METODOLOGIAS ÁGEIS PARA O DESENVOLVIMENTO DE SOFTWARE

Rafael de SANTI
rdesanti@ymail.com

Luiz Ricardo BEGOSSO
begosso@femanet.com.br

RESUMO: Este trabalho faz um estudo sobre algumas das metodologias utilizadas para agilizar o processo de desenvolvimento de software, como, Agile, Scrum, e XP. Aqui serão apresentadas as características de cada processo de desenvolvimento, a fim de demonstrar as vantagens dos métodos ágeis em relação aos métodos tradicionais.

PALAVRAS-CHAVE: Qualidade de software; Orientação a Objeto; Metodologias Ágeis; Extreme Programming; Scrum.

ABSTRACT: This work presents an study about some agile methodologies used in the current market, as Agile, Scrum, XP and other methods to streamline the development process. Here it will be presented the characteristics of each development process in order to demonstrate the advantages of agile methods over traditional methods.

KEYWORDS: Software quality; Object Oriented; Agile; Extreme Programming; Scrum.

0. Introdução

A tecnologia da informação permitiu um grande avanço na forma como as empresas desenvolvem seus negócios. A utilização de produtos de software permite que os diversos setores produtivos agilizem seus processos, gerando maior confiabilidade e reduzindo erros, alavancando os resultados das organizações.

Uma das grandes preocupações da indústria de software está relacionada com o processo de desenvolvimento de software, que deve ser o mais estável possível. Este fato vem de encontro às exigências do mercado globalizado e às necessidades das empresas de implantarem sistemas de Gerenciamento de Qualidade Total em todos os setores, inclusive o de desenvolvimento de software, estabelecendo ganhos tanto em termos de produtividade como de competitividade.

Neste contexto, as pessoas envolvidas tornam-se o ponto central das orientações. Para o sucesso de um programa de qualidade é fundamental que os funcionários das organizações possuam conhecimentos sobre metodologias e técnicas específicas, sejam

bem treinados, e possam compreender o valor de seus conhecimentos sobre os trabalhos que desenvolverão.

É comum as organizações promoverem um clima de aprendizagem a fim de que seus funcionários possam ter acesso ao conhecimento necessário para se engajar num programa de qualidade. Este clima procura aumentar a especialização intelectual dos envolvidos e a habilidade em desenvolver seus serviços na forma mais correta possível, aumentando a eficiência e reduzindo o retrabalho.

A indústria de software reconhece a dificuldade de se adicionar qualidade ao software desenvolvido, mas pode construí-lo com qualidade, passo a passo durante as etapas de seu desenvolvimento. Para isso, é necessário enfocar-se na prevenção de defeitos e em suas remoções imediatas.

Diversas metodologias para desenvolvimento de software foram elaboradas nas últimas décadas, entre elas pode-se citar o Modelo Essencial e o Modelo Orientado a Objetos.

A metodologia de desenvolvimento de software orientada a objetos permitiu que a indústria de software obtivesse ganhos consideráveis na produtividade e na qualidade dos produtos desenvolvidos pelos profissionais. Entretanto, diversos projetos requerem uma forma mais rápida de produção. Este requisito pode ser atendido pelas metodologias ágeis de desenvolvimento de software, como Lean, Agile, Scrum, XP, Kanban, entre outras.

Este trabalho tem o objetivo de realizar um estudo sobre as metodologias ágeis de desenvolvimento de software, contribuindo para a evolução dos conhecimentos necessários para o desenvolvimento de software com qualidade. Para isso, serão estudados os conceitos de Qualidade de Software e algumas metodologias ágeis para desenvolvimento de software.

1. Qualidade de Software

Um software de qualidade deve ser fácil de manusear, funcionar corretamente, ser de fácil manutenção e em caso de falhas, deve também manter os dados a salvo. O preço pago por uma falha de software pode ser irreversível. Isso demonstra o quanto a sociedade está cada vez mais dependente das tecnologias, tornando importante a o desenvolvimento de software com qualidade.

Duas organizações internacionais que atuam no ramo de normatização de software, a International Organization for Standardization – ISO e a International Electrotechnical Comission - IEC, definiram a qualidade de software como “A totalidade de

características de um produto de software que lhe confere a capacidade de satisfazer necessidades explícitas e implícitas" (RODRIGUES et al., 2015).

Necessidades Explicitas: são os requisitos que definem em que a aplicação deverá atuar e os seus objetivos, onde somente o desenvolvedor do projeto terá acesso.

Necessidades Implícitas: são os requisitos óbvios do sistema que não é necessário ser especificado, como um sistema hospitalar não pode deixar que um paciente sofra alguma consequência.

2. Levantamento de Requisitos

O levantamento de requisitos é uma etapa do processo de desenvolvimento de um software, visando a melhor condição para satisfazer e suprir as necessidades e expectativas do cliente em seu negócio e possibilitando oferecer melhorias e eficácia desde o início do projeto, garantindo assim a funcionalidade do sistema.

O início para toda a atividade de desenvolvimento de software é o levantamento de requisitos, sendo esta atividade repetida em todas as demais etapas da engenharia de requisitos. Sommerville (2003) propõe um processo genérico de levantamento e análise de requisitos, que contém as seguintes atividades:

- Compreensão do domínio: Os analistas devem desenvolver sua compreensão do domínio da aplicação;
- Coleta de requisitos: É o processo de interagir com os Stakeholders do sistema para descobrir seus requisitos. A compreensão do domínio se desenvolve mais durante essa atividade;
- Classificação: Essa atividade considera o conjunto não estruturado dos requisitos e os organiza em grupos coerentes;
- Resolução de conflitos: Quando múltiplos Stakeholders estão envolvidos, os requisitos poderão apresentar conflitos. Essa atividade tem por objetivo solucionar esses conflitos;
- Definição das prioridades: Em qualquer conjunto de requisitos, alguns serão mais importantes do que outros. Esse estágio envolve interação com os Stakeholders para a definição dos requisitos mais importantes;
- Verificação de requisitos: Os requisitos são verificados para descobrir se estão completos e consistentes e se estão em concordância com o que os Stakeholders desejam do sistema.

Os requisitos funcionais abordam o que o sistema deve fazer. Por exemplo:

O sistema deve permitir que cada professor realize o lançamento de notas das turmas nas quais lecionou.

O sistema deve permitir que o aluno realize a sua matrícula nas disciplinas oferecidas em um semestre.

A funcionalidade de um software refere-se ao conjunto de funções que devem satisfazer as necessidades explicitas e implícitas

Os requisitos não-funcionais declaram características de qualidade que o sistema deve possuir e que estão relacionadas às suas funcionalidades. Estas características podem ser divididas em termos de confiabilidade, usabilidade, portabilidade e segurança. A definição destas características será realizada a seguir, FILHO (2009).

Confiabilidade: são medidas quantitativas da confiabilidade do sistema, como por exemplo, o tempo médio entre falhas, recuperação de falhas, erros por milhares de linhas de código.

Usabilidade: são os requisitos que se relacionam ou afetam a usabilidade do sistema, como assuntos relacionados à facilidade de uso, ou a necessidade de treinamentos para os usuários.

Portabilidade: refere-se à facilidade de migrar o sistema para outras plataformas.

Segurança: são as particularidades sobre acessos ao sistema, segurança extra em login, restringir acesso de algumas pessoas, entre outros.

3. Orientação a Objetos

As novas aplicações e a manutenção das aplicações antigas são um dos grandes problemas no ciclo de vida do desenvolvimento de software. Esse problema é tido devido à como são particionadas as aplicações, que geralmente são usadas como módulos dependentes entre si.

A orientação a objetos é uma metodologia de desenvolvimento de software que utiliza componentes reutilizáveis, chamados de objetos. Os objetos são usados para acionar dados e funcionalidades do sistema, conforme descrito por SHALLOWAY (2004).

A ideia da programação orientada a objeto (POO) é que toda a estrutura de dados do projeto deve ser ligada às operações que realizam o acesso a ela. Essa conexão entre estrutura e operação é natural e inevitável, pois a estrutura é implementada para que as operações sejam executadas eficientemente e as operações devem conhecer a forma como as estruturas estão implementadas.

O encapsulamento é geralmente definido como uma ocultação de informação, com o objetivo de separar o usuário da classe do seu implementador, isto é, o cliente não pode ver o que há dentro do objeto.

A reusabilidade é o reaproveitamento de componentes para obter um desenvolvimento mais rápido sem diminuir a qualidade. Com isso os objetos podem ser usados diversas vezes sem realizar mudanças na implementação.

4. O Manifesto Ágil

No princípio da década de 1990, foram criados os "processos leves" com o objetivo de eliminar ou diminuir o processo que era excessivo no desenvolvimento do software. De acordo com GOMES (2013), alguns métodos de processos foram criados como o Scrum, Extreme Programming (XP), entre outros. O processo ágil preza quatro valores:

- Indivíduos e a interação entre eles mais que processos e ferramentas: trata-se de entender que uma equipe é formada por pessoas com pontos fortes e fracos, e não podem ser vistos como recursos. Porem as ferramentas também tem seu valor.
- Software em funcionamento mais que documentação abrangente: é uma resposta aos projetos tradicionais, em que por ser realizados por fases costumava-se passar de meses projetando apenas a documentação, que sozinha não acresce muito valor.
- Colaboração com o cliente mais que negociação contratual: é necessário para o processo ágil estar sempre pronto e é necessário se adaptar a mudança no meio dos negócios.
- Responder a mudanças mais que seguir um plano: a capacidade de adaptar-se em um mundo em constante mudança é uma qualidade essencial entregar projetos relevantes e bem-sucedidos.

O manifesto ágil é formado com mais 12 princípios:

- Entrega adiantada do software de qualidade.
- Os requisitos mudam de acordo com o agrado do cliente.
- Entrega de software frequentemente funcionando.
- Pessoas de negócios e desenvolvedores devem trabalhar juntos.
- Time sempre motivado.
- Conversar com a equipe é essencial.
- Software sempre funcionando.
- Ritmo constante entre todos os envolvidos no projeto.
- Excelência técnica e bom design aumenta a agilidade.

- Maximizar a quantidade de trabalho não realizado.
- Os melhores projetos surgem de equipes auto organizáveis.
- Em intervalos regulares a equipe se reúne para se tornar mais eficaz.

Os métodos ágeis são adaptativos por isso estão constantemente sofrendo adaptação. Os benefícios dos métodos ágeis são:

- Melhor Time-to-Market e Maior Retorno Sobre o Investimento: Quanto mais cedo os clientes puderem começar a utilizar o produto, mais rápido receberá o retorno do valor investido do desenvolvimento do produto.
- Maior Satisfação do Cliente e Melhor Gestão de Mudanças de Prioridades: O planejamento interativo permite que o cliente facilmente mude suas prioridades com impacto reduzido na produtividade da equipe, pois o detalhamento é feito apenas quando está próxima da ação.
- Melhor Visibilidade dos Projetos: Faz parte da cultura ágil manter as informações do projeto visíveis e transparentes através de ferramentas, com as quais a equipe pode acompanhar diariamente a evolução do projeto em relação às suas metas.
- Maior Produtividade: Em uma pesquisa, 75% dos entrevistados afirmaram ter alcançado uma maior produtividade após a transição para os métodos ágeis.
- Equipes mais Motivadas: Métodos ágeis promovem os casos em que organizações reportam uma diminuição significativa nas horas extras e madrugadas trabalhadas, são alguns dos fatores que contribuem para equipes mais motivadas e satisfeitas.
- Melhor Disciplina na Engenharia e Melhor Qualidade Interna: A utilização de práticas ágeis contribui para a entrega de produtos com melhor manutenção, extensibilidade e com menos defeitos.
- Processo de Desenvolvimento Simplificado: Os métodos ágeis são mais facilmente compreendidos pela equipe, e oferecem maior margem para otimização e adaptação para maior eficiência no contexto da organização em que está sendo aplicado.
- Redução de Risco: O planejamento interativo e as releases permitem que as prioridades do projeto sejam reajustadas constantemente.
- Para manter o risco sempre balanceado: entregas em prazos curtos oferecem maior velocidade do time para concluir o projeto.
- Redução dos custos: Equipes ágeis são menos propensas a desenvolver funcionalidades de baixa prioridade ou que se tornam desnecessárias ao longo do tempo.

4.1. Testes ágeis

Métodos ágeis defendem a ideia de que é preciso criar software de qualidade desde o início, em vez de assegurar a qualidade apenas no final do processo. Novas funcionalidades serão entregues com alta frequência e não se pode permitir que as alterações realizadas no software para criar as novas funcionalidades façam com que funcionalidades já existentes deixem de funcionar. É para prevenir isso que existem os testes de regressão que podem ser realizados manualmente ou automatizados. Além disso, deve-se prevenir defeitos a todo o momento, porque quanto antes detectado, mais rápido e barato será para resolvê-lo.

Quanto mais cedo o desenvolvedor descobrir que o problema existe, mais rápido e barato será para resolvê-lo.

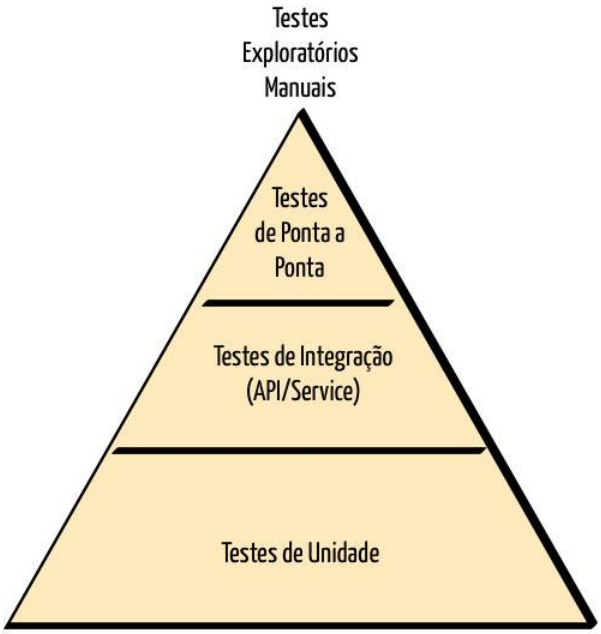

Figura 1- Pirâmide de testes.

A pirâmide de testes, criada por Mike Cohn é um modelo interessante que além de defender que diferentes tipos de testes podem ser combinados, também sugere uma proporção dentre os diferentes tipos, conforme ilustrado na Figura 1.

Testes de Unidade são mais rápidos de se automatizar e também são rápidos de executar, e representam a maior parte dos testes propostos pelo modelo.

Teste de integração (também chamados de testes de serviço ou testes de API) são mais completos do que os testes de unidade e, geralmente, combinam diferentes componentes que trabalham em conjunto para realizar em um determinado sistema.

Testes de ponta a ponta (também chamados de interface de usuário, testes de sistema, ou testes de aceitação) testam a aplicação desde a interface do usuário, por isso são os mais completos e, geralmente, são também os mais caros de se automatizar, e os mais lentos para se executar.

Testes exploratórios são executados sempre manualmente por um membro do time, com a finalidade de ir além dos outros testes. Ele não segue um script pré-planejado e se beneficia da capacidade de investigação, da observação, da análise e da adaptação do testador, conforme especificado por SATPATHY, MACEDO (2012).

4.2. Desenvolvimento guiado por testes

O Desenvolvimento Guiado por Testes (TDD) é uma técnica presente na maioria dos contextos. Essa técnica sugere a escrita de testes de unidade antes mesmo de se codificar a funcionalidade a ser testada. Com isso, é possível garantir um feedback rápido em caso de alterações que possam gerar efeitos colaterais quebrando outras funcionalidades.

O TDD é um processo em que o desenvolvedor começa a implementação de uma nova funcionalidade pelo teste e deve, o tempo todo, fazer de tudo para que seu código fique simples e com qualidade, conforme ilustrado na Figura 2.

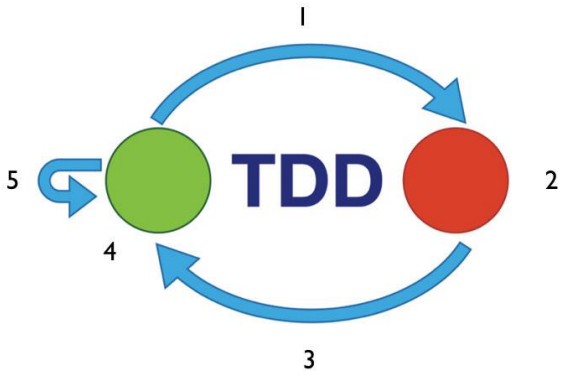

Figura 2- Desenvolvimento Guiado por Testes (TDD)

O TDD é realizado através de um ciclo com os seguintes passos:

- Adicione um teste rapidamente.
- Execute todos os testes e observe o novo teste falhar.
- Faça uma pequena mudança para fazer o teste passar.
- Execute todos os testes e observe se foram bem-sucedidos.
- Refatore.

O uso de TDD assegura que todo código adicionado ao repositório seja testado e, além disso, dá ênfase em se escrever apenas o código necessário para que os testes passem, evitando que o desenvolvedor escreva mais do que é necessário, contribuindo para um código mais simples, enxuto e também mais fácil de se dar manutenção.

4.3. Códigos limpos

Robert C. Martin, um dos criadores do Manifesto Ágil, defende que, ao longo do tempo, a base de código de um projeto vai se tornando mais bagunçada, impactando fortemente na produtividade da equipe.

De fato, muitos softwares, chegam a um tal ponto que precisam ser reescritos do zero para que possam continuar a ser evoluídos.

Algumas características de código limpo são:

- Elegante, fácil de se compreender, e agradável de se ler.
- Simples e direto.
- Segue princípios de programação.
- Sem duplicidade.
- Bem testado.
- Parâmetros, Métodos, Funções, Classes e Variáveis possuem nomes significativos.
- Código é autoexplicativo (sem necessidade de comentários para compreender o código).
- Código bem formatado (é possível ler o código sem precisar utilizar a barra de rolagem).
- Métodos e Classes devem ter uma única responsabilidade (princípio da responsabilidade única).

Trabalhar para manter o código sempre limpo é muito eficiente em termos de custos para a organização, uma vez que a manutenção do software será sustentável a médio e longo prazo.

5. SCRUM

O Scrum é uma das metodologias ágeis mais utilizadas pelas grandes empresas na atualidade, pois ela pode ser aplicada tanto no desenvolvimento de software quanto nas outras áreas.

Segundo SATPATHY (2013), o desenvolvimento do projeto começa com a definição do Backlog do produto em um workshop, que nada mais é do que a coleta dos requisitos do sistema, onde o Product Owner obtém dos Stakeholders ou usuários do sistema uma

lista de requisitos, ordenados de modo que as prioridades do sistema sejam executadas numa sequência correta de prioridades. Isso ocorre num prazo de um ou dois dias. Logo após há uma reunião para o planejamento das Releases e Sprints.

O planejamento das Releases serve para que a equipe identifique todos os dados do projeto, acompanhado de uma agenda de entrega prováveis. Raramente uma reunião dessas deve passar de quatro horas de duração. Já a reunião de planejamento das Sprints, é mais longa chegando a durar até oito horas em casos de Sprints de quatro semanas, e se reajusta quanto menor a Sprint.

Esta reunião é dividida em duas partes, onde na primeira parte o Product Owner lhes apresenta os requisitos em forma de histórias (histórias de usuário), para decidir com o feedback da equipe (também chamada de time) quais requisitos devem fazer partes de quais Sprints, e seus objetivos.

Na segunda parte da reunião o time decide como será feita as tarefas (taks) e o tempo (em horas) que será necessário para incrementar um produto potencialmente entregável.

Ao menos que a equipe trabalhe com algum software de planejamento, essas tarefas da Sprint são anotadas todas em um Quadro de Tarefas (Task Board), onde toda equipe terá acesso fácil a tais anotações.

No término da definição das Releases e Sprints, começa-se o trabalho propriamente dito, com uma reunião diária (Daily Scrum ou Daily Standup) de 15 minutos, para a informar a equipe sobre o progresso rumo ao objetivo da Sprint.

Após a conclusão da Sprint, o Scrum Master reúne a equipe e o Product Owner para uma Revisão de Sprint, que serve como parte de um Mecanismo de Inspeção e Adaptação do Scrum. A reunião dura até quatro horas para Sprints de quatro semanas e assim sucessivamente.

O primeiro objetivo desta reunião é tornar possível a comunicação entre o time de desenvolvimento e o Product Owner para que sejam sanadas as dúvidas do que foi e o que não foi feito na Sprint. Na segunda parte da reunião é feito uma apresentação do que foi feito para o Product Owner, para obter seu feedback. Por fim é feito uma atualização sobre o mercado ou do produto pelo Product Owner (VIEIRA, 2016).

6. XP

O eXtreme Programming (XP) é uma metodologia ágil de desenvolvimento software voltada para pequenas e médias empresas, para sistemas com requisitos vagos e com mudanças frequentes.

6.1. Scrum precisa das práticas técnicas do XP

Em seu livro a respeito do Scrum, SHALLOWAY (2004), refere-se ao método como um framework bastante popular atualmente e muito utilizado para o gerenciamento de projetos, produtos e times. Por dar mais ênfase no gerenciamento de atividades e tarefas, ele pode (e deve) ser utilizado em conjunto com outros métodos e processos que focam em engenharia ágil de software, como o eXtreme Programming.

Jeff Sutherland, cocriador do Scrum, diz que o Manifesto Ágil é sobre Scrum com práticas de engenharia do eXtreme Programming, e essa combinação é o framework preferencial.

6.2. Valores do XP

Este método baseia-se em quatro valores fundamentais sendo eles: o feedback do cliente para com a equipe de desenvolvimento; a comunicação que existe entre o cliente e a equipe de desenvolvimento permite que todos os detalhes do projeto sejam tratados com atenção e de forma adequada; a simplicidade faz a equipe implementar somente o necessário; a coragem trata da equipe acreditar que, utilizando as práticas do XP, será capaz de fazer o software evoluir com segurança e agilidade e por último o respeito com o time e o cliente é muito importante.

6.3. Práticas do XP

A metodologia XP é descrita por WILDT (2015), como um processo com diversas práticas, que serão descritas a seguir.

Cliente Presente: Para que a troca de feedback possa ocorrer e o cliente possa obter o máximo de valor do projeto, é essencial que ele participe ativamente do processo de desenvolvimento.

Jogo do Planejamento: O XP é dividido em releases e interações, de modo que o cliente e a equipe tenham diversas oportunidades de se reunir para revisar o planejamento.

Stand Up Meeting: Trata-se de uma reunião rápida que recebe o nome de stand up meeting.

Programação em Pares: No XP, os desenvolvedores implementam as funcionalidades em pares, ou seja, diante de cada computador, existem sempre dois desenvolvedores. Esta prática permite que o código seja revisado permanente, enquanto é construído.

Desenvolvimento Guiado pelos Testes: Os desenvolvedores escrevem testes para cada funcionalidade antes de codificá-las.

Refactoring: É o ato de alterar um código sem afetar a funcionalidade que ele implementa. É utilizado para tornar mais simples de ser manipulado e se utiliza fortemente dos testes descritos anteriormente.

Código Coletivo: No XP os desenvolvedores têm acesso a todas as partes do código e podem alterar aquilo que julgarem importante sem a necessidade de pedir autorização de outra pessoa.

Código Padronizado: A equipe estabelece padrões de codificação, que servem também para tornar o sistema mais homogêneo e permitir que qualquer manutenção futura seja efetuada mais rapidamente.

Design Simples: Os desenvolvedores implementam uma simplicidade no design onde só é implementado o necessário para o funcionamento perfeito do sistema, e assim as implementações das necessidades futuras fiquem mais simples de ser executadas.

Metáfora: Para facilitar a criação de um design simples, a equipe de desenvolvimento utiliza metáforas, já que elas têm o poder de transmitir ideias complexas de forma simples, através de uma linguagem comum que é estabelecida entre a equipe de desenvolvimento e o cliente.

Ritmo Sustentável: O XP recomenda que os desenvolvedores trabalhem apenas oito horas por dia e evitem fazer horas-extras, visto que é essencial estar descansado a cada manhã, de modo a utilizar a mente na sua plenitude ao longo do dia.

Integração Continua: Quando uma nova funcionalidade é incorporada ao sistema, ela pode afetar outras que já estavam implementadas. Para assegurar que todo o sistema esteja sempre funcionando de forma harmoniosa, a equipe pratica a integração contínua que leva os pares a integrarem seus códigos com o restante do sistema diversas vezes ao dia.

Releases Curtos: A equipe produz um conjunto reduzido de funcionalidades e coloca em produção rapidamente de modo que o cliente já possa utilizar o software no dia-a-dia e se beneficiar dele.

6.4. Papéis do eXtreme Programming

Segundo TELES (2004), uma equipe que utiliza o XP normalmente é composta por pessoas que representam os seguintes papéis: Gerente de Projeto, Coach, Analista de Teste, Redator Técnico e Desenvolvedor. Esses papéis serão descritos a seguir:

Gerente de Projeto: Responsável pela parte administrativa do sistema e é ele também que se relaciona com o cliente assegurando que ele esteja sempre ativo.

Coach: Sua tarefa principal é assegurar o bom funcionamento do processo e buscar formas de melhorá-lo continuamente.
Analista de Teste: fornece feedback para a equipe rapidamente, de modo que ela possa fazer as correções com rapidez e evitar que os problemas se propaguem.
Redator Técnico: O redator técnico ajuda a equipe de desenvolvimento a documentar o sistema.
Desenvolvedor: O desenvolvedor é a pessoa que analisa, projeta e codifica o sistema. Em suma, é a pessoa que efetivamente constrói o software.

7. Conclusão

As empresas de desenvolvimento de software estão cada vez mais competitivas, e com o intuito de destacar das demais, estão evoluindo com a utilização de métodos ágeis, sem perder a qualidade e obtendo maior rapidez no desenvolvimento de software que é produzido.
Neste artigo foi possível obter maior conhecimento sobre a importância dos analistas no levantamento de requisitos agregando qualidade ao software e como é primordial a atenção que se proporciona ao projeto, permitiu-me adquirir um amplo conhecimento em relação a orientação a objeto, Agile, Scrum e o Extreme Programming, que são métodos que tem como peculiaridades e objetivos tornar os processos de desenvolvimento mais ágil.
Enfim, conclui-se que grande parte das empresas que não utilizam estas metodologias são aconselhadas por profissionais da área, que recomendam a utilização destes métodos, para tornar as empresas mais eficientes no desenvolvimento de software.

REFERÊNCIAS BIBLIOGRÁFICAS

FILHO, Wilson de Pádua Paula – Engenharia de software: fundamentos, métodos e padrões. São Paulo: LTC, 2009.
GOMES, André Faria – Agile: Desenvolvimento de software com entregas frequentes e foco no valor de negócio. São Paulo: Casa do Código, 2013.
RODRIGUES, Ana Nathalie; MOURA, Mirtes; RODRIGUES, Paula; SANTOS, Vanusa dos. Qualidade de Software – Parte 01. Disponível em: <http://www.devmedia.com.br/qualidade-de-software-parte-01/9408>. Acesso em: 09.nov. 2015.

SATPATHY, Tridibesh – A Guide to the SCRUM BODY OF KNOWLEDGE. Phoenix: SCRUMstudy™, 2013.

SBROCCO, José Henrique Teixeira de Carvalho; MACEDO, Paulo Cesar – Metodologias Ágeis: Engenharia de Software Sob Medida. São Paulo: Editora Érica, 2012.

SHALLOWAY, Alan – Explicando Padrões De Projeto: Uma Nova Perspectiva Em Projeto Orientado A Objeto. Porto Alegre: Grupo A, 2004.

TELES, Vinícius Manhães – Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec Editora, 2004.

VIEIRA, Denisson. Scrum: A Metodologia Ágil Explicada de forma Definitiva. Disponível em: <http://www.mindmaster.com.br/scrum>. Acesso em: 12.jan. 2016.

WILDT, Daniel; MOURA, Dionatan; LACERDA, Guilherme; HELM, Rafael – eXtreme Programming. São Paulo: Casa do Código, 2015.