FACULDADE DE TECNOLOGIA DE AMERICANA
Curso Superior de Bacharelado em Sistemas e Tecnologia da Informação

Matheus Alves Silveira

MELHORIA DE PROCESSOS E AGILIDADE DE *SOFTWARE* UTILIZANDO *SCRUM* E CMMI

Americana, SP
2013

FACULDADE DE TECNOLOGIA DE AMERICANA
Curso Superior de Bacharelado em Sistemas e Tecnologia da Informação

Matheus Alves Silveira

MELHORIA DE PROCESSOS E AGILIDADE DE *SOFTWARE* UTILIZANDO *SCRUM* E CMMI

Trabalho Monográfico, desenvolvido em cumprimento à exigência curricular do Curso Superior de Bacharelado em Sistemas e Tecnologia da Informação da Fatec-Americana, sob orientação do Prof. Me. Anderson Luiz Barbosa.

Área: Engenharia de *Software*.

Americana, SP
2013

FICHA CATALOGRÁFICA elaborada pela
BIBLIOTECA – FATEC Americana – CEETPS

S589m

Silveira, Matheus Alves

Melhoria de processos e agilidade de software utilizando Scrum e CMMI / Matheus Alves Silveira. – Americana: 2013.

70f.

Monografia (Graduação em Análise de Sistemas e Tecnologia da Informação). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.

Orientador: Prof. Me. Anderson Luiz Barbosa

1. Desenvolvimento de software 2. Scrum – software de projetos I. Barbosa, Anderson Luiz II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.3.05
681.3.077

Matheus Alves Silveira

MELHORIA DE PROCESSOS E AGILIDADE DE SOFTWARE UTILIZANDO SCRUM E CMMI

Trabalho de conclusão de curso apresentado à Faculdade de Tecnologia de Americana como parte dos requisitos para obtenção do título de Bacharelado em Sistemas e Tecnologia da Informação
Área de concentração: Engenharia de Software.

Americana, 06 de dezembro de 2013.

Banca Examinadora:

_____
Anderson Luiz Barbosa. (Presidente)
Mestre
FATEC - Americana

_____
Mariana Godoy Vazquez Miano (Membro)
Doutora
FATEC - Americana

_____
Nivaldo Tadeu Marcusso (Membro)
Especialista
FATEC - Americana

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter permitido que eu chegasse até aqui, por me dar forças para continuar e por ter se mostrado um verdadeiro amparo nas horas mais difíceis que enfrentei. Agradeço aos meus pais pelo apoio e principalmente a minha mãe Ivonete por fazer tudo o que esteve ao seu alcance para me ajudar. Agradeço a minha namorada Priscila por sempre ter me incentivado e apoiado, e pela sua paciência nos momentos em que precisei estar ausente. Agradeço a todos os meus familiares pela força que me deram.

A todos os amigos que conquistei na Fatec, em especial a Amanda, a Aline, o Willian e o Rafael. A todo o corpo docente pelo empenho que tiveram em mostrar o caminho para meu aprendizado. Em especial ao Anderson que foi muito mais que um orientador para mim, que sempre me incentivou e apoiou na minha pesquisa. A minha coordenadora Maria Cristina pela sua disposição e a todos os funcionários da Fatec Americana.

DEDICATÓRIA

Dedico este trabalho a Deus, aos meus pais, pelo apoio, a minha namorada pela paciência e compreensão, aos meus amigos que aqui fiz e que eternamente estarão em minhas lembranças.

“Bem-aventurado o homem que acha sabedoria, e o homem que adquire conhecimento; Porque é melhor a sua mercadoria do que artigos de prata, e maior o seu lucro que o ouro mais fino. Mais preciosa é do que os rubis, e tudo o que mais possas desejar não se pode comparar a ela”.
(Provérbios 3:13-15)

RESUMO

Esta monografia apresenta um breve estudo sobre como aumentar a qualidade, diminuir prazos e otimizar recursos em um processo de desenvolvimento de *software, a* partir de dois temas principais: a utilização de metodologias ágeis e a implementação de melhorias no referido processo. Serão apresentados os principais modelos de processos de *software* utilizados pela Engenharia de *Software* e as características de um processo ágil com o *Scrum*, metodologia ágil de desenvolvimento de *software*. Este trabalho destaca, também, alguns aspectos de planejamento de mudanças de processos e apresenta o CMMI - *Capability Maturity Model Integration* - como um modelo de melhoria de processos de *software*. A pesquisa de campo realizada compara o tempo gasto com a documentação durante o processo e com a manutenção corretiva após o desenvolvimento do *software*, em empresas que utilizam conjuntamente o Scrum e o CMMI, em relação a outras que recorrem a apenas um desses *frameworks* ou nenhum deles. As considerações finais destacam a análise da viabilidade em utilizar o Scrum e o CMMI.

Palavras Chave: Processo de Software; CMMI; *Scrum*

ABSTRACT

This monograph presents a brief study about how to increase quality, reduce time and optimize resources in a software development process, from the two major themes: the use of agile methodologies and the implementation of improvements in that process. This paper presents the main models for software process used by Software Engineering and the characteristics of an agile process with the Scrum, agile methodology on software development. This paper also highlights some aspects about planning and process changes and presents the CMMI - Capability Maturity Model Integration - as a model software process improvement. The field research compares the time spent on documentation during the development process and on corrective maintenance after the software development, in companies using Scrum and CMMI together in relation to others that rely on only one or neither of these frameworks. The final considerations highlight the analysis of the feasibility of using Scrum with CMMI together.

Keywords: *Software Process, CMMI, Scrum*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

Depois que os meios de comunicação revolucionaram em todos os aspectos a nossa vida cotidiana, houve uma evolução nos meios de produção, essa nova revolução é chamada por Sennett (2009, p.9) de “Capitalismo flexível”, onde “... Pede-se aos trabalhadores que sejam ágeis, estejam abertos a mudanças a curto prazo, assumam riscos continuamente, dependam cada vez menos de leis e procedimentos formais.”.

Por causa desse capitalismo flexível surgiu, por parte das empresas que contratam algum tipo de *software* para auxiliar no seu funcionamento, uma cobrança maior em relação ao tempo de entrega do *software*, adequação maior as especificações e conformidades, resposta rápida a alterações requisitos e também uma redução no custo de desenvolvimento de *software*, fazendo com que o processo de desenvolvimento de *software* se modifique para atender essas novas exigências do mercado.

Em relação a este novo cenário de desenvolvimento Sommerville (2011) nos relata que construir um *software* com qualidade demanda tempo e com isso as empresas de desenvolvimento de *software* precisam cada vez mais se empenhar para conseguir solucionar o desafio da entrega que consiste em desenvolver um *software* com máxima qualidade em um tempo muito curto.

A qualidade e a manutenção de um sistema se tornam cada vez mais difíceis, devido a essas mudanças repentinas de requisitos que ocorrem no processo de desenvolvimento de *software* e a cobrança para que o *software* seja entregue no menor tempo possível. Por isso, muitas das vezes, o desenvolvimento é realizado sem o planejamento necessário (KOSCIANSKI e SOARES, 2007; SOMMERVILLE, 2011).

Para ajudar a resolver o desafio da entrega e atender a mudanças de requisitos surgiram as metodologias de desenvolvimento ágil, cujo foco está nas pessoas e é indicado para ambientes em que surgem mudanças repentinas (BECK *et al*., 2001; ZANATTA, 2005; KOSCIANSKI e SOARES, 2007).

Uma metodologia ágil, de acordo com Koscianski e Soares (2007), possui o principal objetivo de entregar o *software* em funcionamento, priorizando o necessário e abrindo mão do que não tem utilidade para o processo de desenvolvimento.

Para Zanatta (2005) é possível um planejamento onde se possa conciliar uma metodologia ágil de *software*, que é uma tendência cada vez mais forte para as empresas de desenvolvimento de *software*, com modelos de melhoria de processos, pois ambos possuem planos para o desenvolvimento de *software* e buscam o melhor para que a organização aumente sua capacidade de produção de *software* com qualidade.

Para tanto o estudo se **justificou** pela necessidade das empresas criadoras de *software* conciliarem o desenvolvimento ágil dessas novas metodologias com a maturidade e capacitação contínua de um modelo de melhoria de processos e conseguir gerar melhores resultados, otimizar seus processos e garantir a qualidade do *software* reduzindo seu tempo de entrega.

O **objetivo geral** consistiu em conhecer o que é um modelo de melhoria de processos no desenvolvimento de *software*, visando conciliá-lo a uma metodologia ágil a fim de mostrar que é possível otimizar o processo desenvolvimento de *software*, garantindo e aumentando sua qualidade sem abrir mão do tempo rápido de entrega.

Os **objetivos específicos** foram:

Conhecer os principais modelos de processo de *software* utilizados na Engenharia de *Software*, visando encontrar qual o modelo de processo que se adequa melhor ao contexto atual do ambiente de desenvolvimento de *software*.

Compreender o que é uma metodologia de desenvolvimento ágil, buscando entender suas características, suas vantagens e ganho de desempenho no processo de desenvolvimento de *software,* utilizando a metodologia *Scrum* como referência.

Conhecer o que é um modelo de melhoria de processos de *software*, demonstrar quais os mais utilizados, focando especificadamente no CMMI (*Capability Maturity Model – Integration).*

Relacionar alguns pontos em comum encontrados no CMMI e no *Scrum*, visando demonstrar que ambos podem ser conciliados e implementados no processo de desenvolvimento de *software* possibilitando melhorar e garantir a qualidade de *software* com maior otimização de tempo possível auxiliando na resolução da problemática que as empresas de desenvolvimento de *software* enfrentam atualmente que é de conseguir entregar um *software* com máxima excelência de qualidade em um curto prazo de tempo.

Como **metodologia** para o desenvolvimento deste trabalho, foi utilizada a metodologia de pesquisa bibliográfica utilizando livros e artigos científicos para

confecção do contexto teórico e para o estudo de caso foi utilizada a pesquisa exploratória, que é entendida por SILVA (2001, p.21) como sendo uma pesquisa que:

> [...] visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses. Envolve levantamento bibliográfico; entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado; análise de exemplos que estimulem a compreensão (...).

Após o levantamento bibliográfico foi feita uma pesquisa de campo realizada através de um formulário onde foram levantadas algumas informações concernentes ao processo de desenvolvimento de *software* de algumas organizações.

O trabalho foi estruturado em cinco capítulos, sendo que o **primeiro** é destinado a introdução, o **segundo** expõe o ambiente de desenvolvimento de *software*. O processo de desenvolvimento de *software* e suas etapas fundamentais e os modelos de desenvolvimento de *software*. Depois explica o que é uma metodologia ágil e o que a caracteriza. E demonstra o *Scrum* através de seus papéis, cerimônias e artefatos.

O **terceiro** capítulo apresenta o que é melhoria de processos de *software* e enfatiza o modelo de capacitação e maturidade CMMI abordando as duas representações disponíveis, suas principais áreas de processo e suas metas práticas genéricas.

O **quarto** capítulo relaciona alguns pontos em comum desses dois *frameworks*[1]. Demonstra os resultados de uma pesquisa desenvolvida para comparar as empresas que utilizam um modelo de melhoria de processos atrelado a uma metodologia ágil em relação às outras empresas de desenvolvimento de *software*. Por fim o **quinto** capítulo expõe as considerações finais em relação ao tema e oferece algumas sugestões para possíveis futuros trabalhos.

[1] Um conjunto de conceitos usado para resolver um problema de um domínio específico.

2. PROCESSO DE *SOFTWARE*

De acordo com Sommerville (2011) as empresas atuais operam em um ambiente global, com mudanças rápidas, onde precisam responder a essas mudanças e obter novas oportunidades e novos mercados através dessas transformações repentinas. Os *softwares* estão presentes em quase todas as operações de negócios dessas empresas e juntamente com a modificação rápida de suas operações vem também a alteração desses *softwares* utilizados para que possam atender as novas especificações.

Ao longo do tempo, diversas metodologias foram criadas para sistematizar o processo de desenvolvimento de *software*. Essas metodologias podem ser divididas em duas linhas. As tradicionais que, segundo Koscianski e Soares (2007), utilizam documentação em cada etapa do processo de desenvolvimento de *software*. E as ágeis que segundo Sommerville (2011) se baseiam em uma abordagem incremental para a especificação, o desenvolvimento e a entrega do *software*.

Vale ressaltar que para alguns tipos de *softwares* que são considerados como sistemas críticos em que é necessária uma análise completa do sistema e suas especificações são quase que constantes a metodologia tradicional se faz a melhor opção. Porém, no ambiente atual de negócio que se caracteriza por mudanças rápidas, essa análise completa poderia causar problemas, pois o *software* poderia correr o risco de já estar desatualizado ou não cobrir as novas expectativas (SOMMERVILLE, 2011).

Muitas organizações não utilizam metodologia alguma para desenvolver *softwares*. Koscianski e Soares (2007) afirmam que geralmente são pequenas e médias organizações que não possuem recursos suficientes para adotar o uso de uma metodologia tradicional ou porque os processos tradicionais não são adequados às suas realidades. Os autores ainda afirmam que essa falta de sistematização na produção do *software* resulta em baixa qualidade do produto final e dificuldade na entrega do *software*, não cumprimento com os prazos e custos predefinidos e ainda inviabilidade de uma futura evolução.

2.1. Modelos de Processos de *Software*

Um processo de *software* é um conjunto de atividades correlatas que se auxiliam na produção de um produto de *software* (KOSCIANSKI e SOARES, 2007; SOMMERVILLE, 2011). Todo processo de *software* deve incluir, segundo Sommerville (2011), quatro atividades fundamentais para a Engenharia de *Software*:

Especificação: definição das funcionalidades e das restrições;

Projeto e Implementação: produção do *software* de acordo com as especificações;

Verificação e Validação: revisão e testes para garantir que as especificações foram desenvolvidas;

Evolução: manutenção e adaptação às novas necessidades do cliente.

Ainda de acordo com o autor, não existe um processo ideal e que a maioria das organizações customiza seus próprios processos de desenvolvimento de *software* para tirarem o melhor proveito das capacidades das pessoas em uma organização e também as características específicas do sistema em desenvolvimento.

Um modelo de processo de *software* é uma generalização simplificada de um processo de *software*. Esse modelo é uma abstração que pode ser utilizado em diferentes abordagens de desenvolvimento de *software*.

O primeiro modelo de processo que foi concebido foi o modelo cascata. Além do modelo cascata os principais modelos utilizados na Engenharia de *Software* são: prototipação, modelo espiral, modelo iterativo e o modelo incremental.

2.1.1. Modelo Cascata

Também conhecido como ***Waterfall*** ou ***Top-Down*** o modelo cascata surgiu de uma visão industrial de construção de produtos (MAGELA, 2006). Esse modelo aborda as atividades fundamentais do processo de *software* como etapas distintas e sequenciais, onde após cada término de uma etapa é gerada uma documentação que precisa ser aprovada para que a próxima etapa se inicie (KOSCIANSKI e SOARES, 2007; SOMMERVILLE, 2011).

O modelo cascata surgiu em uma época em que o custo para fazer alterações e correções era muito alto devido ao acesso a um computador ser muito limitado e não existir ferramentas de apoio ao desenvolvimento do *software*. Por isso o *software*

era todo especificado, planejado e documentado antes de ser propriamente desenvolvido (KOSCIANSKI e SOARES, 2007).

Segundo Sommerville (2011) as principais etapas do modelo em cascata são relacionadas diretamente com as atividades fundamentais do processo de desenvolvimento de *software* onde:

Definição de requisitos: Etapa onde é realizado todo o levantamento dos requisitos do sistema. Todas as funcionalidades e restrições do sistema são definidas e detalhadas;

Projeto de sistema e *software*: Etapa onde é definida a arquitetura geral do sistema. Segundo Pressman (2011) essa fase é um "processo de múltiplos passos que concentra em quatro atributos distintos do programa: estrutura de dados, arquitetura de *software*, detalhes procedimentais e caracterização de interface";

Implementação e teste unitário: Etapa onde o projeto de *software* é desenvolvido e codificado em programas ou unidades de programa e depois é submetido ao teste unitário que é a verificação de cada unidade que foi desenvolvida de acordo com sua especificação;

Integração e teste de sistema: Etapa onde todas as unidades individuais do sistema são integradas e testadas como um sistema completo, afim de assegurar que as especificações foram desenvolvidas corretamente. Após essa etapa o *software* é entregue ao cliente;

Operação e Manutenção: Após a instalação e operação do *software* no cliente alguns erros que não foram identificados durante a confecção do *software* podem ser identificados e alguns serviços podem necessitar de alguma ampliação mediante a necessidade de alteração de requisitos. Normalmente essa etapa é a mais longa do ciclo de vida do *software.*

Cada etapa não pode começar sem que a anterior esteja totalmente concluída. Segundo Sommerville (2011), na prática essas etapas se sobrepõem e colaboram mutuamente para produzir mais informações para o desenvolvimento do *software* fazendo com que muitas das vezes os documentos produzidos em cada etapa necessitem de modificações para refletirem as alterações feitas.

De acordo com Koscianski e Soares (2007), o custo das alterações do *software* se eleva a medida em que o desenvolvimento progride e esse custo pode chegar a ser cem vezes maior do que se fosse realizado ainda na fase de levantamento de requisitos. Sommerville (2011) ainda ressalta que a solução de problemas pode ser

adiada ou até mesmo ignorada podendo significar que o sistema desenvolvido não atenderá as especificações corretas do usuário.

Na etapa de Operação e Manutenção o *software* é colocado em uso e erros e problemas de especificações são descobertos e novas funcionalidades são identificadas obrigando o *software* de ser evoluído para que se mantenha útil (SOMMERVILLE, 2011).

Vale ressaltar que, segundo Magela (2006), esse modelo já foi concebido com críticas. Dificilmente todas as especificações, ou pelo menos as especificações essenciais que o *software* precisa possuir não são levantadas na etapa de Análise de Requisitos. O autor ainda informa que as maiores contradições e inconsistências são encontradas na programação, e conforme já abordado, o custo para que sejam feitas as alterações necessárias torna-se muito alto.

O cenário dinâmico em que os *softwares* atuais encontram não fornece muita vantagem para se utilizar o modelo cascata. O *software* é diferente de um produto qualquer e está em constante evolução e adaptação fazendo com que torne inviável a utilização dessa abordagem neste cenário.

Em outros cenários, onde os requisitos são mais estáveis e bem compreendidos esse modelo se torna muito bem utilizável (KOSCIANSKI e SOARES, 2007; SOMMERVILLE, 2011).

Após o surgimento do modelo cascata e a identificação de suas deficiências surgiram alguns outros modelos de desenvolvimento de *software* que possuíram o propósito de tentar aperfeiçoar ainda mais o processo de desenvolvimento de *software* e suprir essas deficiências encontradas no modelo cascata.

2.1.2. Prototipação

Quando um cliente define uma lista de objetivos gerais para o *software*, mas não se identifica, de forma detalhada, os requisitos. Quando um desenvolvedor encontra-se inseguro em relação à eficiência do que será desenvolvido. Ou quando a interface de usuário precisa ser abordada em caráter de extrema necessidade. O paradigma de prototipação pode ser considerado a melhor escolha de abordagem (PRESSMAN, 2011).

Uma versão inicial do *software* é feita com o objetivo de demonstrar conceitos, e visualizar melhor o que será desenvolvido. Essa primeira versão é um protótipo,

geralmente nenhuma funcionalidade complexa é implementada, mas mesmo assim gera uma visualização melhor para a compreensão do problema a ser resolvido (SOMMERVILLE, 2011).

O modelo baseado em prototipação possui muitas vantagens em relação ao modelo cascata, porque entende a necessidade de diminuir os riscos e dúvidas presentes na construção do *software* (MAGELA, 2006).

Idealmente o protótipo serve como um mecanismo para identificação e validação dos requisitos do *software* (PRESSMAN, 2011; SOMMERVILLE, 2011). Nesse modo a responsabilidade em definir o que será construído é compartilhada com o usuário (MAGELA, 2006).

A prototipação também é ideal para auxiliar na criação da interface do usuário. Devido à dinamicidade das interfaces, diagramas e descrições textuais não são tão eficientes quanto à prototipação para abstrair os requisitos necessários para a criação de uma interface de usuário (SOMMERVILLE, 2011).

De acordo com Pressman(2011), embora a prototipação possa ser utilizada como um modelo de processo isolado é muito comum encontrar a prototipação sendo implementada no contexto de qualquer outro modelo de processo de *software*.

Segundo Sommerville (2011) um problema geral existente na prototipação é que o protótipo pode não ser necessariamente utilizado no sistema final por impossibilidade de adaptação ao sistema devido a especificações que provavelmente foram ignoradas durante a criação do protótipo.

Outro problema que pode ser encontrado é devido a mudanças rápidas durante o desenvolvimento fazerem com que o protótipo não esteja devidamente documentado e para a manutenção a longo prazo isso pode gerar muitos problemas. Além disso mudanças durante o desenvolvimento do protótipo podem degradar a estrutura do sistema tornando difícil e custosa a sua manutenção e evolução (SOMMERVILLE, 2011).

2.1.3. Modelo Incremental

O modelo incremental foi desenvolvido na década de 1980 pela IBM (SOMMERVILLE, 2011) e foi baseado na ideia de desenvolver uma versão inicial do *software* com um prévio levantamento de requisitos (MAGELA, 2006; SOMMERVILLE, 2011), onde as funcionalidades mais essências do *software* são

desenvolvidas e apresentadas ao usuário. A etapas de especificação, desenvolvimento e testes são intercaladas com um breve *feedback* entre todas as etapas (SOMMERVILLE, 2011).

Ao utilizar o modelo incremental, o primeiro incremento é um produto essencial, onde os requisitos básicos são atendidos. Esse incremento inicial é utilizado pelo cliente e como resultado do uso é desenvolvido um planejamento para o incremento seguinte. Esse processo é repetido até que seja produzido o *software* completamente. O foco do modelo incremental é voltado à entrega de um produto operacional a cada incremento. Suas primeiras versões são partes do produto final, porém são capazes de atender ao usuário e oferecer uma plataforma para a avaliação do usuário (PRESSMAN, 2011).

Esse modelo possuiu um ganho enorme em relação aos modelos anteriores, principalmente na eliminação dos riscos do projeto, na rápida apresentação de resultados concretos e eliminação de erros encontrados (MAGELA, 2006).

O desenvolvimento incremental é o modelo que mais contribuiu para o surgimento das metodologias ágeis e é mais recomendável para a maioria dos sistemas de negócios do que o modelo cascata. Além disso o desenvolvimento do *software* de forma incremental é mais fácil e seu custo é menor para realizar modificações no *software* durante o desenvolvimento (SOMMERVILLE, 2011).

Segundo Pressman (2011) esse modelo é muito útil em casos em que não há recursos humanos disponíveis para uma completa implementação na época de vencimento do prazo estabelecido para entrega do projeto.

De acordo com Sommerville (2011), o modelo mais comum atualmente utilizado para desenvolvimento de sistema aplicativos é o modelo incremental. Contudo, como todos os modelos anteriores, esse modelo apresenta dois principais problemas:

Se os sistemas são desenvolvidos com rapidez a produção da documentação em cada versão do *software* se torna inviável;

A arquitetura do *software* pode se degradar a medida em que novos incrementos são inseridos. A menos que seja investido em refatoração para melhorias, as mudanças tendem a corromper a arquitetura do *software* e tornar sua manutenção muito cara e difícil de ser realizada.

2.1.4. Modelo Espiral

O modelo espiral foi proposto por Boehm, em 1980 (PRESSMAN, 2011; SOMMERVILLE, 2011) e foi baseado na junção de algumas características do modelo cascata com o modelo de prototipação, acrescentando um novo elemento denominado **Análise de Riscos** (PRESSMAN, 2011).

O processo de *software* é representado por uma espiral onde cada volta da espiral representa uma fase do processo de *software*. O modelo espiral mistura prevenção e tolerância a mudanças e define que as mudanças objetivam a diminuição dos riscos do projeto de *software* (SOMMERVILLE, 2011).

Em cada novo ciclo, uma nova atividade é incorporada ao desenvolvimento. Porém, antes da execução é realizada uma nova Análise de Risco para depois ser realizado o planejamento. Esse processo é repetido novamente até a versão final do *software* (MAGELA, 2006).

Segundo Sommerville (2011), cada volta da espiral é dividida em quatro quadrantes:

Definição de objetivos: Nessa etapa os objetivos específicos são definidos; as restrições ao processo e ao produto são levantadas; os riscos do projeto são identificados e podem ser planejadas estratégias para diminuição dos riscos;

Avaliação e redução de riscos. Para cada risco identificado no projeto é realizada uma análise detalhada e medidas para a redução de risco são tomadas;

Desenvolvimento e validação: É escolhido um modelo de desenvolvimento para o sistema de acordo com a atividade a ser desenvolvida. Por exemplo, se a atividade em questão for o desenvolvimento de interfaces de usuário provavelmente o modelo escolhido para o desenvolvimento dessa atividade seja o por prototipação;

Planejamento: É realizada uma avaliação no projeto e é decidido se haverá continuidade do modelo com mais uma volta na espiral.

A principal diferença entre o modelo espiral com os outros modelos é seu enfoque explícito nos riscos e na resolução desses riscos através de análise detalhada, prototipação e simulação (SOMMERVILLE, 2011).

2.2. Desenvolvimento Ágil

Como já foi visto, o ambiente atual de *software* está sujeito a muitas mudanças rápidas e se o *software* não acompanhar essas mudanças que podem surgir no decorrer do desenvolvimento ele poderá se tornar obsoleto antes mesmo de ser implementado.

Segundo Sommerville (2011) essa necessidade do processo de desenvolvimento ser rápido e capaz de lidar com mudanças de requisitos já foi diagnosticada por volta da década de 1980 com o surgimento do desenvolvimento incremental, lançado pela IBM e com as chamadas linguagens de 4ª geração.

O surgimento real das metodologias ágeis ocorreu mesmo na década de 1990, com a criação das metodologias **DSDM** (*Dynamic Systems Development Method)*, ***Scrum*** e **XP** (*Extreme Programing*) (SOMMERVILLE, 2011).

O termo **metodologias ágeis** se popularizou em 2001, quando 17 especialistas em processos de desenvolvimento de *software*, representando as metodologias citadas acima e algumas outras, estabeleceram alguns princípios compartilhados entre si e criaram o que foi chamada de **Aliança Ágil** juntamente com **Manifesto Ágil** onde são demonstrados esses princípios (KOSCIANSKI e SOARES, 2007; PRESSMANN, 2011).

Embora difiram entre si em alguns aspectos, todas as metodologias ágeis compartilham algumas características como, por exemplo, a entrega incremental como base, onde geralmente são criadas novas versões do *software* a cada duas ou três semanas. Os clientes amplamente envolvidos no processo de desenvolvimento acompanhando de perto a evolução do *software* diminuindo o risco das especificações não serem desenvolvidas incorretamente e a redução de produtos intermediários e documentações extensivas que são julgadas desnecessárias ou que não serão utilizadas posteriormente (KOSCIANSKI e SOARES, 2007; SOMMERVILLE, 2011).

Segundo Pressman (2011), um processo ágil precisa ter adaptabilidade, ou seja, deverá ser capaz de ser alterado rapidamente o projeto e as condições técnicas de acordo com as mudanças imprevisíveis que surgirem. O autor ainda afirma que essa adaptação deve ser de forma incremental e para que se consiga uma adaptação incremental é necessário que haja *feedbacks* do cliente.

Em uma metodologia ágil podem ser combinados diferentes modelos de processos de *software,* como por exemplo, a entrega é incremental baseado no

modelo incremental mas a evolução do *software* pode ser feita de modo espiral, com o gerenciamento dos riscos em cada mudança. Além de ter em cada incremento um protótipo executável.

Pressman (2011) afirma que uma abordagem iterativa entre a entrega de um protótipo para o cliente e o mesmo fornecer *feedback* influenciando nas mudanças do projeto pode capacitar o cliente para avaliar o incremento do *software* regularmente.

De acordo com Ribeiro (2013), os processos ágeis promovem um ambiente sustentável onde todos os envolvidos trabalham constantemente para chegar em um excelente resultado.

Segundo Sommerville (2011), assim como todos os processos, o desenvolvimento ágil precisa ser gerenciado de modo que sejam otimizados o tempo e os recursos disponíveis para a equipe de desenvolvimento. E isso requer do gerenciamento de projeto uma abordagem diferente mas que seja adaptada as metodologias ágeis.

2.2.1 *Scrum*

Segundo seus criadores, Schwaber e Sutherland (2011), o *Scrum* é “Um *framework* pelo qual pessoas podem tratar e resolver problemas complexos e adaptativos, enquanto produtivamente e criativamente entregam produtos com o mais alto valor possível”. Ele consiste em **papéis**, **artefatos**, **eventos** e **regras.**

Segundo Sommerville (2011), o *Scrum* é um método ágil geral, mas seu foco está no gerenciamento do desenvolvimento iterativo, não possuindo abordagens técnicas específicas da engenharia de *software* ágil.

O *Scrum* possui uma abordagem iterativa e incremental que se fundamenta nas teorias empíricas que afirmam que o conhecimento vem da experiência e da tomada de decisões baseadas no que é conhecido (Schwaber e Sutherland ,2011).

Conforme Pressman (2011, p.9), “O *Scrum* enfatiza o uso de padrões de processos de *software* que provaram ser eficazes para projetos com prazos de entrega apertados, requisitos mutáveis e críticos de negócio”.

De acordo com Ribeiro (2013), existem três pilares que apoiam a implementação do *Scrum*:

Transparência: todos os aspectos importantes do processo devem estar visíveis aos responsáveis pelos resultados. Essa transparência requer uma

padronização para que todos os observadores possam obter um mesmo entendimento do que está sendo visto;

Inspeção: frequentemente devem haver inspeções nos artefatos *Scrum* e no progresso do projeto afim de identificar variações indesejáveis que possam desviar o objetivo. Porém as inspeções não podem ser tão frequentes a ponto de atrapalhar a própria execução das tarefas;

Adaptação: se um ou mais aspectos de um processo desviar para fora de seus limites aceitáveis, ou se o resultado for inaceitável, o processo deve ser ajustado o mais brevemente possível para minimizar os desvios.

Vale ressaltar que a medida em que a complexidade do *software* aumenta a metodologia *Scrum* puramente pode apresentar algumas deficiências e suas práticas podem acabar não satisfazendo plenamente seu papel no gerenciamento do processo de *software*.

Na figura 1 o *Scrum* é ilustrado de uma forma geral. Cada componente do *Scrum* será melhor detalhado no decorrer deste capítulo.

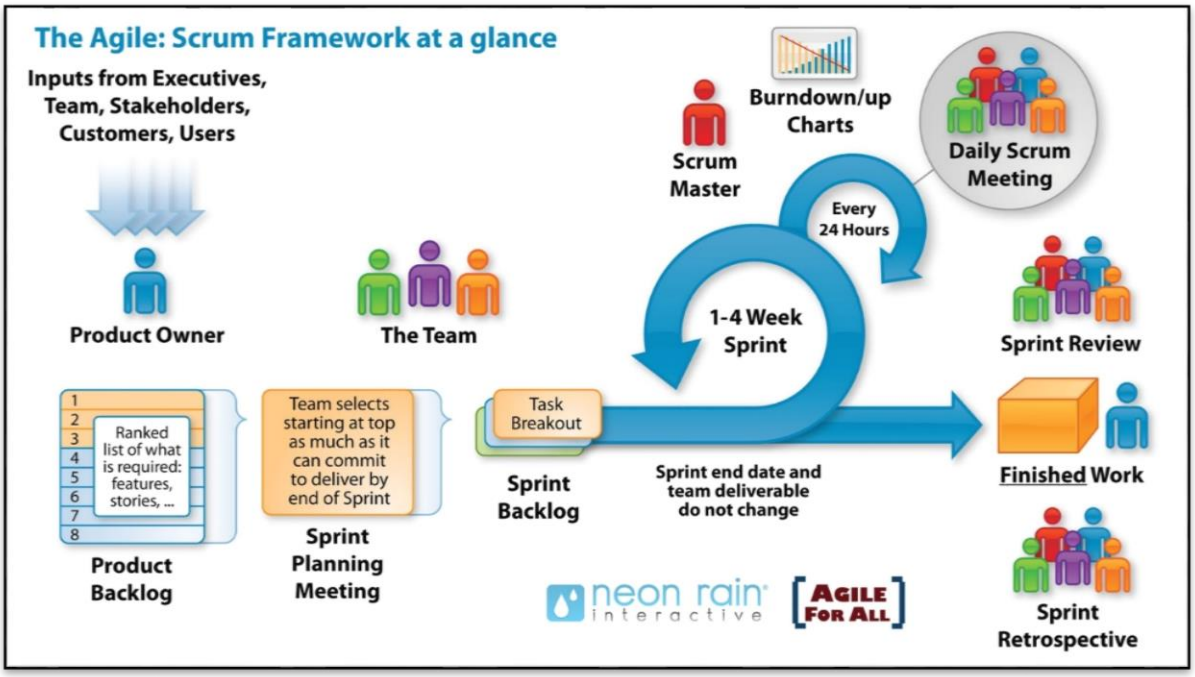

Figura 1 - Processo Scrum Fonte: HARTMAN (2010)

2.2.1.1 Papéis

O *Scrum* possuí somente três papéis: ***Product Owner***, ***Scrum Master*** e **Equipe**. O *Scrum Team*, é composto pelas pessoas que fazem esses papéis, ou seja, pelas pessoas comprometidas no projeto.

2.2.1.1.1 *Product Owner*

O *Product Owner* é a pessoa responsável por gerenciar o *Backlog*[2] do produto. Ele é a pessoa que conhece toda a visão de negócio do que será feito. Além de representar todos os interessados no projeto. Segundo Schwaber e Sutherland (2011), suas principais atividades são:

Ordenar os itens do *Backlog* afim de atingir melhores metas;

Garantir o valor de todo trabalho realizado pela equipe de desenvolvimento;

Garantir a visibilidade, transparência e clareza nas tarefas que os desenvolvedores farão;

Garantir o entendimento de todos os itens do *Product Backlog* ao nível que for necessário.

Segundo os autores acima, o *Product Owner* pode representar o desejo de várias pessoas interessadas mas poderá ser somente uma pessoa designada a função de *Product Owner*, portanto, as alterações nas prioridades dos itens de *Backlog* poderão ser feitas somente pelo *Product Owner*.

2.2.1.1.2 Equipe de Desenvolvimento

Composta pelos profissionais que trabalham diretamente no desenvolvimento do produto. Uma equipe de desenvolvimento possuí os seguintes atributos:

Auto organizável: a própria equipe de desenvolvimento organiza suas tarefas para transformar o *Product Backlog* em um incremento;

Multifuncional: A equipe contém todas as habilidades necessárias para a criação do incremento do Produto;

Responsabilidade coletiva: toda a responsabilidade pertence a equipe como um todo não existem individualidades.

Não existe um tamanho padrão para a equipe de desenvolvimento, porém seu número não pode ser tão pequeno a ponto de restringir as habilidades necessárias e também nem tão grande a ponto de exigir um maior esforço de coordenação que possa ser desnecessário (SCHWABER E SUTHERLAND ,2011).

Segundo Desenvolvimento Ágil(2013), não existe necessariamente uma divisão funcional através dos papéis tradicionais como programador, analista, *tester*,

[2]É uma lista ordenada de tudo o que é necessário ao produto (SCHWABER E SUTHERLAND ,2011).

designer. Todos no projeto trabalham juntos para completar o conjunto de trabalho com o qual são comprometidos de entregar

2.2.1.1.3 *Scrum Master*

O *Scrum Master* é quem garante que o *Scrum* será devidamente entendido e aplicado. Ele é o responsável pela aderência do *Scrum Team* à teoria, práticas e regras do *Scrum*. Além de auxiliar aos que estiverem fora do *Scrum Team* a entender quais de suas interações com a equipe são necessárias ou não (SCHWABER E SUTHERLAND ,2011).

O *Scrum Master* tem as responsabilidades de remover todo impedimento que vier atrapalhar a equipe, proteger a equipe de interrupções e interferências e auxiliar o *Product Owner* na escolha dos itens do *Backlog*. Todo recurso que for necessário ao time deve ser provido através do *Scrum Master*, *frameworks,* boas práticas e outros recursos que melhoram a produtividade da equipe também são providos pelo *Scrum Master*.

2.2.1.2 Eventos *Scrum*

Os eventos presentes no *Scrum* são utilizados para criar uma rotina e minimizar a necessidade de reuniões não definidas no *Scrum*. Todos os eventos possuem um tempo pré-determinado que não pode ser extrapolado. Isso garante que um certo tempo seja otimizado para planejamento sem haver deficiências no processo de planejamento. Cada evento é uma oportunidade para inspecionar e adaptar algo. Esses eventos são especialmente projetados para permitir uma transparência e inspeção criteriosa. Excluir qualquer um dos eventos do *Scrum* afeta diretamente seus três pilares (SCHWABER E SUTHERLAND ,2011).

2.2.1.2.1 *Sprint*

Uma *Sprint* é "…uma unidade de planejamento na qual o trabalho a ser feito é avaliado, os recursos para o desenvolvimento são selecionados e o *software* é implementado..." (SOMMERVILLE, 2011 p.50). O que foi produzido é entregue aos *stakeholders*[3] no final da *Sprint*.

[3] Parte interessada ou interveniente.

A *Sprint* é o evento central do *Scrum*. Geralmente suas durações variam entre duas a quatro semanas de acordo com a quantidade de tarefas a serem feitas. Elas são executadas de forma sequencial, ou seja, uma *Sprint* inicia logo após o término da *Sprint* anterior.

As *Sprint*s são compostas por todos os outros eventos *Scrum* restantes. De acordo com Schwaber e Sutherland (2011), durante a *Sprint*:

Mudanças que afetam o objetivo da *Sprint* não podem ser feitas;

A composição da Equipe permanece constante;

As metas de qualidade não diminuem;

O escopo pode ser renegociado entre o *Product Owner* e a Equipe.

Uma *Sprint* poderá ser cancelada somente pelo *Product Owner*. Uma *Sprint* é cancelada se o objetivo dela se tornar obsoleto, ou se a organização mudar de direção ou então se as condições de mercado ou de tecnologia mudarem.

2.2.1.2.2 *Sprint Planning Meeting*

A *Planning Meeting* é a reunião que define o que será desenvolvido na *Sprint*. Ela possuí um tempo definido de oito horas para uma *Sprint* de um mês e para *Sprint*s menores esse tempo é proporcional. Essa reunião consiste em duas partes, que possuem a mesma quantidade de tempo, que segundo Schwaber e Sutherland (2011), cada parte responde a uma das seguintes questões:

O que deverá ser entregue no final da *Sprint*?

Qual o esforço necessário para que o resultado seja entregue?

Na primeira etapa o *Product Owner* e a Equipe revisam em conjunto o *Product Backlog* e são escolhidos os itens que serão desenvolvidos durante a *Sprint*. Já na segunda etapa, sem o *Product Owner*, a Equipe planeja o desenvolvimento dos itens que foram definidos na primeira etapa, quebrando em tarefas menores e juntamente com outras atividades relacionadas a *Sprint* é formado o *Sprint Backlog*.

2.2.1.2.3 *Daily Meeting*

É uma reunião diária com participação da Equipe e o *Scrum Master*, ela possuí um tempo máximo de quinze minutos, cada integrante da Equipe deve responder as seguintes questões (SCHWABER E SUTHERLAND ,2011):

O que foi feito desde a última reunião?

O que será feito até a próxima reunião?

Existe algum obstáculo causando impedimentos em suas tarefas?

Segundo Desenvolvimento Ágil(2013), a *Daily* não deve ser utilizada como reunião para resolução de problemas. Caso necessária a discussão de um problema essa deverá ocorrer fora da *Daily* e somente com as pessoas que estão envolvidas nesse problema, para que não atrapalhe o andamento da *Daily*. Na *Daily* devem se concentrar no que cada pessoa fez e o que cada pessoa fará, fazendo com que a equipe ganhe uma compreensão melhor do que está sendo feito e o que falta a fazer. Nessa reunião não existe a figura de um chefe coletando informações sobre quem está produzindo ou não. Ao invés disso, nessa reunião os membros assumem seus compromissos perante os outros.

2.2.1.2.4 *Sprint Review Meeting*

No fim de cada *Sprint* é realizado uma *Sprint Review Meeting*, o objetivo dessa reunião é de mostrar ao *Product Owner* e aos demais interessados o que foi alcançado durante a *Sprint*. Durante a *Sprint Review*, o que foi desenvolvido recebe uma avaliação baseado no que foi especificado na *Sprint Planning Meeting*. (DESENVOLVIMENTO ÁGIL,2013).

Essa reunião dura no máximo quatro horas para uma *Sprint* de um mês. Nela o *Product Owner* identifica o que foi “Pronto” e o que não foi “Pronto”. A Equipe de desenvolvimento informa o que foi bem durante a *Sprint* e quais problemas tiveram e como esses problemas foram resolvidos. Além de demonstrar o que foi feito e responder as questões sobre o incremento (SCHWABER E SUTHERLAND ,2011).

2.2.1.2.5 *Sprint Retrospective Meeting*

A *Sprint Retrospective Meeting* ocorre após a *Sprint Review* e antes da próxima *Sprint Planning*, ela é uma oportunidade para o *Scrum Team* fazer uma auto análise e criar um plano de melhorias para serem aplicadas nas próximas *Sprint*s. Ela possuí um tempo máximo de três horas para uma *Sprint* de um mês. Ela é fundamental para que ocorra a inspeção e adaptação do processo *Scrum* (SCHWABER E SUTHERLAND ,2011).

2.2.1.3 Artefatos

Os artefatos do *Scrum* são projetados para maximizar a transparência das informações necessárias para assegurar que o *Scrum Team* consiga entregar com sucesso o incremento “Pronto” como também para oferecer oportunidades de inspeção e adaptação (SCHWABER E SUTHERLAND, 2011).

2.2.1.3.1 *Product Backlog*

O *Product Backlog* é uma lista ordenada de tudo o que é necessário ao produto e é a única origem dos requisitos para qualquer alteração a ser feita no *software.* O *Product Owner* é o único que pode incluir, alterar e excluir itens do *Product Backlog* (SCHWABER E SUTHERLAND, 2011; DESENVOLVIMETO ÁGIL, 2013).

O *Product Backlog* está sempre evoluindo dinamicamente para identificar o que o *software* precisa para ser mais apropriado, competitivo e útil. Enquanto o *software* estiver em evolução existirá o *Product Backlog*. Os itens são ordenados por valor, risco, prioridade e necessidade. Quanto mais perto do topo da lista, mais o item do *Product Backlog* deve ser considerado, detalhado e compreensível (SCHWABER E SUTHERLAND, 2011).

Os itens do *Product Backlog* são priorizados pelo *Product Owner* durante a *Sprint Planning Meeting* e são descritos a equipe. A equipe então determina quais itens conseguirão ser desenvolvidos durante a *Sprint*. Os itens escolhidos pela equipe são transferidos para o *Sprint Backlog* (DESENVOLVIMETO ÁGIL, 2013).

2.2.1.3.2 *Sprint Backlog*

A equipe se compromete a executar um grupo de atividades durante a *Sprint* e o *Product Owner* se compromete em não trazer novos requisitos para a equipe durante a *Sprint*. Os requisitos podem ser trocados somente fora da *Sprint* (DESENVOLVIMETO ÁGIL, 2013).

O *Sprint Backlog* possuí os detalhes suficientes para mostrar as mudanças em progresso e para que elas sejam entendidas durante a *Scrum Daily Meeting*. A equipe de desenvolvimento modifica o *Sprint Backlog* ao longo de toda *Sprint* e sempre que uma nova tarefa é necessária surge um novo item no *Sprint Backlog* desde que essa tarefa esteja totalmente relacionada a um objetivo da *Sprint*. Somente a equipe de

desenvolvimento pode alterar o *Sprint Backlog* durante a *Sprint* (SCHWABER E SUTHERLAND, 2011).

2.2.1.3.3 *Burndown Chart*

A qualquer momento durante a *Sprint*, o total do trabalho remanescente do *Sprint Backlog* pode ser resumido e monitorado. A equipe deve fazer esse monitoramento pelo menos uma vez ao dia para projetar a probabilidade de alcançar o objetivo da *Sprint*. Com esse rastreamento do que está faltando a equipe pode gerenciar seu progresso. Geralmente esse monitoramento é feito através do *Burndown Chart*. Ele consiste em um gráfico onde o eixo horizontal são os dias correntes da *Sprint* e o eixo vertical consiste em um valor que representa o total do trabalho remanescente. É feita uma reta de tendência que ilustra o ritmo ideal que deveria estar ocorrendo na *Sprint* e uma linha de evolução real é traçada onde mostra o ritmo em que a *Sprint* está evoluindo.

Na figura 2 é representado um modelo de gráfico *Burndown Chart*.

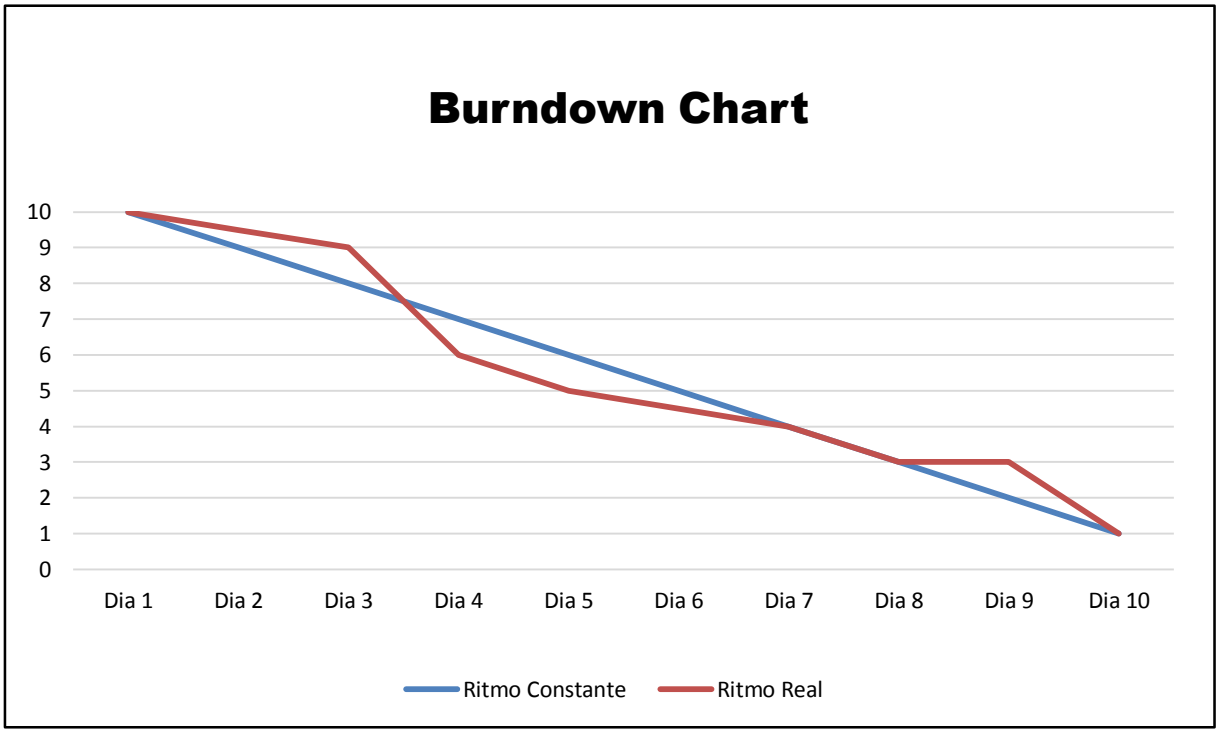

Figura 2 - Exemplo de um gráfico *Burndown Chart* fonte: Dados fictícios

Quando a linha real está posicionada acima da linha constante significa que a *Sprint* pode estar em atraso. Quando a linha real está abaixo da linha constante significa que a *Sprint* pode estar adiantada. Vale ressaltar que se houver muita

discrepância entre o real e o constante, tanto para mais quanto para menos, isso pode significar deficiências no planejamento e isto deve ser resolvido o mais breve possível.

2.2.1.3.4 Incremento

O incremento é o resultado de todos os itens do *Sprint Backlog* que foram completados durante a *Sprint*. Em todo final de *Sprint* um incremento deve estar utilizável e deve atender a definição de “Pronto” do *Scrum Team*. Essa definição deve ser universal para todos os envolvidos. A medida que o *Scrum Team* amadurece é esperado que a definição de “Pronto” seja cada vez mais abrangida para incluir critérios mais rigorosos para aumentar a qualidade do produto (SCHWABER E SUTHERLAND, 2011).

3. MELHORIA DO PROCESSOS DE *SOFTWARE*

Segundo Sommerville(2011), a procura por *softwares* mais baratos e melhores tem crescido constantemente, por isso muitas empresas criadoras de *software* voltam-se para a melhoria de processos de *software* para conseguir melhorar a qualidade de seu *software,* reduzir custos ou acelerar seus processos de desenvolvimento.

Existem inúmeros programas de melhorias de processos, contudo, segundo Paula Filho (2009), um programa de melhoria de processos não pode ser feito simplesmente de maneira intuitiva, mas deve considerar o acervo de experiência dos profissionais e das organizações relacionadas a área. Segundo o autor também, muitas organizações no mundo propuseram paradigmas para a melhoria de processos dos setores produtivos e especificamente algumas desenvolveram modelos de referência para a melhoria dos processos de *software*.

Pressman(2011) atribuí ao termo melhoria do processo de *software* a sigla SPI, que deriva do seu termo em inglês *software process improvement*. Segundo o autor, uma estratégia SPI transforma a abordagem existente para o desenvolvimento de *software* em algo mais focado, repetível e confiável além de produzir um retorno do investimento feito.

3.1. O processo de melhoria de processos

Os processos de *software* podem ser vistos em todas as organizações criadoras de *software* desde empresas formadas por apenas uma pessoa até grandes multinacionais. Os processos de *software* se diferem de acordo com o grau de formalidade do processo, dos tipos de *softwares* desenvolvidos, do tamanho da organização, ente outros fatores, mas o que deve ser considerado é que não existe um processo de *software* ideal ou padrão que pode ser aplicado a todas as organizações. Cada empresa precisa desenvolver seu próprio processo de acordo com suas necessidades (SOMMERVILLE, 2011).

Portanto, segundo Sommerville(2011), a melhoria de processos não é apenas a adoção de métodos particulares ou ferramentas ou de um processo genérico. Sempre existirão fatores organizacionais locais, procedimentos, normas e práticas que influenciarão no processo. Além disso, devem ser levados em consideração os

atributos do processo que são desejáveis de se melhorar de acordo com a necessidade da organização.

Na tabela 1 são listados alguns exemplos de atributos de processo.

Tabela 1 - Atributos de Processo Fonte: Sommerville (2011)

Características	Questões principais
Compreensibilidade	Em que medida o processo é definido explicitamente e quão fácil é entender sua definição?
Padronização	Em que medida o processo é baseado em um processo genérico padrão? Isso pode ser importante para alguns clientes que exigem conformidade com um conjunto definido de padrões de processos. Em que medida o mesmo processo é usado em todas as partes de uma empresa?
Visibilidade	As atividades do processo culminam em resultados claros, de modo que o progresso do processo seja visível externamente?
Capacidade de medição	O processo inclui a coleta de dados ou outras atividades que permitam que as características do produto ou processo sejam medidas?
Capacidade de apoio	Em que medida as ferramentas de *software* podem ser usadas para apoiar as atividades de processo?
Aceitabilidade	O processo definido é aceitável e usável pelos engenheiros responsáveis pela produção do produto de *software*?
Confiabilidade	O processo é projetado de forma que erros de processo sejam evitados ou encontrados antes de resultarem em erros de produto?
Robustez	O processo pode continuar apesar de problemas inesperados?
Manutenibilidade	O processo pode evoluir para refletir a mudança de requisitos organizacionais ou melhorias de processos identificados?
Rapidez	Quão rápido pode ser concluído o processo de entrega de um sistema após a conclusão de uma determinada especificação?

Não é possível otimizar todos os atributos de processos simultaneamente. Ao otimizar um atributo de processo provavelmente algum outro será afetado. Alguns atributos podem ser inversamente relacionados como por exemplo a rapidez e a visibilidade. Para tornar um processo visível, é necessário que as pessoas envolvidas gastem um certo tempo para produzir as informações e isso interfere na rapidez (SOMMERVILLE, 2011).

Tanto Pressman(2011) quanto Sommerville(2011) consideram que o processo de melhoria deve ser feito de forma cíclica para fortalecer a melhoria contínua do processo.

Segundo Sommerville(2011), o processo de melhoria de processos possuí três subprocessos:

Medição de processo. O objetivo é melhorar as medidas do processo de *software* de acordo com os objetivos organizacionais. Por isso os atributos do

processo são medidos para fornecer uma referência que será utilizada para mensurar as melhorias que foram feitas e identificar sua eficácia.

Análise de processo. O processo atual é avaliado e os gargalos são identificados.

Mudanças de processo. Nessa etapa são propostas as mudanças de processos para tentar resolver os gargalos e pontos fracos encontrados na etapa anterior. Após a implementação das mudanças o processo é recomeçado para avaliar se a mudança trouxe melhorias.

Essas etapas serão discutidas de forma mais ampla nos próximos tópicos.

3.1.1. Medição de processos

Segundo Pressman (2011), qualquer tentativa de melhorar o processo de *software* sem antes medir a eficácia das atividades e as práticas de engenharia de *software* associadas a essas atividades irá gastar muitos recursos e não se chegará a lugar algum.

De acordo com Sommerville (2011), as medições de processos são dados quantitativos sobre o processo de *software,* como por exemplo, a medição do tempo necessário para realizar alguma atividade do processo, e podem ser utilizadas para avaliar a melhoria da eficiência desse processo.

Pressman (2011) levanta algumas questões para serem consideradas no processo de avaliação a fim de identificar ações e tarefas que levarão a um processo de alta qualidade. Essas questões possuem também o papel de determinar se a ação em questão é executada conforme as melhores práticas possíveis.

A medição gera evidências sobre o processo e sobre as mudanças de processo. Porém essas evidências precisam ser interpretadas juntamente com outras informações para que as mudanças de processo propostas sejam mais assertivas. Isso implica em conversar com as pessoas envolvidas no processo e receber seus *feedbacks* sobre a eficácia dessas mudanças. Esse *feedback* não só revela outros fatores que são influenciadores no processo como também o grau com que a equipe adotou as mudanças propostas e assim é verificado realmente os impactos gerados pelas mudanças de processo (SOMMERVILLE, 2011).

Em suma, a medição serve para traçar os objetivos de melhorias, definir os resultados que serão considerados ideais, definir os caminhos que deverão ser tomados e por fim mensurar os resultados obtidos e grau de atingimento dos objetivos.

3.1.2. Análise de processos

A análise de processos estuda os processos sob o prisma de ajudar a compreender suas principais características e a forma que são executados na prática, pelas pessoas envolvidas. É necessário realizar algumas análises para saber o que deverá ser medido para compreender melhor o processo que será medido. A tarefa de análise envolve entender as atividades envolvidas no processo, entender as relações que essas atividades possuem dentro do processo assim como suas medições feitas e comparar o processo que está em análise com algum outro processo parecido na organização ou com algum processo idealizado (SOMMERVILLE, 2011).

Quando um processo está em uma fase de melhoria a medição aferirá os resultados que foram gerados bem como o impacto que foi gerado pela melhoria aplicada enquanto a análise aferirá a maneira como foi aplicada a melhoria e o contexto em que o processo foi realizado.

Na tabela 2 são mostrados alguns aspectos do processo que podem ser investigados durante a análise de processos.

Tabela 2 - Aspectos da análise de processos Adaptado de: Sommerville(2011)

Aspecto de processo	Questões
Adoção e padronização	O processo está documentado e padronizado em toda a organização? Se não, isso significa que quaisquer medições efetuadas são específicas apenas para uma única instância de processo? Se os processos não forem padronizados, então as mudanças para um processo podem não ser transferíveis para processos comparáveis, em outros lugares da empresa.
Prática de engenharia de *software*	Existem boas práticas de engenharia de *software*, que não estão incluídas no processo? Por que elas não estão incluídas? A falta dessas práticas afeta as características de produto, tal como o número de defeitos em um sistema de *software* entregue?
Restrições organizacionais	Quais são as restrições organizacionais que afetam o projeto de processo e as maneiras como o processo é executado? Por exemplo, se o processo envolve lidar com material confidencial, pode haver atividades no processo para verificar se as informações confidenciais não estão incluídas em qualquer material que será liberado para organizações externas. Restrições organizacionais podem significar que possíveis mudanças de processo não podem ser realizadas.

Comunicações	Como as comunicações são gerenciadas no processo? Como os problemas de comunicação se relacionam com as medições de processo feitas? Os problemas de comunicação são uma questão importante em muitos processos e, muitas vezes, os gargalos de comunicação são motivo para atraso no projeto.
Introspecção	O processo é reflexo (ou seja, os atores envolvidos no processo pensam e discutem o processo e como ele pode ser melhorado explicitamente)? Existem mecanismos pelos quais os atores de processo podem propor melhorias de processo?
Aprendizagem	Como as pessoas que integram uma equipe de desenvolvimento aprendem sobre os processos de *software* usados? A empresa tem manuais dos processos e programas de treinamento de processos?
Suporte a Ferramentas	Quais aspectos do processo são e não são suportados por ferramentas de Software? Para áreas sem suporte existem ferramentas que podiam ser implantadas efetivamente, para fornecer suporte? Para áreas com suporte as ferramentas são eficazes e eficientes? As melhores ferramentas estão disponíveis?

3.1.3. Mudança de processos

As mudanças de processos podem ser feitas através da inserção de novas práticas, *frameworks*, modificando a sequência das atividades do processo, melhorando a comunicação ou agregando novos papéis e responsabilidades. Essas mudanças devem ser guiadas pelas metas de melhoria e depois de implementadas elas deverão ser medidas e analisadas (SOMMERVILLE, 2011).

Uma mudança de processo começa com a identificação de melhorias, após essa identificação é feito um planejamento para a mudança do processo, depois é necessário o treinamento de todas as pessoas envolvidas para que realmente possam ser introduzidas as mudanças efetivamente e por último essa mudança é revisada para verificar que seja ajustada caso necessário.

Pressman (2011) apresenta os cinco fatores críticos de sucesso mais importantes para que uma organização consiga implementar uma mudança de processos com sucesso:

Comprometimento e suporte da gerência. A gerência deverá estar profundamente envolvida no desenvolvimento de estratégias de melhoria de processo, pois a melhoria de processos de *software* demanda investimento de tempo, dinheiro e esforços e o seu comprometimento e apoio são essenciais para sustentar esse investimento.

Envolvimento do pessoal. A melhoria de processos tem que ocorrer de forma orgânica e não imposta, patrocinada pela gerência e adotada pelos profissionais locais.

Integração e entendimento do processo. Os responsáveis pela melhoria em processos deverão conhecer e entender todos os outros processos comerciais além de conhecer, entender o processo de *software* da maneira em que se encontra e identificar o quão tolerável a mudança será para a cultura local.

Uma estratégia SPI personalizada. O roteiro de melhoria deve ser adaptado ao ambiente local. Devem ser considerados nessa personalização a cultura da equipe, as variedades de produtos, os pontos fortes e os pontos fracos locais.

Sólido gerenciamento do projeto de SPI. A melhoria de processos é um projeto como qualquer outro e precisa de gerenciamento. Se não haver um gerenciamento ativo e eficaz o projeto de melhoria de processos estará fadado ao fracasso.

3.2. CMMI um *framework* de melhoria de processos.

Na década de 1980, o Instituto de Engenharia de Software (SEI, do inglês *Software Engeneering Institute),* patrocinado pelo Departamento de Defesa dos EUA que é popularmente conhecido como Pentágono, criou o primeiro modelo de capacitação para a área de *software*. Esse primeiro modelo foi o SW-CMM (*Software Capability Maturity Model).* Pelo fato de ter sido utilizado pelo Pentágono para avaliar seus fornecedores de *software* esse modelo foi amplamente aceito na indústria americana de *software* (PAULA FILHO, 2009).

O SW-CMM foi base para muitos outros modelos de maturidade e capacitação não só de *software* como para outras áreas, inclusive o próprio SEI criou outros modelos de maturidade e capacitação para outras áreas como por exemplo o CMM-P e o SE-CMM. Em uma tentativa de integrar a multiplicidade desses modelos e de outros que existiam na época, o SEI criou um programa para desenvolver um modelo de capacidade integrado. Surge então como resultado desse programa o CMMI (*Capability Maturity Model Integration)* que substituí os CMM's para Software e Engenharia de Sistemas e integra a outros modelos de maturidade e capacidade (SOMMERVILLE, 2011).

No ano de 2002 ocorreu a publicação do CMMI v.1.1. Em 2006 houve publicação da última versão que é a v.1.2 com algumas alterações (PAULA FILHO, 2009).

3.2.1. Disciplinas do CMMI

De acordo com Koscianski e Soares (2007) existem quatro disciplinas presentes no modelo CMMI, que são elas: Engenharia de Sistemas, Engenharia de Software, Desenvolvimento Integrado do Produto e do Processo e Fontes de Aquisição.

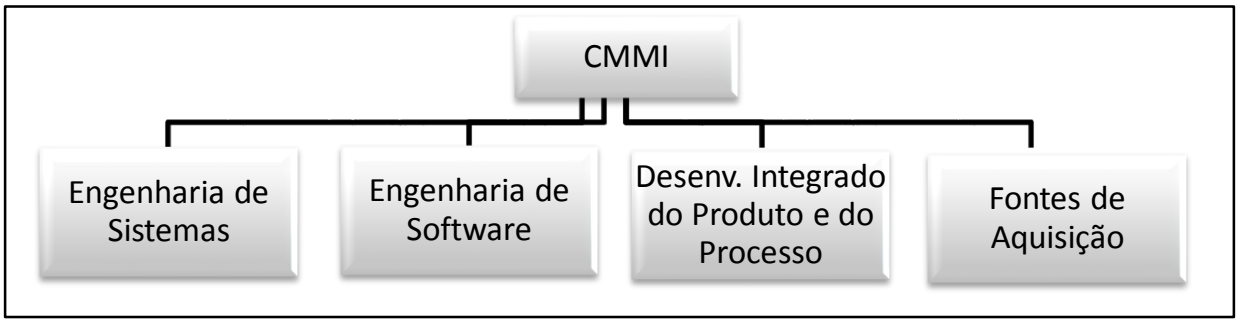

Figura 3 - Disciplinas do CMMI - Adaptado de: Koscianski e Soares (2007)

3.2.1.1. Engenharia de Sistemas

É uma abordagem interdisciplinar, onde seu principal objetivo é o sucesso na abordagem dos sistemas em geral, podendo haver ou não *software*. É consequente às necessidades e requisitos impostos pelo cliente, e através disso os engenheiros de sistemas propõem soluções e produtos com as fases de desenvolvimento, análise, projeto, validação, teste, implementação, treinamento e suporte (KOSCIANSKI E SOARES, 2007).

3.2.1.2. Engenharia de Software

De acordo com a IEEE, conforme citou Koscianski e Soares (2007), a Engenharia de *software* é a “aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção se *software*”. A Engenharia de Software não se aplica apenas ao sistema e suas fases, mas também

as atividades de gerenciamento de projetos, ferramentas e métodos que de apoio para a construção do mesmo (KOSCIANSKI e SOARES, 2007).

3.2.1.3. Desenvolvimento Integrado do Produto e do Processo

É uma abordagem sistemática para melhorar a qualidade e funcionalidades de acordo com as necessidades do cliente, e para realização desse processo é necessário à colaboração dos *stakeholders*. (KOSCIANSKI e SOARES, 2007).

3.2.1.4. Fontes de Aquisição

À medida que o desenvolvimento vai se tornando mais complexo, os processos podem precisar de fornecedores com funções específicas ou modificações em produtos específicos do projeto. No caso dessas atividades muito críticas, o projeto é submetido também a um monitoramento antes de ser lançado (KOSCIANSKI E SOARES, 2007).

Essa disciplina atua na aquisição de produtos que auxiliarão no processo de desenvolvimento como *frameworks* e outros recursos computacionais a fim de facilitar a construção do *software*.

3.2.2 Áreas de Processo

O CMMI possui um conjunto de recomendações que estão agrupadas em **áreas de processo** relacionadas às atividades de processo de *software* (SOMMERVILLE, 2011; PAULA FILHO, 2009). O CMMI é composto por vinte e duas áreas de processo relacionadas à melhoria e capacitação no processo de *software*. (SOMMERVILLE, 2011; SEI, 2006).

Na Representação contínua as áreas de processo estão organizadas em quatro categorias: Gerenciamento de Processos, Gerenciamento de Projetos, Engenharia e Suporte (SOMMERVILLE, 2011). Na representação por estágios, as áreas de processo estão organizadas de acordo com os níveis de maturidade.

Na Tabela 3 as vinte e duas áreas de processo são listadas

Tabela 3 - Áreas de Processo Adaptado de: Paula Filho (2009)

Nível	Sigla	Nome (tradução)	Nome (inglês)	Categoria
2	**REQM**	Gestão de Requisitos	*Requirements Management*	Engenharia
	PP	Planejamento de Projetos	*Project Planning*	Gestão de projetos
	PMC	Monitoração e controle de Projetos	*Project Monitoring and control*	Gestão de projetos
	SAM	Gestão de Acordos com Fornecedores	*Supplier Agreement Management*	Gestão de projetos
	MA	Medição e Analise	*Measurement and Analysis*	Suporte
	PPQA	Garantia da qualidade de processos e Produtos	*Process and Product Quality Assurance*	Suporte
	CM	Gestão de Configurações	*Configuration Management*	Suporte
	RD	Desenvolvimento de Requisitos	*Requirements Development*	Engenharia
	TS	Solução Técnica	*Technical Solution*	Engenharia
	PI	Integração de Produtos	*Product Integration*	Engenharia
	VER	Verificação	*Verification*	Engenharia
3	**VAL**	Validação	*Validation*	Engenharia
	OPF	Focalização dos Processos da Organização	*Organizational Process Focus*	Gestão de processos
	OPD	Definição dos Processos da Organização	*Organization Process Definition*	Gestão de processos
	OT	Treinamento de Organização	*Organizational Training*	Gestão de processos
	IPM	Gestão Integrada de Projetos	*Integrated Project Management*	Gestão de Projetos
	RSKM	Gestão de Riscos	*Risk Monagement*	Gestão de Projetos
	DAR	Análise e Resolução de Decisões	*Decision Analysis and Resolution*	Suporte
4	**OPP**	Desempenho dos Processos da Organização	*Organizational Process Performance*	Gestão de Processos
	QPM	Gestão Quantitativa de Projetos	*Quantitative Project Management*	Gestão de Projetos
5	**OID**	Inovação e Implantação na Organização	*Organizational Innovation and Deployment*	Gestão de Processos
	CAR	Análise e Resolução de Causas	*Causal Analysis and Resolution*	Suporte

Cada área de processo possui um conjunto de **metas específicas**, que são descrições abstratas que definem um estado desejável para cada área de processo. E para se alcançar as metas específicas são utilizadas as **práticas específicas** que definem o modo que essas metas serão atingidas (SOMMERVILLE, 2011; PAULA FILHO, 2009).

3.2.3 Metas e práticas genéricas

Um dos conceitos mais importantes do CMMI é a **institucionalização dos processos**, que garante a execução de boas práticas (SOMMERVILLE, 2011; PAULA FILHO, 2009). Esse grau de institucionalização é determinado pela obtenção das **metas genéricas,** que definem o nível de capacitação em que a organização se encontra (PAULA FILHO, 2009).

Relacionadas às metas genéricas estão às **práticas genéricas** que garantem que os processos envolvidos em uma área a de processo sejam eficazes, repetíveis e duradouros (PAULA FILHO, 2009). Na tabela 5 estão os títulos das metas e práticas genéricas.

Tabela 4 - Metas e práticas genéricas – Adaptado de: Paula Filho (2009).

Metas Genéricas		**Práticas Genéricas**	
Sigla	**Título**	**Sigla**	**Título**
GG 1	Atingir metas específicas	GP 1.1	Executar práticas específicas
GG 2	Institucionalizar um processo gerido	GP 2.1	Estabelecer uma política organizacional
		GP 2.2	Planejar o processo
		GP 2.3	Prover recursos
		GP 2.4	Atribuir responsabilidades
		GP 2.5	Treinar pessoas
		GP 2.6	Gerir configurações
		GP 2.7	Identificar e envolver partes interessadas relevantes
		GP 2.8	Monitorar e controlar o processo
		GP 2.9	Avaliar objetivamente a aderência
		GP 2.10	Rever status com alta direção
GG 3	Institucionalizar um processo definido	GP 3.1	Estabelecer um processo definido
		GP 3.2	Coletar informações para a melhoria
GG 4	Institucionalizar um processo gerido quantitativamente	GP 4.1	Estabelecer objetivos quantitativos para o processo
		GP 4.2	Estabilizar o desempenho de subprocessos
GG 5	Institucionalizar um processo otimizante	GP 5.1	Corrigir causas fundamentais dos problemas

Existe um conjunto de boas práticas recomendadas no CMMI que podem ser associadas com cada meta dentro de uma área de processo. Contudo, o CMMI entende que o que é primordial é que as metas sejam alcançadas e não a maneira como são alcançadas (SOMMERVILLE, 2011).

Quando se utiliza a representação por estágios, para que uma organização enquadre-se no nível 2 deverá ser atingida a meta genérica em todas as áreas de processo desse nível. Já para enquadrar-se no nível 3 deverão ser atingidas a meta genérica do nível 3 em todas as áreas de processo desse nível e a meta genérica do nível 2 (PAULA FILHO, 2009).

Os níveis 4 e 5, só precisam que a meta genérica do nível 3 seja atingida em todas as áreas respectivas, e as metas do no nível 4 e do nível 5 só precisam ser atingidas apenas em subprocessos identificados como críticos no ponto de vista do negócio da organização (PAULA FILHO, 2009).

O SEI permite que sejam desenvolvidos modelos que apoiem diferentes abordagens para a melhoria de processo desde que o modelo contenha os elementos fundamentais de processos efetivos para as disciplinas e descreva um plano de melhoria evolutiva que abrange desde os processos imaturos até processos maduros, disciplinados, com melhoria de qualidade e eficácia. Porém a avaliação de melhoria de processo é realizada através de duas representações diferentes: contínua e por estágios (SEI, 2006).

A principal diferença entre as duas representações é que a representação por estágios é utilizada para avaliar a capacidade da organização como um todo e a representação contínua afere a maturidade de áreas de processo específicas na organização (SOMMERVILLE, 2011).

3.2.4 Representação por Estágios

A **representação por estágios** utiliza conjuntos de metas pré-estabelecidas para as áreas de processo a fim de definir um plano de melhoria, com o qual existe muito mais experiência industrial do que com a representação contínua (PAULA FILHO, 2009).

Essa abordagem é mais simples de entender e provavelmente mais simples de se implantar. Ela é mais aderida e aceita pelas organizações (PAULA FILHO, 2009). Por ser baseada no SW-CMM, muitas organizações que já utilizavam esse modelo

anterior podem preferir manter essa abordagem para continuar algum programa de melhoria que já fora implantado (KOSCIANSKI e SOARES, 2007).

Uma vantagem em se escolher a representação por estágios é que esse modelo define um caminho de melhoria claro para as organizações que podem se planejar para moverem do segundo para o terceiro nível e assim por diante (SOMMERVILLE, 2011).

A principal desvantagem da representação por estágios é o seu caráter prescritivo. Cada nível de maturidade possui suas próprias metas e práticas. Além de essa representação requerer que todas as metas e práticas de um nível sejam implantadas antes da transição para o próximo nível. Porém as necessidades organizacionais podem necessitar que metas e práticas em nível mais alto sejam implantadas antes que as metas e práticas de níveis inferiores (SOMMERVILLE, 2011).

A Representação por Estágios é associada com a maturidade global da organização, de forma que a principal preocupação não é se processos individuais são executados ou incompletos. Portanto, o ponto de partida da representação por estágios é chamado de “inicial”, sendo cinco níveis de maturidade, numerados de 1 a 5. (SEI, 2006)

Um nível de maturidade é composto por práticas específicas e genéricas que são agrupadas em níveis de maturidade, sendo que cada um desses níveis representa a maturidade e amadurecimento de um importante subconjunto dos processos da organização, visando alcançar os próximos níveis. Esses níveis são medidos com base nessas metas estipuladas (SEI, 2006).

Segundo Paula Filho (2009), as práticas específicas são escolhidas de acordo com os seguintes critérios:

Representar fases históricas razoáveis na vida de organizações típicas;

Prover degraus intermediários de maturidade, em sequência razoável;

Sugerir medidas de progresso e objetivos intermediários;

Definir, para cada estágio, prioridades de melhoria.

São 5 níveis de maturidade cada um é uma camada que representa a base para as atividades de melhoria dos processos.

3.2.4.1 Nível 1 – Inicial

No nível 1 de maturidade, geralmente os processos são caóticos, a organização não oferece um ambiente estável de apoio aos processos, esse tipo de organização podem até produzir produtos bons, porém isso acontece com muitas horas extras feitas para compensar o mau planejamento. O Sucesso de uma organização que esteja no nível 1 está totalmente atrelado a competência e heroísmo das próprias pessoas, mas não dos processos comprovados, embora quando isso acontece, nem sempre repetem seu próprio sucesso em projetos futuros (KOSCIANSKI e SOARES, 2007).

Os gerentes dessas organizações geralmente não identificam os reais problemas e não possuem visibilidade real em relação ao progresso dos projetos. Podem até existir processos definidos no papel, mas provavelmente não são seguidos ou são seguidos em fases tranquilas. Além da rotatividade de desenvolvedores ser alta (PAULA FILHO, 2009).

3.2.4.2 Nível 2 – Gerenciado

No nível 2 de maturidade, os projetos da organização já possuem a garantia de que os processos são planejados e executados de acordo com uma política estabelecida.

Os projetos são empregados às pessoas, experientes e treinadas, que possuem os recursos necessários para produzir saídas controladas, envolvem partes interessadas, são monitorados, controlados, revisados e avaliados (PAULA FILHO, 2009).

Quando essas práticas entram em vigor, os projetos são bem executados e gerenciados, podendo facilmente suportar períodos de stress do processo, tudo é realizado em base de procedimentos documentados (SEI, 2006).

Os gerentes acompanham todo o processo aumentando a probabilidade de que os prazos sejam cumpridos. Eventuais não conformidades são identificadas com antecedência, permitindo a ação de medidas corretivas (KOSCIANSKI e SOARES, 2007).

3.2.4.3 Nível 3 – Definido

Neste nível de maturidade, os processos são bem caracterizados e entendidos, seguem padrões, procedimentos, possuem ferramentas e métodos, onde tudo é estabelecido e melhorado ao longo do tempo, de forma que estabeleçam uniformidade ao contexto da organização (KOSCIANSKI e SOARES, 2007).

O escopo da padronização dos processos abrange a organização inteira. Os processos estabelecidos para cada projeto são derivados dos processos padronizados da organização seguindo regras de personalização. A descrição dos processos é mais detalhada, com atividades especificadas, insumos, resultados, critérios de entrada e saída, papéis, métricas e procedimentos de verificação (PAULA FILHO, 2009).

Outro ponto é que no nível 3, os processos e procedimentos são rigorosamente escritos, mais proativos e se relacionam melhor devido aos padrões mais bem definidos (SEI, 2006).

3.2.4.4 Nível 4 – Gerenciado Quantitativamente

No nível de maturidade 4, os processos são controlados usando métodos estatísticos e quantitativos. Os resultados qualitativos devem ser traduzidos em números para serem mais bem compreendidos e comparados. Eventuais problemas específicos que causam variações nas medidas são corrigidos para evitar reincidência. As medidas de qualidade e desempenho são armazenadas para serem utilizadas como referências para decisões futuras. Em relação ao nível 3, uma diferença marcante é o aumento da previsibilidade do desempenho de processos devido ao controle quantitativo. No nível anterior as previsões ocorrem de forma qualitativa e mais subjetiva (KOSCIANSKI e SOARES, 2007).

3.2.4.5 Nível 5 – Em Otimização

No nível de maturidade 5, o foco é a melhora continua dos processos de uma organização com base no entendimento quantitativo das causas comuns de variação de processo. Os objetivos quantitativos de melhoria de processos são continuamente revisados de acordo com os negócios da organização e usados como critério de

gerenciamento. A responsabilidade pela melhoria de processos é transferida a todos os funcionários e colaboradores da organização permitindo que seja criado um ciclo de melhoria contínua dos processos e evitando acomodações ou o retrocesso aos outros níveis inferiores do CMMI (KOSCIANSKI e SOARES, 2007).

No nível de maturidade 4, a organização se preocupa em tratar causas especiais de variação de processo e conseguir previsibilidade estatística dos resultados. Embora os processos possam produzir resultados previsíveis, os resultados podem ser insuficientes para satisfazer aos objetivos estabelecidos. No nível de maturidade 5, a organização preocupa-se em tratar as causas comuns de variação de processo e promover mudanças no processo a fim de melhorar o desempenho de processo e satisfazer aos objetivos quantitativos de melhoria de processo estabelecidos (SEI, 2006).

3.2.5 Representação Contínua

A representação contínua possibilita a organização de escolher uma determinada área de processo e melhorar os processos relacionados a ela (SEI, 2006). Os níveis de capacidade definem uma sequência recomendada para a abordagem da melhoria de processo para cada área de processo especifica (PAULA FILHO, 2009).

A principal vantagem da representação contínua é que as empresas podem escolher os processos de melhoria de acordo com estratégia de negócio (SOMMERVILLE, 2011). Porém isso requer que a organização entenda as interdependências entre as áreas de processo, para que os investimentos em melhorias sejam feitos de modo consistente (PAULA FILHO, 2009).

Para organizações que já utilizam os modelos SE-CMM, EIA/IS 731 ou ISO/IEC TR 15504 provavelmente optarão pela representação contínua, pois a migração para o CMMI é mais natural (KOSCIANSKI e SOARES, 2007).

3.2.5.1 Nível 0 – incompleto

Trata-se da área de processo que, teoricamente, não foi realizada. Caso ela seja implementada pela organização ela recebe essa atribuição se pelo menos um de seus objetivos ou metas não foram atingidos de acordo com os objetivos definidos

pelo CMMI para o nível de capacidade 1 dessa área de processo (KOSCIANSKI e SOARES, 2007; PRESSMAN, 2011).

3.2.5.2 Nível 1 – Realizado

Os critérios do nível 1 de capacidade foram atingidos. Além disso as tarefas necessárias estão sendo executadas de modo que conseguem produzir o que foi definido (KOSCIANSKI e SOARES, 2007; PRESSMAN, 2011).

3.2.5.3 Nível 2 – Gerenciado

Além das metas no nível 1 terem sido atingidas, as pessoas que estão trabalhando recebem os recursos necessários para executar o trabalho, os *stakeholders* são envolvidos ativamente conforme necessário e todas as tarefas e resultados são monitorados, controlados e revisados (KOSCIANSKI e SOARES, 2007).

3.2.5.4 Nível 3 – Definido

Além de cumprir os critérios do nível 2, os processos são adaptados a partir de um conjunto de processos padronizados da organização de acordo com as regras de adaptação estabelecidas e outras informações de melhoria de processos para agregar cada vez mais valores ao processo organizacional (PRESSMAN, 2011).

3.2.5.5 Nível 4 – Gerenciado Quantitativamente

Além de satisfazer os critérios do nível 3 os processos são controlado por meio de técnicas estatísticas e quantitativas. Essas técnicas quantitativas são utilizadas para garantir e melhorar a qualidade e desempenho do processo, estabelecidos e utilizados como critérios na gestão de processos. A qualidade e o desempenho de processo são entendidos em termos estatísticos e gerenciados durante seu tempo de vida (SEI, 2006; PRESSMAN, 2011).

3.2.5.6 Nível 5 – Otimizado

A melhoria contínua de desempenho é o foco principal do processo otimizado. Essa melhoria contínua é buscada através de tecnologias e de inovação. As melhorias são escolhidas com base nas análises quantitativas de sua contribuição esperada em relação ao custo e qualidade no processo (KOSCIANSKI e SOARES, 2007).

Os processos selecionados para melhoria contínua são sistematicamente gerenciados e implantados. Os efeitos da implantação da melhoria de processos são mensurados e avaliados em relação aos objetivos de melhoria quantitativa, previamente estabelecidos (KOSCIANSKI e SOARES, 2007).

A melhoria é feita continuamente através da intervenção nas causas de variação de desempenho. Mesmo os processos de nível 4 serem mais previsíveis que os processos de nível 5 os processos de nível 5 são modificados a ponto de estabilizar as variações diminuindo o risco de falhar em atingir os objetivos (KOSCIANSKI e SOARES, 2007).

4.3 Outras ferramentas para a melhoria de processos

Embora o CMMI seja a estrutura de melhoria de processos mais aplicada no mundo, existem outras propostas de melhoria de processos dentre elas destacam-se os modelos: SPICE, o Bootstrap, PSP e TSP, TickIT (PRESSMAN,2011).

Algumas Normas ISO's (*International Organization for Standardization*) são utilizadas como referências para melhoria de qualidade em diversos segmentos, inclusive no ramo de desenvolvimento de *software,* destacam-se: ISO-9000, ISO/IEC-12207 e a ISO-15504 (PAULA FILHO, 2009).

No Brasil, no ano de 2005 foi lançado o MPS.Br (Melhoria do Processo de Software Brasileiro), um modelo de melhoria de processos desenvolvido pelas instituições SOFTEX, Riosoft, COPPE/UFRJ, CESAR, CenPRA e CELEPAR, seu foco é para as micros, pequenas e médias de *software* brasileiras que possuem poucos recursos para melhoria de processos mas que desejam implementá-la. Esse modelo teve como base o CMMI e as ISO's 15504 e 12207 (KOSCIANSKI e SOARES, 2007).

4. MELHORIA DE PROCESSO COM AGILIDADE.

Segundo GLAZER *et al*. (2008), usar o CMMI e o *Scrum* juntos resulta em uma melhora significativa de desempenho no processo de *software.* E em projetos pilotos que utilizaram o *Scrum* juntamente com o CMMI houve uma redução de até cinquenta por cento do trabalho do projeto (defeitos, retrabalho, o trabalho total necessário, e as despesas gerais de processo). Vale ressaltar que esse estudo foi feito através do próprio SEI que é o órgão criador e mantenedor do CMMI.

Sutherland, Ruseng Jakobsen e Johnson (2008), afirmaram que para as empresas ágeis, o CMMI pode ser usado para institucionalizar as práticas ágeis e apresentam o *Lean Software Development* como uma ferramenta operacional para identificar oportunidades de melhoria em uma empresa CMMI 5.

Marçal (2009) analisou a aderência do SCRUM em relação ao CMMI, especificamente na parte de gerenciamento de projetos e definiu um processo de gestão ágil, chamado Scrummi, no qual combina práticas do *Scrum* com práticas das áreas de processo de gerenciamento de projeto do CMMI.

Além desses diversos outros estudos já comprovaram a viabilidade em utilizar o *Scrum* juntamente com o CMMI.

4.1 Concatenação entre o Nível 2 do CMMI e o *Scrum*

Um processo *Scrum* passa sempre pelo planejamento em praticamente todos os seus eventos. Ele possui metas muito bem definidas e planejadas que devem ser atingidas no final da *Sprint,* satisfazendo a meta genérica **GP 2.2 (Planejar o processo)**. Durante a *Sprint* monitoramentos diários são feitos através do *Burndown Chart* e da *Sprint Daily* que além de auxiliarem no planejamento cumprem com as metas genéricas **GP 2.9 (Avaliar objetivamente a aderência) GP 2.10 (Rever status com alta direção), GP 2.8 (Monitorar e controlar o processo).**

As metas genéricas **GP 2.3 (Prover recursos), GP 2.4 (Atribuir responsabilidades), GP 2.7 (Identificar e envolver partes interessadas relevantes)**, podem ser satisfeitas durante a *Planning Meeting,* onde a equipe de desenvolvimento se compromete em realizar as tarefas que estão ao seu alcance e o *Product Owner* se compromete em fornecer as informações necessárias, além de haver o papel do *Scrum Master,* de angariar recursos para a equipe que auxilia no

cumprimento da meta GP 2.3 e o próprio papel do *Product Owner* que provavelmente afeta diretamente no cumprimento da meta GP 2.7.

As metas genéricas do nível 2 que aparentemente não são satisfeitas por alguma prescrição direta do *Scrum* são: **GP 2.1(Estabelecer uma política organizacional), GP 2.5 (Treinar pessoas)** e **GP 2.6 (Gerir configurações)**, mas como o *Scrum* é um *framework* focado na atividade de desenvolvimento em si e pode ser utilizado em conjunto com outras ferramentas ele não inibe ou prejudica diretamente essas práticas genéricas.

Em relação as metas específicas do nível de maturidade 2 do CMMI o Scrum também alcança alguns estados ideais para umas metas especificas e para outras ele não prescreve diretamente mas também não prejudica o alcance dessas metas.

Na meta de **gerir requisitos,** o gerenciamento dos requisitos pode estar atrelado ao *Product Backlog* e ao *Sprint Backlog,* porém os requisitos desenvolvidos precisam ser armazenados de modo que possam ser rastreáveis caso necessário. Uma sugestão simples é manter um Log de tudo o que já foi desenvolvido.

A meta de **estabelecer estimativas** está diretamente relacionada ao papel do *Product Owner* e a *Planning Meeting*, pois o *Product Onwer* é quem define o escopo do projeto, estabelece as estimativas de resultados e tarefas à equipe durante a primeira etapa da *Planning Meeting* e, em conjunto com a equipe, o ciclo de vida da *Sprint* é definido.

A meta de **desenvolver um plano de projeto** possuí algumas práticas que são muito parecidas com alguns estratégias do *Scrum* como por exemplo a prática de Gerenciamento de Riscos é feita de uma forma muito expressiva no *Scrum no* decorrer de suas iterações, que aliás, uma das características mais fortes na metodologia *Scrum* é a sua adaptabilidade para a redução dos riscos. Outras práticas dessa meta específica são facilmente dissolvidas no *Scrum* como por exemplo as práticas de estabelecimento de cronograma, planejamento de recursos e o envolvimento das partes interessadas

As práticas especificas da meta de **obter compromissos com o plano** também são facilmente relacionadas a adaptabilidade que o *Scrum* oferece.

A meta de **monitorar o projeto contra os planos** possuí muitíssimos pontos em comum com o *Scrum*, como por exemplo, o monitoramento dos parâmetros de planejamento do projeto, o monitoramento do envolvimento dos interessados, as revisões de progresso, as revisões feitas em marcos e o monitoramento de riscos.

A meta de **gerir ações corretivas até o fechamento** está relacionada diretamente a responsabilidade do *Scrum Master* durante a *Sprint* pois é sua função remover os problemas que vierem a surgir durante a *Sprint.*

Em relação as metas de **alinhar as atividades de medição e análise, fornecer resultados da medição, avaliar objetivamente os processos e resultados e fornecer visão objetiva** podem ser aplicados em um processo *Scrum*, como por exemplo a etapa de medição pode ser feita na *Planning Meeting* enquanto as etapas de análise e mudança podem ser aplicadas na *Retrospective Meeting* e em uma próxima *Sprint,* esse processo pode começar novamente partindo das aferições feitas na *Sprint* anterior permitindo que a melhoria seja feita de forma cíclica.

As outras metas específicas presentes no nível 2 não são diretamente relacionadas a atividade de desenvolvimento de *software.* Assim como as metas genéricas que não relacionam diretamente com a atividade de desenvolvimento, o *Scrum* não colabora no cumprimento dessas metas específicas mas também não interfere em suas obtenções e nem mesmo elas prejudicam o funcionamento do *Scrum.* Essas outras metas específicas são: **estabelecer acordos com fornecedores, cumprir acordos com fornecedores, estabelecer linhas de base, rastrear e controlar alterações e estabelecer integridade**.

Tanto as metas e práticas genéricas quanto as específicas do nível 2 do CMMI que são relacionadas diretamente a atividade de desenvolvimento de *software* são visivelmente compatíveis com a metodologia *Scrum.* Sendo assim possível a conciliação da melhoria de processo em um ambiente onde é utilizada a metodologia ágil *Scrum.*

4.2 Comentários sobre os resultados da pesquisa realizada

Foi realizada uma pesquisa de campo para identificar algumas informações concernentes ao processo de *software* em geral. A pesquisa teve duração de 6 meses e houve 17 respostas, entre essas respostas participaram funcionários de organizações criadoras de software das mais diversas origens, desde empresas privadas, órgãos públicos e empresas de economia mista. Houveram também respostas de funcionários de setores diferentes da mesma empresa. O formulário da pesquisa se encontra no apêndice A e as respostas se encontram no apêndice B. Baseado nas respostas dos formulários alguns gráficos foram produzidos afim de mensurar melhor esses resultados obtidos.

Na figura 4 encontra-se o gráfico da metodologia de desenvolvimento mais utilizada pelas organizações criadoras de software. Com quase metade dos resultados a metodologia mais utilizada pelas organizações é a *Scrum.*

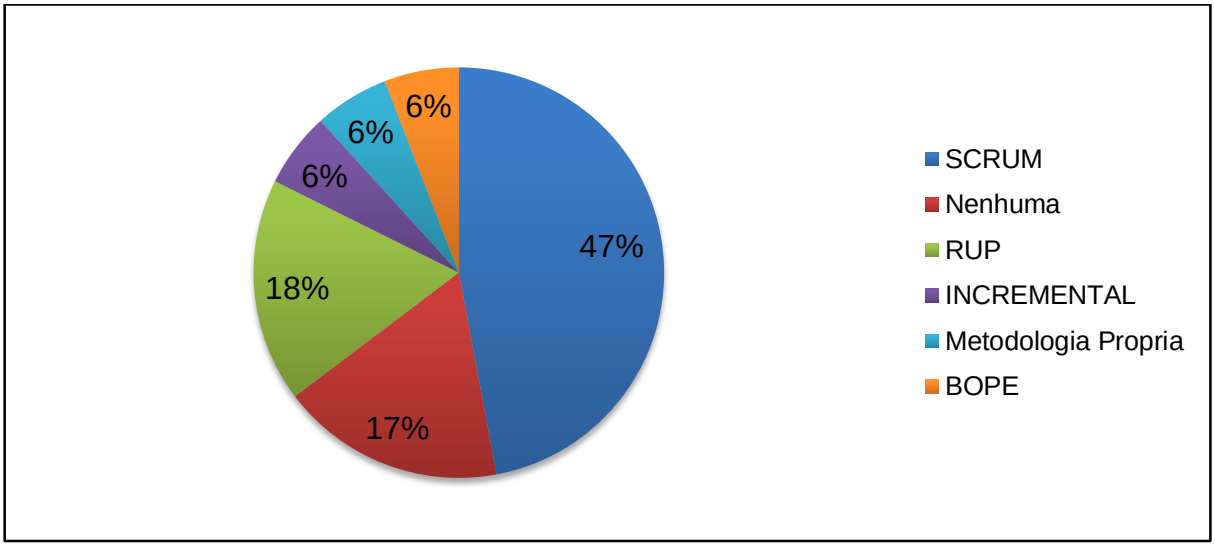

Figura 4 – Metodologia de desenvolvimento mais utilizada por empresas criadoras de *software* Fonte: pesquisa própria

Esse resultado foi muito surpreendente, pois essa metodologia foi escolhida para o estudo desse trabalho devido as suas vantagens que já foram citadas e o seu melhor enquadramento no ambiente atual de desenvolvimento de *software* e pelos dados estatísticos dessa pesquisa essa escolha também foi feita pela maioria das empresas desenvolvedoras de *software.* Esse resultado acabou agregando um maior valor na utilidade desse trabalho devido a assertividade da escolha da metodologia de desenvolvimento a ser pesquisada.

Na figura 5 encontra-se o gráfico que representa qual modelo de gerenciamento utilizado pelas empresas desenvolvedoras de *software* e trouxe um resultado interessante enquanto em relação as metodologias utilizadas, quase metade das empresas utilizam o *Scrum,* que é uma metodologia ágil, em relação a modelo de gerenciamento quase metade das organizações pesquisadas não utilizam modelo algum, inclusive nenhum dos modelos de melhoria em processos. Isto demonstra que uma boa parte de empresas que utilizam o *Scrum,* provavelmente não utiliza um modelo de gerenciamento de projeto, e especificamente não utiliza um modelo baseado na maturidade e capacitação em processos.

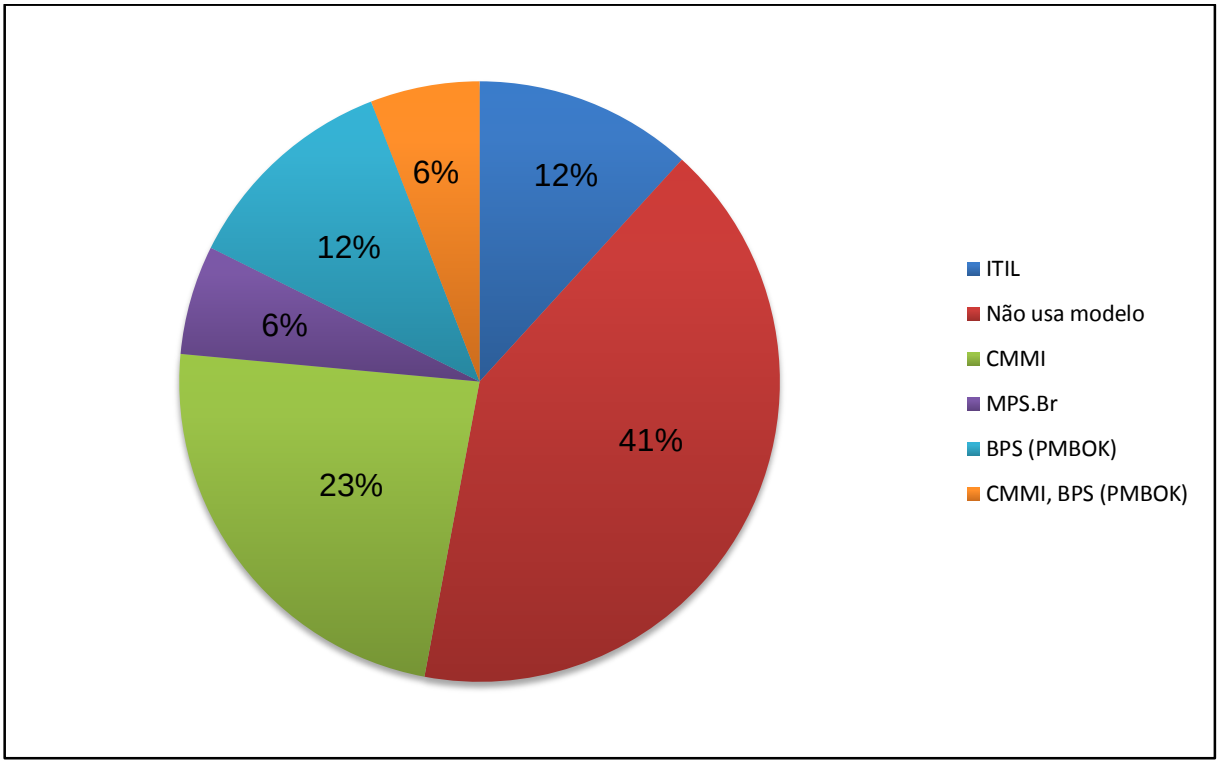

Figura 5 –Modelo de gerenciamento mais utilizado pelas empresas criadoras de *software* Fonte: pesquisa própria

Nos próximos gráficos será comparado o percentual de tempo gasto pelas empresas desenvolvedoras de software em documentação durante o projeto de *software* e o percentual de tempo gasto em manutenção corretiva após o projeto do *software,* em relação ao tempo total de desenvolvimento que o projeto teve. Vale ressaltar que essa manutenção corretiva é apenas correção de falhas e erros que foram encontrados no produto final e que precisam de correção, isto não inclui ampliação ou evolução do *software.*

Como o ambiente de desenvolvimento de *software* é bastante variável e a formalização dos processos tendem a aumentar de acordo com o tamanho da organização e como isso afeta diretamente o processo de desenvolvimento de *software* os resultados foram segmentados de acordo com os portes das organizações.

Na figura 6, são apresentados os resultados das organizações de pequeno porte. Foram classificadas nesse grupo as empresas que possuem de 1 a 200 funcionários. Foram identificadas quatro respostas nesse grupo na pesquisa. Nesse cenário a empresa que utiliza o CMMI juntamente com o *Scrum* teve seu rendimento menor em relação as outras empresas que não utilizam o CMMI e o *Scrum,* isso

provavelmente está relacionado ao fato de que esse tipo de empresa não dispõe de muito recurso, tanto humano quanto computacional e financeiro e a implementação de um modelo robusto como o CMMI seria um pouco dispendiosa.

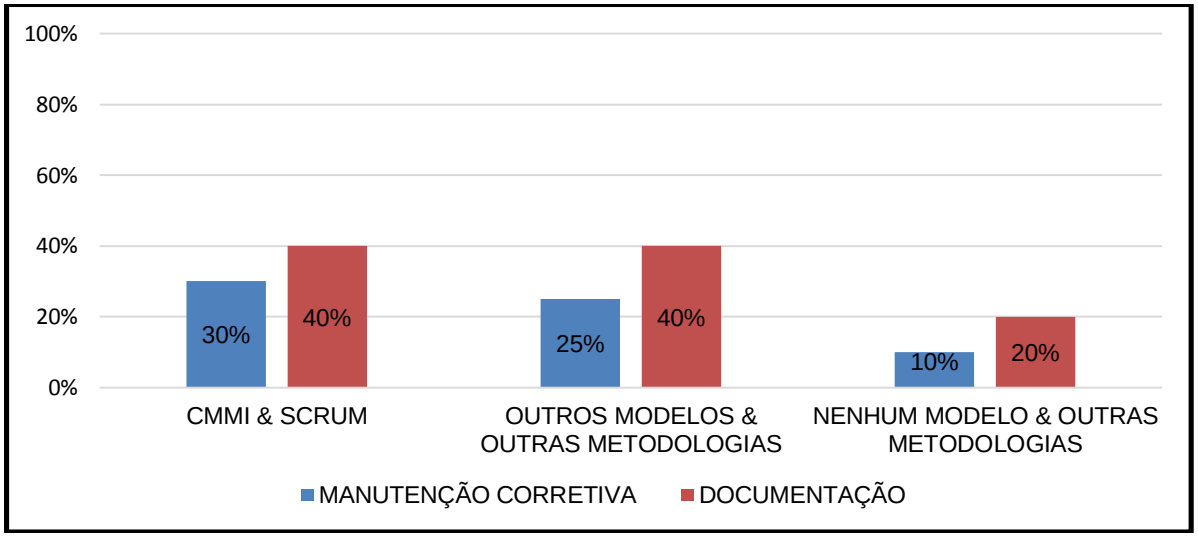

Figura 6 - Tempo médio gasto em manutenção corretiva e documentação no processo de *software* em empresas de Pequeno Porte Fonte: pesquisa própria.

Na figura 7 pode-se ver que no cenário das empresas de médio porte, que são as que possuem de 200 a 500 funcionários, foram identificadas duas respostas e dessas respostas infelizmente nenhuma utiliza o CMMI. Mas em relação a metodologia utilizada o *Scrum* apresentou uma redução de, dez por cento no tempo gasto com documentação, porém o tempo gasto em manutenção corretiva é praticamente o mesmo.

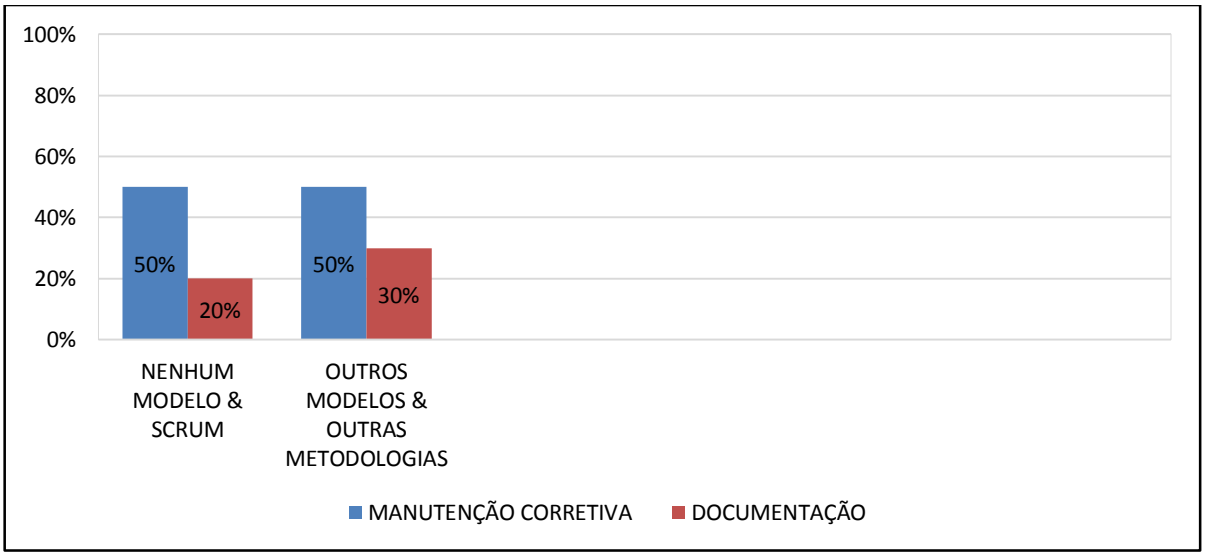

Figura 7 - Tempo médio gasto em manutenção corretiva e documentação no processo de *software* em empresas de Médio Porte Fonte: pesquisa própria.

No cenário das empresas de grande porte, que são as empresas que possuem acima de 500 funcionários foram identificadas onze respostas e a variação de metodologia e modelo de gerenciamento utilizado foi muito maior que nos cenários anteriores. Na figura 8, que demonstra os resultados desse cenário, pode-se ver claramente que o as empresas que utilizam o CMMI combinado com o *Scrum* são as que obtiveram um menor índice de tempo gasto tanto em documentação quanto em manutenção corretiva. Para quem utiliza o CMMI aliado a uma outra metodologia houve um aumento de cinco por cento em relação a documentação, para empresas que utilizam o *Scrum* e não utilizam o CMMI em conjunto o aumento na documentação foi de dez por cento. Para as empresas que não utilizam nenhuma metodologia de desenvolvimento o tempo gasto em manutenção corretiva ou com documentação foram bem maior que das empresas que utilizam uma metodologia de desenvolvimento isso mostra que o uso de pelo menos uma metodologia, quer seja tradicional ou ágil é trivial em um cenário de uma empresa de grande porte.

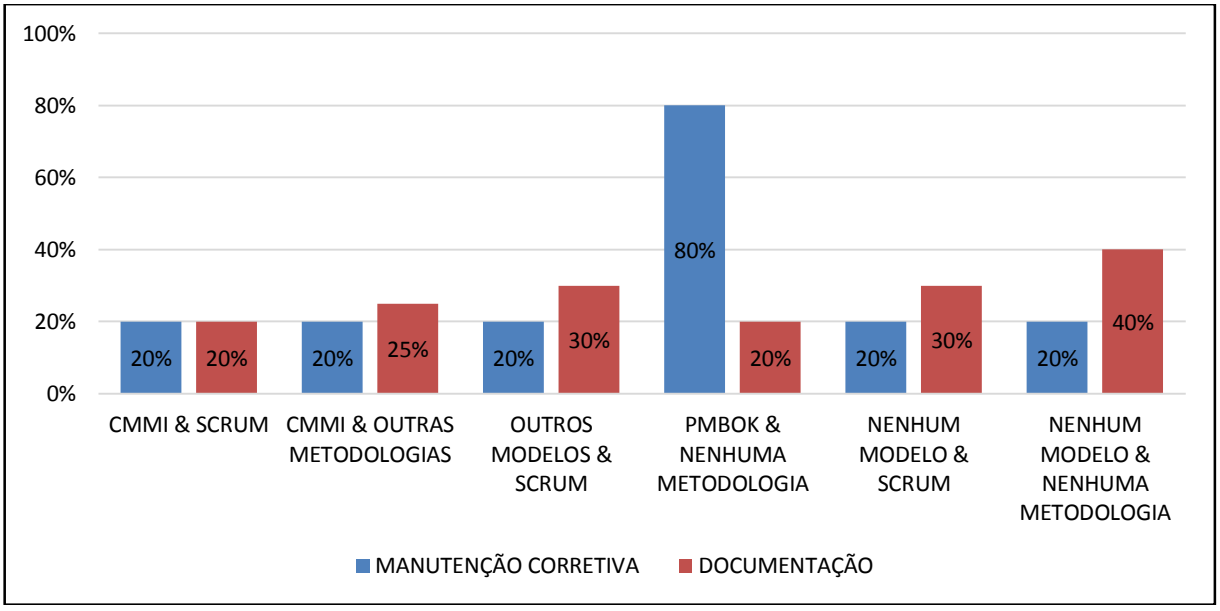

Figura 8 - Tempo médio gasto em manutenção corretiva e documentação no processo de *software* em empresas de Grande Porte Fonte: pesquisa própria.

5. CONSIDERAÇÕES FINAIS

Com o aumento cada vez mais excessivo de demanda por *softwares* de qualidade com um tempo cada vez menor para a entrega estamos vivendo em um cenário cada vez mais desafiador para as empresas criadoras de *software.*

As metodologias ágeis têm se consolidado cada vez mais nesse ambiente, no entanto não basta apenas criarmos *software* em tempos cada vez menores senão pudermos reaproveitar o que já foi feito devido à falta de padronização ou de adaptabilidade do produto. Outro fator a ser ponderado é a integração de sistemas cada vez mais descentralizados e modularizados. A necessidade de institucionalizar os processos de uma organização ainda são as mesmas de quando o processos de criação de *software* era feito de forma demorada. O *Software* precisa ser passível de evolução e integração para ser útil.

Conforme já foi visto existem diversos modelos de processos de *software* e não existe um modelo ideal que possa ser adaptado a qualquer realidade. Cabe a organização escolher seu próprio processo de desenvolvimento e adaptá-lo da melhor forma possível, de acordo com suas questões técnicas e de negócio.

A melhoria de processos surgiu para facilitar uma organização de encontrar o melhor processo possível de acordo com suas necessidades. E a melhoria de processos pode ser utilizada em qualquer processo de software seja tradicional ou ágil. Portanto se alguma empresa deseja se tornar mais competitiva ela deve utilizar de práticas de melhoria de processos para poder otimizar melhor seus recursos e obter uma lucratividade e eficiência maior.

Durante a pesquisa acadêmica, o aluno pesquisador encontrou algumas pessoas que afirmam que o CMMI é totalmente inviável para ser utilizado com uma metodologia ágil, inclusive algumas dessas pessoas ocupavam cargos de gerência e de diretoria em empresas criadoras de *software.* Pesquisando mais afundo sobre o assunto o aluno pesquisador identificou que esse que esse preconceito em relação a compatibilidade desses dois *frameworks* é muito incidente no mundo inteiro por parte de muitas empresas desenvolvedoras de *software*.

Infelizmente muitos acham que o conceito de agilidade está atrelado em produzir *software* com máxima rapidez, cortando todas as etapas possíveis e com isso acabam por cortando as etapas que são necessárias para a implantação de melhoria de processos achando que estão otimizando o tempo quando na verdade esse tempo

inicialmente investido poderia gerar um fator que aumentaria a produtividade da organização.

O CMMI é largamente o modelo mais utilizado para melhoria de processos, suas metas e práticas auxiliam na busca de encontrar um processo que reúne as melhores práticas possíveis com os recursos locais disponíveis. Sua utilização é comprovadamente benéfica para uma organização criadora de *software*.

O *Scrum* tem se tornado a metodologia de desenvolvimento mais utilizada no mundo. Sua simplicidade e adaptabilidade tem atraído muitas empresas de *software.* Nesse cenário de cobranças e instabilidades, o *Scrum* tem destacado como a melhor metodologia de desenvolvimento.

Conciliar um modelo de melhoria de processos que é amplamente utilizado com uma metodologia de desenvolvimento que mais tem se destacado é a chave para a obtenção de um processo produtivo e competitivo. A associação desses dois *frameworks* é possível devido a cada um deles ter propósitos e exigências diferentes mas que no fim buscam um mesmo objetivo que é de aumentar a qualidade e reduzir os trabalhos.

Enquanto o CMMI foca em melhorar a maturidade e capacitação do processo de uma organização abrangendo todas as possíveis áreas dessa organização como por exemplo o suporte, as vendas e o próprio desenvolvimento mas ficando em um nível de abstração alto, o *Scrum* foca explicitamente em otimizar a atividade de desenvolvimento em si. O CMMI não prescreve explicitamente qual metodologia de desenvolvimento será utilizada mas prescreve as metas que deverão ser atingidas.

Em nível de projeto de *software* o *Scrum* foca em gerenciar **o que** será feito, **quem** fará as tarefas e **quando** será entregue o que foi feito, já CMMI foca em verificar o **porquê** de fazer determinada tarefa (se já existe ou não algo que possa ser reaproveitado), **como** essa tarefa deverá ser feita (quais os padrões a serem seguidos) e **onde** os recursos necessários estão para serem disponibilizados.

Enquanto o *Scrum* trabalha para que o processo de *software* seja mais otimizado possível e com menor custo trazendo um ar de “produção enxuta” no processo de desenvolvimento de *software*, o CMMI trabalha padronizar o processo e integrar todas as áreas trazendo um ar de “industrialização” no processo de *software.*

A colaboração entre os dois *frameworks* pode ocorrer de forma mútua, enquanto o CMMI pode oferecer subsídios para os três pilares do *Scrum* (transparência, inspeção e adaptação) o *Scrum* pode auxiliar no cumprimento das

metas específicas do CMMI gerando um poderoso processo de desenvolvimento para a organização.

Com base no referencial teórico, na pesquisa acadêmica e nas publicações a respeito desse tema pode se concluir que não somente é viável utilizar o *Scrum* juntamente com o CMMI no processo de desenvolvimento de *software* como é extremamente benéfico utilizar esses dois *frameworks*, com suas utilizações o processo se torna mais produtivo, eficaz e com excelência em qualidade fazendo com que a organização que implemente esses dois *frameworks* em seu processo consiga aumentar seus resultados no mesmo tempo em que consegue oferecer um produto com alta qualidade.

Os objetivos traçados foram atingidos. Foi comprovado que é possível conciliar um modelo de melhoria de processo de software a uma metodologia ágil sem afetar tanto os princípios de uma metodologia quanto as práticas e metas de um modelo de melhoria de processos.

Os resultados da pesquisa de campo conseguiram comprovar que as empresas que utilizam o CMMI associado ao *Scrum,* na maioria dos casos, apresentaram uma maior qualidade necessitando de uma manutenção corretiva menor, e um tempo de entrega menor necessitando de um tempo menor para documentação do *software* desenvolvido.

Portanto a melhoria contínua em um processo ágil não só é viável de se buscar mas é recompensadora, pois agregará maior valor e fará com que o processo de *software* se torne sustentável e excelente em qualidade.

5.1 Propostas para trabalhos futuros

Durante o desenvolvimento desse trabalho foram identificadas as seguintes possibilidades para trabalhos futuros:

- Utilizar *Lean Software Development* como base para gerar modelos de melhoria de processos;
- Verificar a compatibilidade de outras metodologias de desenvolvimento com o modelo CMMI.
- Verificar a compatibilidade em utilizar o *Scrum* associado ao PMBOK;
- Realizar um estudo comparativo entre o CMMI e o PMBOK;

- Conciliar práticas de testes de *software* com o *Scrum*;
- Adaptar o *Scrum* em um ambiente de prestação de serviços em tecnologia da informação;
- Aprimorar técnicas de medições quantitativas em metodologias ágeis;
- Aprimorar técnicas de documentação e modelagem em processos ágeis;
- Aprimorar técnicas de rastreabilidade de requisitos em processos ágeis;
- Estudar a viabilidade do uso de Design Patterns em métodos ágeis;
- Estudar a criação de modelos de padronização e adaptação de processos.

REFERÊNCIAS

BECK, K. *et al.* **Manifesto ágil.** 2001. Disponível em: <**http://manifestoagil.com.br>.** Acesso em: 25 Set 2012.

Desenvolvimento Ágil. ***Scrum*.** Disponível em **<http://desenvolvimentoagil.com.br/*Scrum*/>**. Acesso em 20 Out 2013.

GLAZER, H. *et al.* ***Cmmi or agile:*** *Why not embrace both!.* 2008.Disponível em: **< http://repository.cmu.edu/cgi/viewcontent.cgi?article=1291&context=sei>.** Acesso em 10 Out 2013.

HARTMAN, B*.* ***Doing scrum isn't the same as living scrum!!!.*** 2010. Disponível em: **<http://www.agileforall.com/wp-content/uploads/downloads/Doing%20Scrum%20isn%27t%20the%20same%20as%20LIVING%20Scrum.pdf>.** Acesso em 05 Nov 2013.

KOSCIANSKI, A.; SOARES, M. S. **Qualidade de *software*:** aprenda as metodologias e técnicas mais modernas para o desenvolvimento de *software*. 2. Ed. São Paulo: Novatec Editora, 2007.

MAGELA, R. **Engenharia de *software* aplicada:** princípios. v. 1, 2006.

MARÇAL, A. S. C. ***SCRUM*MI:** Um processo de gestão ágil baseado no *SCRUM* e aderente ao CMMI. Mestrado em Informática Aplicada, Universidade de Fortaleza, Fortaleza, 2009. Disponível em: **< http://www.cin.ufpe.br/~if720/downloads/SCRUMMI%20-%20AnaSofia.pdf>.** Acesso em: 25 Ago 2013.

PAULA FILHO, W. P. **Engenharia de *software***: fundamentos, métodos e padrões. 3. Ed. Rio de Janeiro: LTC, 2009.

PRESSMAN, R. S. **Engenharia de *software:*** uma abordagem profissional. 7.Ed. Porto Alegre: McGraw-Hill, 2011.

RIBEIRO, R. D. **Agilidade em projetos e desenvolvimento de *software*.** Disponível em <http://www.rafaeldiasribeiro.com.br/downloads/Agilidade.pdf> Acesso em 20 Out 2013.

SENNETT, R. **A corrosão do caráter**: as consequências pessoais do trabalho no novo capitalismo**.**14. Ed. Rio de Janeiro: Record, 2009.

SOMMERVILLE, I. **Engenharia de *software***. 9. Ed. São Paulo: Addison-Wesley, 2011.

SCHWABER, K.; SUTHERLAND, J. **Guia do *scrum***. 2011. Disponível em: **<http://www.*Scrum*.org/Portals/0/Documents/*Scrum*Guides/*Scrum*GuidePortugueseBR.pdf >.** Acesso em 10 Nov 2012.

SUTHERLAND, J.; RUSENG JAKOBSEN, C.; JOHNSON, K. ***Scrum and cmmi level 5:*** *the magic potion for code warriors. In:* ***Hawaii International Conference on System Sciences, Proceedings of the 41st Annual****.* IEEE, 2008. Disponível em: **< http://jeffsutherland.org/*Scrum*/Sutherland*Scrum*CMMIMagicPotionAgile2007.pdf>.** Acesso em 10 Out 2013.

SEI, CMMI. **Guia para desenvolvimento**: Versão 1.2. CMU/SEI-2006-TR-008.

Equipe do Produto CMMI, 2006.

ZANATTA, A. L.; VILAIN**,** P**. Uma análise do método ágil *Scrum* conforme abordagens nas áreas de processo Gerenciamento e Desenvolvimento de Requisitos do CMMI.** Disponível em: **<http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER05/alexandre_zanatta.pdf>.** Acesso em 27 Abr 2013.

APÊNDICE A – Formulário da Pesquisa realizada

Pesquisa - Desenvolvimento de Software.

Esta pesquisa é acadêmica e tem por objetivo realizar um levantamento de informações referentes ao Processo de Desenvolvimento de Software adotado pelas empresas criadoras de Software. Nenhum dado aqui exposto será publicado de forma individual e sim estatísticamente e as empresas não serão identificadas.

*Obrigatório

Nome da Empresa

Porte da Empresa *

Função do funcionário que está respondendo o questionário *
Exemplo: Engenheiro de Projeto, Analista de Teste, Gerente Administrativo, etc

Você conhece o que é Metodologia de Desenvolvimento de Software? *
- Sim
- Não

Qual tipo de metodologia de desenvolvimento que é mais utilizado pela empresa?

Quais Metodologias são utilizadas na Empresa? *
- Praxis
- RUP
- SCRUM
- XP
- A empresa não utiliza Metodologia
- Outro:

Qual a Metodologia mais utilizada na Empresa? *
- Praxis
- RUP
- SCRUM
- XP
- A empresa não utiliza metodologia
- Outro:

Qual a justificativa para a pergunta anterior?

Qual o percentual do tempo gasto para modelagem / documentação? *
Levar em consideração que nenhuma especificação do cliente foi alterada, apenas ajustes de erro de desenvolvimento.

Em relação ao tempo levado para o Desenvolvimento do Projeto. Qual é o percentual de tempo gasto para ajuste do Sistema após sua implementação? *
Levar em consideração que nenhuma especificação do cliente foi alterada, apenas ajustes de erro de desenvolvimento.

A empresa possuí algum modelo de Gerenciamento utilizado no Projeto de Software?

- ( ) Sim
- ( ) Não

Se sim, Qual Modelo?

- [ ] CMMI
- [ ] MPS.Br
- [ ] BPS (PMBOK)
- [ ] ITIL
- [ ] PSP
- [ ] ISO /IEC

Em média qual a quantidade de integrantes das equipes de desenvolvimento? *
Considerando Analistas de Requisitos, Desenvolvedores e Testers.

Como a empresa gerencia as mudanças de prazos e custos?

As equipes geralmente possuem algum problema de comunicação ou trabalham em harmonia?

Enviar

Nunca envie senhas em formulários do Google.

APÊNDICE B – Respostas ao formulário de pesquisa

Respostas do formulário de pesquisa parte 1 de 4

n.	Indicação de data e hora	Função do funcionário que está respondendo o questionário	Você conhece o que é Metodologia de Desenvolvimento de Software?	Qual Metodologia de Desenvolvimento de Software a empresa utiliza?
1	4/28/2013 17:52:58	Analista de Sistemas	Sim	SCRUM
2	5/7/2013 9:27:56	Técnico de TI	Sim	Metodologia Tradicional
3	5/13/2013 20:25:45	Developer	Sim	Metodologia Ágil
4	8/19/2013 12:05:32	Analista de Suporte	Sim	Metodologia Tradicional
5	8/22/2013 19:35:47	Líder de Projetos	Sim	Metodologia Ágil
6	8/23/2013 10:12:28	Diretor	Sim	Metodologia Tradicional
7	8/27/2013 17:12:30	Arquiteto de Software	Sim	Metodologia Ágil
8	8/28/2013 8:55:54	Analista de Software	Sim	Metodologia Ágil
9	8/28/2013 9:25:14	Gerente de Projetos	Sim	Metodologia Tradicional
10	8/28/2013 9:27:46	Product Owner	Sim	Metodologia Ágil
11	8/28/2013 20:18:54	Lider Técnico	Sim	Metodologia Tradicional
12	8/29/2013 13:00:27	gerente de projetos	Sim	Metodologia Ágil
13	8/29/2013 22:28:09	Analista de Sistemas	Sim	Metodologia Tradicional
14	9/3/2013 1:24:17	Gerente	Sim	Metodologia Ágil
15	9/3/2013 16:24:54	Analista de Sistemas e Negócios	Sim	Metodologia Ágil
16	9/4/2013 14:57:03	Engenheiro Junior Desenvolvimento	Sim	Nenhuma
17	9/16/2013 20:10:34	Analista de Sistemas	Sim	Metodologia Tradicional

Respostas do formulário de pesquisa parte 2 de 4

n.	Qual a Metodologia mais utilizada na Empresa?	Qual a justificativa para a pergunta anterior?	Qual o percentual do tempo gasto para modelagem / documentação?
1	SCRUM	Envolvimento do cliente no processo, facilidade de ajuste, pouco tempo com documentação.	21% - 30%
2	A empresa não utiliza metodologia	No ambiente de trabalho, as equipes dividem as taredas de desenvolvimento com análise de negócios, suporte e testes. Desta forma, a organização interna para uso de uma metodologia plena é vedada.	61% - 70%
3	RUP		11% - 20%
4	SCRUM		41% - 50%
5	SCRUM		21% - 30%
6	INCREMENTAL	Como trata-se de um projeto "sem fim definido" e com muitas customizações e evoluções (ERP customizável) a metodologia tradicional escolhida foi a incremental por permitir ter diversas fases de análise, projeto, desenvolvimento e testes (cada etapa é um incremento ao software como um todo).	11% - 20%
7	SCRUM	Adotamos metodologias ágeis, em particular SCRUM, a alguns anos como forma de: - Permitir uma avaliação mais constante por parte do cliente, tanto do escopo quanto da qualidade do produto - Facilitar o gerenciamento do escopo - Reduzir custos através da redução do overhead associado a processos tradicionais - Melhorar nosso tempo de resposta	0% - 10%
8	SCRUM	É uma metodologia ágil cujo processos e métricas, ambos bem definidos, fazem com que grandes empresas a olhem com bons olhos.	0% - 10%
9	RUP		21% - 30%
10	SCRUM		11% - 20%
11	Metodologia Propria		31% - 40%
12	BOPE	Ambiente de pesquisa e desenvolvimento de software. Mescla aspectos empresarias e ambiente acadêmico.	51% - 60%
13	A empresa não utiliza metodologia		11% - 20%
14	SCRUM		21% - 30%
15	SCRUM	Agilidade no desenvolvimento e interação com o usuário.	31% - 40%
16	A empresa não utiliza metodologia		0% - 10%
17	RUP		0% - 10%

Respostas do formulário de pesquisa parte 3 de 4

n.	A empresa possuí algum modelo de Gerenciamento utilizado no Projeto de Software?	Se sim, Qual Modelo?	Em média qual a quantidade de integrantes das equipes de desenvolvimento?	Como a empresa gerencia as mudanças de prazos e custos?
1	Sim	ITIL	4	
2	Não		3	Não há um gerenciamento efetivo de mudanças de prazos, muito menos custos.
3	Sim	ITIL	10	-
4			2	
5	Sim	CMMI, ITIL, ISO /IEC	8	
6	Não		3	As mudanças de prazos e custos são gerenciadas de modo a evitar o menor impacto ao cliente. Sempre que possível são tratadas internamente e ajustes do projeto são realizados. Quando necessário, trata-se com o cliente e o projeto deve ser revisado e aprovado novamente pelo cliente.
7	Não		6	O escopo é definido em alto nível pela visão do projeto e em granularidade menor pela descrição de funcionalidades com valor agregado para o cliente (chamadas estórias). Estas estórias são listadas e priorizadas. Logo o gerenciamento de prazo e custo é negociado sobre estes dois pontos: a visão geral do projeto e do "backlog" de estórias. Novas funcionalidades ou desvios são negociados entre as duas partes.
8	Sim	CMMI, ISO /IEC	10	Adicionando histórias para o próximo sprint
9	Sim	MPS.Br	7	
10	Não		15	Sob demanda.
11	Sim	CMMI	8	
12	Sim	BPS (PMBOK)	20	Gerencia de riscos, escopo do PMBOK e conversa constante com o cliente.
13	Não	BPS (PMBOK)	40	Gerente de Projetos
14	Não		12	
15	Sim	CMMI, BPS (PMBOK)	6	Através do acompanhamento do cronograma, sendo o mesmo atualizado em conjunto com o cliente.
16	Não		6	Dificilmente acontecem mudanças de prazos, porém se houver necessidade acontece aumento de horas extras.
17	Sim	CMMI	12	Há uma área de negócios que negocia diretamente com o cliente, após comunicado da equipe de desenvolvimento.

Respostas do formulário de pesquisa parte 4 de 4

n.	As equipes geralmente possuem algum problema de comunicação ou trabalham em harmonia?	Porte da Empresa	Quais Metodologias são utilizadas na Empresa?	Em relação ao tempo levado para o Desenvolvimento do Projeto. Qual é o percentual de tempo gasto para ajuste do Sistema após sua implementação?
1		Grande Porte - acima de 500 funcionários		11% - 20%
2	Há problemas de comunicação, mas que tem sido enfaticamente trabalhados para melhorar.	Grande Porte - acima de 500 funcionários	A empresa não utiliza Metodologia, Estamos em fase de concepção para utilização de metodologias ágeis.	0% - 10%
3	regular	Pequeno Porte 1 - 200 funcionários	RUP, SCRUM	21% - 30%
4		Grande Porte - acima de 500 funcionários	SCRUM	11% - 20%
5	50% 50%	Grande Porte - acima de 500 funcionários	RUP, SCRUM, XP, kanban	21% - 30%
6	Problemas de comunicação não são frequentes mas acontecem pois quando estamos tratando de recursos humanos sempre existe o fator "pessoa" e as pessoas possuem dias bons e dias ruins. Procuramos tratar os problemas de comunicação de maneira pontual e utilizar como experiência para evitar futuros problemas parecidos.	Pequeno Porte 1 - 200 funcionários	Tradicional: INCREMENTAL	0% - 10%
7	O processo envolve reuniões diárias de status e feedback ao final de cada "sprint", porém já vi casos de sucesso e falha. Vejo que isto depende imensamente do perfil das pessoas envolvidas no projeto e da sua liderança.	Grande Porte - acima de 500 funcionários	SCRUM	11% - 20%
8	Existem problemas de comunicação e politica, como em toda empresa grande.	Grande Porte - acima de 500 funcionários	SCRUM, XP	0% - 10%
9		Médio Porte 201 - 500 funcionários	RUP, SCRUM	41% - 50%
10	Existem problemas pequenos de comunicação.	Médio Porte 201 - 500 funcionários	SCRUM, TDD	41% - 50%

11	Os problemas de comunicação ocorrem pela divisão do time entre cidades e cliente x fornecedor (o cliente tem membros que participam de alguma forma do time).	Grande Porte - acima de 500 funcionários	Metodologia Própria	11% - 20%
12	hoje as equipes priorizam a comunicação face a face. mas o meail eh bastante usado e temos regras para acessar o email no minimo 3 vezes ao dia.	Pequeno Porte 1 - 200 funcionários	RUP, SCRUM, XP, BOPE	11% - 20%
13	Sim.	Grande Porte - acima de 500 funcionários	A empresa não utiliza Metodologia	71% - 80%
14		Grande Porte - acima de 500 funcionários	SCRUM	11% - 20%
15	Trabalham em harmonia.	Pequeno Porte 1 - 200 funcionários	SCRUM	21% - 30%
16	Sempre existem problemas de comunicação, porém acredito que os mais simples podem ser contornados, porém alguns tem de ser corrigidos com urgência.	Grande Porte - acima de 500 funcionários	A empresa não utiliza Metodologia	21% - 30%
17	Sim, possuem.	Grande Porte - acima de 500 funcionários	RUP, SCRUM	11% - 20%