verificar a possibilidade de aplicar o desenvolvimento guiado por testes como alternativa para mitigar as falhas nos sistemas de informação, baseado não só no levantamento bibliográfico, mas, também apoiado nos resultados das aplicações expostos na pesquisa.

Através pesquisa realizada pelo o estudante foi possível observar que os times que utilizaram TDD tornaram os sistemas mais manuteníveis já que a métrica indica que o melhor resultado é o que tende a zero. Para calcular a métrica foram considerados o número de falhas e o tempo de vinte e quatro horas. A tabela 14 mostra o cálculo para a métrica “Tempo gasto para implementar uma mudança pelo mantenedor” dado em time/dia.

Tabela 14: Resultado da avaliação dos times utilizando a métrica “Tempo gasto para implementar uma mudança pelo mantenedor”

Time	Valor da métrica
Com TDD	8.58
Sem TDD	19.54

Conclua-se que projetos que fizeram uso de TDD são mais maduros e confiáveis que os que não fizeram uso da técnica. O aumento no tempo de

desenvolvimento não implica em nenhum critério de qualidade do produto. Apesar disso não deve ser desconsiderado. O aumento no tempo de desenvolvimento pode, por exemplo, impactar diretamente no orçamento do projeto e no seu tempo de entrega e, indiretamente nas demais fases do desenvolvimento do produto.

5.2 COMO A PRÁTICA DE TDD INFLUENCIA O PROJETO DE CLASSES EM SISTEMAS ORIENTADOS A OBJETOS

Dissertação apresentada por Mauricio Finavaro Aniche para obtenção do título de mestre em Ciência da Computação, concluída em 2012, Instituto de Matemática e Estatística da Universidade de São Paulo – IME-USP

O trabalho visa compreender a influência de TDD no projeto de classes. Para isso foi realizado a avaliação entre a prática TDD e as decisões de projeto de classes tomadas pelos desenvolvedores no processo de criação de classes.

A análise foi realizada por meio de dados que foram capturados baseados na percepção de desenvolvedores atualmente na indústria, após a implementação de alguns pequenos problemas especialmente criados a pesquisa.

O objetivo principal foi entender a relação da prática de TDD e as decisões de projeto de classes tomadas pelo desenvolvedor durante o processo de projeto de sistemas orientados a objetos.

Os resultados esperados foram que através do estudo feito, os desenvolvedores de software possam antecipar problemas de projetos de classes, gerando melhorias constantes e, por consequência, diminuindo o custo de manutenção e evolução do sistema. Foi possível identificar que os desenvolvedores que utilizam TDD podem perceber novos *feedback*s. Relacionados ao que não praticam TDD, a leitura do trabalho poderá ajudar na decisão de começar a utilizar TDD ou não durante o ciclo de desenvolvimento de um projeto.

Entretanto, espera-se que através da leitura, o desenvolvedor tende a entender sobre a influência de TDD no projeto de classes e, que experiência e conhecimento são necessários para boas práticas de codificação, assim, TDD ira guiar para um bom projeto de classes.

5.3 USANDO DOJOS DE PROGRAMAÇÃO PARA O ENSINO DE DESENVOLVIMENTO DIRIGIDO POR TESTES

Este artigo foi desenvolvido por Ramiro Batista Luz e Adolfo Neto, sendo apresentado na Universidade Tecnológica Federal do Paraná (UTFPR), na tarefa de usar dojo no desenvolvimento de software dirigido por teste.

Dojo de programação é uma atividade dinâmica e colaborativa inspirada em artes marciais que segue uma disciplina num ambiente de ensino agradável e divertido. Técnicas de desenvolvimento ágil são utilizadas durante o dojo de programação, dentre elas o desenvolvimento guiado por testes, programação em pares e passos de bebê. Foi disponibilizado um questionário eletrônico que respondido por participantes de dojo de programação, de diversas regiões do Brasil e com diferentes níveis de experiência. Além da pesquisa foi realizado ao menos um experimento com alunos de mestrado.

A última fase planejada da pesquisa foi um experimento prático. De acordo comas respostas obtidas nas pesquisas foi realizado um treinamento com uma turma de alunos da disciplina de métodos ágeis. A turma foi dividida em dois grupos, um grupo teve 20 horas-aula expositivas de TDD com teoria e prática intercaladas. O outro grupo teve de 20 horas de dojo de programação. O mesmo conteúdo foi apresentado aos dois grupos, divididos aleatoriamente, pelo mesmo instrutor. Após as 20 horas, ambos os grupos deverão desenvolver um projeto semelhante. O código do projeto foi submetido a um avaliador neutro. Além dessa avaliação o código foi analisado por ferramenta de análise de cobertura de testes, que examina quais partes do programa foram executadas pelo conjunto de testes.

Os resultados do questionário eletrônico indicaram que os participantes concordam que o dojo de programação ajuda o aprendizado de métodos ágeis, as questões relacionadas a programação em par, passos de bebê e TDD receberam valores altos na escala de Likert. Outro ponto considerado forte foi a troca de experiência entre os participantes. Esses resultados direcionaram o planejamento da última etapa da pesquisa, o experimento foi limitado a avaliação de TDD, onde usou-se ferramentas de estatísticas de cobertura detestes e avaliação de código por avaliadores independentes.

Segunda a pesquisa o dojo favorece a participação incluindo os programadores na ambiente de aprendizado. O dojo de programação favorece a socialização dos programadores, segundo constatamos nas entrevistas.

5.4 UMA ABORDAGEM SOBRE TESTES AUTOMATIZADO DE SOFTWARES EM AMBIENTES DE DESENVOLVIMENTO

Este artigo foi desenvolvido por Robson L. Nascimento e Késsia R. C. Marchi, sendo apresentado na Universidade Paranaense (UNIPAR) com objetivo de constituir informações referentes a testes automatizados de software cujo objetivo é enfatizar as informações referente a teste de software.

Torna-se indispensável nos dias atuais, em que as tecnologias evoluem de “um dia para o outro” garantir que os produtos ofertados funcionem de acordo com o especificado. Para se chegar a uma qualidade satisfatória os testes são fundamentais nos *feedback* da funcionalidade dos produtos ofertados. Automatização deste processo é uma maneira eficaz de passar para o computador funcionalidades que as vezes passam despercebido a olhos humanos. “Não é necessário reinventar a roda”, empresas mais competitivas são as empresas que trabalham sob a ótica da melhoria contínua dos processos. [CAETANO 2007]. Utilizar-se de ferramentas já existentes cuja eficiência em ação é confirmada, é uma boa maneira de evitar desperdício de tempo e recursos para que a prática deste trabalho entre em ação.

6 MODELAGEM DE PROCESSO PARA O DESENVOLVIMENTO DE SOFTWARE

Por meio de gestão de processos, as empresas desenvolvedoras de software podem obter resultados positivos na qualidade do software. Clientes necessitam de software a todo momento, neste âmbito, desenvolver softwares torna-se a cada dia uma tarefa mais complexa onde a qualidade, retorno rápido e necessidades atendidas são requisitos primordiais. Visando minimizar essa situação alguns engenheiros de software criaram a metodologia ágil, ou seja, uma forma de desenvolver softwares rápido, com qualidade e com a documentação básica necessária para o desenvolvimento.

O objetivo da concepção de um processo de software é a melhoria e compreensão do que será realmente construído. Os modelos são úteis para elaboração da estrutura do software, além de permitir visualização, especificação, construção e documentação dos artefatos de projeto (DEBONI, 2003). A modelagem agrega outros benefícios como a possibilidade de desenvolver um sistema de forma mais rápida e eficiente, com o mínimo de desperdício e retrabalho de software (FOWLER, 2005; MATOS, 2002).

O processo da automação de testes foi praticado com uso da ferramenta JUnit, baseado no conhecimento e experiência de especialista de testes e especialista em desenvolvimento de software. A meta é, que programadores e testadores que utilizem o processo de desenvolvimento guiado por teste tenham apoio ao desenvolvimento do software, e assim, aumento da qualidade e o bom design do código.

Como já proposto, o presente trabalho é a construção de um processo utilizando o conceito TDD como apoio a melhoria do software guiado por testes, baseado em metodologia ágeis, desta forma, busca uma solução dos problemas ou, minimizando o índice de software de baixa qualidade.

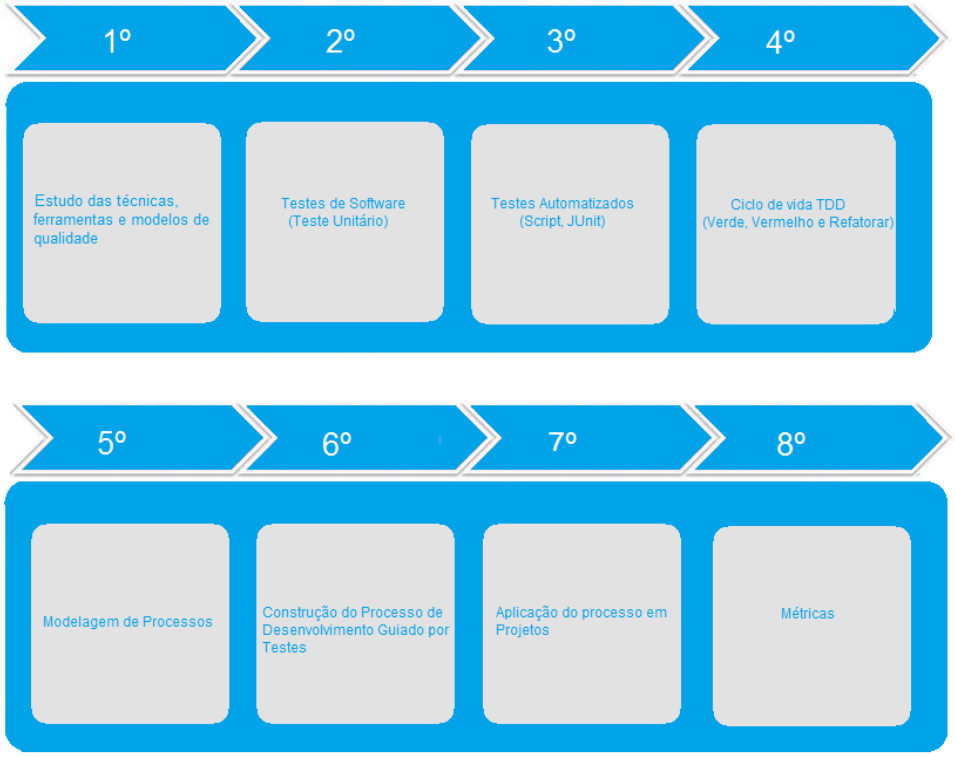
6.1 METODOLOGIA

A metodologia utilizada para a elaboração deste trabalho iniciou-se no levantamento de dados referente aos problemas encontrados no desenvolvimento de software, em seguida uma pesquisa bibliográfica em materiais publicados em

livros, revistas, rede eletrônica, periódicos especializados, dissertações e monografias, para base teórica de conceitos fundamentais de engenharia e qualidade de software, a realização da automação dos testes e a aplicação da técnica TDD na construção do processo de software.

O estudo sobre o processo de desenvolvimento de software utilizando a metodologia TDD foi realizado utilizando os guias e documentos disponibilizados no livro do autor BECK (2002), e entre outras referências conceituais da metodologia, dessa forma, foi possível compreender os conceitos relacionados a este ciclo de desenvolvimento de software. Com base nos estudos realizados, abordamos o desenvolvimento deste trabalho implementando algumas etapas do processo de teste, conforme figura 12.

Figura 12 – Metodologia aplicada



Fonte: Do autor (2014).

A seguir serão descritas de forma detalhada, todas as fases de desenvolvimento do trabalho proposto.

6.1.1 Estudo sobre as Técnicas, Ferramentas, Processos e Modelos de Qualidade da Engenharia de Software

Para construir um software estável e confiável é de suma importância apoiar a execução de processo de software, modelos, técnicas e ferramenta, consequentemente, o software terá maior probabilidade para atingir o padrão de qualidade adequado. Caso contrário, pode afetar significativamente no custo e prazo de entrega do produto. O impacto é tão significante que a melhoria do processo de software é vista por muitos como a mais importante forma para melhorar o produto de software (HENRY, 1994).

Durante o ciclo de vida de um software, várias etapas precisam ser cumpridas para colocá-lo e mantê-lo em funcionamento. O objetivo é entregar um software de qualidade, confiável, estável e que atenda às necessidades do cliente. Mas infelizmente a realidade nem sempre é essa, pois nenhum software está livre de falhas, devido a problemas relacionados a falta de arquitetura no software.

Atualmente existem técnicas que, se aplicadas corretamente, podem ajudar a entregar um software de qualidade. Aliás, a qualidade não está associada somente às falhas, mas a outros fatores como desempenho, usabilidade, confiabilidade, satisfação do cliente, entre outros. Isto é uma preocupação da Engenharia de Software, disciplina que lida com todos os aspectos e etapas da produção de software e que tem como objetivo fornece métodos para construir softwares de alta qualidade (PRESSMAN, 2011). Para que isso seja conseguido é necessário:

a) aplicar teorias, métodos e ferramentas nas situações apropriadas nas diversas etapas do desenvolvimento;
b) trabalhar de acordo com as restrições organizacionais e financeiras procurando soluções que estejam dentro dessas restrições;
c) gerenciar os projetos de software para que o resultado final esteja dentro do escopo, custo e prazos planejados;
d) adotar uma abordagem sistemática e organizada para produzir software de qualidade de maneira mais eficaz.

Para a finalidade de aperfeiçoar o conhecimento em normas, qualidade, testes de software, foram realizados estudos nas normas CMMI, MPS.br e ISO/IEC 9126.

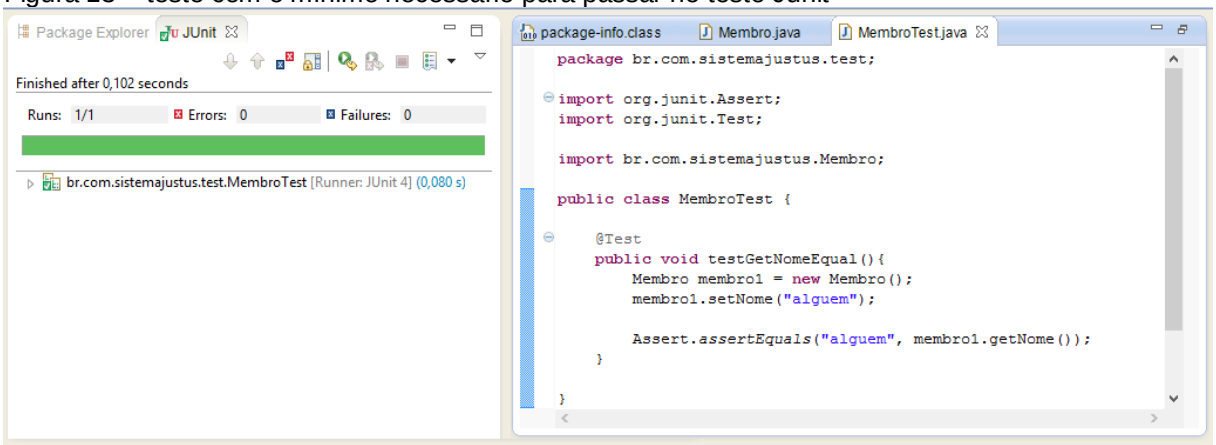
Na construção do processo de desenvolvimento de software foi necessário buscar conhecimento em relação ao desenvolvimento de modelos de processo. Para uma visão geral dos ciclos de vida de desenvolvimento e sua aplicação nas empresas.

6.1.2 Prática de teste unitário

A prática de testes unitários é imprescindível na utilização de TDD. Os casos de testes foram criados de acordo com os requisitos levantados para os processos em questão. A figura 13 mostra os casos de teste feitos para a classe Membro. Para seu funcionamento, ele espera 4 requisitos do seu caso de teste.

a) um método *@Test* no caso JUnit;
b) um método getNome para verificar se está retornando um nome, conforme o esperado;
c) um *assertEquals* que compara a igualdade do conteúdo de objetos;
d) o membro deverá receber um getNome.

Figura 13 – teste com o mínimo necessário para passar no teste Junit

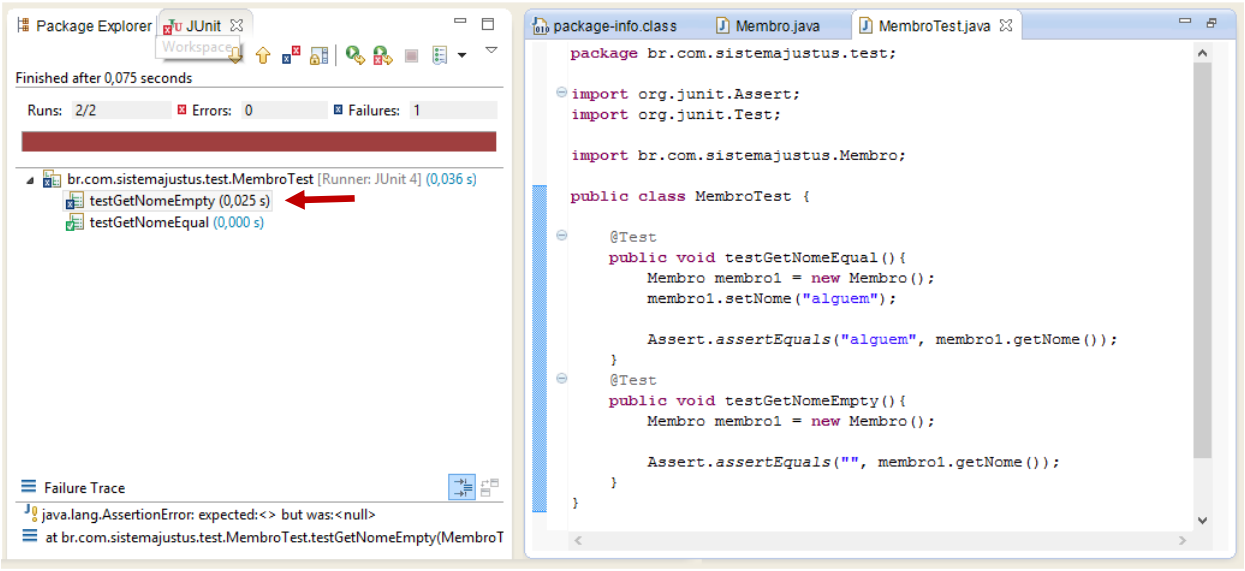


Fonte: Do autor (2014).

Na figura 13 no lado esquerdo é possível observar os requisitos mínimos do teste passou (barra-verde). Neste teste foram implementados todos os casos de testes necessário para a validação do método. Neste momento o desenvolvedor

deverá pensar e analisar sua classe, verificar os requisitos e mais adicionar mais casos de testes para validar seu código. Vejamos um caso de teste no mesmo exemplo (figura 14). Ver o caso de teste para retornar vazio, ao adicionar o método *getNomeEmpty*, nosso código estará preparado.

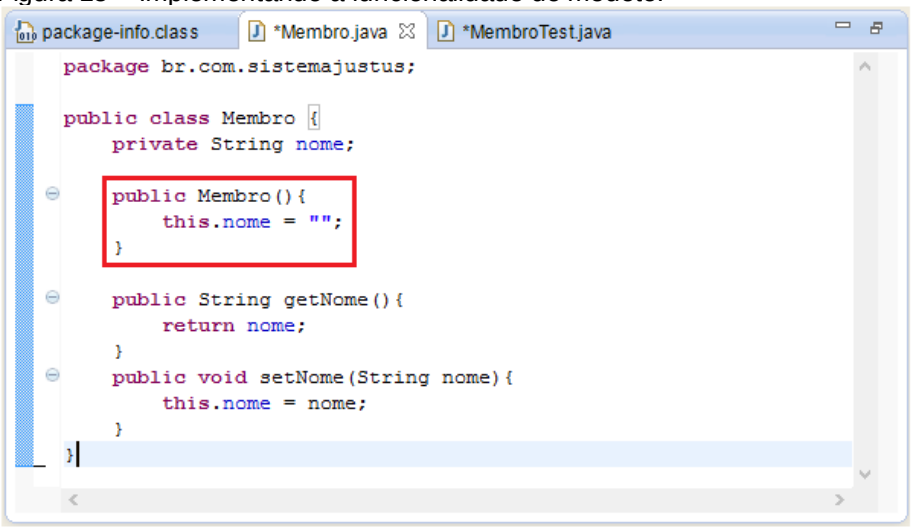
Figura 14 - Erro ao rodar o teste unitário (getNomeEmpty).



Fonte: Do autor (2014)

Ao rodar o teste no JUnit, observa-se que o código não está preparado. Pois era esperado um resultado vazio e o *membro1.getNome* retornou Null. Para corrigir é necessário construir o atributo nome no construtor (figura 15).

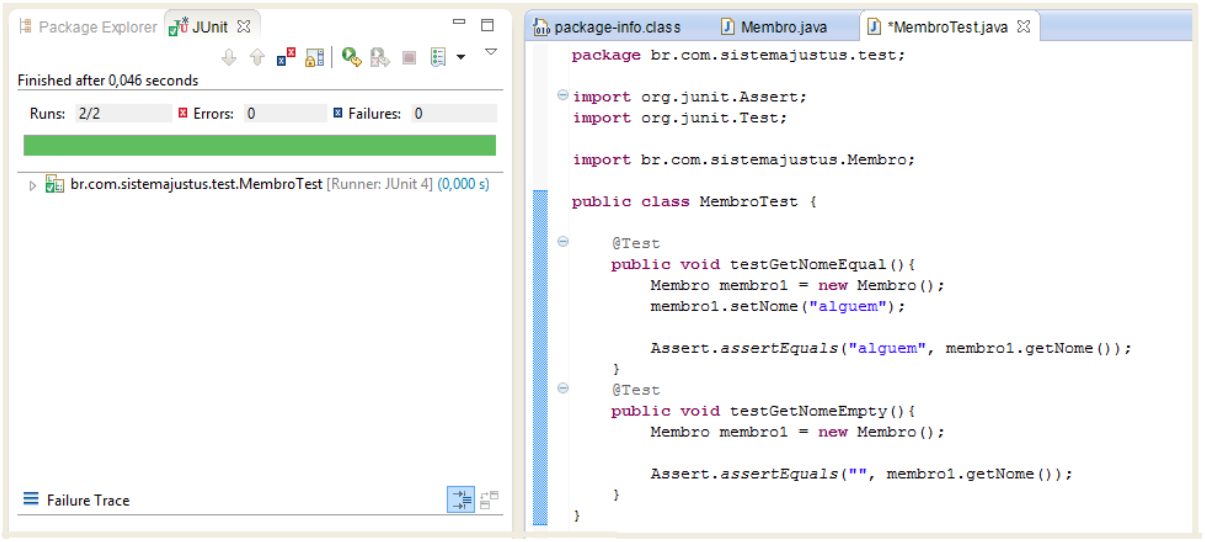
Figura 15 – Implementando a funcionalidade do médoto.



Fonte: Do autor (2014)

Ao corrigir o método Membro e passar o parâmetro, o teste rodou conforme o esperado (figura 16).

Figura 16 – Resultado do teste JUnit (verde).



Fonte: Do Autor (2014)

Para cada sugestão ou pendência desenvolvida foram criados testes unitários. Nesse cenário toda a classe implementada possuía uma classe de teste associada.

6.1.3 Testes automatizados (Script, JUnit)

Testes automatizados são ferramentas extremamente valiosas para feedback, auxiliando a garantir tanto a qualidade externa (perceptível pelo usuário) quanto a qualidade interna (perceptível pelo desenvolvedor).

A facilidade de poder executar continuamente os testes permite a identificação imediata de alterações indesejadas no sistema. Os teste também favorecem a prática da refatotação, que consiste em aplicar pequenas transformações na estrutura interna de um software, preservando o comportamento externo, com objetivo de melhorar o código, tornando-o mais legível e mais aderente ás manutenções pelas quais deverá passar.

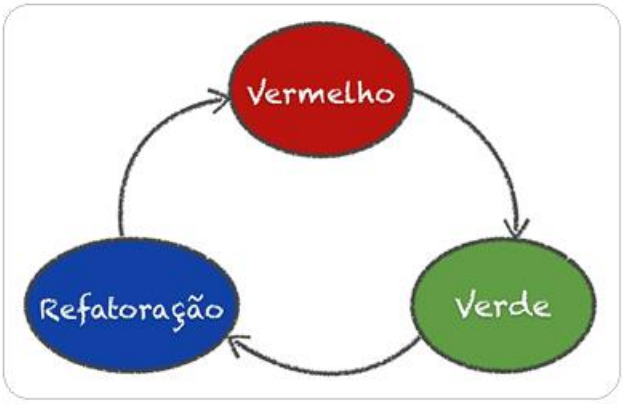
Para colocar em prática os testes automatizados foram feitos com testes unitários, que verificam a classe unitariamente. O teste de unidade tem como foco pequenos trechos de código, a fim de encapsular as unidades e que está produzindo códigos esperados.

Ainda a desenvolvedores que criticam essa prática, por não entender que refatoração é uma técnica disciplinada para manter a boa qualidade da base de código. Ao longo do tempo uma base de código que não passa por refatoração tenderá a ser cada vez menos coesa. Uma classe coesa é justamente aquela que possui apenas uma única responsabilidade.

6.1.4 Ciclo de vida TDD (verde, vermelho e refatorar)

Test Driven Development é uma abordagem iterativa para o desenvolvimento de software baseado em testes automatizados. A ideia é aplicar pequenos ciclos de teste-codificação-refatoração, também conhecido como red-green-refactor (figura 17). Este ciclo será adicionado no processo de desenvolvimento PRD.

Figura 17 – Ciclo TDD Teste-Codificação-Refatoração

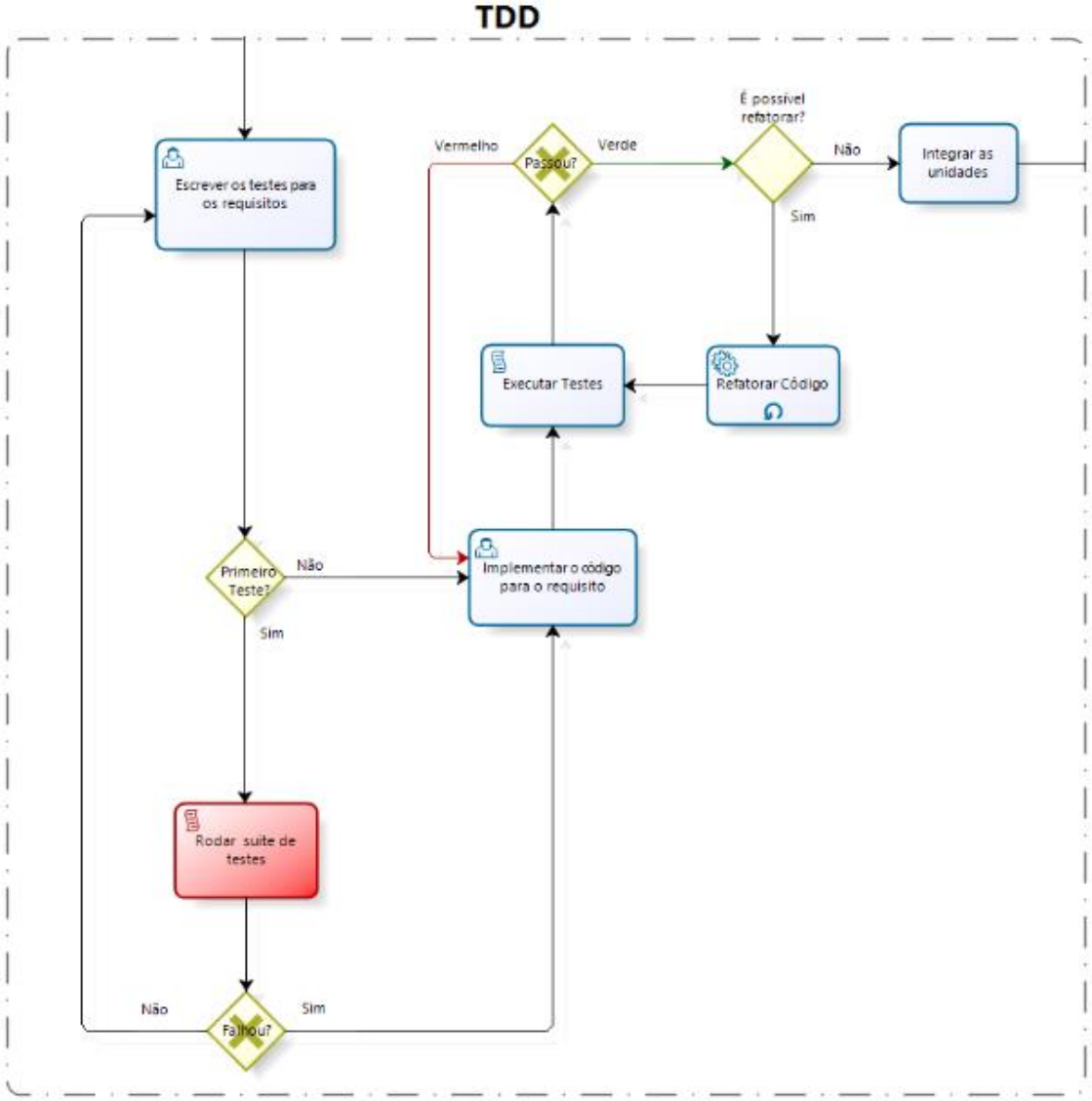


Fonte:

Escrever o teste antes, obrigatoriamente força com que o código produzido seja testável por construção, e código testável implica o código com baixo acoplamento, ou seja, um aspecto importante em qualquer bom design de software. Também é uma boa prática para evitar a implementação de código mais do que necessário. TDD pode ser aplicado para as pequenas partes que compõem um sistema (testes de unidade para métodos de classes), quanto para componentes maiores. Além do benefício em design, TDD traz como benefício de fazer que seja criada uma extensa suítes de testes de regressão para o sistema, com elevado nível de cobertura.

Este ciclo TDD Teste-Codificação-Refatoração foi colocado no processo de desenvolvimento PRD com intuito de melhorar a qualidade do código implementado, em suma, melhorar a qualidade do software em produção (figura 18).

Figura 18 – Aplicação do ciclo TDD dentro do processo de desenvolvimento.



Fonte: Do autor (2014).

A figura 18 ilustra o ciclo TDD aplicado dentro do processo PRD. O método é formado por pequenos passos, que são repetidos para o desenvolvimento de uma nova funcionalidade no sistema. Primeiramente inicializa pela implementação de um caso de teste, seguido para escrita do código, apenas o necessário para passar no teste, e finalmente pela fatoração do código com padrões aceitáveis. Ao aplicar o ciclo TDD no código é preciso:

a) uma suíte de testes;
b) os testes são escritos antes;
c) os testes são a documentação e o guia para implementação do código;
d) nenhum código entra em produção a não ser que tenha testes associados (testes unitários e automatizados);

6.1.5 Modelagem de Processos

Elaborar um processo de desenvolvimento de software significa determinar de forma precisa e detalhada de como implementar o código. Um processo pode ser visto como uma instância de um método com suas técnicas e ferramentas associadas, elaborado durante a etapa de planejamento, no qual as atividades que o compõem são alocadas aos membros da equipe de desenvolvimento, com prazos definidos e métricas para se avaliar como elas estão sendo realizadas (LEITE, 2000).

A modelagem de processos auxiliam as empresas a obterem um melhor controle e gerenciamento dos projetos. O processo deverá ser modelado em uma linguagem que possa ser entendida facilmente por todas o time envolvido no desenvolvimento do software. Neste caso, para a modelagem do processo pode-se aplicar o SPEM (Software Process Engineering Metamodel) que é um dos padrões de modelagem de processos de software mais difundidos e aceitos na comunidade de Engenharia de Software.

De forma similar, também é necessário controlar os processos de negócio para que todas as atividades e tarefas estejam de acordo com as metas e estratégias da organização. Para tanto, foi desenvolvido o BPMN (Business Process Modeling Notation) que é um padrão de modelagem para processos de negócio.

Dessa forma, este trabalho visa somar à ferramenta de modelagem de processo, que utiliza a notação SPEM, a opção de mapear para o padrão BPMN a modelagem de um processo de software específico, visto que o processo de software é também um processo de negócio da empresa.

O propósito é garantir a qualidade do desenvolvimento do software e fornecer aos envolvidos no projeto, tanto gerentes quanto desenvolvedores, uma

visão clara sobre métodos ágeis e onde, como e quando a técnica TDD pode ser utilizada para obter um bom design de código, a fim de produzir um software de qualidade.

6.1.6 Construção do Processo de Desenvolvimento Guiado por Testes

É perceptível que cada vez mais a qualidade do produto é exigido, e ferramentas, métodos e processos são fartos para auxiliar na qualidade, mas para isso é necessário aperfeiçoar o conhecimento e colocar em prática esses métodos. Como já citado, TDD é uma forma diferente para o desenvolvedor trabalhar. Pois como o próprio nome diz a metodologia é baseada no desenvolvimento guiado por testes, desta forma, o desenvolvedor deixa de pensar, em primeiro lugar, no código e passa a pensar em como realizar o teste no código que será implementado.

Já vimos que os testes em modo geral, são a melhor maneira de avaliar a obtenção da qualidade do software produzido. Este processo de testes é realizado somente no final de cada projeto, o que leva a descoberta de possíveis erros tarde, ou no pior caso, quando o produto já está em produção. Então porque não trazer testes para dentro da implementação do código? Neste cenário, surge como resposta a metodologia ágil, que aborda o conceito TDD, “desenvolvimento dirigido por testes”. O TDD é uma abordagem iterativa para desenvolvimento de software cujo princípio está em escrever testes automatizados. Ao praticar os testes unitários a equipe estará automatizando o desenvolvimento e trazendo funcionalidades extremamente importante para obtenção dos resultados produzidos e esperados pela equipe, durante a construção do produto.

Um processo de software compreende um conjuntos de atividades e resultados associados que auxiliam na produção de software. Dentre as várias atividades associadas, é possível citar requisitos e a codificação. O resultado do processo é a construção do produto final. Embora existam vários processos para desenvolvimento de software, existem atividades fundamentais comum entre todos: (SOMMERVILLE, 2003):

a) especificação de Software: nessa fase é a identificação das funcionalidades (requisitos) e das restrições do software. Geralmente é

a fase que há integridade entre desenvolvedor e cliente para definir características do produto.

b) Projeto e Implementação de Software: o software é produzido de acordo com as especificações. Nesta fase são propostos modelos através de diagramas e estes modelos são implementados em algumas linguagens de programação.
c) Validação do Software: o software é validado para verificar se todas as especificações foram implementadas.
d) Evolução do Software: o software precisa evoluir para continuar sendo útil ao cliente.

No decorrer dos anos de experiência e estudos feitos, é possível avaliar e apontar que ainda exista muitas organizações sem nenhum tipo de processo para construção de software. Geralmente pelo fato de processos tradicionais não se adaptarem a realidade da empresa. Acredita-se que, organizações de pequeno e médio porte não possuem recursos suficientes para adotar o uso de processos pesados. Ocasionando muitas organizações não utilizarem nenhum tipo de processo. E o resultado desta falta de sistematização na produção de software é, a baixa qualidade no produto final, e em consequência, atraso na entrega do software, custos elevados e inviabilidade de futura evolução no software.

Existem vários processos de software definidos na literatura da Engenharia de Software. É comum mesmo algumas organizações criarem seu próprio processo ou adaptar algum processo à sua realidade. Dentre os vários processos existentes, existem as metodologias tradicionais, que são orientadas a documentação, e as metodologias ágeis, que procuram desenvolver softwares com mínimo de documentação, mais com resultados mais imediatos.

De acordo com o levantamento bibliográfico e o cenário atual, foi realizado um estudo detalhado de metodologias ágeis e a utilização de TDD para o desenvolvimento de software. Em um modelo de processo que atendem os mínimos requisitos (SOMMERVILLE, 2010), foram adicionado o ciclo TDD para o desenvolvimento de software. Propus construir um processo de desenvolvimento de software que se adapte a este cenário, com proposito de aplicado em empresas de pequeno, médio e grande porte, assim, garantindo maior estabilidade no

desenvolvimento do produto e contribuindo para melhorar a qualidade do código. O conjunto de fases que compõe o projeto pode ser definido como “ciclo de vida do projeto” (figura 19). Para o modelo de desenvolvimento foi dividido em 4 fases: concepção, planejamento de desenvolvimento, desenvolvimento e entrega.

Figura 19 – Fase do projeto de desenvolvimento

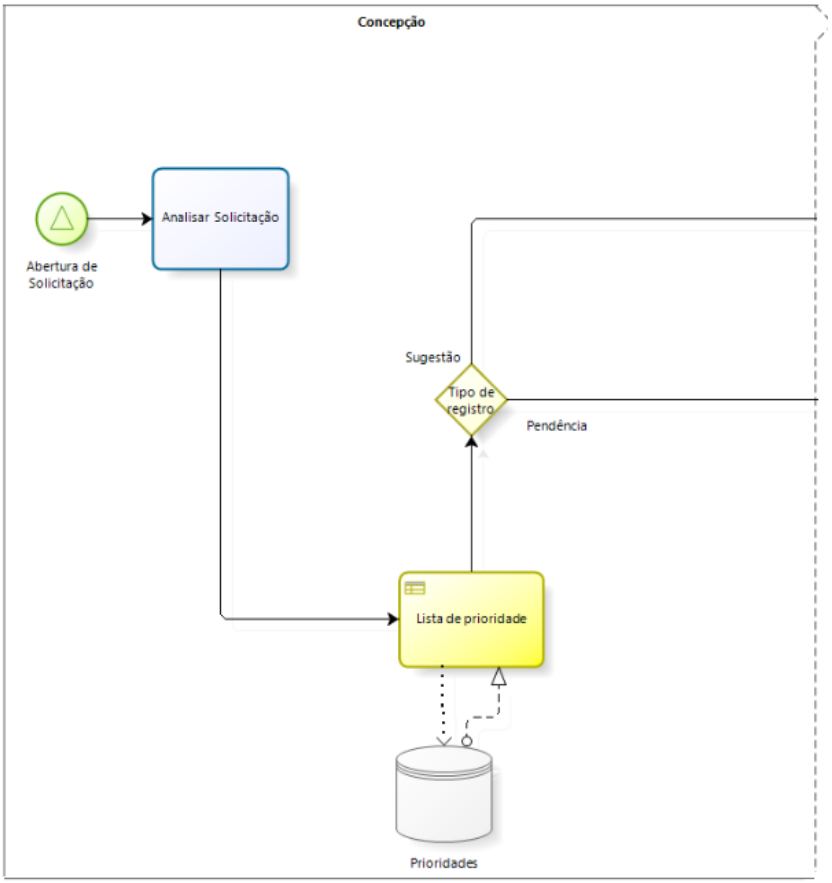
Fonte: Do autor (2014).

A seguir cada etapa do modelo de desenvolvimento será detalhada de forma simples e organizacional. As fases compõe o mínimo necessário para construção de um produto de software. A base é a qualidade, como resultado final.

6.1.6.1 Concepção

A fase de concepção é o início do processo de desenvolvimento onde ocorre a entrada e análise das solicitações (figura 20). Na abertura da solicitação seguida pela análise é imprescindível identificar o tipo de registro para priorização, podendo ser uma sugestão ou pendência. Para sugestão o fluxo deverá seguir para a análise de requisitos e para pendência deverá seguir diretamente para a correção, ambas na fase de desenvolvimento.

Figura 20 – Fase de concepção



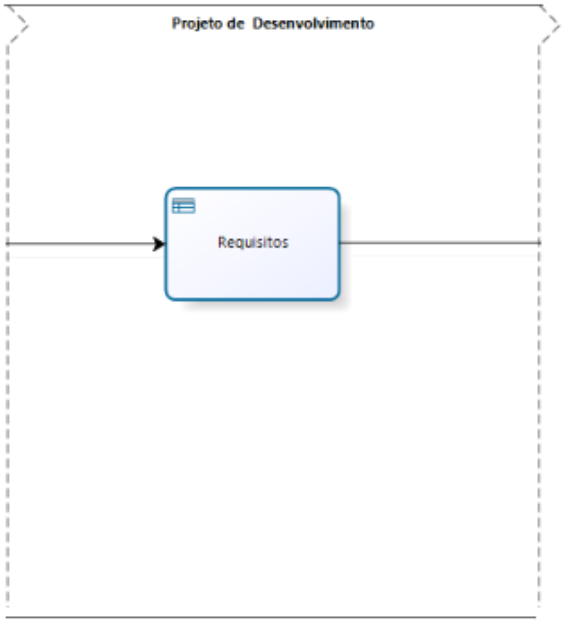
Fonte: Do autor (2014).

Nesta fase ocorre o processo de definição da arquitetura, dos componentes, das interfaces, bem como de outras características relacionadas ao sistema. O objetivo é verificar a viabilidade do projeto, bem como os riscos e definições de casos de uso mais críticos.

6.1.6.2 Projeto de Desenvolvimento

Após passar pela fase de concepção, a fase de projeto de desenvolvimento tem como objetivo o levantamento dos requisitos da solução a ser desenvolvida (figura 21).

Figura 21 - Fase de Projeto de Desenvolvimento



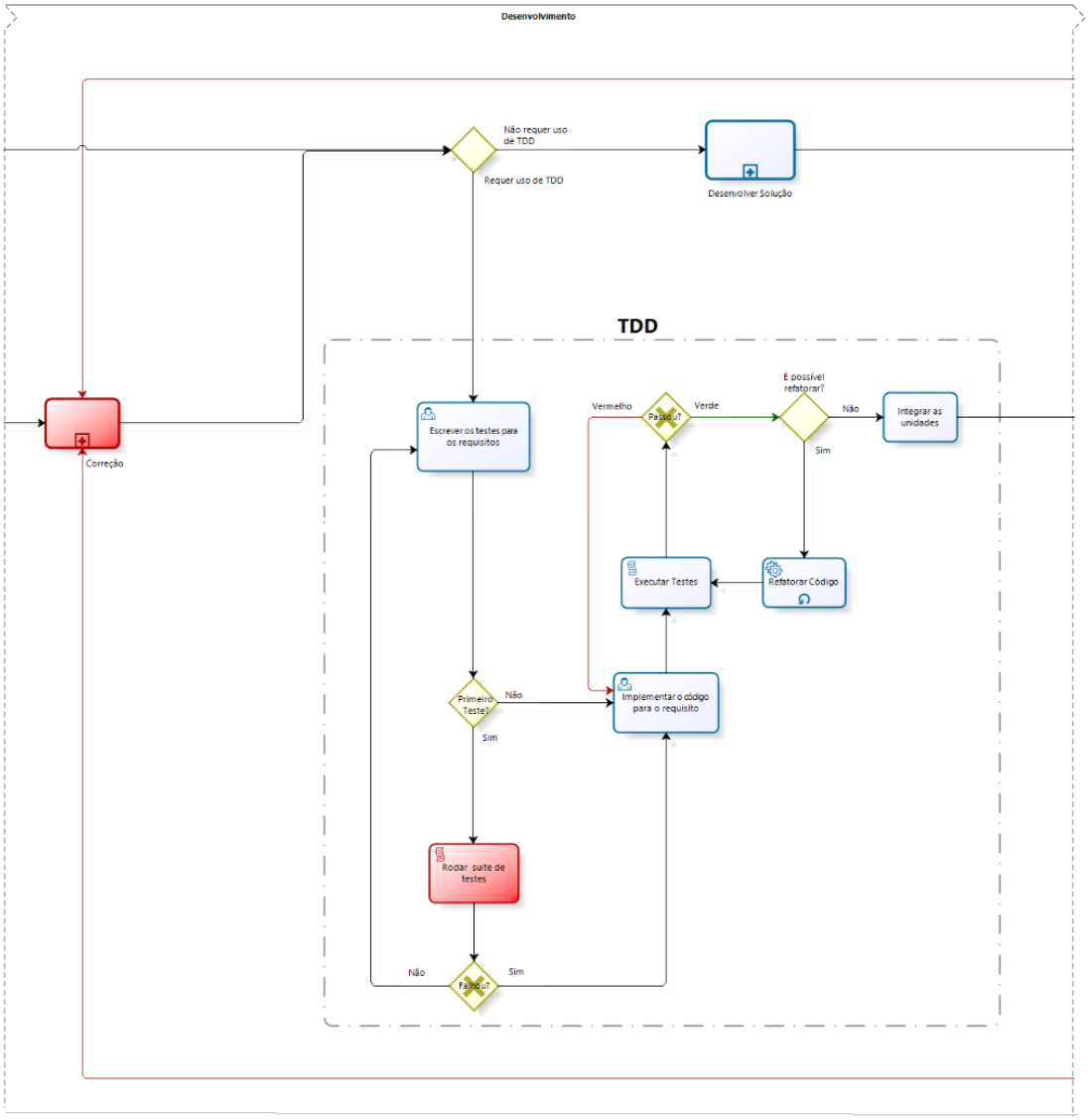
Fonte: do autor

No levantamento de requisitos, além da análise do negócio, alguns aspectos devem ser considerados como: arquitetura do sistema, linguagem de programação utilizada, Sistema Gerenciador de Banco de Dados (SGBD), padrões de interface gráfica, dentre outros. Essa fase tem como propósito descrever o quê o sistema deve fazer, promovendo o entendimento dos requisitos aos desenvolvedores.

6.1.6.3 Desenvolvimento

Nessa etapa, após o levantamento de requisitos proveniente de uma sugestão ou a identificação de uma pendência que vai exigir uma correção é iniciada etapa de codificação (figura 22).

Figura 22 – Fase Desenvolvimento



Fonte: Do autor

Para seguir para a implementação do código é necessário verificar se o projeto ira abordar o ciclo TDD na sua implementação, que encontra-se dentro da fase de desenvolvimento do processo. Conforme já citado no trabalho, alguns casos, o ciclo TDD, não são recomendáveis. Em suma, tudo vai depender da parte do projeto em que está trabalhando, da cultura da empresa e da “vontade” e

experiência do desenvolvedor, as vantagens serão reconhecidas por quem pratica, podendo ser observadas no design do código.

No cenário tradicional, muitas vezes a falta de coesão ou excesso de acoplamento é causado pelo fato do desenvolvedor pensar somente na implementação e esquecer como a classe vai funcionar perante o todo. Ao utilizar o processo de desenvolvimento PRD o desenvolvedor tem como refletir sobre como sua classe se comporta perante outras classes do sistema. O teste atua como o primeiro cliente da classe que está sendo escrita. Nele, o desenvolvedor toma decisões como o nome da classe, os seus métodos, parâmetros, tipos de retorno e entre outras decisões (figura 23).

Figura 23 – Requisitos de testes .

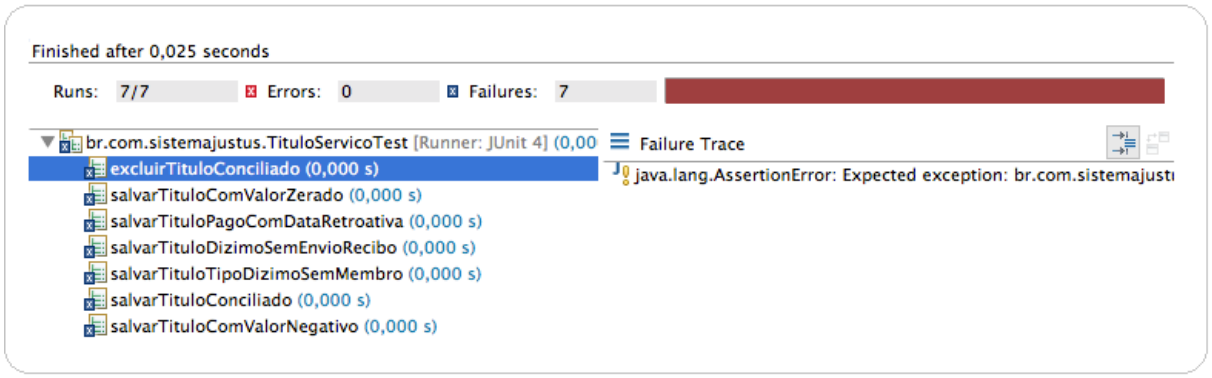
```
1  package br.com.sistemajustus;
2
3  import java.text.ParseException;
4  import java.text.SimpleDateFormat;
5  import java.util.Date;
6
7  import org.junit.Test;
8
9  public class TituloServicoTest {
10
11     private SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
12
13     @Test(expected = AplicacaoException.class)
14     public void salvarTituloTipoDizimoSemMembro() {
15         Titulo titulo = new Titulo();
16         titulo.setCodigo(1);
17         titulo.setConciliado(Boolean.FALSE);
18         titulo.setData(new Date());
19         titulo.setReciboEmail(Boolean.TRUE);
20         titulo.setTipo(TipoTitulo.DIZIMO);
21         titulo.setValor(10D);
22         titulo.setPago(Boolean.FALSE);
23
24         TituloServico servico = new TituloServico();
25         servico.verificarDizimoSemMembro(titulo);
26     }
```

Fonte: Do Autor (2014).

Após implementar os requisitos de testes para a classe, o programador deverá executa-los e os mesmos não deverão passar, pois ainda não existe código implementado, caso o teste passe, significa que esse teste não foi implementado corretamente e o desenvolvedor deverá voltar e analisar código de teste escrito.

Observação: para todas as classes foi implementado casos de testes, o exemplo na figura 24 só está ilustrando os requisitos de teste da classe *savarTituloTipoDizimoSemMembro.*

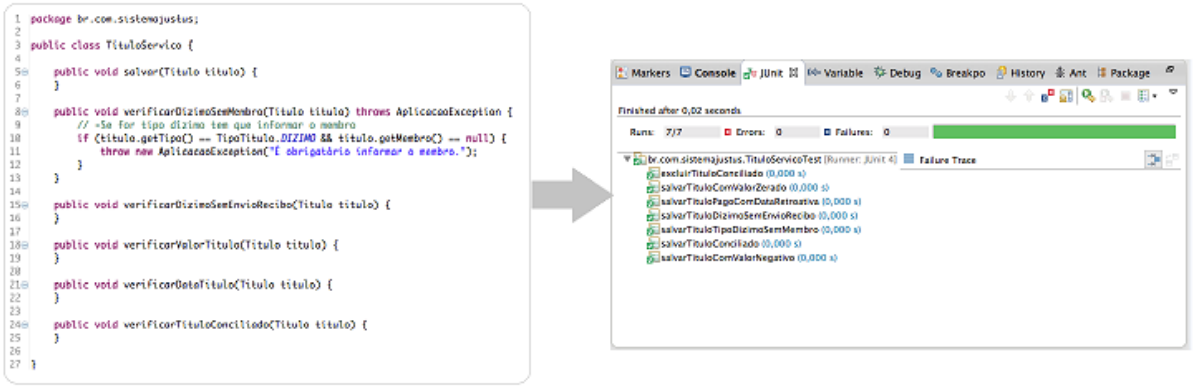
Figura 24 – Resultado do Teste (vermelho)



Fonte: Do autor (2014).

A figura 25 ilustra o resultado dos testes, após os requisito implementados, o desenvolvedor dá início a implementação do código para o requisito e em sequência executa o teste novamente, e veja passar (verde).

Figura 25 – Resultado dos testes.



Fonte: Do autor (2014).

Em casos que o teste não executa (barra-vermelha) deverá voltar e reimplementar o caso de teste para a funcionalidade, pois há algo de errado Em seguida é necessário analisar o código e verificar se é uma possível *refatoração do* código. Quando não existir mais possibilidade de *refatorar* o código, o mesmo estará pronto para ser integrado as demais unidades. A correta prática do ciclo TDD

proporciona que todo o código de produção possua pelo menos um teste de unidade.

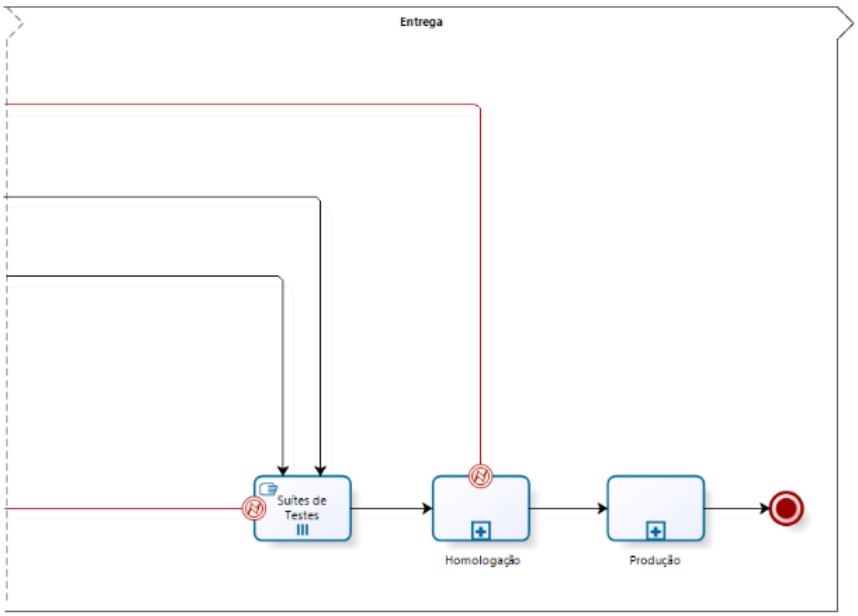
6.1.6.4 Entrega

Na fase de entrega temos testes, homologação e produção (figura 26). Na etapa de teste são aplicadas diversas atividades de casos testes a fim de validar o produto de software. São feitos testes em cada funcionalidade de cada módulo, levando em consideração a especificação feita na fase de projeto. O principal resultado é o relatório de testes, que contém as informações relevantes sobre erros encontrados no sistema, e seu comportamento em vários aspectos.

Em casos onde são encontrados erros o fluxo volta para a etapa de codificação para correção e posterior repetição dos testes.

Na etapa de homologação é feita a validação dos requisitos com o cliente ou com o gestor do produto com objetivo de alinhar a solicitação com o resultado gerado na saída do processo (figura 26).

Figura 26 – Fase Entrega



Fonte: Do autor.

Finalmente na etapa de produção, o software é atualizado com as novas funcionalidades solicitadas para o cliente final.

6.1.7 Aplicação do processo em Projetos

Foi realizado um estudo comparativo de dois projetos com o objetivo de avaliar os ganhos da utilização do processo de desenvolvimento PRD na linha de desenvolvimento de softwares. O estudo comparativo foi realizado entre dois processos contas a pagar e contas a receber. Ambos processos, foram desenvolvidos na empresa AG3 Sistema, localizada em Criciúma-SC, que realiza o desenvolvimento de sistemas voltados para Web, com soluções inovadoras que agregam conceitos de planejamento estratégico aplicados na gestão eclesiástica.

6.1.8 Métricas

O TDD tem se tornado uma técnica significativa para diversas empresas de software, uma vez que, aumentar o nível de entendimento do código e proporciona a qualidade do produto. A sua adoção torna a equipe mais homogênea, com revisão contínua e maior *feedback*.

Pode-se observar através dos indicadores que o projeto que utilizou TDD obteve um índice de defeitos de alta severidade muito menor comparado ao projeto que não utilizou. Sendo que o nível de complexidade de pontos de função e tecnologias de ambos os projetos eram muito semelhantes. A diferença, de fato, era a utilização da técnica de TDD aplicada apenas no projeto em um projeto.

Foi apresentado resultados de uma análise entre dois projetos de sistemas de informação, onde é possível observar uma grande quantidade de erros de baixa severidade identificada ao longo do desenvolvimento do sistema em ambos os projetos. Porém, esses erros, se comparados com outros de alta severidade, tiveram pequeno impacto no tempo e custo dos projetos.

No projeto contas receber, que não utilizou TDD, foi identificado um maior percentual de erros de alta severidade quanto comparado ao Projeto contas a Receber, que utilizou a prática. Esse fator aumentou o tempo de desenvolvimento do projeto e, através do *feedback* do cliente e outras métricas analisadas, comprometeu a qualidade do mesmo. No projeto que utilizou as práticas do TDD, apesar de ainda ocorrerem erros, foi possível notar que suas falhas foram identificadas e sanadas

facilmente ao longo da etapa de desenvolvimento, não impactando no prazo total do projeto, otimizando a qualidade do produto final.

6.2 RESULTADOS OBTIDOS

Em linhas gerais, pode-se dizer que o estudo alcançou os resultados esperados, pois ao aplicar o processo PRD na empresa AG3 Sistema, obteve-se casos de sucesso na implementação do código e dessa forma garantindo melhor qualidade no software, em suma, clientes satisfeitos.

O objetivo da construção do processo de desenvolvimento não foi comparar ou avaliar com modelos de referências, como o Modelo MPS.BR, mais sim, enfatizar que metodologias ágeis, como o método TDD, podem estar presente em processos de desenvolvimento e contribuir na qualidade dos sistemas produzidos. Sendo que, existe dois eixos de qualidade a serem considerados: a qualidade interna e a qualidade externa.

Muitos fatores contribuem para que inúmeros programadores deixem de utilizar TDD logo no início, primeiro, ao iniciar o desenvolvimento orientado a testes é preciso ler documentações, estudar metodologias ágeis e entender como realmente funciona o ciclo de TDD em um processo de desenvolvimento.

De fato ao utilizar TDD o programador “perde” em linhas de código por hora, mas ganha horas, dias, e até semanas na prevenção de novos erros e correção dos que ocorrem durante a implementação de uma nova f*eature*. Além disso, a classe terá teste unitário para ser rodada sempre que houver alterações no software.

Vimos que iniciar um processo TDD pode ser um trabalho árduo, mais os benefícios são relevantes, pois em pouco tempo utilizando testes o programador percebe mudanças relevantes em sua forma de programar. Em suma, o uso de TDD ajuda o programador a elaborar um código mais limpo e objetos concisos e com menos dependências. Também existem questões como segurança em qualquer alteração do software, uma vez que, todos as classes possuem pelo menos um teste unitário para cada alteração. Fazer qualquer alteração sem automatização de testes é realmente complicado, pois até então não sabe-se (ou lembra-se) ao certo quem afeta quem no sistema. Com a prática de TDD cada pequeno passo do software está

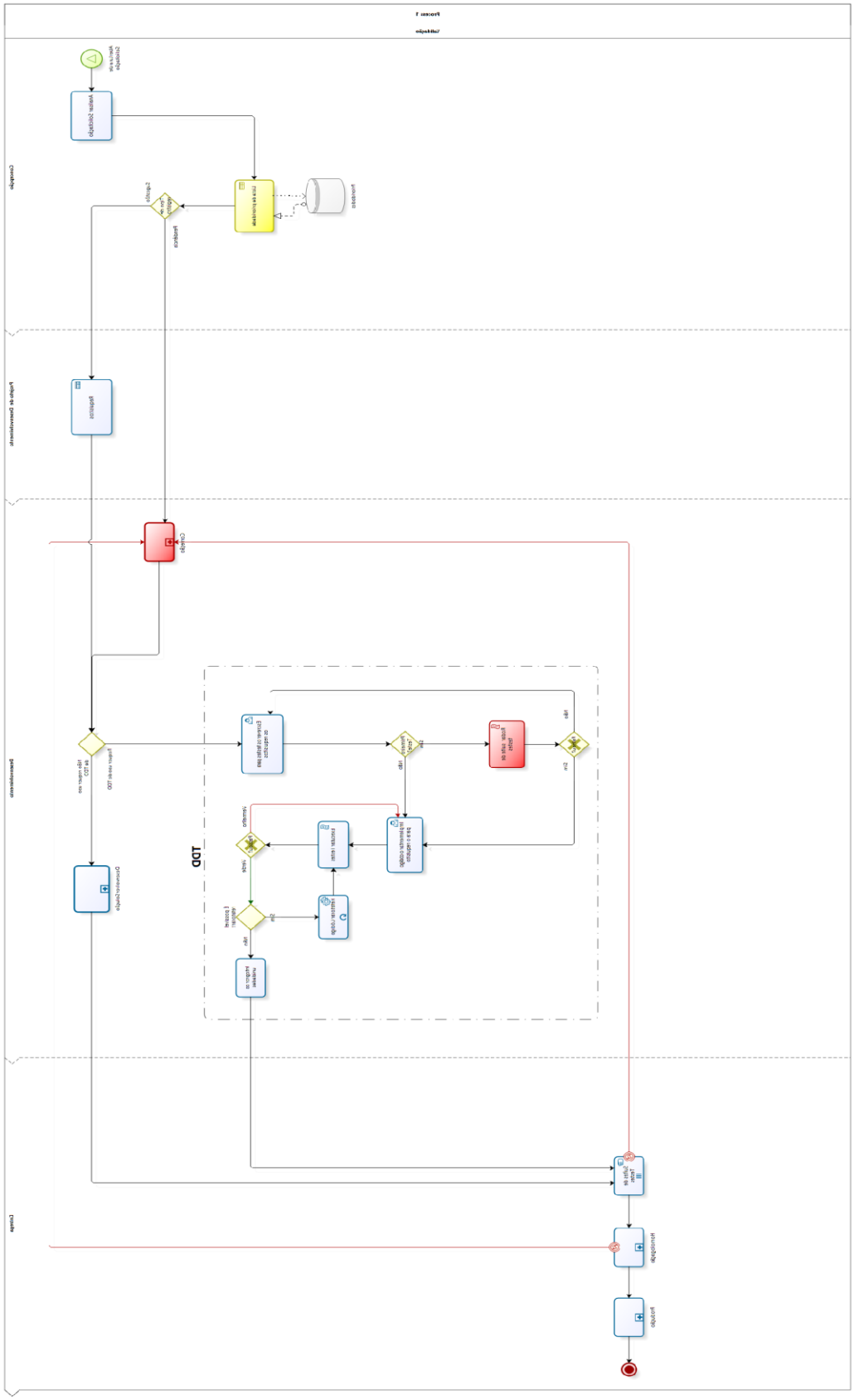
devidamente testado. Ou seja, com este cenário o programador pode realizar qualquer alteração sem medo e sem culpa.

Por prover mais segurança o trabalho em equipe torna-se muito mais proveitoso porque elimina discussões e dúvidas desnecessárias.

Ao criar testes descritivos, os mesmo servem como documentação do software. Isso é perceptível pelo fato que quando um desenvolvedor for rodar os testes (*scripts*), a documentação estará pronta. Com os testes unitários a "documentação" é gerada antes mesmo da nova *feature* ser implementada e permanece fiel a qualquer alteração.

A aplicação do processo nos projetos, obtiveram entradas e saídas, e suas características, bem como os artefatos, técnicas e ferramentas envolvidas. A figura 27 ilustra o processo de software PRD construído.

Figura 27 – Processo de desenvolvimento



Fonte: Do autor (2014).

A avaliação realizada na pesquisa mostram o processo PRD, auxiliou e garante melhor qualidade no desenvolvimento, diminui o tempo de constatar possíveis falhas. Além do tempo, o desenvolvimento guiado por testes faz com que a base de código cresça, devido aos testes criados durante o ciclo, e aumento no tempo de manutenção, já que conforme o sistema for sendo alterado os testes também deverão ser revisitados para que seja possível utilizá-los para garantir que o sistema se comporta como deveria e a qualidade ser mantida. Portanto, foi possível verificar que ao aplicar o processo de desenvolvimento as falhas no software foram mitigadas, baseado não só no levantamento bibliográfico, mas, também apoiado nos resultados e avaliações feitas no processo.

A produtividade no início do desenvolvimento do software foi um pouco demasiada, mais ao longo do projeto percebeu uma crescente produtividade, pois no surgimento de novas funcionalidades e serem implementadas, os desenvolvedores não precisaram testar as funcionalidade já existente. Apenas foi rodado os scripts prontos, necessitando testar apenas as novas funcionalidades.

Percebe-se que o projeto que foi implementado TDD ficou muito mais seguro, estável e testável do que o projeto que não utilizou TDD no desenvolvimento. Garantindo maior qualidade no software.

7 CONCLUSÃO

Em tempos de tecnologia e a procura constante por processos de desenvolvimento que visam promover a qualidade de software, os temas engenharia de software, processos e métodos ágeis se tornam um tanto presente entre as empresas. Nesta busca, adoção de métodos ágeis de desenvolvimento de software tornam-se objeto de averiguação, por serem conhecidos como metodologias leves e pregarem a satisfação do cliente como principal prioridade no desenvolvimento de software, e, em contrapartida, a procura das empresas pela melhoria de seus processos através métodos ágeis, como TDD é primordial.

Com a prática TDD nos ajuda a escrever um software melhor, com mais qualidade, e um código melhor, mais fácil de ser mantido e evoluído. Esses dois pontos são importantíssimo sem qualquer software, e TDD nos ajuda a alcançá-los.

Devida a abordagem indutiva da pesquisa foram expostos resultados de aplicações reais da técnica na indústria de desenvolvimento de sistemas. Os resultados foram produtivos e mostraram que o desenvolvimento guiado por testes torna o sistema mais maduro e manutenível do ponto de vista de qualidade proposto. Ao adotar o processo que apoia o desenvolvimento guiado por testes a quantidade de defeitos foi menor e mostrou que o projeto que utilizou a técnica foi possível localizar e remover as falhas de forma mais rápida.

A prática TDD no processo de desenvolvimento, pode-se obter resultados de testes realmente automatizados, com redução dos defeitos na fase funcional do desenvolvimento, aumento da produtividade da equipe, aumento da frequência de testes em projetos. Além disso, agregando valor ao produto final que será entregue ao cliente.

Muitos desenvolvedores imaginam que trabalhar com TDD pode ser difícil e desmotivador no início, mas ao persistir com essa metodologia é possível perceber um grande aumento na velocidade e produtividade de trabalho e fazer testes antes se torna proveitoso. Visto que, no momento que o desenvolvedor escrever um teste e faz o mesmo passar, ficará com a sensação de que pelo menos um pedaço da funcionalidade está pronto e testável.

Mensurar produtividade aplicando o ciclo TDD em projetos, significa levar em consideração que produtividade é a relação entre a quantidade de funcionalidades implementadas e o tempo que elas levam para serem implementadas, entendendo que a definição de “implementar a funcionalidade x” é construir um software capaz de atender a uma necessidade ou a um desejo do cliente. Não descartando a hipótese que softwares escritos com TDD não possuem bugs, mas o que estudos e desenvolvedores que utilizam a prática garante é que, os bugs em softwares escritos com TDD aparecem com menor frequência e, quando aparecem, são detectados com maior facilidade.

Por fim, baseado em relatos de experiências, estudos existentes, e com a concepção deste trabalho, pode-se concluir que metodologias ágeis como TDD aumentam a qualidade de software, principalmente quando é utilizado em um contexto industrial e que presam pela qualidade de seus produtos. TDD é uma boa prática de desenvolvimento de software, e que ao ser utilizada corretamente traz a empresa software bons resultados. O objetivo final é escrever código de qualidade, afim de entregar ao cliente um software de qualidade e dentro do prazo.

7.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Considerando que a aplicação do processo PRD em uma empresa de software de grande porte tomaria um tempo muito grande e em relação ao tempo de conclusão do trabalho, sugere-se que seja abordado em trabalhos futuros.

a) validar a solução em maior abrangência;
b) aplicar o trabalho em uma organização de grande porte;
c) validar uma maior quantidade de requisitos funcionais e não funcionais.

Ao aplicar o processo e realizar um treinamento intensivo aos desenvolvedores de software, a metodologia TDD poderá trazer resultados positivos a empresa.

REFERÊNCIAS

ANICHE, Mauricio Finavaro. **Como a prática de TDD influencia o projeto de classes em sistemas orientados a objetos.** Dissertação de Mestrado (Mestre em Ciência da Computação) – Instituto de Matemática e Estática – Universidade de São Paulo - IME-USP. São Paulo, 2012.

ASTELS, D. **Test-Driven development: A Practical Guide.** Prentice Hall. 2ª Ed. 2003.

BASTOS, Lucas. **Quais vantagens do desenvolvimento ágil de software?** ComputerWorld, 2013.
Disponível em: <http://computerworld.com.br/blog/opiniao/2013/10/21/quais-vantagens-do-desenvolvimento-agil-de-software/>. Acesso em 18 mai. 2014

BERNARDO, Paulo Cheque; KON, Fabio. **A Importância dos Testes Automatizados:** Controle ágil, rápido e confiável de qualidade. Artigo publicado na Engenharia de Software Magazine; 2008. Disponível em: <http://www.ime.usp.br/~kon/papers/EngSoftMagazine-IntroducaoTestes.pdf/>.
Acesso em 19 abr 2014.

BECK, Kent; ANDRES C. Extreme Programming Explained: Embrace Change. 2ª Ed. Addison-Wesley, 2004.

BECK, Kent. **Test-Driven Development:** By Example. Ed. Addison-Wesley Professional, 2002.

BERNARDO, Paulo Cheque. **Padrões de Testes Automatizados**. Monografia (Mestre em Ciência da Computação) – Instituto de Matemática e Estática – Universidade de São Paulo - IME-USP. São Paulo, 2011.

CASTELLS, M. **O poder da identidade – A era da informação:** economia, sociedade e cultura. São Paulo: Paz e Terra, 1999. v. 2.

CORTÊS, M. L.; CHIOSSI, T. C. S. **Modelos de qualidade de software**. Unicamp, Campinas, 2001.

DEBONI, J. **Modelagem Orientada a Objetos com UML.** São Paulo: Futura, 2003.

ERDOGMUS, H. et al. **On the effectiveness of the test-first approach to rogramming.** Ieee Transactions on Software Engineering, 2005.

FREEMAN, S., MACKINNON, T., PRYCE, N., and WALNES, J. **Mock Roles, not Objects. Berkshire House.**
Disponivel em: < http://jmock.org/oopsla2004.pdf/>2004. Acesso: 10 mar 2014.

GUERRA, Ana Cervigni; COLOMBO, Regina Maria Thienne. **Tecnologia da Informação:** Qualidade de Produto de Software. Brasília: MCT/SEPIN, 2009.

HIGHSMITH, J. **Agile Software Development Ecosystems**. Boston: Addison Wesley, 2002.

KOSKELA, Lasse. **Test Driven: TDD and Acceptance TDD for Java Developers.** Manning Publications. 2007

HENRY, J; Kafura, D; HENRY S.; **Matherson, L. Improving Software Maintenance at Martin Marietta**. IEEE Software, v. 11. 1994.

MANIFESTO, Agi Le. **Manifesto for Agile Software Development.** 2008**,** Disponível em <http://www.agilemanifesto.org>. Acesso em: 03 mai. 2014.

MOLINARI, Leonardo. **Inovação e Automação de Testes de Software**. Érica, 2010.

MOLINARI. Leonardo. **Testes Funcionais de Software**. Florianópolis: Visual Books, 2008. 224 p.

MYERS, Glenford J. **The Art of Software Testing,** Ed. John Wiley & Sons, New Jersey, USA, 1979.

PAN, Jiantao. **Software Testing**. Spring: Carnegie Mellon University, 1999 Disponível em: <http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/>. Acesso em: 18 mar. 2014.

PIRES, Cassio. **Extreme Programming:** um novo conceito em Metodologias de desenvolvimento. Acesso dia 31 mar. 2014.
Disponível em: <http://www.gojava.org/files/artigos/ExtremePrograming.pdf/>

PRANCHES, Henrique Feliciano. **Uma Avaliação Empírica de um Ambiente Favorável para o Desenvolvimento Dirigido por Teste**. 2007. 117 f. Tese (Mestrado) – Pontifícia Universidade Católica do Rio de janeiro.

PRESSMAN, Roger S. **Software Engineering**: A practitioner''s approach. 5ª.ed New York, USA: McGraw-Hill, 2001.

_____. **Engenharia de Software**. 6ª. ed. São Paulo: McGraw-Hill, 2006.

_____. **Engenharia de Software**, Sexta Edição. Editora MCGrawHill: Porto Alegre, 2010.

_____. **Engenharia De Software:** Uma Abordagem Profissional. 7ª. ed. São Paulo: Bookman. 2011.

SANTOS, Rafael Liu. **Emprego de Test Driven Development no Desenvolvimento de Aplicações.** 2010. 108 f. Dissertação (Bacharelado) – Universidade de Brasília, 2010.

SOARES, Michel Dos Santos. **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software.** Conselheiro Lafaiete: Unipac. Disponível em: <http://wiki.dcc.ufba.br/pub/Aside/ProjetoBitecIsaac/Met._Ageis.pdf/>. Acesso em: 10 jan 2014

SOMMERVILLE. Ian. **Engenharia de Software.** Addison Wesley, São Paulo, 2001.

_____. **Engenharia de software.** 6. ed. São Paulo: Addison Wesley, 2003.

_____. **Engenharia de software.** 8º. ed. São Paulo: Pearson, 2007.

TURNER R. BOEHM B. **Balancing Agility and Discipline: A Guide for the Perplexed.** Addison-Wesley Longman Publishing Co., 2005.

WILLIAMS, Laurie, MAXIMILIEN, E. Michael. **Assessing test-Driven Development at IBM.** In: ICSE 03,25., 2003, Portland. Artigo. Washington: Ieee Computer Society, 2003.

APÊNDICE(S)

Construção de um processo utilizando o conceito Test Driven Development (TDD) Como apoio e melhoria do processo de software

Morgana Cadorin Nazário[1], Gustavo Bisognin[2]
[1]Universidade do Extremo Sul Catarinense (UNESC) Caixa Postal 3167 – 88806-000 – Criciúma – SC – Brasil

[2]MSc. Professor do Curso de Ciência da Computação - Universidade do Extremo Sul Catarinense (UNESC)
Caixa Postal 3167 – 88806-000 – Criciúma – SC – Brasil
morcadorin@gmail.com, gbisog@gmail.com

Abstract. *With the growth of new technologies, software factories act strongly on the quality of the products developed. The search for solutions is persevering and the emergence of new processes, techniques and tools help design software, however, do not guarantee software quality and delivery on time. To minimize common problems and possible return of anticipated errors, agile methodologies come into play, and propose obtaining practical results in a shorter period, thus taking the focus of the case and putting in the product and the people who produce it. Extreme Programming (XP) has become quite popular among software developers, since the code discusses focused practices. Guided by growth tests (TDD) XP originated by inheriting various aspects in the development of software, focusing the use of TDD cycle to ensure the quality of the implemented code and therefore minimize failures.*

Resumo. *Com o crescimento de novas tecnologias, fabricas de softwares atuam fortemente na qualidade de seus produtos desenvolvidos. A busca por soluções é perseverante e o surgimento de novos processos, técnicas e ferramentas auxiliam na concepção de software, no entanto, não garantem a qualidade do software e a entrega dentro do prazo. Para minimizar problemas frequentes e retorno de possível erros antecipados, as metodologias ágeis entram em cena, e propõem a obtenção de resultados práticos em um período menor, assim, tirando o foco do processo e o colocando no produto e nas pessoas que o produzem. Programação extrema (XP) se tornou bastante popular entre desenvolvedores de software, uma vez que, discute práticas focadas no código. Desenvolvimento Guiado por Testes (TDD) foi originado através da XP, herdando diversos aspectos para o desenvolvimento de software, com foco a utilização do ciclo TDD para garantir a qualidade do código implementado e, por consequência, minimizar falhas.*

1. Introdução

Com avanço de novas tecnologia o crescimento da utilização de software aumenta, desta forma, à dependência se torna pleno e com maior probabilidade de enfrentar possíveis problemas. Neste cenário, procura-se incorporar a qualidade nos produtos ofertados pelas empresas de software, sendo que nem sempre e assa a realidade oferecida. Os problemas

derivados das falhas de software demonstram dados estatísticos que comprovam grandes prejuízos financeiros para empresas de software e também para o consumidor do produto, assim, ocasionando um desgaste muito maior em relação à qualidade.

Para obter a qualidade e diminuir o índice de defeitos na construção de um produto de software, a engenharia de software constitui e utiliza sólidos princípios de engenharia a fim de atrair softwares econômicos, confiáveis e eficientes. Os requisitos são necessidades explícitas do cliente e, para suprir e conquistar a satisfação do próprio, é essencial o desenvolvimento de um produto com a baixa existência de defeitos, erros ou falhas. (GUERRA; COLOMBO, 2009).

Para construir um produto de software é necessário lidar com problemas decorrentes, garantir a qualidade e diminuir os atrasos na entrega. Com isso as empresas de software buscam por métodos eficientes, que tem como objetivo melhorar a qualidade dos produtos de software e aumentar a produtividade no processo de desenvolvimento.

Nesse cenário, houve a necessidade, então, de se abordar a construção de software de modo que permita criar e evoluir produtos de software com qualidade, mais rápido e eficaz. Para este tipo de desenvolvimentos as metodologias ágeis são essenciais e o melhor método a ser adotado.

Metodologia ágil é definida como um conjunto de processos para desenvolvimento de software, ou seja, é uma coleção de metodologias baseada na prática para modelagem efetiva de sistemas de software. O termo “Método Ágil” identifica metodologias de desenvolvimento que adotam os valores e princípios do manifesto ágil. Dentre os inúmeros métodos ágeis existentes, podemos destacar: Extreme Programming (XP), Test Driven Development (TDD), SCRUM, Crystal Clear, entre outros.

Neste artigo será destacado a metodologia ágil TDD. O Desenvolvimento Guiado por Testes, do inglês Test Driven Development (TDD) é uma prática que se popularizou com metodologias ágeis, e sua origem foi através da Extreme Programming (XP), cujo objetivo é acelerar o desenvolvimento do software visando a melhoria contínua do processo (SANTOS, 2010). Para garantir a qualidade do software, a metodologia TDD é uma prática de desenvolvimento para progredir na qualidade do software, antecipando a identificação de erros, contribuir para o aumento da qualidade e como resultado reduzir custos.

Dentro do contexto apresentado, encontrou-se a possibilidade de desenvolver um processo de desenvolvimento de software baseado na metodologia TDD. Como também a utilização de testes automatizados na cobertura de testes unitários, empregando-se critérios ágeis. Com isso, objetiva-se reduzir o número de defeitos, código refatorado e aumento na qualidade do código implementado, assim, garantindo redução dos custos associados ao projeto de desenvolvimento de software.

2. Métodos Ágeis de Desenvolvimento de Software

Um software é caracterizado por tarefas e requisitos que tendem a mudar constantemente, deste modo, desenvolver um software se torna algo complexo (TURNER; BOEHM, 2005). Na busca por processos de desenvolvimento de software mais flexíveis e capazes de suportar ambientes de negócios altamente voláteis e imprevisíveis, várias organizações optam pela adoção de metodologias ágeis. Os métodos ágeis se propõe resolver diversos desses problemas através de práticas que favorecem o produto final.

Segundo Erdogmus (2005, tradução nossa), na última década o paradigma mais importante que surgiu no desenvolvimento de software foi o desenvolvimento ágil. Mesmo

que não represente o mais popular deles, é um assunto relevante e bastante discutido entre as empresas de software.

No Brasil o crescimento de métodos ágeis está se tornando imprescindível dentro das organizações. Um fator que faz isso acontecer é relacionado a vantagem do desenvolvimento ágil ser flexível às mudanças. Diferente das metodologias tradicionais que tentam prever todas as possibilidades sem deixar espaço para descobertas durante o desenvolvimento (BASTOS, 2013), metodologias ágeis são definidas como maneiras eficientes de guiar um projeto do início ao fim, sem complicações e mantendo o tempo inteiro o foco na entrega de valor para o cliente.

Ao aplicar os métodos ágeis nas empresas, há um relativo aumento na efetividade das equipes de desenvolvimento de produto, criando softwares que agregam valores ao negócio do cliente. Como também, o índice de projetos concluídos dentro do prazo e de acordo com os requisitos. Cada empresa aborda o método ágil conforme a sua necessidade.

O uso de metodologias ágeis são importantes no atual cenário, com demandas crescentes e fracassos recorrentes. A qualidade de software não está somente ligada em encontrar mais defeitos, mas também, é uma questão de não introduzir defeitos. Por isso é essencial que as equipes de desenvolvimento sejam capazes de reduzir a incidência de defeitos e os custos associados a depuração e correção dos mesmos.

Os objetivos de metodologias, como XP e TDD, é deixar o processo de desenvolvimento de software mais rápido, simples e com qualidade. A sua popularidade cresce devido a velocidade do mercado atual, onde respostas rápidas são cada vez mais exigidas. As empresas devem se aperfeiçoar constantemente para encontrar novas soluções que tragam ganhos para a qualidade do software e o aprendizado de novos conceitos deve ser adequada a realidade em que a empresa está inserida.

3. Desenvolvimento Guiado por Teste (TDD)

Desenvolvimento Guiado por Testes ou Test Driven Development (TDD), é um conjunto de técnicas associadas com Extreme Programming (XP). Essa técnica de desenvolvimento de sistemas tem sido usada esporadicamente por décadas. A referência mais antiga do seu uso foi na data de 1960. Mas somente se popularizou em 2001, por meio do livro TDD: By Example do autor Kent Beck.

A prática de TDD é conhecida por guiar o desenvolvedor durante o processo de criação do projeto de classes por meio de constante *feedback* sobre a qualidade do projeto. Ela se baseia na repetição de um pequeno ciclo de atividades. No processo tradicional de desenvolvimento de um sistema, o primeiro passo é projetar, em seguida implementar o projeto e por último testar a implementação do projeto (KOSKELA, 2007). Neste cenário, o desenvolvedor só descobrirá falhas na última etapa do ciclo, ou seja, depois de já ter implementado tudo que o projeto contemplava, a quantidade de código já é grande tornando a tarefa de corrigir a falha mais complexa.

A metodologia TDD repete pequenos ciclos de Desenvolvimento Iterativo e Incremental (IID), ou seja, constrói um sistema requisito por requisito, ao invés de construir todas as camadas e componentes e os integrando ao final do desenvolvimento (FREEMAN et al, 2004, tradução nossa). Portanto a implementação do requisito é melhorada de maneira progressiva até que a mesma esteja boa o bastante.

É comum alguns desenvolvedores relacionarem TDD a práticas de testes de software. Embora a criação de testes seja algo intrínseco ao processo, é dito que TDD auxilia o desenvolvedor a criar classes mais legíveis, mais coesas e menos acopladas. O teste é a

ferramenta que o programador utiliza para avaliar o projeto da classe que está sendo criada, dessa forma, guiar sua lógica de programação.

É importante que os testes sejam *baby steps*, ou seja, avance em passos bem curtos, para ter um bom controle do código. Com o passar do tempo, o programador começa a adquirir segurança nessa técnica, e pode ir um pouco além, mas é sempre prudente e recomendado que não se avance muito na hora de escrever um teste.

Ao aplicar TDD em um projeto, o programador estará trabalhando da seguinte forma: escreve um pouco de teste, um pouco de implementação e recebe *feedback* sobre o código. Isso acontece ao longo do desenvolvimento de maneira frequente. O desenvolvedor que não pratica TDD espera um tempo (às vezes longo demais) para obter o mesmo feedback do software.

Ao antecipar os possíveis erros, para o desenvolvedor será mais fácil melhorar o código, corrigir problemas, e reduzindo custo, já o desenvolvedor que recebeu a mesma mensagem muito tempo depois não receberá este mesmo *feedback*. De fato é muito mais fácil e rápido alterar o código que ele implementado recentemente do que o código escrito 3 dias atrás (ANICHE, 2012).

A prática TDD pode ser definido como uma prática de projeto de classes. De acordo com Beck, Astels e Martins (2010) a mudança na ordem do ciclo de desenvolvimento tradicional, apesar de simples, agrega diversos outros benefícios ao código codificado, entre os benefícios estão: maior simplicidade, menor acoplamento e maior coesão das classes criadas. Por ser flexível TDD pode ser adaptada a outros métodos, sejam Métodos Dirigidos por Planejamento ou Métodos Ágeis. Um bom exemplo é o processo XP, que incorpora suas técnicas de projeto e testes de unidades.

4. Modelagem de processo para o Desenvolvimento de Software

Por meio de gestão de processos, as empresas desenvolvedoras de software podem obter resultados positivos na qualidade do software. Clientes necessitam de software a todo momento, neste âmbito, desenvolver softwares torna-se a cada dia uma tarefa mais complexa onde a qualidade, retorno rápido e necessidades atendidas são requisitos primordiais.

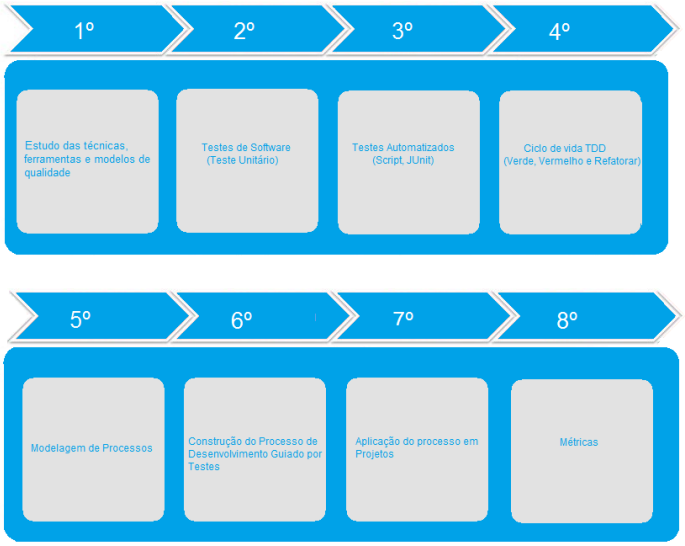
A meta é, que programadores e testadores que utilizem o processo de desenvolvimento guiado por teste, tenham apoio ao desenvolvimento do software, e assim, aumento da qualidade e o bom design do código.

O processo de desenvolvimento de software foi elaborado, tendo como base o conceito TDD, como apoio a melhoria do software guiado por testes. Baseado em metodologia ágeis, busca uma solução dos problemas ou, minimizando o índice de software de baixa qualidade.

Levando em consideração os pontos citados anteriormente, foi desenvolvido um processo genérico, ou seja, onde são definidas as fases e as atividades, mas a forma na qual as atividades serão executadas fica a critério do usuário, podendo ser aplicado no desenvolvimento do software.

Para o desenvolvimento do processo foram realizadas as etapas demonstradas na figura 1.

Figura 1 – Metodologia aplicada



Fonte: Do autor (2014)

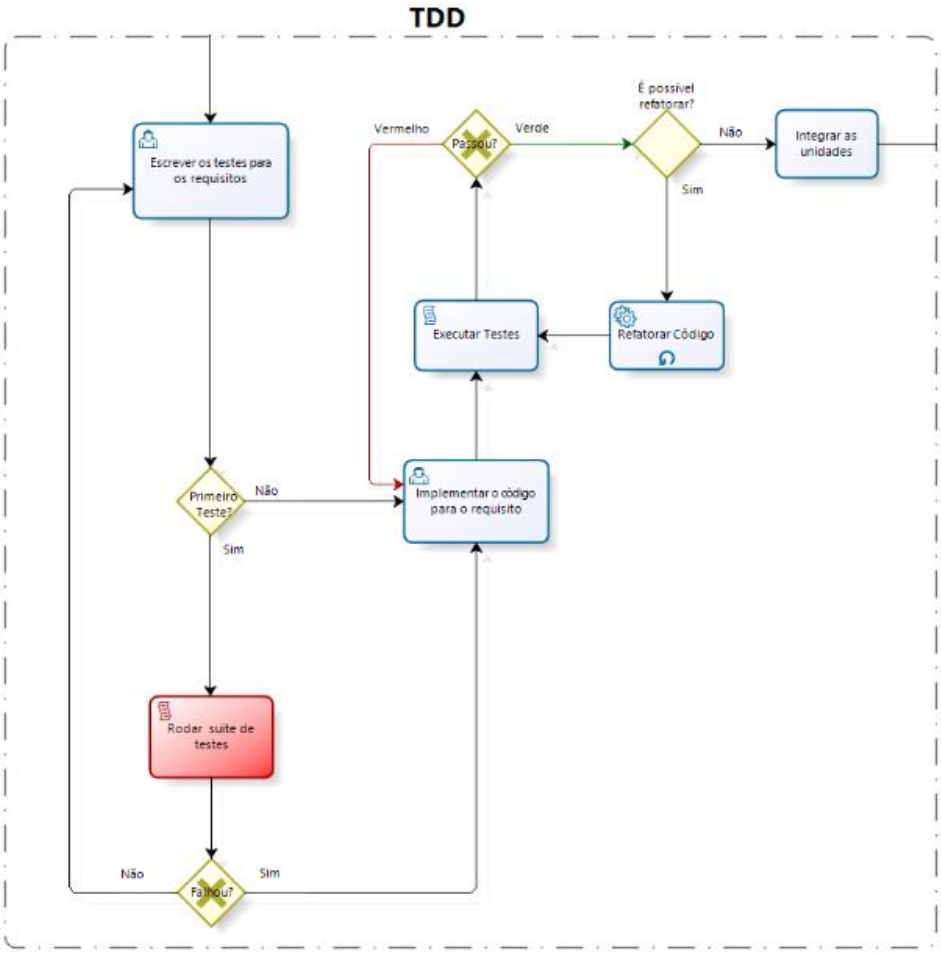
Na primeira etapa de elaboração do processo foi realizada a identificação de metodologias ágeis, no qual foi escolhido XP e TDD como principal fonte de pesquisa. Na construção do processo de desenvolvimento de software PRD, foi necessário buscar conhecimento em relação ao desenvolvimento de modelos de processo. Para uma visão geral dos ciclos de vida de desenvolvimento e sua aplicação nas empresas.

Na segunda etapa de desenvolvimento do trabalho, foram sintetizados os conceitos de testes unitários como base a utilização de TDD. Os casos de testes foram criados de acordo com os requisitos levantados para os processos em questão.

Na terceira etapa, foram reunidos os conceitos referente a automatização de testes unitários, utilizando a ferramenta jUnit como apoio aos testes, assim, gerando os scripts para o projeto.

Na quarta etapa mostra o ciclo TDD aplicado dentro do processo de desenvolvimento PRD (figura 2). O método é formado por pequenos passos, que são repetidos para o desenvolvimento de uma nova funcionalidade no sistema. Primeiramente inicializa pela implementação de um teste para o requisito, seguido para escrita do código, apenas o necessário para passar no teste, e finalmente a refatoração do código com padrões aceitáveis.

Figura 2 – Aplicação do ciclo TDD dentro do processo de desenvolvimento.



Fonte Do autor (2014)

O ciclo TDD possibilita ser aplicado em pequenas partes que compõe o sistema (teste de unidade para métodos de classe) e, permite que seja mais complexo para o desenvolvedor se afastar da especificação do projeto, dessa forma consiga controlar, com maior facilidade, estimativas para a conclusão do mesmo.

4.1. Construção do processo de desenvolvimento guiado por testes PRD

É perceptível que cada vez mais a qualidade do produto é exigido, e ferramentas, métodos e processos são fartos para auxiliar na qualidade, mas para isso é necessário aperfeiçoar o conhecimento e colocar em prática esses métodos. O desenvolvimento guiado por teste é uma forma diferente para o desenvolvedor trabalhar. Dessa forma, o desenvolvedor deixa de pensar, em primeiro lugar, no código e passa a pensar em como realizar o teste no código que será implementado.

Teste de software em modo geral, é a melhor maneira de avaliar a obtenção da qualidade do software produzido. Este processo de testes é realizado somente no final de cada projeto, o que leva a descoberta de possíveis erros um pouco tarde, ou no pior caso, quando o produto já está em produção. Então porque não trazer testes para dentro da implementação do

código? Neste cenário, surge como resposta a metodologia ágil, que aborda o conceito TDD, desenvolvimento dirigido por testes.

Um processo de software compreende um conjuntos de atividades e resultados associados que auxiliam na produção de software. Dentre as várias atividades associadas, é possível citar requisitos e a codificação. O resultado do processo é a construção do produto final. Embora existam vários processos para desenvolvimento de software, existem atividades fundamentais comum entre todos: (SOMMERVILLE, 2003). Especificação de software, projeto e implementação de Software, validação do Software e evolução do software.

No decorrer dos anos de experiência e estudos feitos, é possível avaliar e apontar que ainda exista muitas organizações sem nenhum tipo de processo para construção de software. Geralmente pelo fato de processos tradicionais não se adaptarem a realidade da empresa. Acredita-se que, organizações de pequeno e médio porte não possuem recursos suficientes para adotar o uso de processos pesados. Ocasionando muitas organizações não utilizarem nenhum tipo de processo. E o resultado desta falta de sistematização na produção de software é, a baixa qualidade no produto final, e em consequência, atraso na entrega do software, custos elevados e inviabilidade de futura evolução no software.

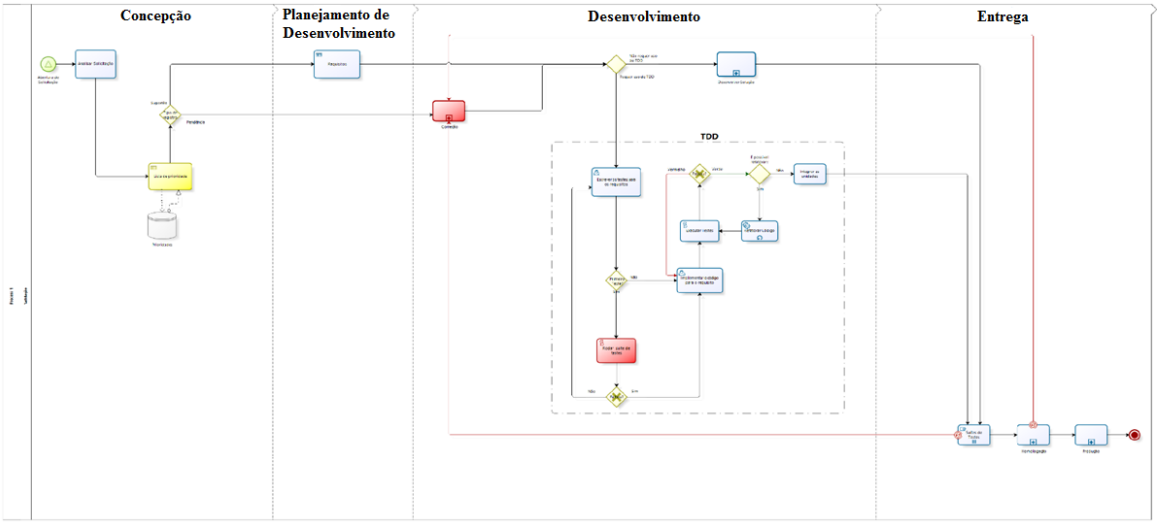
De acordo com o levantamento bibliográfico e o cenário atual, foi realizado um estudo detalhado de metodologias ágeis e a utilização de TDD para o desenvolvimento de software. Em um modelo de processo que atendem os mínimos requisitos (SOMMERVILLE, 2003), foram adicionado o ciclo TDD para o desenvolvimento de software. Foram construído um processo de desenvolvimento de software que se adapte a ao cenário atual, com proposito de ser aplicado em empresas de pequeno, médio e grande porte, assim, garantindo maior estabilidade no desenvolvimento do produto e contribuindo para melhorar a qualidade do código. O conjunto de fases que compõe o projeto pode ser definido como “ciclo de vida do projeto”. Para o modelo de desenvolvimento foi dividido em 4 fases: concepção, planejamento de desenvolvimento, desenvolvimento e entrega (figura 3).

Figura 3 – Fase do projeto de desenvolvimento

Fonte: Do autor (2014).

A seguir cada etapa do modelo de desenvolvimento será detalhada de forma simples e organizacional no processo de desenvolvimento PRD (figura 4). As fases compõe o mínimo necessário para construção de um produto de software. A base é a qualidade, como resultado final.

Figura 4 – Processo de desenvolvimento



Fonte: Do autor (2014).

A construção do processo de desenvolvimento PRD, pode auxiliar e garantir melhor qualidade no desenvolvimento e diminui o tempo de constatar possíveis falhas. Além do tempo, o desenvolvimento guiado por testes faz com que a base de código cresça, devido aos testes criados durante o ciclo, e aumento no tempo de manutenção, já que conforme o sistema for sendo alterado os testes também deverão ser revisitados para que seja possível utilizá-los para garantir que o sistema se comporta como deveria e a qualidade ser mantida. Portanto, foi possível verificar que ao aplicar o processo de desenvolvimento as falhas no software foram mitigadas, baseado não só no levantamento bibliográfico, mas, também apoiado nos resultados e avaliações feitas no processo.

5. Conclusão

Com o decorrer dos anos, o setor de desenvolvimento de software tem crescido significativamente, e a procura por processos de desenvolvimento que visam promover a qualidade de software é constante. Processos e métodos ágeis se tornam um tanto presente entre as empresas. Nesta busca, adoção de métodos ágeis de desenvolvimento de software tornam-se objeto de averiguação, por serem conhecidos como metodologias leves e pregarem a satisfação do cliente como principal prioridade no desenvolvimento de software, e, em contrapartida, a procura das empresas pela melhoria de seus processos através métodos ágeis, como TDD é primordial.

Devida a abordagem indutiva da pesquisa foram expostos resultados de aplicações reais da técnica na indústria de desenvolvimento de sistemas. Os resultados foram produtivos e mostraram que o desenvolvimento guiado por testes torna o sistema mais maduro e manutenível do ponto de vista de qualidade proposto. Ao adotar o processo que apoia o desenvolvimento guiado por testes a quantidade de defeitos foi menor e mostrou que o projeto que utilizou a técnica foi possível localizar e remover as falhas de forma mais rápida.

A prática TDD no processo de desenvolvimento, pode-se obter resultados de testes realmente automatizados, com redução dos defeitos na fase funcional do desenvolvimento, aumento da produtividade da equipe, aumento da frequência de testes em projetos. Além disso, agregando valor ao produto final que será entregue ao cliente.

Visto que, no momento que o desenvolvedor escrever um teste e faz o mesmo passar, ficará com a sensação de que pelo menos um pedaço da funcionalidade está pronto e testável.

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

Mensurar produtividade aplicando o ciclo TDD em projetos, significa levar em consideração que produtividade é a relação entre a quantidade de funcionalidades implementadas e o tempo que elas levam para serem implementadas, entendendo que a definição de “implementar a funcionalidade x” é construir um software capaz de atender a uma necessidade ou a um desejo do cliente. Não descartando a hipótese que softwares escritos com TDD não possuem bugs, mas o que estudos e desenvolvedores que utilizam a prática garante é que, os bugs em softwares escritos com TDD aparecem com menor frequência e, quando aparecem, são detectados com maior facilidade.

Conclui-se que os objetivos propostos foram atendidos, e metodologias ágeis como TDD aumentam a qualidade de software, principalmente quando é utilizado em um contexto industrial e que presam pela qualidade de seus produtos. TDD é uma boa prática de desenvolvimento de software, e que ao ser utilizada corretamente traz a empresa software bons resultados. O objetivo final é escrever código de qualidade, afim de entregar ao cliente um software de qualidade e dentro do prazo.

6. REFERENCES

ANICHE, Mauricio Finavaro. **Como a prática de TDD influencia o projeto de classes em sistemas orientados a objetos.** Dissertação de Mestrado (Mestre em Ciência da Computação) – Instituto de Matemática e Estática – Universidade de São Paulo - IME-USP. São Paulo, 2012.

BASTOS, Lucas. **Quais vantagens do desenvolvimento ágil de software?** ComputerWorld, 2013.
Disponível em: <http://computerworld.com.br/blog/opiniao/2013/10/21/quais-vantagens-do-desenvolvimento-agil-de-software/>. Acesso em 18 mai. 2014.

ERDOGMUS, H. et al. **On the effectiveness of the test-first approach to rogramming.** Ieee Transactions on Software Engineering, 2005.

FREEMAN, S., MACKINNON, T., PRYCE, N., and WALNES, J. **Mock Roles, not Objects. Berkshire House.**
Disponivel em: < http://jmock.org/oopsla2004.pdf/>2004. Acesso: 10 mar 2014.

GUERRA, Ana Cervigni; COLOMBO, Regina Maria Thienne. **Tecnologia da Informação:** Qualidade de Produto de Software. Brasília: MCT/SEPIN, 2009.

SANTOS, Rafael Liu. **Emprego de Test Driven Development no Desenvolvimento de Aplicações.** 2010. 108 f. Dissertação (Bacharelado) – Universidade de Brasília, 2010.

SOMMERVILLE. Ian. **Engenharia de Software.** Addison Wesley, São Paulo, 2003.

TURNER R. BOEHM B. **Balancing Agility and Discipline: A Guide for the Perplexed.** Addison-Wesley Longman Publishing Co., 2005.