UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

***CAMPUS* LUIZ MENEGHEL - CENTRO DE CIÊNCIAS TECNOLÓGICAS**

SISTEMAS DE INFORMAÇÃO

RONI LUCAS FRANCISCO XAVIER

METODOLOGIAS TRADICIONAIS E METODOLOGIAS ÁGEIS: UMA ANÁLISE COMPARATIVA ENTRE RATIONAL UNIFIED PROCESS (RUP) E SCRUM (FRAMEWORK ESTRUTURADO)

Bandeirantes

2017

RONI LUCAS FRANCISCO XAVIER

METODOLOGIAS TRADICIONAIS E METODOLOGIAS ÁGEIS: UMA ANÁLISE COMPARATIVA ENTRE RATIONAL UNIFIED PROCESS (RUP) E SCRUM (FRAMEWORK ESTRUTURADO)

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Me. Fabio de Sordi Junior.

Bandeirantes

2017

RONI LUCAS FRANCISCO XAVIER

METODOLOGIAS TRADICIONAIS E METODOLOGIAS ÁGEIS: UMA ANÁLISE COMPARATIVA ENTRE RATIONAL UNIFIED PROCESS (RUP) E SCRUM (FRAMEWORK ESTRUTURADO)

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

COMISSÃO EXAMINADORA

_____
Prof. Me. Fabio de Sordi Junior.
UENP – *Campus* Luiz Meneghel

_____
Prof. Me. Jose Reinaldo Merlin
UENP – *Campus* Luiz Meneghel

_____
Prof. Dr. Mauricio Massaru Arimoto
UENP – *Campus* Luiz Meneghel

Bandeirantes, 26 de junho de 2017

Dedico este trabalho ao Mestre dos mestres, Jesus.

AGRADECIMENTOS

Agradeço ao Mestre Jesus e à toda hierarquia da luz que me instruiu com diversos tipos de conhecimentos e inspirou a continuar desenvolvendo este trabalho mesmo diante das lições a serem aprendidas; agradeço meu último orientador Fábio, que mesmo o trabalho estando em desenvolvimento aceitou assumir tal responsabilidade me auxiliando em meio ao processo de construção; agradeço a professora Juliana que também foi minha orientadora antes do professor Fábio, que me mostrou diversos saberes e também me ajudou na construção e sugestões em relação à pesquisa, sendo o principal diferencial, o fato de já não mais estar vinculada à instituição; agradeço também à minha mãe Nadir que me apoio fortemente nesta jornada acadêmica, sempre sendo paciente e atenciosa; ao meu irmãozinho Robson por entender meus posicionamentos diante de certas mensagens entregues; agradeço ao meu super grande amigo Dionatan, que mesmo tendo igualmente sua pesquisa para realizar ainda conseguiu desprender grandes momentos para me auxiliar na construção desta pesquisa e na verdadeira busca do conhecimento maior; agradeço ao meu pai Silvio (Xavier), que mesmo não estando mais fisicamente comigo, ajudou grandiosamente no desenvolvimento do rapaz que me tornei; agradeço minha vó Conceição e minha Tia Fátima que me deram grandes forças também em diversos seguimentos da minha vida; agradeço meus outros tios pelo apoio nessa jornada terrena; agradeço à minha prima Jheniffer, que se souber que agradeci todo mundo e não a mencionei ficará brava comigo (brincadeira), agradeço pelo apoio e pelas brincadeiras também; agradeço a dirigente da igreja onde me sinto em meio à família, irmã Cidinha pelas orações e palavras de conforto quando mais houve a necessidade; agradeço por fim, mas de modo algum menos importante, alguns professores que me mostraram que a busca do conhecimento pode ser bem mais atraente do que de fato imaginamos, sendo os mais importantes: Luiz Lomba, Tamara Baldo, Wellington Della Mura, Luiz Fernando, Thiago Coleti, André Menolli, Rafaella Neitzel e outros.

Quanto mais busco o conhecimento, menos pareço saber.
(Roni Lucas F. Xavier)

RESUMO

Devido a popularização cada mais maior de sistemas computacionais voltados à gestão empresarial, emissão de nota, controle de produção, entre outros e o aumento nas exigências sobre como seu comportamento pode ou não influenciar os envolvidos. Levando em consideração este aumento na popularização de programas, observa-se a necessidade de escolha sobre como todo este processo de desenvolvimento deverá ser conduzido de modo a comportar as mudanças provenientes de uma sociedade cada vez mais exigente e volátil. Assim, o presente trabalho visa elicitar as características do Rational Unified Process – RUP, uma metodologia tradicional de desenvolvimento de software, e do SCRUM que é um framework estruturado de metodologia ágil, para fins de comparação e entendimento sobre o comportamento de ambas no processo de desenvolvimento de um novo software. Esse paralelo entre as metodologias será realizado objetivando a análise frente aos diversos tipos de projetos, equipes e organizações. A comparação será realizada por meio de atividades consideradas comuns a um projeto genérico de desenvolvimento de software, procurando com isso, o embasamento em suas características de modo mais geral, para fins de decisão de implantação ou não em dado projeto.

Palavras-chave: Metodologia Tradicional. Metodologia Ágil. Rational Unified Process. RUP. Framework Estruturado. SCRUM. Análise Textual. Desenvolvimento de Software. Processo de Software.

ABSTRACT

Due to the ever greater popularization of computational systems aimed at business management, note issuance, production control, among others and the increasing demands on how their behavior may or may not influence those involved. Taking into account this increase in the popularization of programs, there is a need to choose how the entire development process should be conducted in order to accommodate changes from an increasingly demanding and volatile society. Thus, the present work aims to elicit the characteristics of the Rational Unified Process - RUP, a traditional software development methodology, and the SCRUM that is a structured framework of agile methodology, for comparison and understanding of the behavior of both in the process of Development of new software. This parallel between the methodologies will be carried out aiming at the analysis of the different types of projects, teams and organizations. The comparison will be carried out through activities considered common to a generic software development project, seeking with this, the basement in its characteristics more generally, for purposes of decision of implantation or not in a given project.

Keywords: Traditional Methodology. Agile Methodology. Rational Unified Process. RUP. Structured Framework. SCRUM. Textual Analysis. Software development. Software Process

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ATD	Análise Textual Discursiva
IBM	International Business Machines
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
TDD	Test Driven Development
T.I.	Tecnologia da Informação

SUMÁRIO

1. INTRODUÇÃO

A engenharia de software surge para oferecer métodos definidos e consistentes para o desenvolvimento de *software* embasando-se no aprimoramento técnico da qualidade envolvida no projeto para que o mesmo apresente qualidade em todas as fases, nos processos de desenvolvimento garantindo um roteiro lógico, nos métodos que são responsáveis por promover toda a informação técnica que a equipe precisa para desenvolver e por fim nas ferramentas que servirão de apoio para os anteriores. (PRESSMAN, 2011).

Em dado momento da história os métodos tradicionais foram os responsáveis por ordenarem de modo sistemático o processo de desenvolvimento de software para tentar terminar com a desordem, que por sua vez, foi crescendo conforme o tempo foi passando e ainda existe (PRESSMAN, 2011).

Com a finalidade de minimizar o caos no processo de desenvolvimento de software, surgem novas metodologias, como por exemplo os métodos ágeis apresentando quantidade reduzida de documentação e novos enfoques dentre suas fases de construção, ou seja, apenas atividades que realmente são necessárias são as que criam, enquanto os métodos tradicionais possuem documentação em todas as suas fases de desenvolvimento, gerando grandes quantidades de artefatos. (PRESSMAN, 2011).

Devido ao rápido avanço na TI - Tecnologia da Informação, a cada instante surgem novas tecnologias (BALTZAN, 2016), tornando-se necessário adequar os diversos processos da empresa, principalmente aqueles cujo foco principal é o desenvolvimento de *software*. Diante do exposto surge a seguinte pergunta: Qual a metodologia de desenvolvimento de *software* mais adequada para cada projeto quando estima-se construir um produto de qualidade que atenda às necessidades e expectativas do cliente, a um preço justo e entrega de acordo com o prazo definido inicialmente de modo a cumprir os requisitos acordados do projeto?

O presente trabalho fará uma análise das características, pontos positivos e negativos apresentados por ambas as metodologias o RUP – Rational Unified Process e SCRUM por meio de atividades pré-definidas tomando por base as fases de desenvolvimento de *software* comuns às metodologias. Após a análise comparativa das metodologias propostas, será feita uma análise textual discursiva (ATD) com o

objetivo de aferir a aplicabilidade das metodologias de desenvolvimento e situações DE USO.

1.1 FORMULAÇÃO DO PROBLEMA

Diante de uma sociedade que muda constantemente principalmente no quesito tecnológico, é recomendável analisar de modo consistente como o desenvolvimento de novas tecnologias está acontecendo na sociedade atual denominada cultura digital (RAQUEL, 2014).

O desenvolvimento de um novo software passa por diversos processos até se chegar ao produto final. Partindo deste pressuposto de que os que desejam construir um novo software, profissionais ou não, devem antes estabelecer o que deverá ser feito, de modo a utilizar da melhor forma possível os recursos disponíveis da organização.

A escolha sobre a melhor forma de organizar as tarefas envolvidas em todo o processo de desenvolvimento deve estar em harmonia com a cultura da empresa visando alcançar de modo satisfatório, tanto para equipe de desenvolvimento quanto para o cliente, as necessidades e expectativas do produto de software, ou seja, a qualidade. (PRESSMAN 2011).

Desse modo, Pressman (2011), diz que não há como chegar a um bom nível de qualidade de software sem, no entanto, efetuar a gestão de todo seu processo, assim, Sommerville (2011) define um modelo de processo como sendo uma visão geral acerca de todas as atividades e tarefas a serem desempenhadas para se alcançar o objetivo final, em que esse conjunto de tarefas trata-se da metodologia a ser aplicada.

Diante do exposto, além das metodologias faz-se necessário um estudo na íntegra para identificar qual possui uma melhor adaptabilidade com o projeto escolhido antes mesmo de se iniciar a fase de desenvolvimento, sendo uma pequena, média ou grande organização.

1.2 OBJETIVOS

1.3 OBJETIVO GERAL

Efetuar análise sobre Metodologias Ágeis e Metodologias Tradicionais de Desenvolvimento de Software representadas pelo RUP e pelo SCRUM, com o objetivo de aferir qual a metodologia mais adequada para cada tipo de desenvolvimento de software.

1.3.1 OBJETIVOS ESPECÍFICOS

Como subpartes do objetivo geral os objetivos específicos decorrentes foram os seguintes:

I. Identificar os principais tipos de metodologias tradicionais e ágeis, promovendo um estudo sobre suas características nativas, que posteriormente servirão para objeto de comparação;

II. Elencar as características de análise para efetuar a comparação entre ambas as metodologias;

III. Aplicar as características de análise ao RUP e ao SCRUM, a fim de levantar pontos positivos e negativos de ambas que poderão servir de base para profissionais da área sobre a forma de organização das atividades do desenvolvimento de software;

IV. Efetuar comparação entre as metodologias objetivando obter os pontos que divergem e convergem entre ambas que podem ser úteis ao apoio do profissional de T.I.;

V. Criar Análise Textual Discursiva dos resultados obtidos para consolidar de fato o estudo realizado de modo claro e objetivo.

1.4 JUSTIFICATIVA

Apesar das metodologias ágeis ainda se encontrarem em pleno desenvolvimento em sua linha de evolução, já vem apresentando resultados positivos e consistentes em sua utilização em projetos de pequeno porte, já que não há vastos registros de sua eficácia em projetos de maior porte, segundo Soares (2004), contudo, como alguns anos se passaram, outros autores podem ter produzidas obras com opiniões e relatos distintos, mas devido ao acesso não puderem ser consideradas.

Dessa forma surge a necessidade cada vez maior de que os profissionais da área estejam cada vez mais inteirados no que tange métodos de desenvolvimento de software já que a qualidade tem sido um dos principais objetivos há décadas. (PRESSMAN, 2011).

Como a sociedade muda consideravelmente rápido, faz-se necessário que o desenvolvimento de software esteja aberto a novas mudanças e alterações, a fim de suprir a necessidade dos usuários e ganhar vantagem competitiva (SOMMVERVILLE, 2011), já que a cultura empresarial existente apresenta como características principais o rápido desenvolvimento de inovações e elevado nível de concorrência. (PRIKLADNICKI, WILLI e MILANI, 2014).

Partindo do princípio que os requisitos[1] iniciais do projeto podem sofrer alterações no decorrer de seu desenvolvimento, o presente trabalho justifica-se pela ausência de trabalhos publicados comparando as metodologias tema do presente trabalho, já que para Dybå e Dingsøyr (2008), ainda não há um grande entendimento acerca dos métodos ágeis, e que, para Sommerville (2011), as metodologias pesadas tendem a estender consideravelmente o tempo de desenvolvimento de um software, surgindo por tanto, a necessidade de montar um referencial teórico, principalmente para recém-formados dos cursos de tecnologia e também para empresários da área de desenvolvimento de software, objetivando a clareza das características de ambas as metodologias.

[1] Segundo Sommerville (2011) “Requisitos são as descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento”.

A relevância desta etapa está contida no fato de que as alterações em dado momento do projeto diante da utilização de uma ou outra metodologia podem acarretar em um aumento massivo em relação aos recursos relativos sobre seu custo, como mostrado na Figura 1. (PRESSMAN, 2011).

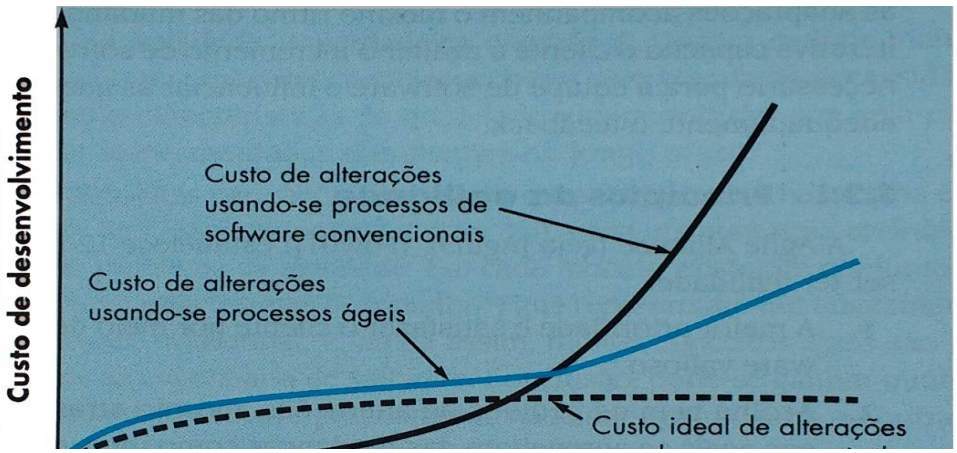

Figura 1. Custos de alterações como uma função do tempo em desenvolvimento. Fonte: (PRESSMAN, 2011).

Desse modo o presente trabalho almeja consolidar uma base teórica de esclarecimentos no que tange ao uso de métodos ágeis e tradicionais por iniciantes ou não, levando em consideração as mudanças e os custos.

1.5 ORGANIZAÇÃO DO TRABALHO

O restante do trabalho está organizado da seguinte forma. A Seção 2 apresenta os métodos pelos quais o trabalho será conduzido. A Seção 3 apresenta a fundamentação teórica, sendo a seção 3.1 responsável por tratar sobre os problemas de qualidade envolvendo software que remontam décadas, a seção 3.2 uma visão geral em relação ao desenvolvimento de software, a seção 3.3 os modelos de processos com atividades genéricas utilizados parametrizar o desenvolvimento de um software, a seção 3.4 concede embasamento teórico em relação aos métodos tradicionais de desenvolvimento de software e na seção 3.5 os métodos ágeis são fundamentados. A seção 4 mostra as pesquisas que foram conduzidas, sendo que na subseção 4.1 são detalhadas as metodologias tradicionais; na subseção 4.2, são detalhadas as metodologias ágeis; na subseção 4.3 são mostradas as características individuais de ambas; na subseção 4.4 são mostrados os resultados da comparação realizados por meio da análise textual discursiva. A Seção 5 apresenta as conclusões da pesquisa.

2. MÉTODOS

A presente pesquisa será desenvolvida tomando por base a pesquisa bibliográfica em livros de engenharia de software e pesquisa exploratória em artigos e periódicos nacionais e internacionais dos últimos anos sobre a temática, que segundo Santos e Candeloro (2006), revisão bibliográfica é parte de um projeto de pesquisa, que revela explicitamente o universo de contribuições científicas de autores sobre um tema específico.

Neste tipo de modalidade de pesquisa é importante desenvolver o texto tomando por base as publicações de autores que falam do tema a ser trabalhado, contudo, as colocações dos autores devem ser contextualizadas a fim produzir melhor efeito sobre o tema proposto.

Para Macedo (1994) ““revisão bibliográfica” ou “revisão da literatura” consiste numa espécie de “varredura” do que existe sobre um assunto e o conhecimento de autores que tratam desse assunto”.

Após apreciação das obras publicadas da área, com o objetivo de coletar dados para a pesquisa e embasar a aplicabilidade do RUP e do SCRUM possibilitando uma comparação atual entre as metodologias, os critérios para a comparação serão evidenciados tomando por base os principais autores da área como Sommerville, Pressman e Koscianski.

A ideia ligada por meio da análise comparativa consiste em comparar um ou mais elementos a fim de inferir algo. Este tipo de metodologia compara as igualdades entre as partes bem como suas divergências, independentemente de sua distância no espaço tempo/geográfico (MARCONI e LAKATOS, 2010).

Esta etapa utilizando a metodologia de comparação será realizada a fim de evidenciar os pontos positivos e negativos das metodologias levantadas objetivando contribuir com o conhecimento do profissional da área, possibilitando esclarecimentos para sua utilização ou não.

Caracterizada como um tipo de experimentação indireta, este tipo de metodologia pode ser empregue tanto em trabalhos de cunho qualitativos como quantitativos. (MARCONI e LAKATOS, 2010).

Apesar de alguns autores defenderem a posição de que o método comparativo apresenta características superficiais no quesito comparação entre um ou mais objeto de estudo, é provável e totalmente possível de que existam trabalhos a serem desenvolvidos que necessitaram de uma maior granularidade[2], porém, ainda assim este tipo de método ofertaria total suporte, ou seja, sua utilização pode ser bem-sucedida em diversos tipos de trabalhos. (PRODANOV e FREITAS, 2013).

No trabalho aqui preconizado, a análise comparativa será de cunho qualitativo. Na etapa de finalização do trabalho será desenvolvido uma análise textual discursiva sobre a temática mostrando as principais diferenças, bem como qual a melhor indicação dependendo do tipo de projeto.

A análise textual pode ser definida como um conjunto de artefatos que possuem e apresentam informações válidas e confiáveis em relação ao tema desejado. Este conjunto de artefatos denominado *corpus* pode ou não serem produzidos única e exclusivamente para o trabalho que está sendo produzido como outros objetivos. (MORAES, 2003).

A Figura 2 mostra a ideia geral ligada ao presente trabalho, por meio da evidenciação das características do RUP e do SCRUM.

[2] Granularidade, segundo Baltzan (2016), está ligado diretamente ao nível de aprofundamento que se deseja fazer uso de determinado grupo de dados. Desse modo quanto maior for a granularidade de certos dados maior será a riqueza de detalhes que se possui sobre eles.

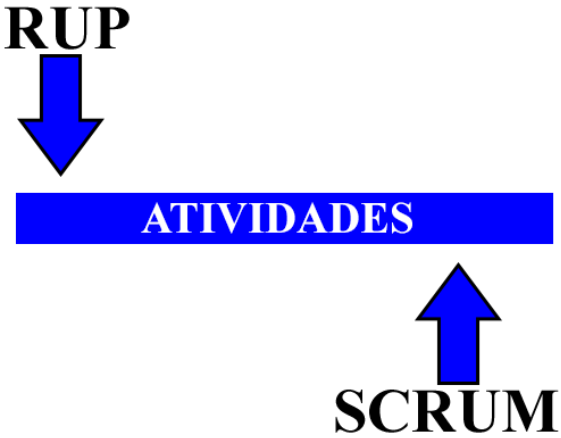

Figura 2. Comparativo das metodologias de desenvolvimento por meio das atividades propostas.
Fonte: (AUTOR, 2016).

3. FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta uma visão geral sobre revisão de literatura, apresentando os principais conceitos do tema.

3.1 CRISE DO SOFTWARE

Em meados da década de 70 surgiram muitos problemas acerca do desenvolvimento de software, pois, segundo Rodrigues (2008), nesta época a engenharia de software era praticamente inexistente, os códigos eram difíceis de manter pela inexistência de métodos eficientes para o seu desenvolvimento.

Com o decorrer da história surgiram os métodos denominados tracionais pela engenharia de software, que tentaram minimizar os erros e perdas decorridos durante processo de desenvolvimento de software, contudo, os resultados obtidos não foram os mais promissores e a desordem que permeava tal processo acabou se perdurando até os dias de hoje. No entanto, os esforços empregados a fim melhorar a qualidade do processo para então melhorar o produto em si ainda continua. (PRESSMAN, 2011).

A ideia por trás do tema ‘crise do software’ bem como seu uso remonta a um artigo publicado por Dijkstra em 1972 embasando-se na pouca experiência da engenharia de software como profissão de desenvolvimento de software e a grande oscilação de valores, cronograma, qualidade e prazo de projetos que estes desenvolvimentos apresentavam (REISSWITZ, 2009).

Dentre as possíveis causas para o surgimento da crise, Reisswitz (2009) enfatiza a imaturidade da engenharia de software e alta complexidade do desenvolvimento, resultando em custos elevados, não cumprimento dos prazos, qualidade baixa, funcionalidades inexistentes, manutenção de alta complexidade.

Uma das possíveis soluções é a utilização de um processo de desenvolvimento de software, levando em consideração que as atividades serão previamente planejadas aumentando com isso a qualidade (REISSWITZ, 2009).

3.2 DESENVOLVIMENTO DE SOFTWARE UMA VISÃO GERAL

São várias as definições do termo Software, para Sommerville (2011), utilizar a palavra software para designar programa de computador não está errado, contudo, não exprime o sentido completo que a engenharia de software considera. Para a engenharia de software, não se trata apenas de um programa de computador, mas todo artefato gerado em seu desenvolvimento e os dados de configurações necessários para que esse programa funcione corretamente.

Desse modo pode-se definir software como sendo um programa de computador bem como todos os artefatos gerados ou associados ao seu desenvolvimento (SOMMVERVILLE, 2011).

Laudon e Laudon (1999) define o termo Software como instruções detalhadas que controlam a operação de um sistema de computador.

Um software pode encaixar-se em dois grupos, sendo software de sistema responsáveis por promover a gestão de recursos utilizados no computador como o sistema operacional e o grupo de software aplicativos que são desenvolvidos visando a interação com o usuário e este com o computador (LAUDON e LAUDON, 1999).

Independente do objetivo final a que o software foi destinado, é conveniente encontrar em organizar cada uma das tarefas a serem desenvolvidas durante seu desenvolvimento, surgindo a necessidade de estabelecer processos sistémicos. (PRESSMAN, 2011).

Estas metodologias de processo, que abrigará um pequeno conjunto de atividades consideradas comuns a todos tipos de software, a fim de fornecer uma base procedimental consistente para o desenvolvimento de software, segundo Pressman (2011), são a base de todo o projeto, sendo esta, a forma pela qual o projeto alcançará seu objetivo final.

3.3 MODELOS DE PROCESSOS DE SOFTWARE

Modelos de processos de software podem ser definidos como passos previsíveis que auxiliarão no desenvolvimento do software, cabendo aos engenheiros de software bem como seus gerentes adequarem o processo definido à realidade do projeto a fim de obter um resultado de alta qualidade (PRESSMAN, 2011).

Para Sommerville (2011) processo de software é um conjunto de atividades relacionadas que levam à produção de um produto de software.

Partindo de uma visão mais específica, Reisswitz (2009) afirma que um processo de desenvolvimento de software caracteriza-se por um conjunto de atividades, parcialmente ordenadas, com a finalidade de obter um produto de software.

Tomando como base uma das visões em engenharia de software, Sommerville (2011) define quatro atividades genéricas a todas as metodologias de processos de software, sendo:

a) Especificação: promover a interação entre cliente e engenheiro de software a fim de evidenciar as funcionalidades do software;
b) Desenvolvimento: codificação do software;
c) Validação: validar a proposta do cliente com o desenvolvido;
d) Evolução: mudanças mercadológicas ou alterações de requisitos.

Partindo de uma segunda visão proposta por Pressman (2011), têm-se que cinco são as fases principais ao desenvolvimento de software, sendo:

a) Comunicação: interação com o cliente a fim de compreender o escopo do projeto a ser desenvolvimento;
b) Planejamento: estruturar a forma como o projeto transcorrerá;
c) Modelagem: mostrar o projeto como um todo de modo que todos os

envolvidos entendam onde se pretende chegar;

d) Construção: codificação do software;

e) Entrega: implantação do software e retorno do cliente quanto ao que foi desenvolvido.

A organização das atividades envolvidas no processo de software bem como o que cada atividade fará pode ser denominado como fluxo de processo, sendo:

a) Fluxo de processo linear: executa cada atividade metodológica de forma sequencial, como descrito na Figura 3, onde num primeiro momento acontece a comunicação com o cliente, o planejamento das atividades envolvidas, a descrição geral de todo o projeto, a sua construção e codificação e sua disponibilização ao usuário como última fase;

O fluxo linear possui características sequencial, em que, a fase seguinte apenas inicia-se com a finalização da anterior e entrega dos artefatos oriundos da mesma, de modo dar suporte as demais fases.

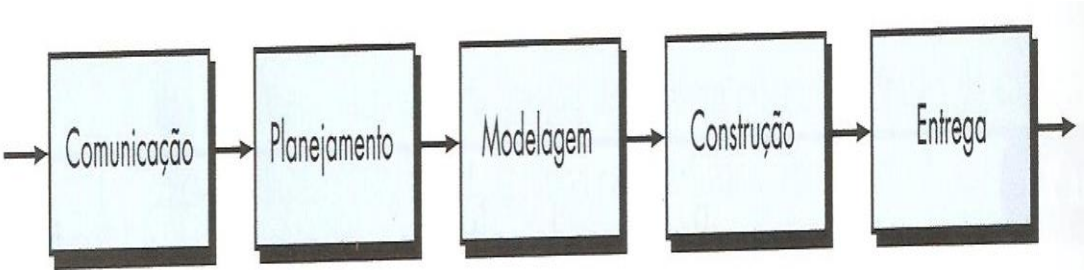

Figura 3. Modelo de processo linear.
Fonte: (PRESSMAN, 2011).

b) Fluxo de processo iterativo: executa as atividades de modo sequencial, contudo, pode haver a repetição de uma ou mais atividades metodológicas, como mostra a Figura 4, em que as fases são as mesmas, contudo, é possível retroceder fases caso os responsáveis da

fase seguinte determinem que os artefatos gerados não estão suficientemente maduros para prosseguir.

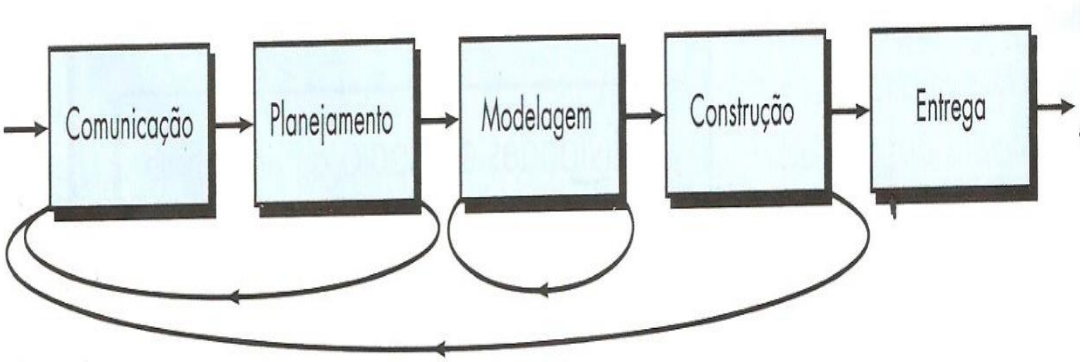

Figura 4. Modelo de processo iterativo.
Fonte: (PRESSMAN, 2011).

c) Fluxo evolucionário: executa as atividades de modo circular, ou seja, sempre após a última atividade ‘entrega’ o ciclo pode recomeçar, mostrado na Figura 5.

Este fluxo de processo apresenta a característica de realizar todo o processo objetivando a geração de incrementos e não o software como um produto final.

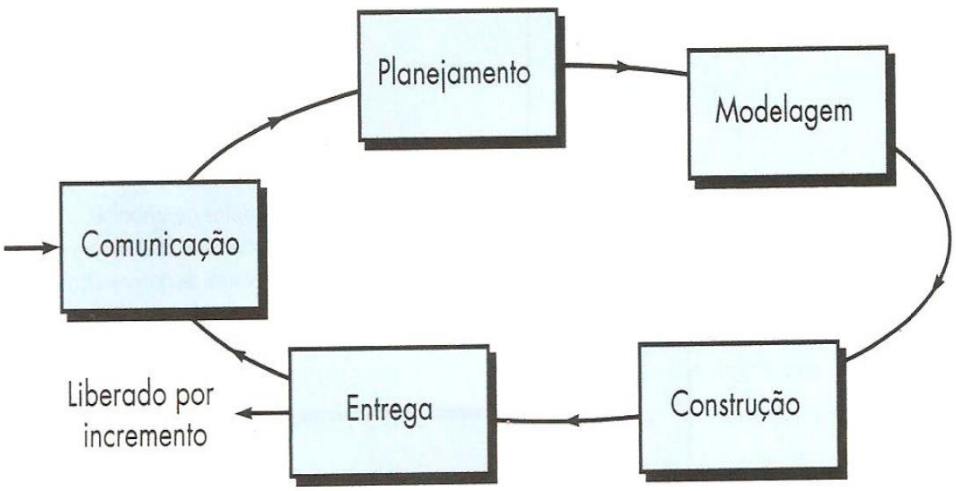

Figura 5. Modelo de processo evolucionário.
Fonte: (PRESSMAN, 2011).

d) Fluxo paralelo: pode executar uma ou mais tarefas em paralelo, demostrado na Figura 6. A característica marcante deste processo é o fato de que mais de uma atividade pode ser executada simultaneamente. (PRESSMAN, 2011).

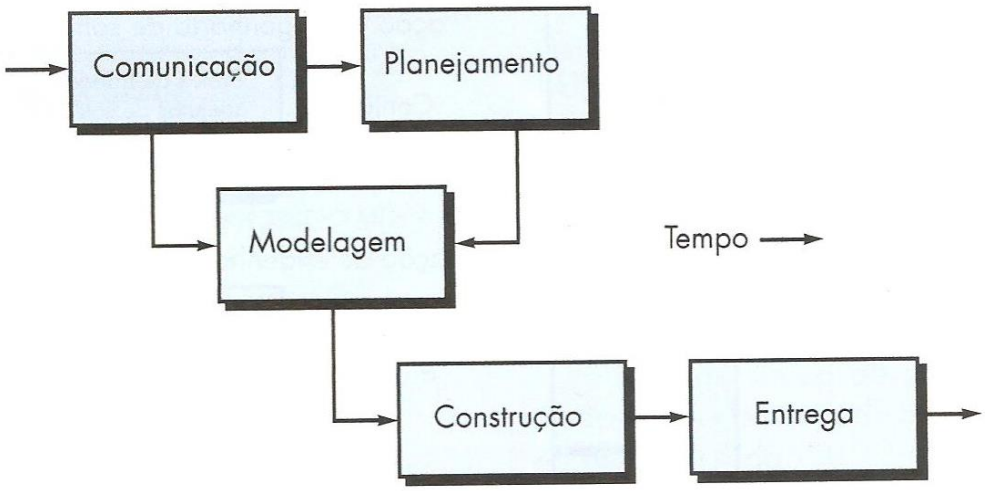

Figura 6. Modelo de processo paralelo.
Fonte: (PRESSMAN, 2011).

As atividades metodológicas fornecem um roteiro para todo o processo de criação do *software* e são sustentadas ao longo do projeto pelas atividades de apoio a fim de contribuir com a equipe em relação à gestão de todas as etapas do processo como qualidade, riscos, mudanças, entre outras. (PRESSMAN, 2011).

As atividades de apoio devem ser aplicadas ao longo de todo o processo de desenvolvimento a fim de complementar as atividades metodológicas facilitando o gerenciamento, o controle, a qualidade, as mudanças e o risco. São elas:

a) Controle e acompanhamento do projeto: permite que a equipe analise os acontecimentos reais e os compare com o planejamento a fim de atestar que o projeto está ocorrendo da forma como deveria;

b) Administração de riscos: avaliar os riscos que podem comprometer as etapas de desenvolvimento do projeto bem como sua qualidade

c) Garantia da qualidade de software: são as definições de métricas utilizadas em todo o desenvolvimento do software a fim de garantir um produto de qualidade na fase de entrega do produto.

d) Revisões técnicas: promove uma avaliação junto a todo artefato[3] gerado em determinada fase de desenvolvimento a fim de sanar quaisquer erros que possam prejudicar o desenvolvimento da fase seguinte.

e) Medição: responsável por fazer o balanceamento entre aquilo que se encontra em desenvolvimento (processo, projeto ou produto) para estabelecer um comparativo com os requisitos a fim de obter um *feedback*[4] de estabelecer se aquilo se dada a amostra existe ou não uma congruência com as necessidades do cliente.

f) Gerenciamento da configuração de software: possui a responsabilidade de promover a gestão de atualizações durante todo o processo de desenvolvimento do software.

g) Gerenciamento da reusabilidade: promoverá uma gestão consistente acerca daquilo que pode ou não ser utilizado como re-uso do desenvolvimento do software bem como a reutilização de partes que encontram-se fora do escopo do projeto atual.

h) Preparo e produção de artefatos de software: trata-se das atividades necessárias para geração de qualquer tipo de artefatono desenvolvimento do software. (PRESSMAN, 2011).

[3] Artefato: [..] modelos, documentos, logs, formulários e listas. (PRESSMAN, 2011).

[4] Segundo Pressman (2011), feedback são as “Recomendações derivadas da interpretação de métricas de produto transmitidas para a equipe de software”.

3.4 METODOLOGIA TRADICIONAL

Metodologias tradicionais de desenvolvimento de software utilizam-se do conceito de que as atividades envolvidas são executadas de forma sequencial e ordenadas e com passos bem estabelecidos. (PRIKLADNICKI, WILLI e MILANI, 2014). Estes métodos tendem a gerar incrementos que são artefatos de software (PRESSMAN, 2011), além de serem burocráticos e consideravelmente lentos, até mesmo pelo volume de artefatos gerados. (PRIKLADNICKI, WILLI e MILANI, 2014).

Os métodos tradicionais foram os responsáveis por formatarem o desenvolvimento de software de modo a ordenarem as atividades do processo a fim de roteirizar as atividades proporcionando uma melhor experiência para os envolvidos. (PRESSMAN, 2011).

Em outras palavras os métodos tradicionais possuem as mesmas atividades para construção do processo de desenvolvimento, contudo, a diferença entre eles está na diferença como as atividades são tratadas, caracterizando um fluxo de processo.

3.5 METODOLOGIA ÁGIL

Os métodos ágeis são caracterizados como métodos incrementais de software com entregas rápidas com no máximo três semanas de intervalo e grande participação dos clientes durante o processo de concepção do projeto. (SOMMERVILLE, 2011).

A ideia de desenvolvimento incremental surgiu com a *International Business Machines* (IBM) na década de 80 em que seu principal objetivo era lidar com as mudanças dos requisitos durante o desenvolvimento do software.

Para que um processo de software seja ágil antes é necessário que ele seja adaptável e que haja um processo de grande interação com o cliente a fim de obter feedback acerca do que está sendo desenvolvido, interação esta, permeada por protótipos. (PRESSMAN, 2011).

Apesar de apresentar muitos aspectos positivos, estes métodos apresentam a desvantagem de a equipe responsável ser descontinuada do projeto e as ideias e conceitos do projeto simplesmente ser levado outras vertentes, já que estas metodologias documentam realmente o que é necessário para promover o máximo de agilidade possível. (SOMMERVILLE, 2011).

Pressman (2011) cita doze princípios para que o projeto ou método apresente agilidade, sendo:

1. Surpreender o cliente com partes funcionais do software;
2. Receber de forma positiva as atualizações sugeridas pelos clientes e utilizar isso como meio de destaque frente a outras empresas;
3. Disponibilizar ao cliente partes funcionais do software em breves períodos de tempo;
4. Promover a união total da equipe que está envolvida no desenvolvimento;
5. Sempre motivar a equipe e promover um ambiente descontraído para a equipe;
6. Marcar reuniões com a equipe de modo geral a fim de discutir sobre assuntos do projeto e/ou transmissão de informações;
7. O crescimento se dará com a entrega de “pedaços de software” em funcionamento;
8. Todos os envolvidos devem manter um ritmo adequado ao andamento do projeto;
9. Promover o máximo de atenção possível para alcançar o máximo de aumento na agilidade do projeto;
10. Optar pelo caminho mais simples, a fim de reduzir a quantidade de trabalho;
11. Para alcançar os melhores resultados cada equipe deve ser comportar e organizar como cada processo será trabalho como;
12. Com cada equipe organizando-se procurando obter os mais altos resultados deverá estabelecer a melhor forma estabelecer pausas.

4. DESENVOLVIMENTO

Tomando por base a revisão da literatura em periódicos nacionais e internacionais da área de engenharia de software, chegou-se a um conjunto de atividades das quais servirão de base para análise individual das características das metodologias RUP e SCRUM.

Esse conjunto é delimitado pelas seguintes atividades:

I. Requisitos;
II. Modelagem de requisitos;
III. Planejamento;
IV. Desenvolvimento;
V. Testes;
VI. Entrega;
VII. Comunicação e colaboração;
VIII. Gerenciamento de mudanças;

Baseadas em autores como Koscianski e Soares, Sommerville, e Pressman, esse conjunto técnico de atividades compreende, de modo genérico, todo o desenvolvimento de um novo software desde a geração da ideia até sua entrega ao usuário final.

4.1 ATIVIDADES DE ANÁLISE

Nesta seção serão apresentadas as atividades que servirão de base para a comparação entre as metodologias.

4.1.1 REQUISITOS

Antes de dar início ao desenvolvimento de um novo software é importante ter em mente quais os requisitos que este novo sistema deverá apresentar, o que, para a engenharia de software trata-se de uma das partes mais complexas a serem desenvolvidas, já que muitas vezes o próprio cliente não sabe o que realmente deseja. (PRESSMAN, 2011).

Sommerville (2011) define requisitos de um novo sistema como sendo “as descrições do que o sistema deve fazer, os serviços oferecem e as restrições a seu funcionamento”.

A importância dessa atividade de especificação acerca das funcionalidades que o produto final deverá apresentar são encaradas como o principal causador de falhas durante o processo de desenvolvimento. (KOSCIANSKI e SOARES, 2007).

Desse modo, segundo Koscianski e Soares (2007), fazer a gestão dos requisitos é de suma importância, a fim de que o projeto apresente o máximo de qualidade ao seu término.

Como uma das atividades mais importantes ao desenvolvimento, segundo Pressman (2011), está atividade é dividida em sete tarefas a fim de garantir que o pedido do cliente será atendido em sua totalidade. São elas:

I. Concepção: o problema como um todo será analisado, bem como todos os envolvidos no processo e as respectivas interações;

II. Levantamento: nesta o objetivo central envolvendo o novo software deverá ser investigado junto ao usuário e de que forma este deverá resolver determinados problemas;

III. Elaboração: os levantamentos descritos nas fases anteriores passarão por um filtro e darão origem um modelo que servirá como guia descrevendo todos os aspectos que envolvem diretamente o software, como comportamento, interação, informações, entre outros;

IV. Negociação: os conflitos, caso existam, serão negociados junto às partes interessadas, já que quando o sistema possui interação com mais de um usuário, pode acarretar opiniões divergentes;

V. Especificação: todo o levantamento realizado nas fases tarefas anteriores dará origem a um documento ou algum tipo de representação, onde será possível entender de forma simplificada quais as características que deverão estar presentes no novo software.

VI. Validação: a consolidação de tudo quanto foi descrito na especificação será analisado, a fim de que os artefatos[5] já desenvolvidos possuem ou não a qualidade necessária para o projeto;

VII. Gestão de requisitos: nesta serão identificados os requisitos que podem apresentar alteração durante a evolução do projeto.

Para Sommerville (2011), a especificação de requisitos deve apresentar de forma clara e objetiva aquilo a que se propõe, de modo que qualquer pessoa envolvida no projeto consiga entender sobre as características que o software apresentará.

Objetivando a redução dos erros e falhas ligados ao levantamento de requisitos, autores como Koscianski (2007) e Sommerville (2011), apontam algumas atividades tidas como as mais comuns tratando-se da elicitação, como entrevista, etnografia, questionários, casos de uso[6] e cenários.

4.1.2 MODELO DE REQUISITOS

No mundo real do desenvolvimento de software, as partes geralmente não são bem treinadas ou não possuem informações relevantes, conhecimento e experiência no campo do projeto que irão atuar. Tais fatores poderão aumentar a probabilidade de que este projeto falhe. Se o desenvolvimento do sistema puder ser complementado com gráficos sistemáticos e Diagramas, facilitará a distinção em relação às funções características do sistema, evitando possíveis equívocos durante o projeto. (CHANG, 2010).

O princípio da modelagem propõe a ideia de mostrar o conjunto de atributos ligados ao software, objetivando seu melhor entendimento em diferentes níveis de descrição como mais abstrato (usuário) e mais técnico (desenvolvedores). (PRESSMAN, 2011).

---

[5] Artefatos são todos os componentes oriundos do desenvolvimento de um software, como documentação, diagramas, especificações, etc. (WAZLAWICK, 2013).

[6] Caso de uso, para Pressman (2011), “é uma narrativa textual ou modelo que descreve uma função ou recurso de um sistema do ponto de vista do usuário. Um caso de uso é escrito pelo usuário e serve como base para criação de um modelo de requisitos mais amplo”.

Nesta fase, para Sommerville (2011), “os engenheiros de software trabalham com clientes e usuários finais do sistema para obter informações sobre o domínio da aplicação, os serviços que o sistema deve oferecer, [..] e assim por diante”.

Ainda segundo Pressman (2011), este modelo é entendido pela engenharia de software como sendo o modelo mais técnico inicial ao desenvolvimento de um novo projeto, e para Hasegawa et. al (2009), esta fase é responsável pela geração de vários documentos ou modelos abstratos, denominados modelos de requisitos, dos quais servirão de roteiro durante todo o processo.

Compondo o modelo de requisitos, segundo Pressman (2011), têm-se os seguintes modelos:

I. Baseado em cenários;
II. Dados;
III. Orientados a classes;
IV. Orientados a fluxos;
V. Comportamentais.

4.1.3 PLANEJAMENTO

Durante o desenvolvimento de um novo software não há como saber com exatidão quais os possíveis defeitos[7] ou falhas[8] surgirão, já que cada projeto está sujeito ao seu ambiente do escopo, e justamente por esta condição é que se faz necessário um bom planejamento antes do início do desenvolvimento do projeto. (PRESSMAN, 2011).

Uma das atividades mais críticas no desenvolvimento de produtos de software é o processo decisório que atribui recursos a lançamentos futuros sob restrições técnicas, de recursos, de risco e de orçamento. Este processo centrado na decisão é referido como planejamento. (FRANCH e RUHE, 2016).

[7] Segundo Koscianski e Soares (2007), “defeito é uma imperfeição de um produto. O defeito faz parte do produto e, em geral, refere-se a algo está implementado no código de maneira incorreta”.

[8] Segundo Koscianski e Soares (2007), “falha é o resultado errado provocado por um defeito ou condição inesperada.

Para Pressman (2011), em muitos casos a falta de planejamento pode deixar a desejar no projeto, contudo, o excesso dessa atividade compreende igualmente um problema. Desse modo é importante e recomendável que seja feito uso da moderação de acordo com as características do projeto e das pessoas envolvidas.

De modo bem simplificado, Pressman (2011), afirma que esta atividade ou fase do desenvolvimento de software, compara-se a um mapa, onde todos os envolvidos no projeto faram uso, cada um em seu ramo de atuação, objetivando o sucesso do projeto.

4.1.4 IMPLEMENTAÇÃO

Objetivando o sucesso do projeto a qual se está em desenvolvimento é preciso que todas as atividades que fazem parte do desenvolvimento de software tenham sido realizadas corretamente, incluindo a parte de codificação propriamente dita, já que esta não pode ser entendida como uma parte para geração de código, mas sim, uma atividade continuativa e totalmente atrelada ao modelo de requisitos. (KOSCIANSKI e SOARES, 2007).

Para Pressman (2011), “a atividade de construção engloba um conjunto de tarefas de codificação e testes que conduzem ao software operacional pronto para ser entregue ao cliente e ao usuário final”.

Koscianski e Soares (2007) enfatizam que muitos dos erros decorrentes no ato da geração de código de um sistema, tem como causa principal a falta de comunicação e entendimento entre os responsáveis.

Dependendo do modelo de processo utilizado para desenvolvimento do projeto a codificação possuir mais ou menos interação com o cliente e também apresentar testes junto a codificação ou apenas ao final quando o software já possuir todas as suas características definidas. (PRESSMAN, 2011).

4.1.5 TESTES

Segundo IEEE Std 829 (1998), teste é tido como um conjunto de um ou mais casos de teste ou plano de teste.

Este prescreve o escopo, a abordagem, os recursos e o cronograma das atividades de teste. Identifica os itens, recursos a serem testados, as tarefas de teste a serem executadas, o pessoal responsável por cada tarefa e os riscos associados ao plano. (IEEE STD 829, 1998).

O objetivo do plano de teste é prescrever o escopo, a abordagem, os recursos e o cronograma das atividades de teste, identificar os itens sendo testados, os recursos a serem testados, as tarefas de teste a serem executadas, o pessoal responsável por cada tarefa e os riscos associados a este plano. (IEEE STD 829, 1998).

Esta atividade planejada de modo consistente e bastante sistemática poderá apontar falhas ainda por serem descobertas e contribuir consideravelmente para um projeto voltado a segurança e maior qualidade. (KOSCIANSKI e SOARES, 2007).

Koscianski e Soares (2007) apontam cinco tipos de distintos de testes de software como sendo os principais, sendo:

I. Caixa preta e Caixa Branca;
II. Estresse;
III. Integração;
IV. Orientado a objetos;
V. Aceitação

4.1.6 ENTREGA

Para Pressman (2011), esta atividade pode ser entendida como a finalização de todos os processos ligados ao desenvolvimento de um novo software e sua disponibilização ao cliente como um produto acabado e pronto para uso ou apenas um incremento a ser testado. Ainda nesta atividade o autor cita o feedback ligado o usuário em relação ao uso do mesmo.

Ainda tomando como base as colocações feitas por Pressman (2011), o autor deixa claro a importância da opinião dos usuários após a disponibilização do produto como um todo ou particionado, bem como o estabelecimento da comunicação entre organização desenvolvedora e seus utilizadores.

4.1.7 COMUNICAÇÃO E COLABORAÇÃO

Segundo Koscianski e Soares (2007), em um ambiente empresarial é bastante comum e fácil de ocorrer intrigas, desunião, falta de trabalho em equipe, entre diversos outros conceitos ligados comportamento entre pessoas, contudo, esta gestão deve ser tratada como uma das prioridades no projeto, já que este pode refletir positiva ou negativamente no quesito qualidade.

O referido autor salienta que um bom relacionamento entre a equipe pode facilitar consideravelmente no que tange ao entendimento do escopo ligado ao projeto a ser desenvolvido.

Dentre as causas mais preponderantes ao fracasso de projetos de software está a falta de comunicação entre a própria equipe e desta com o cliente, segundo Koscianski e Soares (2007), já que um cliente possui a habilidade de fazer bons ou maus argumentos acerca dos serviços prestados. (PRUITT e WILEY, 2012).

4.1.8 GERENCIAMENTO DE CONFIGURAÇÃO

O surgimento dessa disciplina de configuração remonta a década de 1950 quando a documentação e a produção de naves espaciais experimentaram dificuldades causadas por mudanças de engenharia inadequadamente documentadas. (ESTUBLIER, 2002).

Gerenciamento de configuração de software é a disciplina de controle de mudanças em sistemas grandes e complexos. Seu objetivo é prevenir o caos causado por numerosas correções, extensões e adaptações que são aplicadas a qualquer sistema grande durante sua vida útil. (ESTUBLIER, 2002).

Para Pressman (2011), a gerência de configuração é responsável pela gestão do seguimento de quaisquer mudanças relativas ao processo de desenvolvimento de um software.

Outro objetivo bastante consistente do gerenciamento de configuração é garantir uma sistemática e rastreável acerca do processo de desenvolvimento e manutenção, de modo que um sistema esteja em um estado bem definido com especificações claras e precisas e atributos de qualidade em todas as estapas do desenvolvimento. (ESTUBLIER, 2002).

4.2 CARACTERÍSTICAS DO SCRUM

Esta seção do desenvolvimento destina-se ao levantamento das características do framework SCRUM ligados ao processo de desenvolvimento de software.

Processos de desenvolvimento ágeis de software descrevem de modo iterativo o desenvolvimento incremental. Os requisitos iniciais para o produto acabado evoluem de estreita colaboração com o cliente e há um foco criando várias sub-entregas de software que o cliente pode experimentar antes mesmo que haja a entrega definitiva. (WOLFF, 2012).

As metodologias de desenvolvimento ágil de software como SCRUM são amplamente usados na indústria em diversos níveis, já que ofertam abordagens ágeis, práticas e flexíveis para a elicitação de requisitos, para priorização e organização do trabalho real de desenvolvimento, e para testes contínuos e de integração. (ELORANTA e KAI, 2012).

Para Eloranta e Kai (2012) essas metodologias consideram as mudanças provenientes do desenvolvimento do projeto, como alterações na regra de negócio, sendo parte do mesmo, ou seja, este tipo de metodologia adotam as mudanças e minimizam consideravelmente a ideia de seguimento de planos engessados.

Segundo Koscianski e Soares (2007), o SCRUM é uma metodologia de desenvolvimento de software que oferece parâmetros consistentes objetivando prover um ambiente em que possam existir mudanças relativas às regras de negócio do software a ser desenvolvido, bem como a flexibilidade com que isso ocorrerá.

O SCRUM, de acordo com Pressman (2011) e Sommerville (2011), é composto por cinco fases distintas que servem de base para o desenvolvimento de qualquer produto de software, sendo:

- Requisitos: pela especificação acerca das funcionalidades que o software deverá possuir;
- Análise: pelo levantamento sobre a viabilidade ou não baseado nos requisitos coletados;
- Projeto: pela geração e codificação propriamente dita do software;
- Evolução: pelas mudanças que o software deverá sofrer para continuar atendendo as necessidades do cliente;
- Entrega: pela disponibilização total ao cliente do que foi desenvolvido.

Estas fases seguidas pelo SCRUM, citadas por Pressman, são organizadas, de acordo com Koscianski e Soares (2007), em três fases:

- Pré-Planejamento;
- Desenvolvimento;
- Pós-Planejamento.

A fase do pré-planejamento acontece o levantamento acerca das funcionalidades desejadas pelo cliente (requisitos) e não possui características fortemente extensas, já que apenas alguns requisitos, os mais importantes, serão coletados em um primeiro momento. (KOSCIANSKI E SOARES, 2007), pois, para Blokehead (2017), os requisitos em muitos casos podem ser incompletos ou ausentes das especiações realizadas.

Uma das características que faz o SCRUM um método ágil com requisitos voláteis[9] em relação as mudanças, segundo Pressman (2011), é o fato de poder receber inserções de alterações do cliente em qualquer momento durante o desenvolvimento do software e atualizá-la dentro do registro *Backlog*[10].

Integrando esta primeira fase do SCRUM, segundo Koscianski e Soares (2007), também acontece todo o planejamento e levantamentos do projeto, a fim de garantir a não existência de riscos não computados e que o projeto seguirá com alterações já identificadas, ou seja, esta fase dará origem a um modelo de documento contento as principais informações acerca do desenvolvimento do software.

Após o término de toda a análise e do levantamento das principais funcionalidades do software, é o momento de dar início a segunda fase da metodologia SCRUM.

Junto a segunda fase observa-se as atividades de análise em relação a fase anterior, construção do projeto para visualização maximizada sobre o que será este software, desenvolvimento do código fonte do software e por fim os testes fechando a segunda fase. (KOSIANSKI e SOARES, 2007).

Esta fase que engloba geração de código e testes é denominada *Sprint*[11]. São nos sprints que cada requisito encontrado na fase anterior serão desenvolvidos. Normalmente estes possuem a duração média de trinta dias, podendo variar de acordo com a organização, tempo, projeto, etc. (PRESSMAN, 2011).

Em cada sprint o usuário tem acesso a uma parte executável do software a fim de obtenção de *feedback*[12] por parte dos *stakeholders*[13], objetivando garantir que o projeto está dentro das conformidades estabelecidas. (SOMMERVILLE, 2011).

---

[9] Requisito volátil significa que poderá sofrer alterações no decorrer do projeto. (VAZQUEZ e SIMÕES, 2016).

[10] Segundo Pressman (2011), Backlog é “uma lista com prioridades dos requisitos ou funcionalidades do projeto que fornecem valor comercial ao cliente”.

[11] Sprints, segundo Pressman (2011), “consistem de unidades de trabalho solicitadas para atingir um requisito no registro de trabalho (backlog)”.

[12] Feedback, para Pressman (2011), é entendido como uma opinião em forma de resposta do cliente após receber um executável do software.

[13] Stakeholders, de acordo com Sommerville (2011), é “Um stakeholder é uma pessoa ou papel que, de alguma maneira, é afetado pelo Sistema”.

Na última fase do desenvolvimento do projeto, têm-se os ajustes finais dos testes, alinhamento dos documentos propostos no decorrer do projeto e por fim a disponibilização ou implantação total do projeto ao cliente. (KOSIANSKI e SOARES, 2007).

De acordo com Koscianski e Soares (2007), o SCRUM, trata-se de uma metodologia voltada a projetos onde o usuário não consegue definir de modo claro o que deseja, é recomendável, de acordo com Pressman (2011), que a equipe destinada ao uso da metodologia SCRUM não ultrapasse a quantidade de dez integrantes, ao passo que a comunicação se torna a maior e melhor forma de construir um projeto de qualidade, pois, segundo Soares (2004), a comunicação com o cliente é indispensável.

Para Pressman (2011) a ideia de comunicação entre os membros da equipe não muda muito em relação a esta com o cliente, e ressalta a importância da comunicação entre os integrantes técnicos atrelados ao mesmo objetivo a fim de compartilhar experiências sobre cada processo individual.

4.3 CARACTERÍSTICAS DO RUP

Esta seção do desenvolvimento destina-se ao levantamento das características do método de desenvolvimento Rational Unified Process - RUP ligados ao processo de desenvolvimento de software.

O processo unificado ou Rational Unified Process, foi assim nomeado como forma de condecoração pela contribuição em diversos sentidos da organização *Rational Corporation* para com este processo de desenvolvimento de software. (PRESSMAN, 2011).

Para Sommerville (2011) os métodos usados na atualidade apresentam apenas uma visão unificada do processo de desenvolvimento, contudo, o RUP apresenta a característica de mostrar o processo sob diferentes aspectos como

dinâmico, estático e prático a fim de facilitar ainda mais a compreensão de todo o processo.

De acordo com as considerações elencadas por Pressman (2011), este modelo de processo de desenvolvimento de software possui características de diversos modelos, tanto tradicionais como ágeis, objetivando incorporar suas melhores qualidades.

O RUP em um nível mais geral é representado por quatro fases de iteração, sendo a de concepção, elaboração, construção e transição. Essas fases agrupam um conjunto de *workflows* ou atividades que nortearão ainda mais o projeto. SOMMERVILEE (2011).

Como sendo uma das atividades iniciais do RUP têm-se os requisitos. Esta possui o objetivo de descrever aquilo que o software deverá ou não fazer de modo a permitir que os desenvolvedores estejam cientes do que será desenvolvido e os clientes possam ser agentes integrantes em todo o processo. (SOFTWARE, 2001).

De acordo com Software (2001) esta atividade é responsável por criar um documento de visão e identificar as necessidades dos interessados. Os atores também são identificados, representando os usuários, e qualquer outro sistema que possa interagir com o sistema em desenvolvimento. Os casos de uso são identificados, representando o comportamento do sistema. Como estes são desenvolvidos de acordo com as necessidades do ator, o sistema é mais provável de ser relevante para os usuários.

Outra fase no modelo desenvolvimento é a de modelagem de negócios, capaz de auxiliar a entender e gerenciar as diferenças nas formas em que as organizações farão uso do sistema a fim de tornar mais simplificada a priorização de certas funcionalidades que o mesmo deverá apresentar. (KRUTCHEN, 2004).

Após todo o levantamento e modelagem das regas de negócios que o novo software deverá apresentar, a próxima etapa que entra em cena é o do planejamento, onde todas as informações técnicas relativas ao projeto deverão ser definidas, a fim de nortear todos os envolvidos. (PRESSMAN, 2011).

Para Krutchen (2004), o RUP trata-se de um modelo atual de criação e desenvolvimento de software, cujo projeto foi elaborado em conjunto com UML (Unified Modeling Language – linguagem de modelagem unificada), que, segundo Sommerville (2011), servirá para nortear o processo utilizando-se de diversos diagramas, como de sequência, classes, entre outros, objetivando propiciar uma visão geral acerca de todo o projeto.

A fase de planejamento serve principalmente para evidenciar e representar os componentes que serão desenvolvidos na etapa de implementação, bem como estes colaboram-se para executar uma ou mais funcionalidades oriundas do usuário. (SOFTWARE, 2001).

Após todo o processo de modelagem do projeto chega-se a fase de geração de código, que de acordo Sommerville (2011), os componentes evidenciados anteriormente serão de fato implementados e, segundo Pressman (2011), poderão ou não ser complementados a fim de atingir as expectativas do usuário.

Para Sommerville (2011) esta fase de construção propriamente dita do sistema é considerada relativamente rápida, já que a realização da modelagem anterior tende a facilitar e guiar os desenvolvedores no decorrer de suas atividades.

Junto à atividade de implementação são realizados os testes das funcionalidades do software (SOMMERVILLE, 2011), pois, à medida que os componentes vão sendo desenvolvidos os testes de unidade[14] servirão de base para garantir que cada componente ou funcionalidade a ser utilizado pelo usuário final de fato corresponde a sua especificação. (PRESSMAN, 2011).

O RUP propõe uma abordagem iterativa, o que significa que você prova ao longo do projeto. Isso permite que os defeitos sejam encontrados o mais cedo possível, o que reduz radicalmente o custo de corrigir falhas. (SOFTWARE, 2001).

Os testes são realizados as dimensões de qualidade, confiabilidade, funcionalidade, desempenho da aplicação e desempenho do sistema. Para cada uma dessas dimensões de qualidade, o processo descreve como se passará pelo ciclo de

[14] Segundo Pressman (2011) "*teste de unidade* focaliza o esforço de verificação na menor unidade de projeto do software — o componente ou módulo de software. Usando como guia a descrição de projeto no nível de Componente [..]".

vida do teste de planejamento, design, Implementação, execução e avaliação. (SOFTWARE, 2001).

Concluído os testes necessários da aplicação chega-se a fase de disponibilização da versão beta ou incremento do sistema a fim de garantir e validar que aquilo que o usuário final usará corresponde ao à suas expectativas e coletar suas opiniões. (PRESSMAN, 2011).

A fase de entrega e disponibilização ao usuário final, conta com diversas atividades, como instalação do software, planejamento e realização de testes beta, distribuição, assistência aos usuários, entre outras. (SOFTWARE, 2001).

Além dessas tarefas, Pressman (2011), salienta que "...a equipe de software elabora material com as informações de apoio ([..], manuais para o usuário, guias para a resolução de problemas, [..]) que são necessários para o lançamento da versão".

Ao que tange a comunicação entre os próprios integrantes técnicos da equipe e os usuários, leva-se em consideração as colocações feitas por Pressman (2011), onde o autor coloca que essa comunicação acontece de modo mais intenso na fase de concepção do RUP (atividades iniciais do projeto, como levantamento de requisitos, modelagem, etc.) e na fase de entrega, quando software é entregue ao usuário em forma de incremento ou produto acabado.

Esta ausência de comunicação, em certos aspectos, pode ser compensada pela quantidade de documentos e modelos gerados, incluindo principalmente a UML, durante as atividades iniciais a fim de assegurar e garantir aquilo que foi acertado junto aos stakeholders do sistema. (PRESSMAN, 2011);

Sommerville (2011) explica que a comunicação entre grandes grupos de desenvolvimento é mais difícil de acontecer, já que normalmente o que acontece é que cada integrante ou subgrupo de integrantes ficará responsável por determinada tarefa.

De acordo Sommerville (2011), o RUP contempla, por meio de atividades, as mudanças que um dado software pode apresentar. O referido autor diz que tal processo é capaz de "gerenciar as mudanças do software, usando um sistema de

gerenciamento de mudanças e procedimentos e ferramentas de gerenciamento de configuração".

Essas configurações são necessárias já que o software pode apresentar erros ou situações inusitadas durante o seu desenvolvimento ou mesmo durante o cenário de utilização pelos stakeholders. (SOMMERVILLE, 2011).

Complementada por Pressman (2011), onde autor diz que não importa o tipo de mudança, em um dado momento ou outro o mesmo terá que enfrentá-las.

4.4 ANÁLISE COMPARATIVA ENTRE RUP E SCRUM

Baseado no levantamento das atividades desenvolvidas por cada método de desenvolvimento de software, foi possível desenvolver a Tabela 1, tornando-se possível a análise em relação a forma como os métodos SCRUM e RUP, utilizam tais atividades no decorrer do processo de desenvolvimento de um novo software.

As características selecionadas para a realização da análise foram selecionadas de modo que pudessem mostrar o comportamento da metodologia no quesito documentação gerada no decorrer do processo, formalidades no que tange a documentos e o período em que uma dada atividade ocorre, sendo no início, meio ou fim, como mostra a Tabela 1.

Tabela 1 - Comparação entre SCRUM e RUP.

	SCRUM (Sprint)					RUP (Todo)				
ATIVIDADES	**DOCUMENTOS**	**FORMALIDADE**	**INÍCIO**	**MEIO**	**FIM**	**DOCUMENTOS**	**FORMALIDADE**	**INÍCIO**	**MEIO**	**FIM**
Requisitos	X	-	X	-	-	X	X	X	-	-
Modelagem de Requisitos	-	-	-	-	-	X	X	X	-	-
Planejamento	X	-	X	X	X	X	X	X	X	X
Implementação	-	-	-	X	-	X	X	X	X	X
Testes	X	-	-	X	X	X	X	-	-	X
Entrega	X	X	-	-	X	X	X	-	-	X
Comunicação e Colaboração	X	-	X	X	X	X	X	X	-	X
Gerenciamento de Configuração	X	X	-	-	X	X	X	-	-	X

Fonte: (AUTOR, 2017).

Neste caso, como observado na Tabela 1, a metodologia SCRUM é analisada dividida por sprints e não como um único processo sistémico. Diante disso todas as atividades são executadas várias vezes durante o mesmo projeto, apontando com isso, segundo Sommerville (2011), a principal característica do método, que é ser incremental.

Por meio desta comparação o SCRUM é visto pela vertente de seu desenvolvimento médio que é trinta dias, período este denominado sprint, onde os requisitos especificados na fase de levantamento são desenvolvidos afim de apresentar as funcionalidades ou cliente afim de poder efetuar possíveis alterações ou correções. (PRESSMAN, 2011).

Desse modo cada sprint pode ou não apresentar as atividades descritas na tabela, já que as metodologias apresentam uma especificação diferente a cada atividade.

Como observado na Tabela 1, a única atividade em que não faz parte do processo do SCRUM é a modelagem de requisitos, já que, segundo Pressman (2011), suas características envolvem o desenvolvimento considerável de documento. Já as demais atividades são em menor ou maior nível desempenhadas por ambas.

Baseado na descrição do processo utilizado pelo SCRUM e pelo RUP, a Tabela 2 exibe uma comparação com um nível de granularidade mais baixo, evidenciando de fato a diferença entre ambas de modo geral.

Tabela 2 - Análise específica entre RUP e SCRUM.

ATIVIDADES	SCRUM	RUP
Requisitos	X	X
Modelagem de Requisitos	-	X
Planejamento	X	X
Implementação	X	X
Testes	X	X
Entrega	X	X
Comunicação e Colaboração	X	X
Gerenciamento de Configuração	X	X

Fonte: (AUTOR, 2017).

Mesmo o SCRUM não apresentando documentações formais para a representação dos requisitos coletados dos usuários, ainda assim, segundo Ambler (2001-2012), o principal objetivo acerca dos requisitos é fornecer um entendimento sobre o que será desenvolvido e não a geração de documentos.

5. CONSIDERAÇÕES FINAIS

Com este trabalho, estima-se contribuir com outros profissionais da área de desenvolvimento de sistemas mediante aferição realizada nas metodologias propostas.

Independentemente do tipo de *software* que se pretende construir verifica-se a necessidade de estruturar logo no início do projeto quais serão as atividades a serem desempenhas durante todo o processo, bem como os responsáveis por cada uma.

Diante do exposto, o presente trabalho poderá servir de base na escolha da metodologia que melhor atende as necessidades do profissional que não possui vasto conhecimento sobre o assunto.

Cada tipo de *software* e organização possui características e exigências diversas acerca do processo de desenvolvimento como um todo. Assim também são as metodologias.

Partindo desse princípio cada caso carecerá de uma atenção diferente, seja na documentação, na prototipação, na comunicação com o cliente ou em qualquer outra área isso deverá estar previamente definido antes do início do projeto.

Objetivando um maior entendimento sobre as características individuais tanto do SCRUM como do RUP, foi-se analisado ambas as metodologias sobre atividades distintas a fim de evidenciar o comportamento individual.

O resultado obtido no presente refere-se ao fato de o SCRUM é um método geral de desenvolvimento de software como qualquer outro, incluindo os tradicionais, contudo, seu foco não está na documentação gerada, ao invés disso preocupa-se em disponibilizar incrementos ao usuário a fim de validar as funcionalidades especificadas anteriormente, mas não somente no início projeto.

Mesmo o RUP possuindo uma quantidade maior de documentação, sua implantação não pode ser vetada de modo algum, já que a pesquisa sobre tal metodologia aponta que esta característica para uma grande organização e visualização do projeto, enquanto que o SCRUM, por não possuir tanta

documentação, acaba por perder quando um ou mais membros deixam de integrar a equipe.

Por se tratar de um framework estruturado, o SCRUM, de acordo com Pressman (2011), não possui regras de como realizar as tarefas no desenvolvimento de um software, mas sim o que deve ser realizado.

Desse modo não há vínculo entre tal conjunto de passos e tipos de testes a serem realizados a fim de validar os incrementos desenvolvidos, ou os testes a serem desenvolvidos, já que eles entram no processo no momento em que se dá início às primeiras codificações das funcionalidades (requisitos) coletados por meio do usuário. (PRESSMAN, 2011).

Levando tal situação em consideração, o presente trabalho conclui que uma boa forma de testar os incrementos desenvolvidos em cada sprint é utilizando Test Driven Development - (TDD), que de acordo com Sommerville (2011), também é utilizado pelo Extreme Programming, já que as funcionalidades são antes testadas e depois implementadas, objetivando a extinção apenas no final de todo o processo.

Ainda segundo Sommerville (2011), é possível constatar que os requisitos dos usuários ficam mais fáceis de serem atualizados, já que os mesmos serão coletados conforme o projeto for evoluindo.

No que diz respeito a utilização do SCRUM em um projeto, pode-se dizer que as condições mais plausíveis são aquelas em que o não há certeza por parte do usuário quanto às funcionalidades que o software deverá apresentar, pois, existem grandes chances de a especificação inicial sofrer alterações conforme o projeto avançar.

Outro ponto consideravelmente importante levantado na pesquisa, é que no tange à equipe de desenvolvimento, onde esta metodologia (SCRUM) não é aconselhável quando a equipe a trabalhar no projeto for demasiadamente grande, já que não há documentações formais sobre o que se deverá realizar. É importante que a equipe seja reduzida para facilitar a troca de informação entre os membros e contribuir o máximo possível para o bom desenvolvimento do projeto.

Baseado naquilo que o profissional necessita e deseja alcançar ao final do projeto, o trabalho poderá servir como um guia teórico cujo principal objetivo é sanar quaisquer dúvidas oriundas sobre os métodos ágeis ou tradicionais no processo de desenvolvimento do *software*, bem como o melhor momento para aplicar um outro método e suas características individuais.

Assim, tomando por base as análises realizadas sobre as metodologias ágeis e tradicionais, constatou-se que não existem testes vinculados a metodologia SCRUM, que por sua vez apresenta testes já na segunda fase, que na verdade é o início da parte de codificação das funcionalidades acordadas junto ao usuário.

Como o TDD é voltado para projetos ágeis e o SCRUM é comumente um conjunto de atividades tidas como ágeis, torna-se interessante seu estudo mais aprofundado a fim de evidenciar potencialidades entre esta junção.

Sendo um dos principais diferenciais dos métodos ágeis o fato de puderem lidar com requisitos faltantes, ausentes ou mutantes, o desenvolvimento orientado a teste daria ainda mais um suporte ao framework, possibilitando maior agilidade no desenvolvimento e ajudando a evitar falhas.

BIBLIOGRAFIAS

AMBLER, S.W. **Agile Requirements Best Practices. Agile Modeling**. Disponível em: <http://www.agilemodeling.com/essays/agileRequirementsBestPractices.htm>. Acesso em: 23 jun. 2017.

BALTZAN, Paige. **Tecnologia Orientada para Gestão**. Porto Alegre: AMGH, 2016.

BLOKEHEAD, The. Scrum - **Guia de Práticas Ágeis Essenciais com Scrum**. Babelcube Inc, 2017.

CHANG, Chih-Hung; LU, Chih-Wei; CHU, William C.; SHIH, Chih-Hsiong; YANG Chao-Tung; HSIUNG, Pao-Ann; HSUEH, Nien-Lin; KOONG, Chorng-Shiuh. **SysML-based requirement modeling environment for multicore embedded system**. ACM Symposium on Applied Computing, (Mar. 2010), 2224-2228.

DYBÅ, T.; DINGSØYR, T. **Empirical studies of agile software development: A systematic review. Informatics**. Software Technology. 50, (Ago. 2008), 833-859.

ELORANTA, Veli-Pekka; KOSKIMIES, Kai. **Aligning Architecture Knowledge Management with Scrum**. WICSA/ECSA 2012 Companion Volume, (Aug, 2012), 112-115.

ESTUBLIER, Jacky; LEBLANG, David; CLEMM, Geoffrey; CONRADI, Reider; HOCK André van der; TICHY, Walter; WIBORG-WEBER, Darcy. **Impact of the Research Community for the Field of Software Configuration Management**. 24th International Conference on Software Engineering, (May. 2002), 643-644.

FRANCH, Xavier; Ruhe, GUENTHER. **Software release planning**. 38th International Conference on Software Engineering Companion, (May. 2016), 894-895.

HASEGAWA, Ryo. KITAMURA, Motohiro; KAIYA, Haruhiko; SAEKI, Motoshi. **Extracting conceptual graphs from japanese documents for software requirements modeling**. Sixth Asia-Pacific Conference on Conceptual Modeling, (Jan. 2009), 87-96.

IEEE, Institute of Electrical and Electronics Engineers. **IEEE Standard for Software Test Documentation.** New York, USA, 1998.

KOSCIANSKI, André; SOARES, Michel dos Santos. **Qualidade de software: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software**. 2. Ed. São Paulo: Novatec, 2007.

KRUTCHEN, Philippe. **The Rational Unified Process: An Introduction**. 3. ed. Boston, MA: Addison-Wesley, 2004.

LAUDON, Kenneth C.; LAUDON, Jane Price. **Gerenciamento de Sistemas de Informação**. 3. ed. Rio de Janeiro: LTC, 1999.

MACEDO, Neusa Dias de. **Iniciação à Pesquisa Bibliográfica**. 2. ed. São Paulo: Edições Loyola, 1994.

MORAES, R. **Uma Tempestade de Luz: A Compreensão Possibilitada Pela Análise Textual Discursiva**. Ciência & Educação: Bauru, SP, v. 9, n. 2, p. 191-210, 2003.

MARCONI, Marina de Andrade; LAKATOS, Maria Lakatos. **Fundamentos de Metodologia Científica**. São Paulo: Atlas, 2010.

PRESSMAN, Roger S. **Engenharia de Software: Uma Abordagem Profissional**. 7. ed. Porto Alegre: AMGH, 2011.

PRIKLADNICKI, Rafael; WILLI, Renato; MILANI, Fabiano. **Métodos Ágeis Para o Desenvolvimento de Software**. Porto Alegre: Bookman, 2014.

PRODANOV, Cleber Crstiano; FREITAS, Ernani Cesar de. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico**. 2. ed. Novo Hamburgo: Feevale, 2013.

PRUITT, Steve; WILEY, Anthony. **HP Relate – A customer communication system for the smb market**. 2012 ACM symposium on Document Engineering, (Sep. 2012).

RAQUEL, Alves Rodrigues. **A Era da Cibercultura: as mudanças na sociedade e a evolução da internet**. Rio de Janeiro: IETEC - Instituto de Educação Tecnológica, 2014.

REISSWITZ, Flavia. **Análise de Sistemas V.4: Processos de Desenvolvimento de Software**. [S.l.: s.n.]. 2009.

RODRIGUES. Edson Junior Lobo. **Curso de Engenharia de Software: Métodos e Processos para Garantir a Qualidade no Desenvolvimento de Software**. São Paulo: Digerati Books, 2008.

SANTOS, Vanice dos; CANDELORO, Rosana J. **Trabalhos Acadêmicos: Uma Orientação para a Pesquisa e Normas Técnicas**. Porto Alegre: AGO, 2006.

SOARES, M. d. S. **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software**. Infocomp Journal of Computer Science, [S.l.], v. 3, n. 2, (Nov, 2004), 8-13.

SOFTWARE, Rational. **Rational unified process: best practices for software development teams**. TP026B, Rev 11/01. 21p

SOMMERVILLE, Ian.**Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

VAZQUEZ, Caros Eduardo; SIMÕES, Guilherme Siqueira. **Engenharia de requisitos: software orientado ao negócio**. 1. ed. Rio de Janeiro: Brasport, 2016.

WOLFF, Sune. **Scrum Goes Formal: Agile Methods for Safety-Critical Systems**. IEEE. FormSERA 2012, Zurich, Switzerland.