UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
MBA EM ADMINISTRAÇÃO DA TECNOLOGIA DA INFORMAÇÃO

GABRIEL SCHMITT KOHLRAUSCH

PROPOSTA PARA IMPLANTAÇÃO E GESTÃO DE UM PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE* BASEADO EM MÉTODOS ÁGEIS

São Leopoldo - RS
2013

Gabriel Schmitt Kohlrausch

PROPOSTA PARA IMPLANTAÇÃO E GESTÃO DE UM PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE* BASEADO MÉTODOS ÁGEIS

Trabalho de Conclusão de Curso de Especialização apresentado como requisito parcial para obtenção do título de Especialista em Administração de TI, pelo MBA em Administração de TI, da Universidade do Vale do Rio dos Sinos - UNISINOS

Orientador: Prof. Me. Edson Wobeto

São Leopoldo-RS

2013

"Nós somos aquilo que fazemos com frequência. Excelência, então, não é um ato, mas um hábito."

(Aristóteles)

RESUMO

Este trabalho faz um estudo de caso para realizar a proposta de implementação de um processo de desenvolvimento de *software* baseado em métodos ágeis para times de desenvolvimento de uma empresa de TI de pequeno porte. O desenvolvimento de *software* baseado em princípios ágeis faz com que o time possa entregar *software* com maior retorno de investimento, mais rápido, permitindo alterações de escopo e colaboração real entre os membros do time. Primeiramente são investigadas questões relacionadas a cultura organizacional, gestão do conhecimento, engenharia de *software* e metodologias ágeis. Posteriormente é realizada uma análise de dados obtidos através de documentos internos, sistemas de informação da empresa estudada, e entrevistas com os times envolvidos no processo de desenvolvimento de *software* para identificar o processo de desenvolvimento atual e seus problemas. Por fim, o resultado apresentado indica um projeto para implantação de um novo processo de desenvolvimento de *software* baseado em métodos ágeis e com gestão visual para o time de P&D da empresa estudada.

Palavras-chave: Engenharia de *Software*. Cultura organizacional. Metodologias ágeis. *SCRUM.* Desenvolvimento de *Software*.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

P&D	Pesquisa e desenvolvimento
ERP	*Enterprise Resource Planning*
ES	Engenharia de *Software*
BSC	*Balanced Scorecard*
CASE	*Computer-Aided Software Engineering*
SECI	Socialização, Externalização, Combinação e Internalização
XP	*Extreme Programming*
FDD	*Feature-Driven Development*
LSD	*Lean Software Development*
DSDM	*Dynamic Systems Development Methodology*
TDD	*Test Driven Development*
AUP	*Agile Unified Process*
YAGNI	*You Aren't Gonna Need It*
WIP	*Work in Progress*

SUMÁRIO

1 INTRODUÇÃO

Um dos grandes desafios, para equipes de desenvolvimento de *software*, é demonstrar para seus clientes, internos ou externos, que o *software* desejado muitas vezes é mais complexo de ser criado comparado a outros projetos que não são de origem criativa. Isto gera um ambiente de muitas mudanças e incertezas, afetando diretamente o processo de desenvolvimento de *software*, o time envolvido e a gestão do negócio envolvido.

Utilizando-se de uma metáfora, pode-se dizer que requisitos funcionais são uns dos principais insumos para o desenvolvimento de um produto de *software*. Porém, como vivemos em um ritmo acelerado de mudanças em tecnologia da informação e busca por inovações (BOEHM, 2006) os requisitos estão sujeitos a alterações, estas que podem ser frequentes conforme a demanda do cliente. A alteração frequente de requisitos, torna o desenvolvimento de *software* um desafio (CARVALHO; MELLO, 2009, p.01), principalmente para equipes pequenas onde os integrantes da equipe não podem dedicar-se a um projeto exclusivo.

1.1 SITUAÇÃO PROBLEMÁTICA E PERGUNTA DE PESQUISA

A empresa estudada, STI - Society Tecnologia da Informação LTDA., é uma pequena empresa que desenvolve uma solução *Enterprise Resource Planning* (ERP[1]) especializada para clubes sociais, esportivos, náuticos, agremiações, sindicatos e parques. Ela está no mercado desde 1986, buscando o foco no desenvolvimento e aprimoramento de seu produto de *software* principal, o Society Associados. Com os anos, a STI, tornou-se referência no segmento devido ao *know-how* embutido no produto Associados. A STI está dívida em cinco times operacionais são eles: Suporte técnico, Customização, Pesquisa & Desenvolvimento, Administração e Comercial.

O mercado de *software* para clubes sociais é limitado, isto, devido ao fato de que não existe a abertura de novos clubes sociais, fazendo com que o mercado se

[1] ERP – Enterprise Resource Planning (Planejamento de Recursos da Corporação). São sistemas de informação que integram os dados e processos de uma organização em um único sistema e podem ser vista sub uma perspectiva funcional e sistêmica.
CRUZ, Tadeu. Sistema de Informações Gerenciais – Tecnologias da Informação e a Empresa do Século XXI. 3ª. ed. São Paulo: Atlas, 2009;

concentre em clubes tradicionais, os quais normalmente possuem muitos anos de existência e se tornaram referência na comunidade. Outro fator importante neste mercado é falta de uma gestão mais profissional, isso porque clubes sociais possuem uma administração, muitas vezes, política fazendo com que a equipe gerencial seja substituída após a eleição de uma nova diretoria e presidência. Em contra partida, a Confederação Brasileira de Clubes e as Federações Estaduais de Clubes, realizam um trabalho de conscientização para uma administração mais profissional, tornando a gestão mais próxima a de empresas privadas. Devido a este avanço na gestão dos clubes sociais a STI tem investindo no desenvolvimento de módulos que auxiliam a gestão de outras áreas de um clube.

Estes módulos podem ser de uso administrativo como, por exemplo, módulos de controle patrimonial, compras, estoque e contabilidade ou módulos bem específicos que atendem áreas como Gestão de Eventos, Gestão de Acervos (Biblioteca) e Gestão de Estacionamento. Uma última opção de módulo seria o desenvolvimento customizado conforme necessidade de automatização e/ou gestão que um clube venha a necessitar. A customização pode ser realizada sobre um módulo existente ou até mesmo a construção de um módulo especifico para o cliente. A composição do módulo Associados com os módulos administrativos, módulos específicos e customizados tem o objetivo de compor um ERP específico para clubes sociais, clubes de lazer, condomínios e clubes de futebol.

Com o surgimento de novos centros de lazer, com características de clubes sociais e surgimento de programas de sócio torcedores em clubes de futebol a concorrência da empresa aumentou significativamente. Hoje a empresa possui concorrentes diretos, os quais possuem um diferencial importante que é uma plataforma tecnológica mais atual.

A STI se destaca frente a concorrência devido ao produto de *software* final, Society ERP, ser mais abrangente que os produtos oferecidos pelos concorrentes. Outra vantagem competitiva é o fato do produto de *software* Society ERP ser construído para permitir uma adaptação facilitada, sendo focado na adequação do produto as diversas necessidades dos clientes e aos estatutos sociais dos clubes. Outro ponto que se destaca, frente a concorrência, é o tempo de mercado que a empresa possui refletindo assim na especialização do time da STI para atender clubes sociais, criando um serviço de atendimento personalizado e único.

A adaptabilidade do produto de *software* oferecido pela STI permite que ela possa captar clientes de diferentes nichos dentro do mercado de *software* para clubes e de diferentes tamanho, seja área física, quadro social ou fluxo de caixa. Entre os principais nichos são clubes sociais, clubes de futebol, parques e clubes condomínio espalhados pelos principais estados Brasileiros.

Operacionalmente a STI conta com os times de Administração e Comercial para realizar trabalhos de cunho menos técnico, sendo que o time de Administração fica responsável pelo faturamento, contas a pagar, controle financeiro, folha de pagamento e outros processos internos. Já o time Comercial realiza prospecção de clientes, propostas e contratos, porém colaboradores dos times técnicos realizam demonstrações e visitas a futuros clientes. Todo material de publicidade necessário é em parte desenvolvido internamente pelo setor comercial e em parte desenvolvidos por agencias terceirizadas.

Compondo a parte técnica da STI temos os times de Suporte Técnico, Customização e o time de Pesquisa & Desenvolvimento (P&D). O time de Suporte Técnico é responsável em manter o atendimento de *HelpDesk* aos clientes, solucionando dúvidas e realizando análise de demandas. O trabalho do time de Suporte Técnico é integrado com o time de Customização, o qual realiza as adaptações nos produtos para atender demandas especificas dos clientes, bem como executa projetos específicos oriundos das necessidades de clientes. É importante ressaltar que o time de Customização é formado por pessoas multidisciplinares, as quais realizam atividades de análise, programação e testes. Todo trabalho de programação é desenvolvido com base no framework disponibilizado pelo time de P&D. Novas implantações em clientes são realizadas por times multidisciplinares com integrantes de vários setores operacionais. Já o time de P&D se responsabiliza pela saúde do núcleo do framework e pesquisa de novas tecnologias para utilização dos demais times operacionais. Como também se trata de um time multidisciplinar tarefas de análise, programação e testes ficam a encargo de todos os integrantes do time. Os times de Suporte Técnico e P&D contam, cada um, com um coordenador, responsáveis pelo andamento das operações nos setores e a integração dos mesmos.

O processo de desenvolvimento adotado pelos times técnicos é baseado em interações, onde um conjunto de demandas, oriundas de projetos, desejos de

clientes, novas funcionalidades e afins, são priorizadas para que os times tenham uma lista de demandas priorizadas para trabalhar na semana. O processo de escolha de prioridades é feito em uma comissão formada pelos coordenadores dos times de Suporte Técnico e P&D, responsável pelo time comercial e um integrante do time de Customização. Cada membro da comissão se responsabiliza em montar sua lista de demandas a fim de compartilhar com os demais membros, buscando assim a melhor priorização de demandas conforme a visão compartilhada com todos. Após a definição de demandas prioritárias, as mesmas, são alocadas ou para a Customização ou P&D, conforme a origem, para que os times possam trabalhar durante a semana. O processo se repete na semana seguinte, sempre com a revisão das demandas da semana que passou.

Hoje a STI não possui indicadores de desempenho no processo de desenvolvimento, prejudicando a tomada de decisões e gestão do processo de desenvolvimento. Este fato ainda impacta diretamente na percepção de qualidade dos produtos entregues, fazendo com que ações na área de qualidade de *software* sejam adiadas. Os times de Customização e P&D, por serem multidisciplinares, acabam trabalhando em demandas de origens diferentes, ou seja, eventualmente o time de P&D realiza customizações e eventualmente o time de Customizações realiza trabalhos para o time de P&D. A falta de estimativas nas demandas torna muito difícil o planejamento semanal para os times, isto aliado a falta de indicadores de produtividade, acarreta muitas vezes na entrega parcial do planejamento realizado, resultando na frustração dos times. Na busca de melhores resultados os coordenadores dos times acabam alterando processos, porém essas alterações não são baseadas em fatos e sim no “sentimento” de que existe algo errado, este movimento, acaba comprometendo a melhoria contínua do processo e o aprendizado organizacional. Outro sintoma resultante da falta de indicadores para a gestão, é a troca constante de prioridades entre demandas, levando muitas vezes a produção parcial de funcionalidades ou a produção de funcionalidades que não agregam o devido valor de negócio ao cliente. Todos estes sintomas, afetam diretamente o poder de planejamento a médio e longo prazo da STI e consequentemente o acompanhamento de objetivos estabelecidos.

Este problema, determinou a questão de pesquisa deste trabalho, que é: Como implementar um processo de desenvolvimento de *software* e seus indicadores de gestão baseado em metodologias ágeis?

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Esta monografia tem por objetivo relacionar o atual processo de desenvolvimento de *software,* da empresa STI, com metodologias ágeis sugeridas pelos autores, a fim de analisar o processo de desenvolvimento de *software* atual e problemas relacionados para então propor melhorias que possam ajudar a definir um novo processo de desenvolvimento de *software* e a gestão do mesmo baseado em métodos ágeis, para a empresa STI.

1.2.2 Objetivos Específicos

- Pesquisar e descrever o processo atual de desenvolvimento de *software,* da empresa STI e os principais problemas encontrados
- Pesquisar metodologias ágeis e a aderência de suas práticas, no processo de desenvolvimento de *software* da empresa STI.
- Propor um processo de desenvolvimento de *software* buscando alinhamento com os princípios ágeis.
- Propor ferramentas que apoiem a gestão do processo de desenvolvimento de *software* afim de buscar a melhoria contínua do processo.

1.3 JUSTIFICATIVA

Devido ao ritmo acelerado de mudanças e exigência de constante inovações para os produtos desenvolvidos, pela STI, a metodologia de desenvolvimento utilizada atualmente não permite que todos os setores da empresa possam trabalhar com uma visão mais longa e sistemática do todo.

A STI já passou por várias experiências e desafios, relacionados a metodologias de desenvolvimento de *software*, mas nada resultou em uma metodologia consistente para a empresa. Assim o sentimento dos times envolvidos é de que “no fim tudo vai dar certo”, porém não é possível mais aceitar a “sorte” como delimitadora de sucesso. É necessário planejar, medir e controlar para, assim, aumentar a “sorte”.

Como a STI trabalha com um ambiente de negócio dinâmico, e segundo Conforto, Amaral (2007), tal ambiente é caracterizado pela dificuldade em prever o futuro, incertezas e grandes desafios. O modelo tradicional de desenvolvimento de *software* e gestão de projetos passa a ser questionável quanto a sua eficácia. Como alternativa ao modelo tradicional, menos flexível a mudanças, metodologias ágeis de desenvolvimento e gerenciamento de projetos objetivam atender as necessidades de rápidas mudanças, impostas no decorrer dos projetos. (LERMEN, 2012).

Então, devido ao fato que o pesquisador deste trabalho é responsável pela área de desenvolvimento de *software* da STI, um estudo correlacionado a teoria sobre métodos, práticas e técnicas ágeis de desenvolvimento de *software*, alinhado com o estudo de indicadores de desempenho e qualidade que possam auxiliar a gestão do processo de desenvolvimento de *software* afim de sinalizar a melhoria contínua do processo, deve produzir contribuições nos times de Customizações e P&D, da STI, fazendo com que uma metodologia de desenvolvimento de *software* possa ser estabelecida e monitorada.

Este trabalho está dividido em 4 capítulos sendo que o capítulo 2 será de fundamentação teórica seguido do capítulo 3 que apresentará a metodologia de pesquisa. Ainda no capítulo 3, teremos a coleta e análise dos dados da empresa STI. No capítulo 4, Será apresentado a proposta estudada. Por fim, o capítulo 5 está reservado para as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Para que possa ser estudado o processo de desenvolvimento de *software* da STI, deve-se buscar na teoria conceitos que auxiliarão a identificação das etapas realizadas pelos times de Customização e P&D da STI, bem como aspectos na cultura organizacional que afetem a busca da melhoria contínua do processo de desenvolvimento de *software* da empresa. Então na primeira parte abordado conceitos sobre gestão do conhecimento e cultura organizacional afim de identificar pontos importantes que possam impactar a adoção de novas metodologias, práticas e processos pela empresa e seus times.

Na segunda parte buscará o embasamento de engenharia de *software* e os processos padrões de desenvolvimento de *software* afim de entender o processo utilizado pela STI.

Por último serão abordadas conceitos sobre metodologias ágeis de desenvolvimento de *software*, cuja compreensão é necessária para o seguimento do presente estudo, quando buscar-se subsídios para a identificação de melhores práticas a fim de propor melhorias no processo de desenvolvimento de *software* e seus principais indicadores de desempenho e qualidade.

2.1 CULTURA ORGANIZACIONAL

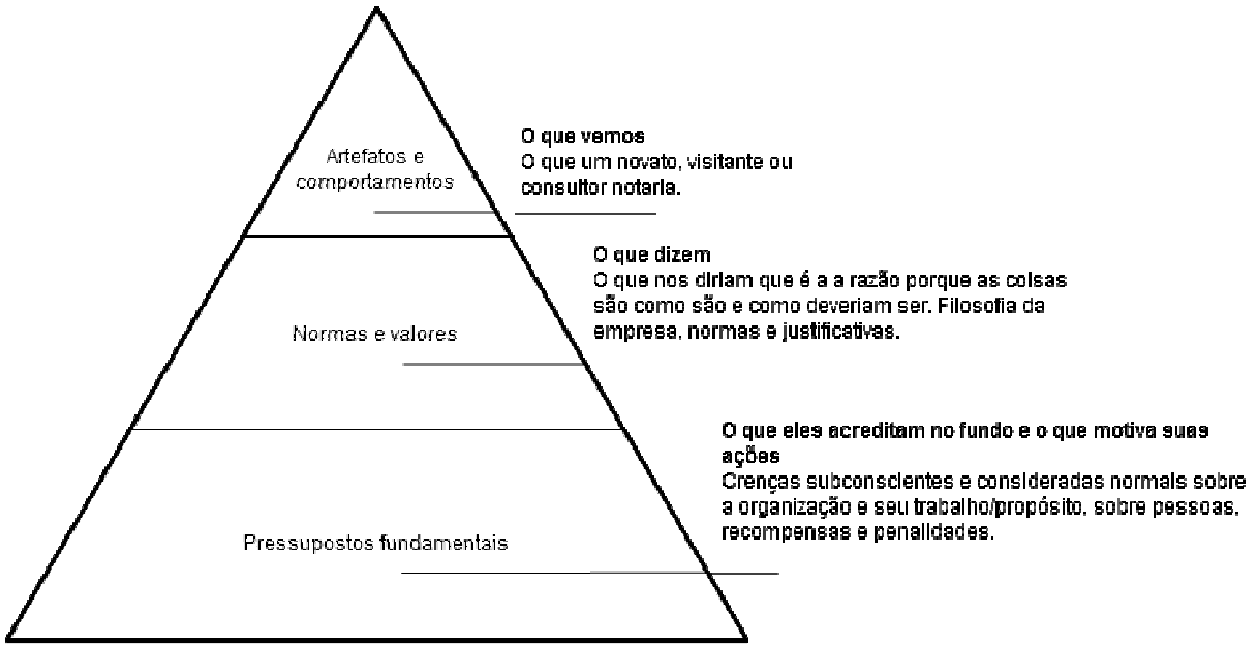
É difícil identificar a cultura das organizações porque devemos decifrar o que se passa na cabeça das pessoas. Para isso definir a palavra “cultura” é importante e segundo Housser e Liker (2009, p. 35) refere Schein[2] cultura é:

> *[...] O padrão de pressupostos básicos inventado, descoberto ou desenvolvido por dado grupo para aprender a lidar com seus problemas de adaptação externa e integração interna, e que funcionou bem o suficiente para ser considerado válido e, portanto, para ser ensinado aos novos membros como o modo correto de perceber, pensar e sentir-se em relação àqueles problemas.*

Angeloni et al. (2008), aponta como a cultura sendo algo que estaria permanentemente sendo restruturada, pois os indivíduos de uma organização constantemente estariam inventando, descobrindo e desenvolvendo formas para se

adaptarem a problemas externos ou internos perpetuando essas formas pelo grupo. A cultura poderia ser representada por um *iceberb*, com três níveis (Figura 1): Artefatos e comportamentos, Normas e valores e Pressupostos fundamentais (HOUSSER; LIKER, 2009, p. 36).

Figura 1: Os três níveis da cultura



Adaptado: Housser e Liker (2009, P. 36).

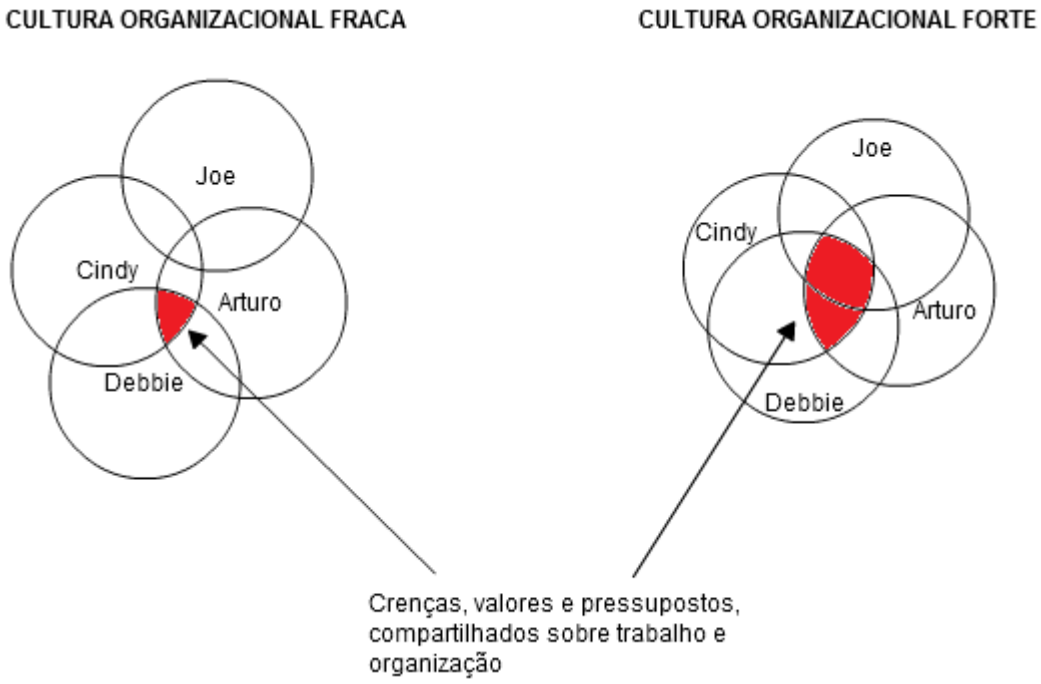
É possível exercer certa influência na cultura através da manipulação e controle sobre variáveis como mitos, normas e tabus, porém o resultado não é certo e muito menos definitivo (ANGELONI et al., 2008). Porém, segundo Angeloni et al.(2008, p.56), se vermos a cultura "como um reflexo da forma pela qual a organização é interpretada pelos seus integrantes" enfatizamos a maneira pela qual as pessoas entendem a organização.

Pensando na cultura organizacional em termos de diagramas de Venn[3] temos uma cultura organizacional fraca quando as crenças, os valores e os pressupostos compartilhados entre os indivíduos no trabalho têm pouco em comum, porém apesar das diferenças pessoais (próprias crenças, valores e pressupostos), em uma cultura organizacional mais coesa, todos indivíduos compartilham algumas crenças, valores

[2] Schein, Edgar. "Coming to a new awareness of organizational culture," *Sloan Management Review,* Winter 1984, Vol. 25, No. 2, p 3-16.

e pressupostos sobre o trabalho. (HOUSSER; LIKER, 2009). Desenhando um diagrama de Venn dos dois cenários descritos teríamos algo como a Figura 2.

Figura 2: Cultura Organizacional



Adaptado: Housser e Liker (2009, P. 39).

Muitos aspectos da cultura organizacional parecem estar localizados no próprio domínio psíquico dos integrantes da organização. Assim mecanismos (Quadro 1) criados pelos indivíduos para lidarem com situações na vida acabam intervindo na formação da cultura organizacional (ANGELONI et al., 2008). O meio (pais, cidade, região e etc.) onde a organização está inserida também afeta a cultura organizacional, ou seja, a aplicação de uma prática advinda de outro país com outra cultura pode ser desastroso (ANGELONI et al., 2008 e HOUSSER;LIKER, 2009).

Quadro 1 - Mecanismos utilizados pelo homem

Repressão	Empurrar impulsos não desejados e ideias para o inconsciente.
Negação	Recusa em admitir um fato, sentimento ou lembrança que evoque um impulso.

[3] Diagrama de Venn: Designa-se por diagramas de Venn os diagramas usados em matemática para simbolizar graficamente propriedades, axiomas e problemas relativos aos conjuntos e sua teoria. Fonte: Wikipédia (http://pt.wikipedia.org/wiki/Diagrama_de_Venn, acessado em 31/07/2013).

Deslocamento	Remessa dos impulsos ligados a uma pessoa ou situação para outro alvo mais seguro.
Fixação	Adesão rígida a uma atitude em particular ou comportamento.
Projeção	Atribuição dos próprios impulsos ou sentimentos a outras pessoas.
Introjeção	Internalização de aspectos do mundo exterior no psiquismo de uma pessoa.
Racionalização	Criação de elaborados esquemas de justificação para disfarçar motivos e intenções subjacentes.
Formação de reação	Conversão de uma atitude ou sentimento em sua forma oposta.
Regressão	Adoção de padrões de comportamento considerados satisfatórios na infância, a fim de reduzir o atual nível de solicitação do ego.
Sublimação	Canalização de impulsos primários para formas sociais mais aceitáveis.
Idealização	Valorização dos aspectos positivos de uma situação para se proteger dos negativos.
Desintegração	Fragmentação dos diferentes elementos da experiência, frequentemente a fim de proteger o bom do mau.

Fonte: Angeloni et al. (2008) p. 64

Para uma mudança na cultura organizacional seja alcançada, de maneira menos traumática possível, a atuação do líder pode ser fundamental redefinindo processos, rotinas e trabalhando nos modelo mentais[4] dos indivíduos (ANGELONI et al. 2008).

[4] SENGE, 1990

2.1.1 *Liderança organizacional*

Segundo o Modelo Toyota[5] os líderes da empresa ensinam a cultura Toyota para todos os seus membros de equipe, assim os líderes são vistos como professores. Assim a parte mais importante do trabalho dos líderes ensinar a novos membros o modelo Toyota para definição, análise, comunicação e solução de problemas. Farias (2013), complementa que o líder é quem se responsabiliza pelo treinamento e desenvolvimento da carreira das pessoas, fornece *feedback* sobre o trabalho, monitora o trabalho de forma sistêmica, porém a questão maior é se o papel de líder está dividido entre todas as pessoas envolvidas ou se está representado por apenas uma pessoa. Senge (2010, p.412) estimula a visão da liderança como:

> "A nova visão da liderança nas organizações que aprendem se centra em tarefas mais sutis e mais importantes. Numa organização que aprende, os líderes são designers, professores e navegadores." Nos dias de hoje, vejo esses papéis fundamentais como mais importantes do que nunca [...]

Já Kanter (1996) apud Angeloni et al. (2008) se refere ao líder como alguém que possa reafirmar que a mudança pode ser conduzida, tendo em sua voz a vontade do grupo, moldando-a para fins construtivos ainda deve ter uma personalidade que possa inspirar os liderados a sentirem autonomia para aumentar e empregar as próprias capacidades.

Porém nos dias de hoje a palavra "líder" refere-se principalmente a uma posição de autoridade, sinônimo para a alta direção (SENGE, 2010). Ainda, Senge (2010) implica que essa confusão, na definição da palavra "líder", declara que líderes são apenas aqueles no topo da hierarquia, ou seja, os demais não são líderes e tem pouco poder de fazer mudanças acontecerem. Porém Farias (2013), lembra que em organizações em que o conhecimento é algo crucial, os times devem se auto organizar e para isso precisam de empoderamento, autorização e confiança da gestão, então, o líder não pode ser visto como alguém que delega responsabilidades e sim como um líder facilitador. Angeloni et al. (2008) e Senge (2010) apontam no sentido que existem diferenças entre o líder na visão tradicional e o líder da era do conhecimento, tais diferenças Angeloni et al. (2008) sintetiza no Quadro 2.

---

[5] LIKER; HOSEUS, 2009. A Cultura Toyota: A alma do modelo Toyota, p. 39.

Quadro 2 - O líder tradicional *versus* o líder da era do conhecimento

Líder tradicional	Líder na era do conhecimento
Apoia-se em regras, normas e procedimentos	Apoia-se nas pessoas, suas capacitações e habilidades
Rotina é uma batalha constante a ser vencida	Rotina é o reinício de novas oportunidades
Distingue suas ações das dos colaboradores, tendo cada um o seu papel	Distingue suas ações pela competência
Comunica o suficiente para manter as coisas funcionando	Debate, pesquisa
Vê, acompanha e controla tudo	Vê, acompanha e controla o que é mais importante
Cultura específica de uma tarefa	Cultura ampla, visando entender e criar alternativas
Delega o que fazer	Delega como fazer
Motivado pelo poder e pelo dinheiro	Motivado pelo desafio da auto realização
Poder baseado no cargo	Poder baseado na competência
Trabalho é simples troca de econômica	Trabalho é um processo de enriquecimento cultural, além de uma troca econômica
Visão de especialista	Visão ampla de generalista

Adaptado: Angeloni et al. (2008 p. 91)

Angeloni et al. (2008), Farias (2013) e Senge (2010) acreditam que os líderes são os agentes de mudança, são pessoas que agem como projetistas, professores e regentes, utilizando habilidades como maestria pessoal, construir uma visão

compartilhada, trazer à tona modelos mentais vigentes e incentivar padrões mais sistêmicos de pensamento, influenciando diretamente a cultura organizacional da empresa. A importância do líder adaptar-se à situação atual, aprendendo a aprender e até mesmo desaprendendo regras que até então eram válidas é fundamental para buscar o comprometimento dos colaboradores, fazendo assim, com que os colaboradores estejam cientes que não serão mais comandados e sim orquestrados (FARIAS, 2013 e ANGELONI et al., 2008). Angeloni et al. (2008) complementa dizendo que “essas características dos líderes somente serão aceitas na organização se a cultura organizacional estiver voltada para a aprendizagem contínua.”.

2.2 CONHECIMENTO ORGANIZACIONAL

A permanência de uma organização no mercado é fundamentalmente ligada a sua capacidade de realizar mudanças para aumentem suas vantagens competitivas, porém a realização destas mudanças é dificultada pela falta de conhecimento da organização sobre como seus processos de negócio são realizados e sua própria estrutura organizacional (CASTRO, 2012). Para Wiig (2011) e Teece (1998) apud Junges (2011) a vantagem competitiva gerada pelas organizações está associada a capacidade e a forma que as mesmas gerenciam seus ativos baseados em conhecimento e mais importante quão efetivos são esses processos para que criem valor e desempenho superior ao longo do tempo.

Para Silva, L. (2011) conhecimento organizacional pode ser definido:

> [...] a capacidade da organização em criar, difundir internamente e incorporar conhecimentos aos produtos, serviços, regras de negócios e sistemas, obtidos pela sinergia entre as pessoas que gravitam em torno da organização e entre estas e a própria organização.

A capacidade de redirecionar processos ajustando-se a cenários de mutação constantes é resultado do processo contínuo de aprendizado organizacional, onde as organizações deveriam estar tornando-se mais do que são no presente (SUZIN, 2010). Para este trabalho conhecimento organizacional será definido como o conhecimento que a organização possui sobre seus processos de negócio, sobre o

relacionamento entre os diversos setores organizacionais, sobre o mercado, tecnologias, clientes e competidores (CASTRO, 2012).

2.2.1 Dado, informação e conhecimento

Carvalho (2012) diz “Dado não é informação; informação não é conhecimento; conhecimento não é dado.”. Essa afirmação, segundo Carvalho (2012), é para lembrar que não raras as vezes usamos o termo dado, informação e conhecimento como sinônimos e reforçando o autor referência Davenport e Prusak[6] sobre o assunto:

> A confusão entre dado, informação e conhecimento – em que diferem e o que significam – gera enormes dispêndios com iniciativas de tecnologia que raramente produzem resultados satisfatórios. [...] Por mais primário que possa soar, é importante frisar que dado, informação e conhecimento não são sinônimos. O sucesso ou o fracasso organizacional pode depender de se saber qual deles precisamos, com qual deles contamos e o que podemos ou não fazer com cada um deles. Entender o que são esses três elementos e como passar de um para outro é essencial para a realização bem sucedida do trabalho ligado ao conhecimento.

O dado é um registro de um evento, onde um evento pode ser registrado de diversas maneiras, gerando, assim diversos dados (CARVALHO, 2012). Ainda Carvalho (2012), sugere um exemplo para ilustrar o entendimento de um dado, ilustrando o evento da compra de uma televisão, que pode gerar vários dados tais como: quando a compra foi feita; em que loja; valor pago; modelo do aparelho; etc. Porém o autor finaliza dizendo que estes dados isolados não representam nada sobre qual razão levou a escolha do aparelho, como foi o atendimento, etc.

Para Nonaka e Takeuchi (2008) informação é um fluxo de mensagens, necessário para criar o conhecimento. Para Carvalho (2012, p.7) informação é “um conjunto de dados com determinado significado para o sistema.”. A informação é especifica a um contexto, ou seja, um conjunto de dados sem a implicação de um sujeito coordenando significativamente não passa de um acúmulo de coisas sem significado (CARVALHO, 2012 e NONAKA; TAKEUCHI, 2008). Angeloni et al. (2008) entende que a informação é um conjunto de dados que devem ser selecionados e agrupados segundo um critério logico para consecução de um objetivo.

---

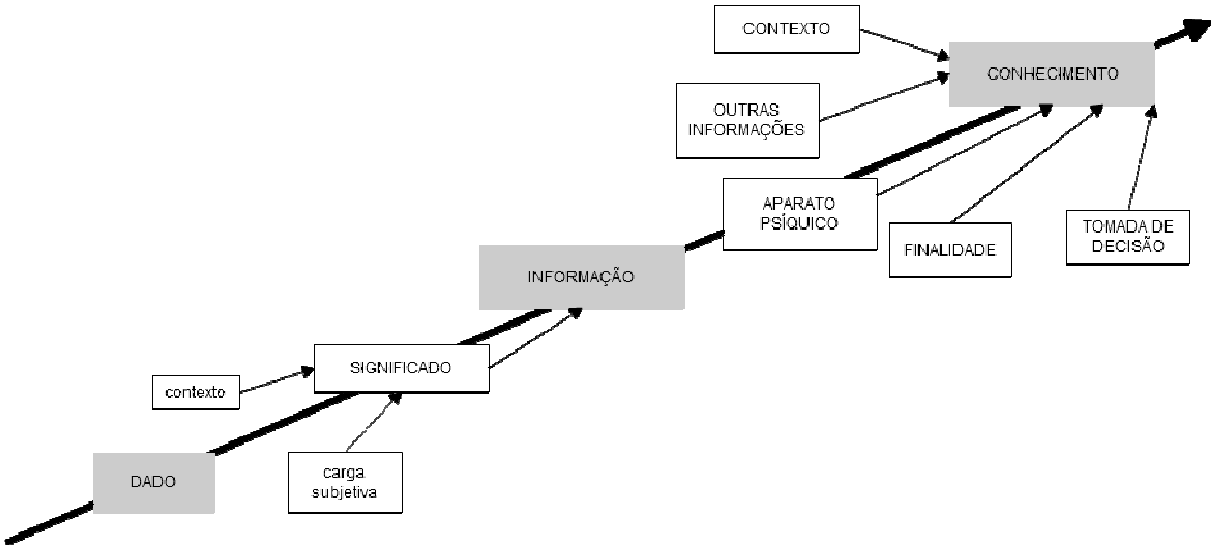
[6] DAVENPORT, Thomas; PRUSAK, Laurence. Conhecimento empresarial: como as organizações gerenciam seu capital intelectual. 15 edição. Rio de Janeiro: Elseiver, 1998.

O conhecimento segundo Davenport (1998) apud Carvalho (2012, p.9) é “a informação, que devidamente tratada, muda o comportamento do sistema. Conhecimento é compreender todas dimensões da realidade, é realizar um agrupamento de informações cognitivamente, empiricamente e emocionalmente (ANGELONI et al. 2008). Nonaka e Takeuchi (2008, p.56), com grifos do original, sintetizam conhecimento como:

> [...] o conhecimento, ao contrário da informação, é sobre *crenças* e *compromisso.* O conhecimento é uma função de uma determinada instância, perspectiva ou intenção. [...] o conhecimento, ao contrário da informação, é sobre *ação*. É sempre conhecimento “para algum fim”. [...] o conhecimento, como a informação, é sobre *significado*. É específico ao contexto e relacional. [...] consideramos o conhecimento como *um processo humano dinâmico de justificação da crença pessoal dirigida à “verdade”.*

Carvalho (2012) sintetiza o desenvolvimento do dado em informação e desta em conhecimento na Figura 3.

Figura 3: Transformação de dado em conhecimento



Adaptado: Carvalho (2012, P. 11).

2.2.2 Duas dimensões do conhecimento

Segundo Nonaka e Takeuchi (2008) podemos identificar, dentro do conhecimento, dois componentes aparentemente opostos: conhecimento tácito e o conhecimento explícito. Conhecimento tácito é o conhecimento implícito, difícil de ser articulado em palavras, consequentemente difícil de ser transmitido (ANGELONI

et al. 2008). Já o conhecimento explícito refere-se ao conhecimento que é transmissível na linguagem formal, sistemática (NONAKA; TAKEUCHI, 2008).

O conhecimento explícito é mensurável, mais racional, pois trata-se de um conhecimento cristalizado, sendo possível transmiti-lo por palavras, números, fórmulas e etc.; pode ser armazenado e transportado através de meios como livros, artigos, banco de dados e etc.; enfim o conhecimento explícito pode ser ministrado em palestras e aulas (CARVALHO, 2012, p.12).

Segundo Carvalho (2012, p. 12), o conhecimento tácito "não é um conhecimento palpável, muito menos explicável. Ele é profundamente pessoal e, por isso, muito mais difícil de ser compartilhado.". O conhecimento tácito está enraizado nas ideias que o indivíduo incorpora, assim como suas ações e experiência, ou seja, intuições e palpites subjetivos (NONAKA; TAKEUCHI, 2008). Para Carvalho (2012) o conhecimento tácito não pode ser ensinado facilmente, pois é fluído e adaptável, por outro lado ele poder ser aprendido a partir de relações pessoais. O Quadro 3 demonstra as diferenças entre conhecimento tácito e explícito.

Quadro 3 - Dois tipos de conhecimento

Conhecimento tácito	**Conhecimento explícito**
Subjetivo	Objetivo
Conhecimento da experiência (corpo)	Conhecimento da racionalidade (mente)
Conhecimento simultâneo (aqui e agora)	Conhecimento sequencial (lá e então)
Conhecimento análogo (prática)	Conhecimento digital (teoria)

Adaptado: Nonaka e Takeuchi (2008 p. 58)

Para Nonaka e Takeuchi (2008) o conhecimento tácito inclui elementos técnicos e cognitivos. Elementos técnicos, segundo Nonaka e Takeuchi (2008), incluem o *know-how*, as habilidades concretas. Carvalho (2012), inclui elementos como *insights,* intuições, palpites e inspirações, os quais são adquiridos por meio da experiência corporal. Carvalho (2012, p. 13), sobre elementos técnicos, cita:

> [...] encontramos as habilidades difíceis de serem discernidas. São técnicas que nós incorporamos inconscientemente graças a nossas experiências. Por isso, podemos considera-las habilidades informais. [...] o futebol pode ilustrar a questão. Consideremos o Pelé. Nenhuma escolinha de futebol no

> mundo vai ensinar você ou seus filhos a terem aquela habilidade magistral com a bola.

O elemento cognitivo centralizam-se em modelos de trabalho do mundo que os seres humanos criam ao fazerem e manipularem analogias em suas mentes, ou seja, modelos mentais como esquemas, paradigmas e pontos de vista que ajudam os seres humanos a perceberem e definirem seu mundo (NONAKA; TAKEUCHI, 2008). Para Carvalho (2012), o elemento cognitivo é constituído pelas nossas crenças, percepções, ideias, valores, emoções e modelos mentais, onde estes elementos comumente os consideramos naturais, ignorando o fato, de que foram adquiridos e moldados ao longo de nossas vidas sendo determinados por nossas escolhas. As diferenças entre os elementos do conhecimento tácito são resumidos no Quadro 4.

Quadro 4 - Duas dimensões do conhecimento tácito

	Conhecimento tácito	
Dimensão	Técnica	Cognitiva
Ponto-chave	*Know-how*	Modelos mentais
Elementos	*Insights*, Intuições, palpites, inspirações, experiências corporais	Esquemas, paradigmas, perspectivas, crenças, valores, emoções, pontos de vista, ideais
Resultado	Habilidades informais	Visão de mundo, “o que é” e “o que deveria ser”

Fonte: Carvalho (2012, p. 14)

2.2.3 Aprendizagem organizacional

Para Nonaka e Takeuchi (2008) conhecimento é criado através da interação entre o conhecimento tácito e o explicito, assim os autores definem quatro modos para conversão do conhecimento (SECI):

- Socialização: Conhecimento tácito para conhecimento tácito
- Externalização: Conhecimento tácito para conhecimento explícito
- Combinação: Conhecimento explícito para conhecimento explícito
- Internalização: Conhecimento explícito para conhecimento tácito.

As relações que um indivíduo estabelece com outro promovem o compartilhamento de experiências, tratando-se assim de uma socialização (CARAVALHO, 2012). Para Nonaka e Takeuchi (2008) e Angeloni et al. (2008) a socialização é o modo onde se cria conhecimento tácito e para tal o indivíduo pode adquirir conhecimento sem usar a linguagem, bastando apenas a observação, imitação e prática. Nonaka e Takeuchi (2008, p.61) dizem que:

> [...]a chave para a aquisição de conhecimento tácito é a experiência. Sem alguma forma de experiência compartilhada, é extremamente difícil que uma pessoa projete-se no processo de raciocínio de outro indivíduo. A mera transferência de informação, frequentemente, tem pouco sentido, se for abstraída das emoções associadas e dos contextos específicos nos quais as experiências estão inseridas.

Para Carvalho (2012, p.18), após produção de um *insight,* por meio da socialização, um grupo de indivíduos comovidos em torno do conhecimento tácito, tendem a realizar conversas, discussões e reflexões levando a uma externalização do conhecimento. Segundo Nonaka e Takeuchi (2008, p. 62) a externalização é quando “conhecimento tácito torna-se explicito, tomando forma de metáforas, analogias, conceitos, hipóteses ou modelos.”.

Segundo Angeloni et al. (2008) e Nonaka e Takeuchi (2008) a combinação é quando indivíduos trocam e combinam o conhecimento explícito através de meios como rede de computadores, reuniões, documentos e conversas telefônicas. Carvalho (2012, p.18) define o processo de combinação, como o seguinte:

> Uma vez que um grupo de indivíduos explicitou o conhecimento por meio de um novo conceito, cabe à organização disponibilizar esse conhecimento explícito de mo9do que todos os demais grupos sejam capazes de fazer a *combinação* desse conhecimento explícito com outros que já existem em seu ambiente interno e externo. Assim, eles poderão combinar os conjuntos de conhecimentos explícitos e sistematizar cada conceito em um sistema de conhecimento.

A internalização está ligada ao “aprender fazendo”, isto é, quando as experiências são internalizadas nas bases do conhecimento tácito do indivíduo, na

forma de modelos mentais ou *know-how* (NONAKA; TAKEUCHI, 2008). Para Carvalho (2012, p. 19) o processo que viabiliza a disseminação do conhecimento dentro da organização é a combinação, mas para que isso aconteça com sucesso é preciso que haja a internalização do conhecimento, ou seja, a organização deve processar o conhecimento explícito e, por outro lado, capacitar o indivíduo a não só assimilar esse conhecimento, mas também incorpora-lo a seu conhecimento tácito.

Para Nonaka e Takeuchi (2008) o conhecimento tácito individual é a base do conhecimento organizacional, com isso, a organização deve mobilizar este conhecimento individual para que possa organizacionalmente amplificar o conhecimento através do modelo SECI. Carvalho (2012) ressalta que este movimento não é em linha reta nem em círculo, mas em espiral para tanto referência Nonaka e Tayama[7] (2008):

> É importante observar que o movimento através dos quatro modos de conversão do conhecimento forma uma espiral, e não um círculo. Na espiral da criação do conhecimento, a interação entre o conhecimento tácito e o conhecimento explícito é amplificada por meio de quatro modos de conversão do conhecimento. A espiral torna-se maior em escala à medida que sobe para os níveis ontológicos.

Para Angeloni et al. (2008, p.141) a aprendizagem organizacional surge com uma forma de propiciar aos indivíduos e a organização maneiras de aprender e reaprender, de acordo com o caos ou a estabilidade que se formam no momento, alavancando o conhecimento.

2.3 ENGENHARIA DE *SOFTWARE*

Inúmeras pessoas escrevem programas de *software,* seja para uso profissional, pessoal ou apenas por hobby, porém quando se trata de programas de *software* para uso profissional, estes normalmente são criados por equipes e devem ser mantidos e alterados durante a sua vida (SOMMERVILLE, 2011, p. 3). Mediante a isto, o autor, cita “A Engenharia de *Software* tem por objetivo apoiar o desenvolvimento profissional de *software, mais do que a programação individual”.*

Segundo, Boehm (2006), na década de 50 e 60 muitas organizações adotavam a abordagem “*Code and Fix”* no desenvolvimento de *software*, isto é, que

o processo era realizar o desenvolvimento e ajustar o que deu errado. Porém, segundo o autor, esta abordagem era cara para as organizações então em 1968 o termo Engenharia de *Software* (ES) foi concebido, a fim de expressar o desejo de transformar o desenvolvimento de software em uma disciplina da engenharia, como engenharia elétrica ou mecânica (SCHNEIDER, 2009).

De acordo com IEEE Standard 610.12 (1990), ES pode ser definida como uma aplicação de uma abordagem sistemática, disciplinada e quantificável para desenvolvimento, operação e manutenção de *software*. Sommerville (2011), simplifica a ES, para uma disciplina da engenharia cujo foco está em todos os aspectos da produção de *software,* desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo usado. Já para Pressmann (2001), a ES é uma tecnologia em camadas (Figura 4), as quais são: ferramentas, métodos, processos e qualidade.

Figura 4: Engenharia de software em camadas.

Fonte: Pressman (2001, P. 17).

Segundo, Pressman (2001, p. 17) “qualquer abordagem de engenharia (inclusive a engenharia de *software*) deve se apoiar num compromisso organizacional com a qualidade.”. A camada de qualidade serve como sustentação para as demais camadas, possibilitando assim um desenvolvimento de *software* de maior qualidade (NOVELLO, 2006).

Como ferramentas para ES, Novello, 2006, p.27 entende:

> As ferramentas da ES proporcionam o apoio automatizado ou semi-automatizado aos métodos. As ferramentas que apoiam os diversos métodos podem ser integradas de forma que a informação criada por uma ferramenta possa ser utilizada por outra, sendo estabelecido um sistema de suporte ao desenvolvimento de software chamado engenharia de software auxiliada por computador (CASE – *Computer-Aided Software Engineering*).

---

[7] NONAKA, Ikujiro; TOYAMA, Royoko. Criação do conhecimento como processo sintetizador. Porto Alegre: Bookman, 2008.

A camada de métodos devem proporcionar os detalhes de como desenvolver um *software* (PRESSMAN, 2001), oferecendo um amplo conjunto de tarefas, tais como: análise de requisitos, planejamento e estimativa do projeto, projeto da estrutura de dados, arquitetura do sistema, algoritmos de processamento, codificação, testes entre outros (PRESSMAN, 2001 apud NOVELLO, 2006).

Segundo Novello (2006, p.27), “os processos constituem o elo de ligação entre as ferramentas e os métodos[...]”. Não distante, Pressman (2001, p. 17) diz:

> O alicerce da engenharia de *software* é a camada de *processo.* O processo de engenharia de *software* é o adesivo que mantém unidas as camadas de tecnologia e permite o desenvolvimento racional e oportuno de *software*s de computador. [...] Os processos de *software* formam a base para o controle gerencial de projetos de *software* e estabelecem o contexto no qual os métodos técnicos são aplicados, os produtos de trabalho (modelos, documentos, dados, relatórios, formulários, etc.) são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são adequadamente geridas.

Uma abordagem sistemática e organizada do trabalho é adotada por engenheiros de *software*, afim de produzir *software* de alta qualidade. Assim ES, tem tudo a ver com a seleção do método mais adequado para o desenvolvimento de um produto de *software* conforme um conjunto de circunstâncias (SOMMERVILLE, 2011). Ainda o autor cita dois motivos importantes para o uso da ES:

- Cada vez mais, indivíduos e sociedades dependem dos sistemas de *software* avançados. Temos de ser capazes de produzir sistemas confiáveis econômica e rapidamente.

- Geralmente é mais barato, a longo prazo, usar métodos e técnicas da engenharia de *software* para sistemas de *software*, em vez de simplesmente escrever os programas como se fossem algum projeto pessoal. Para a maioria dos sistemas, a maior parte do custo é mudar o *software* depois que ele começa a ser usado.

2.4 MODELOS DE PROCESSO DE *SOFTWARE*

Utilizar uma abordagem sistemática, tal qual a ES utiliza, é às vezes, chamada processo de *software* (SOMMERVILLE, 2011). A busca de estratégias de que considerem processos, métodos e ferramentas para resolução de problemas no desenvolvimento de *software* tem sido o objetivo das organizações (NOVELLO,

2006). Estas estratégias são chamadas de modelos de processo de *software*, porém devemos considerar estes modelos como abstrações que podem ser usadas para explicar diferentes abordagens de desenvolvimento de *software* (SOMMERVILLE, 2011).

Para realizar o desenvolvimento de um produto de *software*, segue-se um processo, que de forma simplificada é um conjunto de atividades e resultados relacionados que conduzem à produção de um *software* (JACOBSON; BOOCH, 1999). Segundo, Rosioto (2008), processo de *software* é o conjunto de atividades parcialmente ordenadas que um projeto deve seguir para desempenhar alguma tarefa. Esta tarefa deve ter como objetivo atingir uma determinada meta e geralmente está associada ao desenvolvimento de um ou mais produtos.

Segundo, Sommerville (2011, p. 18), existem vários processos de *software diferentes*, porém existem algumas atividades são fundamentais para a ES:

1. *Especificação de software*: Atividades de Engenharia de Requisitos para definir as funcionalidades e restrições do *software.*
2. *Projeto e implementação de software:* Atividades para projetar o *software* e produzi-lo conforme especificações.
3. *Validação de software:* Atividades que vão garantir que o *software* atende as demandas do cliente.
4. *Evolução do software:* Atividades que garantem a evolução do *software* confirme a mudança nas necessidades dos clientes.

Neste Capítulo serão abordados processos de *software* que são dirigidos a planos, como Sommerville, (2011, p.19) define: “processos dirigidos a planos são aqueles em que todas as atividades são planejadas com antecedência, e o progresso é avaliado por comparação ao planejamento inicial.”. Exemplos de processos dirigidos a planos, segundo Sommerville (2011) e Pressman (2001) são: Modelo cascata, Modelo Espiral, *Rational Unified Process (RUP)* e Desenvolvimento baseado em componentes.

No Capítulo 2.5 serão abordados processos ágeis que segundo Pressman (2001), tem uma abordagem mais informal, onde o planejamento é gradativo, e é mais fácil alterar o processo conforme mudanças do cliente (SOMMERVILLE, 2011). O foco não está em encontrar o modelo de processo ‘ideal’ de *software*, para a organização e sim a melhoria e adaptação de modelos existentes criando processos

de ES mais específicos (SOMMERVILLE, 2011). Exemplos de métodos ágeis, segundo Farias (2013) são: *Scrum, Extreme Programming (XP), Crystal Clear e Feature Driven Development (FDD)*.

A diferença entre processos ágeis e dirigidos a planos, de acordo com Farias (2013, p.7) é:

> *Métodos ágeis são adaptativos ao invés de prescritivos, por isso, incentivam a melhoria contínua (implicando em um constante estado de mudanças e transformação, visando alcançar um estado melhor) através de ciclos inspeção e adaptação. Esse é o motivo pelo qual métodos ágeis utilizam processos empíricos, em vez de prescritivos. Enquanto processos empíricos são apropriados para domínios instáveis e com alto nível de mudanças, processos prescritivos são indicados para atividades ordenadas que podem ser alcançadas através de uma sequência de passos.*

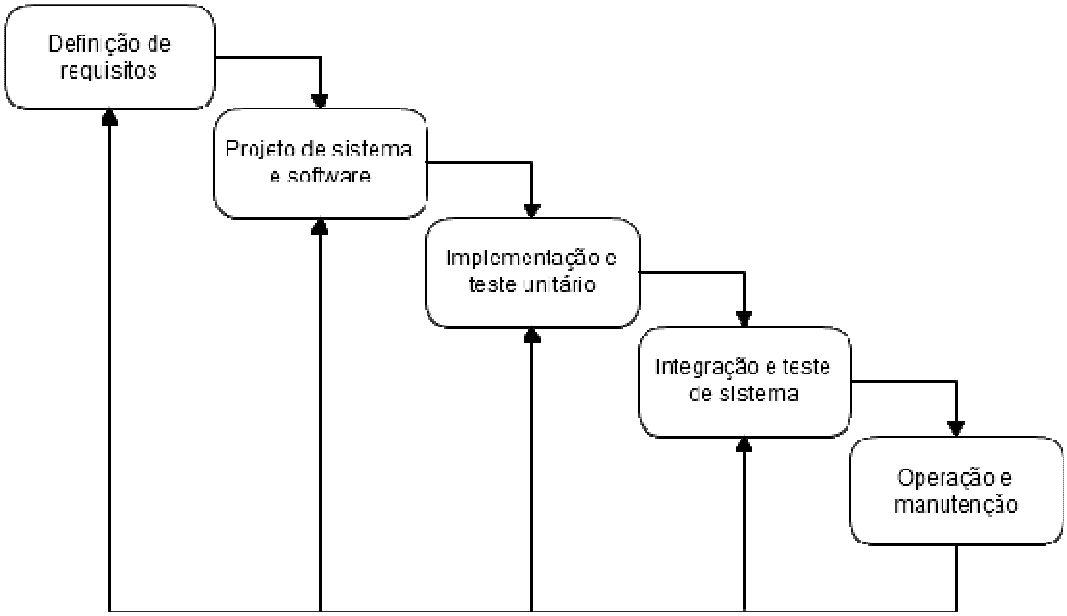
2.4.1 Modelo em cascata

Dentre os processos para desenvolvimento de *softwares* mais antigos e mais utilizado, destaca-se, o processo em cascata, termo originado por W.W.Royce em 1970 (FARIAS, 2013). Este processo costuma ser realizado através de fases, sendo que, uma fase inicia somente quando a anterior termina.

O modelo em cascata foi definido em uma época onde o custo de realizar alterações no produto de *software* era muito alto, devido a dificuldades no acesso a computadores e ferramentas de apoio ao desenvolvimento serem rudimentares (SOARES, 2004). Devido a estes fatores o *software* era todo planejado e documentado antes de ser implementado, ou seja, cada fase de desenvolvimento tem associada ao seu término uma documentação que deve ser aprovada para que a próxima fase inicie. De forma geral, fazem parte do modelo clássico as fases de análise de requisitos, projeto, implementação, testes, integração e manutenção (FARIAS, 2013 e SOARES, 2004).

Para Sommerville (2011, p. 20), as atividades fundamentais (Figura 5) do desenvolvimento no modelo cascata são: Análise e definição de requisitos; Projeto de sistema e software; Implementação e teste unitário; Integração e teste de sistema; Operação e manutenção

Figura 5: O Modelo em cascata.



Adaptado: Sommerville (2011, P. 20).

Este modelo foi amplamente utilizado por organizações de desenvolvimento de *software* até o início da década de 90 (SOARES, 2004). Não distante, Sommerville (2011), diz que pela semelhança com outros projetos da engenharia, o modelo em cascata, oferece um modelo de gerenciamento comum para todo o projeto tornando-o comumente utilizado.

O princípio da utilização do modelo em cascata é planejamento e programação de todas as atividades do processo, antes de começar trabalhar nelas (SOMMERVILLE, 2011). Requisitos de um sistema bem compreendidos e razoavelmente estáveis são características para a adoção do modelo em cascata (PRESSMAN, 2001 e SOARES, 2004). Sommerville (2011), explica que, no modelo em cascata, a inflexibilidade do processo é premissa que os compromissos, mediante ao produto de *software*, devem ser assumidos no início do processo, dificultando as mudanças de requisitos no produto de *software* desenvolvido.

A eficácia do modelo em cascata, é questionada quando o *software* a ser desenvolvido é um produto que atende ambientes de negócios dinâmicos, os quais se caracterizam pela dificuldade em prever o futuro, gerando assim muita incerteza quanto aos requisitos (CONFORTO; AMARAL, 2007). Assim, assumindo que a incerteza, gera mudança nos requisitos de *software* e estas mudanças são comuns (SOARES, 2004), Schawaber (2007) apud Capareto (2012) cita:

> [..] em um projeto típico, 50% do tempo é gasto antes de se construir qualquer funcionalidade para o *software* nas definições dos requisitos, a arquitetura e as especificações, sendo que 35% dos requisitos mudam no decorrer do projeto e 65% das funcionalidades descritas nunca ou raramente serão utilizadas pelas pessoas.

Devido a inflexibilidade do modelo em cascata, Sommerville (2011, p. 21) diz: “Para a maioria dos sistemas, esse processo não oferece custo-benefício significativo sobre outras abordagens, para o desenvolvimento de sistemas.”. Pressman (2001) colabora dizendo que o modelo em cascata, exige que o cliente tenha paciência, pois uma versão do produto de *software* desejado não vai ficar disponível até o final do projeto, ou seja, Pressman (2001, p. 39) resume dizendo: “um erro grosseiro pode ser desastroso se não for detectado até que o programa executável seja revisto.”.

2.4.2 Modelo incremental

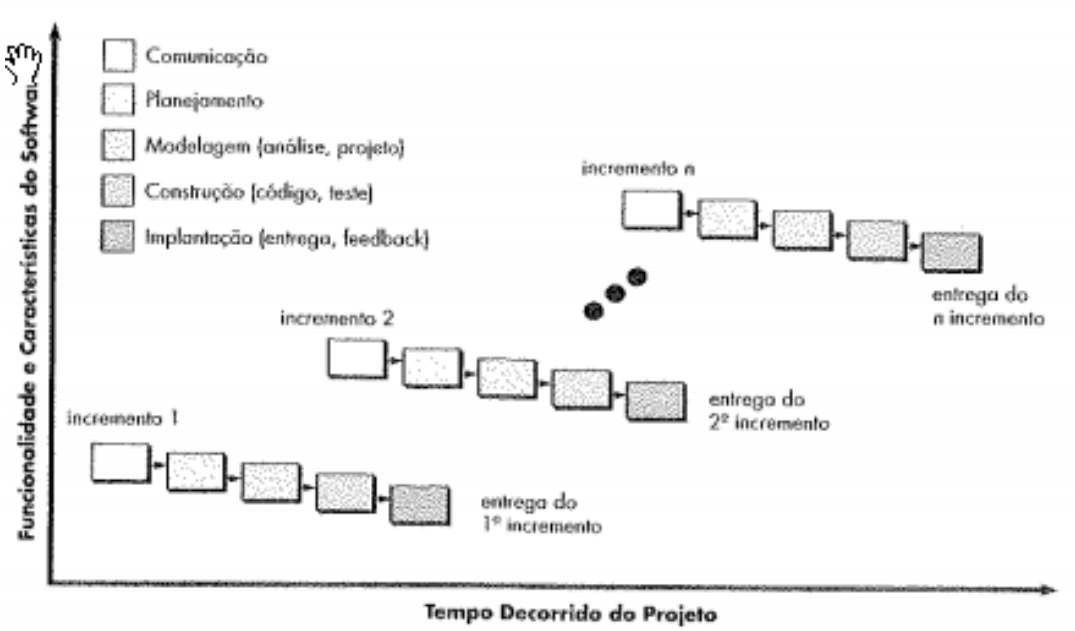
O modelo em cascata trabalha na suposição que as coisas são estáveis a longo prazo, porém tal suposição é falha pois “modificação” é uma constante em projetos de *software* (LARMAN, 2007). Sommerville (2011), complementa dizendo que o desenvolvimento incremental de *software* é melhor que a abordagem do modelo em cascata pois reflete a maneira como resolvemos os problemas, geralmente movendo-se passo a passo em direção a uma solução. Não distante, Pressman (2001) ressalta que requisitos de negócio e do produto mudam frequentemente, prazos reduzidos do mercado tornam impossível o desenvolvimento de um produto de *software* abrangente e nestas situações modelos incrementais de desenvolvimento de *software* auxiliam para evolução de um produto de *software* com o tempo.

A ideia do desenvolvimento de *software* no modelo incremental é desenvolver uma implementação inicial, do produto de *software,* expô-la aos comentários de usuários e através do *feedback* dos mesmos promover a evolução do produto até que um sistema adequado seja desenvolvido (SOMMERVILE, 2011). Para Larman (2007), o modelo incremental envolve a imediata programação e teste de uma parte do produto de *software* repetindo essas atividades em ciclos buscando a realimentação dos requisitos a cada ciclo. Não distante, Pressman (2001, p. 40) diz “O modelo *incremental* combina elementos do modelo em cascata aplicado de

maneira iterativa.”, representando graficamente o modelo através da Figura 6. Não distante, Cunha (2006, p. 25) afirma que:

> Iterações são pequenas etapas do ciclo de vida definidas por sequencias distintas de atividades com planejamento detalhado e critérios de validação, resultando sempre em algum produto, interno ou externo. Como o planejamento das iterações nunca é todo feito no início do projeto os riscos de mau planejamento reduzem-se, pois os elementos são integrados progressivamente.

Figura 6: O modelo incremental



Adaptado: Pressman (2001, P. 40).

No modelo incremental atividades de especificação, desenvolvimento e validação são intercaladas, havendo um rápido *feedback*, entre as atividades (SOMMERVILLE, 2011). Em cada iteração existe um pouco de análise, implementação, testes e implantação, onde requisitos e riscos mais críticos são abordados fazendo com que na próxima iteração novos requisitos serão trabalhados e riscos serão mitigados através de analise, implementação, testes e implantação (MARTINS, 2007). Pressman (2001) complementa dizendo que a cada iteração se apresenta um produto operacional, onde nas primeiras iterações temos uma versão simplificada do *software*, onde o usuário pode trabalhar e avaliar o resultado do *software* entregue. Sommervile (2011) reforça dizendo que cada incremento adiciona ou melhora alguma funcionalidade necessária para o cliente, ou seja, o cliente avalia

o sistema no final de cada iteração verificando se o mesmo oferece o que foi requisitado.

Além de propiciar uma visão melhor do progresso do projeto, desde o início (Larman, 2007) o desenvolvimento de *software* utilizando modelos incrementais oferece as seguintes vantagens:

- Custo de alteração do *software* é relativamente baixo, pois com o *feedback* e envolvimento do usuário a cada iteração as alteração de requisitos são detectadas mais cedo, além de ajudar na mitigação de riscos (SOMMERVILLE, 2011 e LARMAN, 2007).
- Evita “estados de bloqueio” (PRESSMAN, 2001) pois não há necessidade de membros do time aguardem outros membros terminarem tarefas dependentes para só assim ir para próxima fase do ciclo de vida de desenvolvimento de *software* (PRESSMAN, 2001 e LARMAN, 2007).
- Entrega de valor para os clientes antecipada, mesmo sem todo *software* implementado. (SOMMERVILLE, 2011).

Atualmente o desenvolvimento incremental é a abordagem mais comum para desenvolvimento de sistemas, pois permite que possa ser adaptada para ser dirigida a planos (mais prescritiva), mais ágil (menos prescritiva) ou ambos (SOMMERVILLE, 2011). Ainda Sommerville (2011), sugere a mescla de características do modelo incremental com o modelo em cascata para desenvolvimento de sistemas de grande porte, promovendo espaço para melhorias no processo de desenvolvimento de *software*.

2.5 MÉTODOS ÁGEIS DE DESENVOLVIMENTO PARA *SOFTWARE*

Para Farias (2013), Sommerville (2011), Larman (2007), Cockburn (2006) e Beck (2000) métodos ágeis consistem em métodos de desenvolvimento de software que utilizam ciclos curtos de desenvolvimento, chamados de iterações, os quais normalmente possuem duração de poucas semanas garantindo, assim, o feedback constante e reposta rápida às mudanças. Teles (2006, p. 41) afirma que “o desenvolvimento ágil se baseia na premissa de que o cliente aprende ao longo do desenvolvimento, à medida que é capaz de manipular o sistema.”.

Outras características do desenvolvimento ágil são práticas e princípios que refletem leveza, comunicação, equipes auto organizadas, entre outras (Larman, 2007). Sommerville (2011) ressalta que ao envolver os clientes no processo de desenvolvimento, pode-se obter *feedback* rápido sobre a evolução dos requisitos, assim minimiza-se a documentação, uma vez que, a comunicação realizada é mais informal.

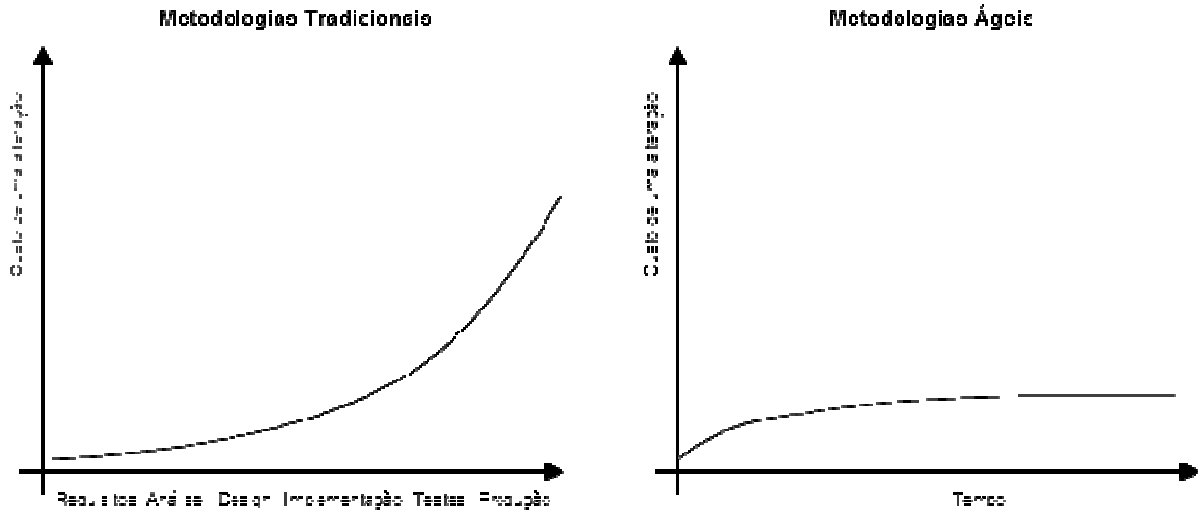
Farias (2013) ressalta que cada iteração do desenvolvimento ágil contém todas as atividades descritas pelas ES, ou seja, cada iteração se realiza um incremento no produto de *software* através de atividades de planejamento, análise, design, codificação, testes e documentação. Ou seja, metodologias ágeis não acrescentam nada de novo em relação a ES, porém a diferença estão nos valores praticados pelas metodologias ágeis, tais como *feedback* constante, foco nas pessoas e entrega de valor (COCKBURN, 2006).

Pelo fato das empresas operarem em um ambiente de mudanças rápidas e neste ambiente é praticamente impossível obter um conjunto completo de requisitos de *software* estável metodologias ágeis de desenvolvimento de *software* são mais adequadas (SOMMERVILLE, 2011). Farias (2013) e Sommerville (2011) colaboram afirmando que métodos tradicionais, como o processo em cascata, baseiam-se em realizar as atividades da ES uma após a outra, uma única vez e em sequência, assim nos métodos tradicionais requisitos de *software* estáveis são a premissa básica para o processo ao contrário das metodologias ágeis.

Para Teles (2006) o cliente não consegue especificar os detalhes do produto de *software* que deseja no começo do projeto porque simplesmente ele não os conhece ainda, assim, alguns aspectos *software* desejado só vão se tornar claros quando o cliente possuir a oportunidade de utilizar o sistema construído. Com isso considera-se que o cliente também está aprendendo sobre o que precisa e cada *feedback* pode mudar os requisitos de *software* que não fazem mais sentido neste novo contexto (FARIAS, 2013). Assim Farias (2013), conclui que métodos ágeis abraçam a imprevisibilidade natural, no desenvolvimento de *software*, através de seus ciclos de *feedback* constante, os quais permitem que o cliente e o time de desenvolvimento possam adaptar-se às mudanças.

Teles (2006) demonstra o custo de realizar uma alteração no produto de *software* no decorrer do projeto, utilizando métodos tradicionais e utilizando metodologias ágeis (Figura 7). O fato de metodologias tradicionais necessitarem de um planejamento prévio de tudo que pode acontecer no desenvolvimento do produto de *software*, assim, o custo de uma alteração no decorrer do projeto aumenta na proporção que o projeto chega ao final (SOARES, 2004). Por outro lado metodologias ágeis são mais adaptativas, ou seja, incentivam a melhoria continua através de ciclos de inspeção e adaptação, aceitando as mudanças naturais no decorrer do projeto de um produto de *software* (FARIAS, 2013).

Figura 7: Custo de alteração no *software* ao decorrer do tempo



Adaptado: Teles (2006, P. 36-43).

Segundo Sommerville (2011) o ponto não é decidir qual metodologia é melhor, dirigida a planos (tradicional) ou ágil. O melhor caminho seria uma abordagem híbrida das metodologias (SOMMERVILLE, 2011). Colaborando com essa afirmação, Farias (2013, p. 6) diz que o desenvolvimento ágil não é a única forma de encarar o desenvolvimento de *software*, muito menos é a mais eficiente. Frederick Brooks[8] (1996) complementa afirmando que não existe nenhuma técnica de gestão ou tecnologia que resolva todos os problemas de todos os contextos, resumindo essa ideia ele diz: “não há bala de prata”.

Desenvolvimento de *software* baseado em métodos ágeis tem se tornado muito popular, grandes *players* do mercado vem utilizando: Google, Yahoo,

Symantec, Microsoft (SHORE; WARDEN, 2007). Com isso várias metodologias estão disponíveis para o desenvolvimento ágil de *software,* algumas delas são: *Extreme Programming* (XP), *Scrum, Crystal, Feature-Driven Development (*FDD*), Lean Software Development (*LSD*), Dynamic Systems Development Methodology (*DSDM), *Test Driven Development* (TDD), *Agile Unified Process* (AUP) e Kanban (SILVA, F., 2009).

Devido ao grande número de referências a processos leves (métodos ágeis de desenvolvimento de *software*), em fevereiro de 2001 um grupo de profissionais extraordinários do desenvolvimento de *software* reuniu-se em um *Resort* de Ski em Wasatch Range para discutir melhores maneiras de desenvolver *software*, assim nasceu o manifesto ágil, uma declaração com os princípios que regem o desenvolvimento ágil (FARIAS, 2013).

2.5.1 Manifesto ágil

Segundo Sommerville (2011) a filosofia por trás dos métodos ágeis é refletida no manifesto ágil. Os profissionais que deram origem ao manifesto ágil[9] foram: Kent Beck, Mike Bedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland e Dave Thomas. O manifesto ágil (Quadro 5) contém 4 valores considerados fundamentais para o desenvolvimento ágil de *software*.

Quadro 5 - Manifesto Ágil

O Manifesto Ágil
Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazê-lo. Através desse trabalho, passamos a valorizar: • **Indivíduos e a interação entre elas** mais que processos e ferramentas;

[8] Brooks, Frederick P. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley Professional, 1996

[9] http://agilemanifesto.org, acesso em 15 de maio de 2013.

- **Software em funcionamento** mais que documentação abrangente;
- **Colaboração com o cliente** mais que negociação contratual;
- **Responder a mudanças** mais que seguir um plano.

Mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

Adaptado: Farias (2013, p. 3)

Segundo Farias (2013) o primeiro valor, “indivíduos e a interação entre eles mais que processos e ferramentas”, se refere ao fato de uma equipe é formada por pessoas, e cada uma é diferente e única, possuindo pontos fortes e fracos, assim elas não podem ser vista como recursos homogêneos e substituíveis. Segundo Cohn (2005), pessoas estão em primeiro lugar pois excelentes ferramentas e processos sem pessoas excelentes envolvidas, muito provavelmente, produzirão um resultado medíocre em vez de excelente.

O segundo valor, “*software* em funcionamento mais que documentação abrangente”, significa entregar um produto de *software* estável a cada iteração, assim é possível coletar *feedback* tanto do produto como o processo (COHN, 2005). Ao invés de passar meses produzindo documentação, comum em metodologias tradicionais e que não agrega muito valor, a natureza iterativa dos métodos ágeis permite a entrega de valor para os clientes através de *software* funcional e é claro uma vez que a documentação seja importante para o projeto ela poderá ser produzida a cada entrega (FARIAS, 2013).

Segundo Cohn (2005) e Farias (2013) contratos são necessários, mas os seus termos e detalhes podem exercer grande influência se as partes estarão mais colaborativas ou mais competitivas. Assim o terceiro valor, “colaboração com o cliente é mais valorizada do que negociação contratual” quer dizer que metodologias ágeis tem foco em entregar produtos que agreguem valor, para isso é preciso sempre estar pronto para adaptar-se às mudanças que afetam o escopo do produto e para isso a colaboração do cliente é fundamental, além de um contrato que mantenha o máximo de opções abertas (FARIAS, 2013). Ainda segundo Farias (2013) métodos ágeis procuram trazer o cliente para perto da equipe, ou seja, o cliente faz parte do projeto e tem um papel importante para que o projeto seja bem sucedido.

O último valor, “responder a mudanças é mais importante do que seguir um plano”, remete ao fato dos times ágeis focarem na entrega do máximo de valor para clientes e usuários do projeto (COHN, 2005). Segundo Farias (2013, p.9), “planejar é preciso, mas planos não precisam ser “escritos em pedras”, eles podem ser apagados, corrigidos e refeitos.”.

Farias (2013), Cohn (2005), Soares (2004) e Silva, F. (2009) ressaltam que metodologias ágeis não ignoram totalmente os processos, ferramentas, documentação, negociação de contratos ou planejamento, porém elas partem do princípio que para obter *software* em funcionamento que agregue valor, indivíduos e interações, *software* funcional, colaboração com cliente e resposta rápida a mudanças devem ser mais priorizados. Assim Farias (2013, p.8) resume: “[...] para todos os valores do manifesto ágil: o elemento da esquerda é mais importante do que o elemento da direita, porém o da direita também é importante e relevante.”

Além dos quatro valores, o manifesto ágil é composto por 12 princípios (BECK et al., 2001).

- Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de *software* com valor agregado.
- Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
- Entregar frequentemente *software* funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
- Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
- *Software* funcionando é a medida primária de progresso.
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.

- Contínua atenção à excelência técnica e bom design aumenta a agilidade.
- Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial.
- As melhores arquiteturas, requisitos e designs emergem de equipes auto organizáveis.
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Segundo Mallmann (2011, p.21), “com base nos doze princípios cada um dos métodos ágeis apresenta um conjunto de atividades a serem adotadas durante o processo de desenvolvimento do sistema.”. A seguir serão descritos três métodos, *Scrum, XP* e *Kanban*, que se baseiam em valores e princípios ágeis.

2.5.2 *Scrum*

Segundo Marçal (2009), o *Scrum* é um método ágil que aceita a imprevisibilidade do desenvolvimento de *software* e foi criado em 1996 por Ken Schwaber e Jeff Sutherland. Farias (2013), Sommerville (2011) e Marçal (2009) concordam que o Scrum é método ágil com foco nos aspectos gerenciais do desenvolvimento de *software*. Para Mallmann (2011) o *Scrum* tem como foco encontrar uma forma de trabalho para os membros do time para que possam produzir *software* de forma flexível e em um ambiente em constante mudança. Marçal (2009, p. 39) complementa dizendo: “é uma abordagem empírica baseada na flexibilidade, adaptabilidade e produtividade em que a escolha das técnicas de desenvolvimento fica a cargo do time.”.

De acordo com Schwaber (2004) ao invés de realizar um planejamento detalhado prescritivo do projeto, o *Scrum*, assume um modelo empírico de controle de processos para garantir a visibilidade, inspeção e adaptação do planejamento do projeto. O *Scrum* tem como princípios: times pequenos (em torno de sete pessoas), requisitos poucos estáveis ou desconhecidos, iterações curtas, com entrega do produto para o cliente (MARÇAL, 2009, MALLMANN, 2011, COHN, 2005 e SILVA, F., 2009).

Para o *Scrum* cada iteração é chamada de *Sprint*, onde estes intervalos costumam ter um mês, podendo variar de poucos dias a algumas semanas. (FARIAS, 2013 e SILVA, F. 2009). Dentro de um time de desenvolvimento, no *Scrum*, existe três papeis principais: o *Product Owner* (dono do produto), o time e o *Scrum Master* (FARIAS, 2013, SCHWABER, 2004 e MARÇAL, 2009).

O *Product Owner* é aquele que teve ou que representa quem teve a necessidade do produto e assim tem a visão do produto, podendo ser o próprio cliente ou alguém que represente os desejos do cliente (FARIAS, 2013). Ou seja, ele representa os interesses de todos no projeto, definindo e mantendo um artefato chamado de *product backlog* que contém uma lista de funcionalidades priorizada. Atraves do *product backlog* o *Product Owner* garante que as funcionalidades de maior valor sejam construídas prioritariamente (FARIAS, 2013, MARÇAL, 2009 e SCHWABER, 2004). Farias (2013), também, define como responsabilidade do *product owner*, que qualquer informação de negócio que for necessária para o time, seja repassada para assim o time possa transformar as ideias, do *product owner*, em *software.*

Para Farias (2013) o time deve ser composto de pessoas dos mais variados perfis, como testadores, desenvolvedores, designers, analistas de negócios etc., ou seja, o time deve ser *cross-funcional*. A responsabilidade do time é desenvolver as funcionalidades do produto, ou seja, transformar o *product backlog* em incremento de funcionalidades a cada *Sprint* (MARÇAL, 2009 e SCHWABER, 2004). Marçal (2009) ressalta que o time deve ser responsável por gerenciar seu próprio trabalho e coletivamente pelo sucesso da *Sprint* e consequentemente pelo projeto como um todo.

O *Scrum Master* deve ensinar o *Scrum* a todos os envolvidos no projeto e implementar o *Scrum* de modo que esteja adequado à cultura da organização (MARÇAL, 2009). Farias (2013), define o *Scrum Master* como um facilitador, ou seja, é responsável em remover os impedimentos do time, ajudando a garantir assim o objetivo do *Sprint* seja atingido. Schwaber (2004), afirma que o *Scrum Master* deve manter o processo em funcionamento, assegurando que todas as regras e práticas necessárias estão sendo aplicadas, ou seja, ser um líder que proponha a colaboração entre os membros do time extraindo o que cada um pode dar de melhor. Farias (2013, p. 13) ressalta dizendo:

> É importante ressaltar que o Scrum Master não atua como um gerente ou chefe da equipe porque ela é auto organizável. O Scrum Master não determina o que cada membro da equipe deve ou não fazer: a equipe se compromete com a entrega das funcionalidades e, então, se auto organiza, definindo por quem e em qual momento as tarefas serão realizadas.

De acordo com Schwaber (2004), os artefatos utilizados pelo *Scrum* no seu fluxo de desenvolvimento são: *Product Backlog, Sprint BackLog* e incremento de funcionalidade do produto.

O *Product Backlog*, como citado anteriormente, é uma lista de funcionalidades (requisitos funcionais e não funcionais) priorizadas do sistema/produto que está sendo desenvolvido (MARÇAL, 2009). Schwaber (2004) salienta, que o *Product Backlog*, nunca está completo e irá evoluir conforme a evolução do produto e o ambiente ao qual está inserido.

Já o *Sprint Backlog* corresponde a lista de funcionalidades que o time se compromete em entregar ao final da *Sprint* corrente (MALLMANN, 2011). Através de uma reunião de planejamento (*Sprint Planning)* o time e o *Product Owner* selecionam, a partir do *Product Backlog*, quais funcionalidades serão implementadas (SCHWABER, 2004). Ao final da *Sprint* o time deverá entregar as funcionalidades, selecionadas no *Sprint Backlog*, testadas e funcionais, caracterizando assim um incremento de funcionalidade do produto (SCHWABER, 2004).

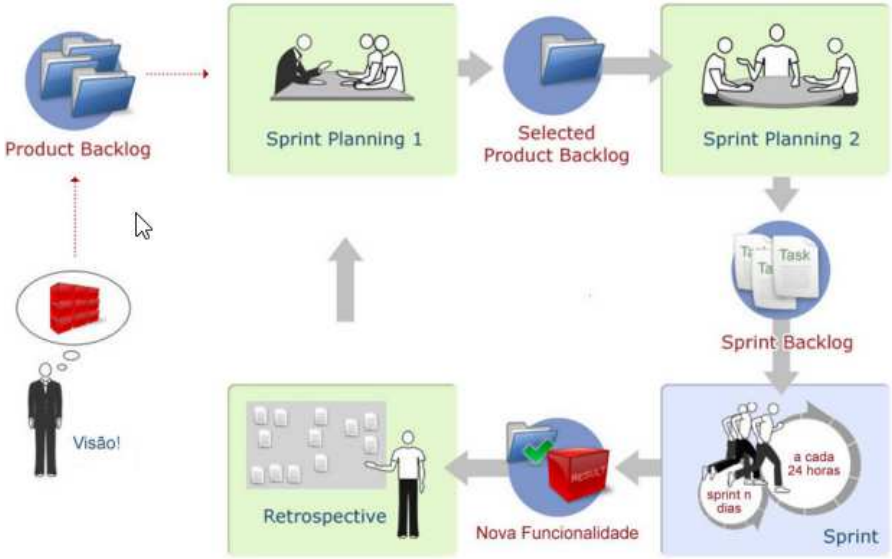
Segundo Larman (2004) apud Marçal (2009, p. 41) o *Scrum* possui um ciclo de vida composto de quatro fases: Planejamento, Preparação, Desenvolvimento e Entrega. Conforme Marçal (2009) a fase de planejamento tem como objetivo estabelecer a visão do projeto, enquanto a fase de preparação identifica requisitos e iniciais alocando-os no *product backlog* devidamente priorizados a cada *Sprint*. Na fase de desenvolvimento o time realiza a implementação para a fase de entrega realizar a entrega incremental de partes do produto funcionando.

O fluxo de desenvolvimento de um projeto, utilizando *Scrum*, inicia-se com o desenvolvimento da visão do produto, que deve conter as principais características do produto, restrições e premissas (SCHWABER, 2004). Após a visão do produto, uma lista de requisitos conhecidos é criada, tal lista consiste na versão inicial do *product backlog* que então é priorizado e divido em *releases* (MARÇAL, 2009).

O ciclo de desenvolvimento do produto (Figura 8) é realizado dentro de *Sprint's* e segundo Schwaber (2004) antes de iniciar o desenvolvimento é realizado uma reunião de planejamento (*Sprint Planning Meeting*), onde o *product owner* e o

time decidem em conjunto o que deverá ser implementado (*Selected Product Backlog*). Tal reunião de planejamento, é dividida em duas partes, a primeira com enfoque mais estratégico afim de definir o que será feito, estimar tamanho do que será feito e esclarecer dúvidas com o *product owner* (FARIAS, 2013). Segundo, Farias (2013) e Marçal (2009) nesta primeira parte é importante considerar a capacidade de produção do time (velocidade do time) para alocar os itens que serão desenvolvidos no *Sprint*. A segunda parte da reunião possui um enfoque mais tático, procurando decidir como as tarefas serão feitas (FARIAS, 2013). Farias (2013), explica que nesta parte de reunião as tarefas devem ser analisadas mais detalhadamente onde é possível tomar diversas decisões técnicas e de negócios.

Figura 8: Ciclo de desenvolvimento *Scrum*



Fonte: Marçal (2009, P. 43).

Farias (2013) explica que ao final da reunião de planejamento o time começa a trabalhar nas tarefas, respeitando as prioridades que foram estabelecidas, ou seja, fazendo as tarefas mais importantes que irão agregar mais valor ao produto, de acordo com o *product owner*. Durante a execução de *sprints*, o time deve fazer uma reunião diária (*Daily Scrum Meeting*) para que se possa conversar sobre o andamento da *Sprint* (FARIAS, 2013, MARÇAL, 2009 e SCHWABER, 2004). Nesta reunião, normalmente rápida e feita em pé (*stand-up meeting*), todos os membros do time devem compartilhar informações que descrevam o progresso desde a última reunião, problemas encontrados e o planejamento futuro (SOMMERVILLE, 2011). Para facilitar o compartilhamento das informações, citadas anteriormente, Schwaber (2004) sugere três perguntas, que cada membro do time deve responder, são elas:

- O que eu fiz desde a última reunião?
- O que irei fazer até a próxima reunião?
- O que está impedindo de realizar meu trabalho?

Ottinger[10] apud Farias (2013) sugere outras perguntas que adicionam maior foco na entrega, aprendizagem e colaboração:

- Que tarefas você ajudou a terminar desde a última reunião?
- Que tarefa você ajudará a terminar hoje?
- Como o resto do time pode ajudar a concluir uma tarefa?
- O que você aprendeu de novo?

Ao final do *Sprint* duas reuniões acontecem, uma reunião de revisão (*Sprint Review Meeting*) e uma reunião de retrospectiva (*Sprint Retrospective Meeting*) (FARIAS, 2013, MARÇAL, 2009 e SCHWABER, 2004). A reunião de revisão tem como objetivo a apresentação do trabalho desenvolvido para o *Product Owner*, para que ele possa oferecer *feedback* e aprovar ou não tudo o que foi produzido (FARIAS, 2013). Na reunião de retrospectiva o time busca discutir sobre lições aprendidas e melhorias que podem ser aplicadas no processo (FARIAS, 2013). Farias (2013) ressalta, que nenhum processo é perfeito e por essa razão é necessário trabalhar continuamente na melhoria e adaptação do processo, neste contexto as reuniões de retrospectiva são uma excelente forma de melhoria contínua.

Durante cada *Sprint,* o time deve buscar se auto organizar, mas para isso precisam de empoderamento, autorização e confiança da gestão (FARIAS, 2013). Segundo, Sommerville (2011) a ideia por trás do *Scrum* é que toda a equipe tem poder para tomar decisões, ou seja, não é necessário um “gerente de projetos” para gerenciar o time. Farias (2013) afirma que para que ajudar o time a manter um ritmo sustentável todas cerimonias devem ter um tempo definido (*timebox*) e este deve ser respeitado pelo time.

Farias (2013) e Sommerville (2011) citam algumas vantagens na utilização da metodologia *Scrum*:

- Visão sistêmica de todo o time;
- Comunicação entre o time é melhorada;

[10] Ottinger, Tim. *Continuous Improvement In a Flash: A Guide For Scrum Masters.* LeanPub.

- Através da entrega incremental de funcionalidades o *feedback* do cliente é melhorado;
- A confiança entre o time e o cliente fica fortalecida, criando-se uma cultura positiva, na qual todos esperam e trabalham juntos para o sucesso do projeto.

Times grandes ou times que estão localizados em locais distribuídos geograficamente possuem problemas ao utilizar a metodologia *Scrum*, necessitando uma adaptação do processo (FARIAS, 2013 e SOMMERVILLE, 2011).

2.5.3 *Extreme Programming* - XP

O *Extreme Programming* (XP) é um processo de desenvolvimento de *software* baseado em valores e princípios do manifesto ágil, onde se busca assegurar que o cliente receba o máximo de valor de cada dia de trabalho do time de desenvolvimento (TELES, 2006). Segundo, Farias (2013), o XP é um dos métodos ágeis que melhor cobre aspectos técnicos do desenvolvimento de *software*, tais como codificação, *design* e testes. Para Sommerville (2011) o XP é o mais conhecido e o mais utilizado dos métodos ágeis, pois foi desenvolvido para impulsionar práticas reconhecidamente boas a níveis “extremos”. Beck (2000) resume o XP como uma metodologia ágil para pequenas e médias equipes que desenvolvem *software* baseado em requisitos vagos e que se modificam rapidamente.

Para assegurar que o cliente sempre receba um alto retorno do investimento realizado no *software*, o XP, é organizado em torno de um conjunto de valores e práticas que atuam de forma harmônica e coesa (TELES, 2006). Teles (2006, p.23) cita os quatro valores fundamentais do XP:

- *Feedback*: “Quando o cliente aprende com o sistema que utiliza e re-avalia as suas necessidades, ele gera *feedback* para a equipe desenvolvimento.”
- Comunicação: “A comunicação entre o cliente e a equipe permite que todos os detalhes do projeto sejam tratados com a atenção e a agilidade que merecem.”

- Simplicidade: “Implementar apenas aquilo que é suficiente para atender a cada necessidade do cliente.”
- Coragem: “A equipe precisa ser corajosa e acreditar que, utilizando as práticas e valores do XP, será capaz de fazer o *software* evoluir com segurança e agilidade.”

Beck (2000), sugere doze práticas que norteiam o processo XP: Cliente presente; Planejamento; Programação em pares; Testes; Refatoração; Propriedade coletiva; Código padrão; *Design* simples; Metáfora; Ritmo sustentável; Integração contínua e Entregas frequentes.

Soares (2004), explica que a prática do Cliente presente deve colocar o cliente como parte integrante do time de desenvolvimento, estando sempre disponível para sanar dúvidas de requisitos. Teles (2006) colabora afirmando que a presença do cliente viabiliza a simplicidade do processo, especialmente na comunicação.

A prática de Planejamento, segundo Beck (2000), consiste em assegurar que a equipe esteja trabalhando naquilo que é mais importante para o cliente e decidindo em adiar o escopo futuro. Para auxiliar o planejamento, o XP, é divido em *releases* e iterações, onde *releases* são módulos que geram um valor bem definido para o cliente e iterações são períodos de tempo de poucas semanas onde o time implementa um conjunto de funcionalidades (TELES, 2006). Para definir e aprender mais sobre as funcionalidades de cada iteração é realizada uma reunião chamada de “jogo do planejamento” (*Planning Game*), onde o cliente escreve “histórias de usuário” em cartões, a fim de representar a necessidade da funcionalidade a ser desenvolvida, e após explica para o time de desenvolvedores tudo o que for necessário para implementar tal história (FARIAS, 2013).

A implementação do código, segue a prática Programação em pares, que define que toda implementação deve ser feita em dupla, ou seja, dois desenvolvedores trabalham em um único computador (SOARES, 2004). Tal prática permite que o código seja revisado permanentemente enquanto é construído, da mesma forma que contribui para que o código fique mais simples e eficaz (TELES, 2006). Segundo, Farias (2013), a programação em par, traz muitos benéficos entre eles: o produto é entregue com menos defeitos, aumento de produtividade e compartilhamento do conhecimento (eliminação de ilhas de conhecimento).

Segundo, Farias (2013, p. 15) testar é uma atividade de extrema importância para garantir a qualidade do *software* e não é algo opcional com XP. A prática de Testes, do XP, é a forma que asseguramos que o *software* está correto, portanto, todo código deve possuir testes de unidade e estes testes devem ser executados com sucesso antes que a entrega seja feita (BECK, 2000, FARIAS, 2013 e TELES, 2006). Segundo Farias (2013), devido a importância dos testes, o XP, utiliza a técnica de escrever os testes antes do código de produção, tal técnica é conhecida como Desenvolvimento Guiado por Testes ou *Test Driven Development* (TDD).

Para Teles (2006), o time deve fazer com que o código escrito esteja sempre claro e fácil de compreender, auxiliando a evolução do sistema incremental mente. Para atingir esse objetivo, a prática de Refatoração (*Refactoring*), indica que o time possa modificar partes do sistema que estejam funcionando para facilitar sua manutenção (TELES, 2006). Soares (2004), ressalta, que o *refactoring*, deve ser feito apenas quando necessário, ou seja, apenas quando for percebido que é possível simplificar o código atual sem que o mesmo perca sua funcionalidade.

Qualquer pessoa que percebe que pode adicionar valor ao código, mesmo que ele próprio não tenha desenvolvido, pode fazê-lo, esta é a base da prática Propriedade coletiva (SOARES, 2004). Para Teles (2006) esta prática, aliada a prática de *refactoring* e testes, fornece mais um mecanismo que garante a revisão, verificação e simplicidade do código. Para que exista a prática de propriedade coletiva, a prática de Código padrão é fundamental, ou seja, o time deve estabelecer padrões de codificação (SOARES, 2004 e TELES, 2006).

Segundo Beck (2000), a prática de *Design* simples, significa que o time ao invés de criar generalizações dentro do código, de modo a prepara-lo, para possível necessidade no futuro, o time, opta por um código mais simples, que seja suficiente para atender às necessidades atuais. Farias (2013), ressalta que essa prática do XP diz basicamente que devemos fazer as coisas somente no momento que precisarmos delas, o que na verdade é a base do princípio YAGNI (*You aren't gonna need it*). A prática da Metáfora, é utilizada para facilitar a criação de um *design* simples, ou seja, como as metáforas tem poder de transmitir ideias complexas de forma simples o time pode-se utilizar dessa prática para formar uma linguagem comum entre cliente e time (TELES, 2006).

A qualidade dos desenvolvedores e a capacidade que eles tem de se manter atentos, criativos e dispostos a solucionar problemas está ligado diretamente com a qualidade do código e sistema sendo desenvolvido (TELES, 2006). O XP, assume a prática do Ritmo sustentável para garantir que o time tenha sempre o máximo de rendimento, tal prática consiste em assumir que o time não irá fazer horas extras constantemente (TELES, 2006). Soares (2004), afirma que caso o time tenha que trabalhar mais de 40 horas pela segunda semana consecutiva, existe um sério problema e tal problema não será resolvido com aumento de horas trabalhadas e sim com melhor planejamento.

Integração contínua é a pratica de interagir e integrar várias partes do sistema de *software* várias vezes por dia (SOARES, 2004). Teles (2006) ressalta que uma nova funcionalidade recém incorporada no sistema, pode afetar outras que já estavam implementadas, então para assegurar que todo o sistema esteja sempre funcionando de forma harmoniosa, o time deve utilizar a prática de integração continua.

Por fim a prática de Entregas frequentes tem como objetivo gerar um fluxo continuo de valor para o cliente (TELES, 2006). Beck (2000) afirma que para entregar valor frequentemente o time precisa trabalhar com *releases* curtas, produzindo um conjunto reduzido de funcionalidades que agregam maior valor para o *software* em questão.

Segundo, Silva F. (2009) as práticas do XP se relacionam entre si e a combinação destas práticas possibilita que o time poupe esforços com a concepção do sistema, mantendo a flexibilidade diante de mudanças nos requisitos. Pressman (2001), sintetiza os valores e práticas do XP através da Figura 9.

Figura 9: O Processo XP

Fonte: Pressman (2001, P. 88).

2.5.4 Métricas ágeis

Para ajudar um time a compreender onde estão e principalmente para ajuda-los a comparar com o estado em que querem estar, métricas podem ser úteis (FARIAS, 2013). Rebelo (2011) afirma que em projetos tradicionais o foco se concentra em quatro métricas (escopo, prazo, custos e qualidade) porém em um projeto que utiliza práticas e filosofia ágil a equipe aparece como um indicador de peso e ganha destaque sobre os demais. Segundo McHugh (2012) e Farias (2013) ao focar métricas individuais pode-se desencorajar a colaboração dentro do time, prejudicando o propósito da métrica. McHugh (2012) cita dois problemas que podem afetar as métricas o efeito observador[11] (*observer effect*) e o efeito luz do poste[12] (*Streetlight effect*).

Um time de desenvolvimento de *software* pode ter vários objetivos (aumento na qualidade, aumento na produtividade, produzir melhores estimativas, aumento na satisfação do usuário etc.), onde cada objetivo deve ser acompanhado por no mínimo um indicador indutor e um indicador de resultado (FARIAS, 2013). Para Farias (2013, p. 98):

[11] *Observer effect (physics)*: http://en.wikipedia.org/wiki/Observer_effect_%28physics%29

> [...] quando um indicador indutor muda, significa que você pode estar no caminho para atingir um objetivo. Por exemplo, aumentar a cobertura de testes ou a quantidade de testes criados pode indicar mais qualidade no produto. [...] já indicadores de resultado são métricas que verificam que você atingiu seu objetivo depois de ter completado o trabalho. Por exemplo, reduzir o número de defeitos reportados por clientes aponta que a qualidade do produto de fato melhorou.

Rebelo (2011) e Farias (2013) sugerem a utilização de *burndown charts* para que o time possa ter *feedback* de como está o andamento da iteração em relação ao planejado. Farias (2013) descreve na Figura 10 o exemplo de *burndown* onde o time não conseguiu entregar tudo que foi planejado.

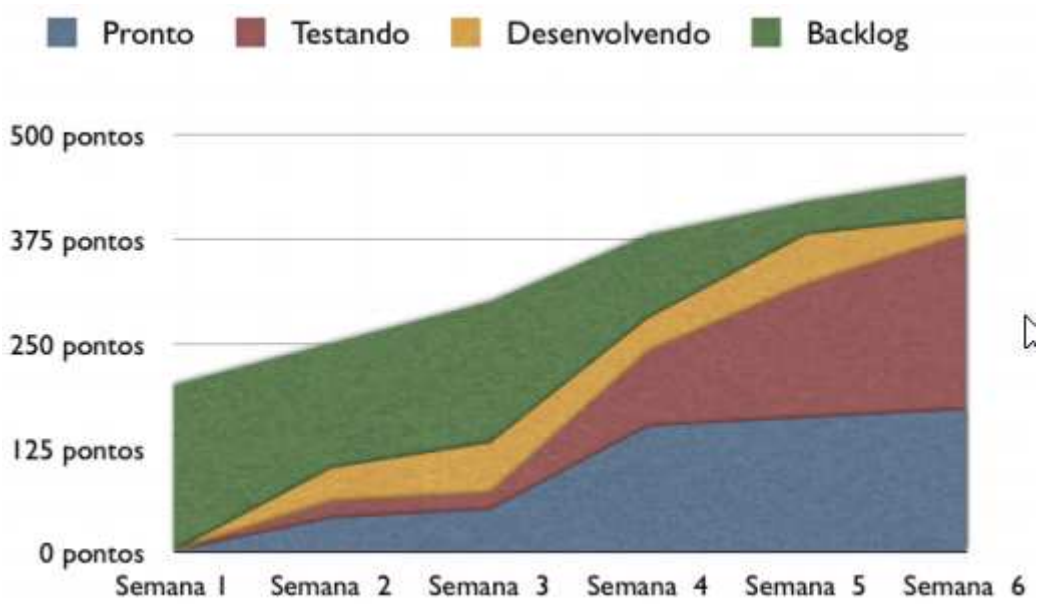
Figura 10: *Burndown chart*

Fonte: Farias (2013, P. 95).

Para acompanhar o escopo do projeto, visualizando a variação do trabalho em progresso ao longo do tempo, pode-se utilizar um diagrama de fluxo cumulativo (*Cumulative Flow Diagram*), exemplificado na Figura 11 (FARIAS, 2013 e REBELO, 2011).

[12] *Streetlight effect:* http://en.wikipedia.org/wiki/Streetlight_effect

Figura 11: *Cumulative Flow Diagram*



Fonte: Farias (2013, P. 97).

McHugh (2012), sugere as seguintes métricas que trazem benefício ao produto, sem estimular comportamentos que prejudiquem o time, consequentemente o projeto:

- Valor entregue: Essa métrica visa medir o impacto do que foi entregue e não o desempenho do time.
- Entregas na data: Ao final de um *realese* conseguir visualizar um percentual de funcionalidades (histórias) que foram entregues dentro do planejamento de cada iteração.

Já Farias (2013), cita alguns indicadores comuns em ambientes ágeis, porém ressalta que o uso dos indicadores deve fazer sentido no contexto da organização:

- Velocidade do time: quantidade de pontos de história que o time consegue entregar por iteração;
- Aceleração: Alteração da velocidade do time, ao longo das iterações;
- *Lead Time*: O tempo entre o pedido do cliente e a entrega;
- *Cycle Time*: O tempo total entre o início e o fim da produção das funcionalidades;
- Tempo de vida: Quantidade de tempo em que uma história está no *backlog*;
- Quantidade de histórias impedidas: Histórias que estão impedidas por falta de informação ou recursos;

- Taxa de Defeitos por história: Defeitos encontrados por história;
- Cobertura de Testes: Percentual de cobertura de testes de unidade;
- Quantidade de Cenários de testes: Contagem dos cenários de testes disponíveis;
- *Throughput* (Produção): Quantidade de histórias entregues por período de tempo;
- *Happiness Index* do Time: Indica a motivação do time;
- *Work in Progress* (WIP): Quantidade de histórias começadas e não entregues.

3 MÉTODOS E PROCEDIMENTOS

Este capítulo descreve o método de pesquisa utilizado no trabalho, a unidade-caso, técnicas de análise de dados e limitações do método de pesquisa utilizado.

Os dados a serem coletados para este estudo de caso, serão buscados no processo de desenvolvimento de *software* utilizado atualmente pela empresa estudada.

3.1 DELINEAMENTO DA PESQUISA

Este trabalho foi realizado por meio de uma pesquisa de campo de caráter qualitativo na qual o pesquisador faz observações sobre o fenômeno estudado e uma análise intuitiva sobe os dados obtidos (YIN, 2010).

Segundo Yin (2010), o método de estudo de caso é recomendado em pesquisas onde o tipo de questão é da forma “como?” e “por quê?”; quando o controle que o pesquisador tem sobre os eventos é muito reduzido; ou quando os fenômenos estudados são atuais.

Desta forma, considerando a pergunta de pesquisa formulada e em concordância com Yin (2010), o estudo de caso é a metodologia escolhida de pesquisa, pois o fenômeno estudado categoriza-se como contemporâneo dentro do contexto da vida real além da capacidade de lidar com uma ampla variedade de evidências – documentos, artefatos, entrevistas e observações, além daquelas que podem estar disponíveis no estudo histórico convencional.

3.2 DEFINIÇÃO DA UNIDADE DE ANÁLISE

O estudo de caso proposto é o processo de desenvolvimento de *software* adotado pela empresa STI, no ano de 2013. Como o pesquisador é funcionário da empresa e responsável pela área de desenvolvimento de *software* percebeu-se uma oportunidade, através deste trabalho, de pesquisar melhorias no processo de desenvolvimento de *software,* bem como na gestão do processo. A escolha pela pesquisa em metodologias mais adaptativas ocorreu pelo alinhamento com os objetivos da empresa de entregar *software* constantemente que gere valor para os clientes e alinhamento com os princípios ágeis.

3.3 TÉCNICAS DE COLETA DE DADOS

Para coleta de dados serão utilizados as seguintes técnicas: análise de documentos internos, observação participantes e entrevistas semiestruturadas.

Os documentos internos, de origem técnica, tem origem do setor de Customizações e P&D. Será utilizado como fonte de dados o sistema de gerenciamento interno de demandas MYSUITE, fornecido pela empresa Brazip. O MYSUITE é utilizado por ambos os setores afim de organizar parte do escopo de trabalho. Ainda serão utilizados documentos do planejamento interno do setor de P&D, que são: Manual integração do setor P&D e Planilhas internas para controle de projetos.

A observação participante, segundo Yin (2010), é uma modalidade especial de observação, onde você não é apenas um observador passivo, em vez disso você pode participar dos eventos sendo estudados. Com isso a observação participante será feita nas reuniões semanais para priorização de projetos e em reuniões do setor P&D de planejamento e estimativa de tarefas relacionadas aos projetos priorizados, durante o período de 2 meses.

Entrevistas semiestruturadas devem ser utilizadas com a finalidade de que através de perguntas abertas, entrevistados tenham a possibilidade de responder sobre o questionado sem a condição de respostas prefixadas pelo pesquisador. O pesquisador prevê realizar entrevistas com os dois integrantes do setor de P&D, quatro integrantes do setor de Customizações, três Diretores e dois Clientes internos, visando obter maiores informações sobre a visão de cada um sobre o

processo de desenvolvimento de *software*, aspectos de liderança e colaboração dos times.

3.4 TÉCNICAS DE ANÁLISE DE DADOS

Como foram utilizadas múltiplas fontes de evidência, a técnica para análise de dados foi baseada na triangulação de dados. Segundo Vergara (2010) a triangulação de dados, fornece novas perspectivas, ou segundo Yin (2010) novas avaliações, do fenômeno estudado.

Através do referencial teórico, os documentos coletados na empresa estudada, as informações obtidas pelas entrevistas realizadas e a observação participante do processo foi possível realizar a triangulação dos dados.

3.5 LIMITAÇÕES DO MÉTODO

Ocorreram dificuldades para conseguir realizar todas as entrevistas necessárias dentro do período planejado. Isto ocorreu devido ao conflito de agendas entre o pesquisador e os entrevistados.

Como os processos realizados na empresa não são documentados, caracterizou-se certa dificuldade de identificar os processos que realmente são executados pelos envolvidos no âmbito desta pesquisa.

Referente a literatura sobre gestão do conhecimento e aprendizagem organizacional, obteve-se certa dificuldade de sintetizar as informações necessárias para o presente trabalho. Isto devido a vasta literatura sobre os assuntos. Por outro, lado, ocorreu certa dificuldade em obter material teórico sobre métricas ágeis, sendo encontrado apenas artigos em sites de portais.

Por fim, este estudo realizado não é algo que possa ser generalizado.

4 APRESENTAÇÃO E ANÁLISE DOS DADOS

Neste capítulo serão apresentados o processo de desenvolvimento de *software* que a empresa STI utiliza no momento, bem como problemas que o processo apresenta. Após serão descritos uma proposta de um novo processo de desenvolvimento de *software* e práticas de gestão do processo.

A análise dos dados através da técnica de triangulação de dados foi utilizada para verificar o processo atual de desenvolvimento de *software* da empresa STI levando a desenvolver uma proposta de um novo processo para o desenvolvimento de *software* e a gestão deste processo.

4.1 PROCESSO ATUAL DE DESENVOLVIMENTO DE *SOFTWARE* NA STI

O processo de desenvolvimento de *software* que a empresa STI utiliza nos times de Customização e P&D não está documentando nos documentos internos utilizados pela empresa. Assim o processo é passado de colaborador para colaborador através das etapas de Socialização e Externalização conforme o modelo SECI de Nonaka e Takeuchi (2008). Neste ponto foi identificado uma falta de padrão do processo, pois segundo entrevistas com os colaboradores, percebeu-se que algumas etapas do processo não eram citadas por todos e outras introduzidas por um único colaborador.

A etapa inicial do processo de desenvolvimento de *software* foi identificada pela maioria dos desenvolvedores como sendo a introdução de uma solicitação na ferramenta de *helpdesk* interno, o *Mysuite*. Foi notado que os desenvolvedores que participam do time de P&D tem uma leve percepção da etapa inicial do processo de desenvolvimento, a justificativa é que algumas funcionalidades que o time desenvolve não estão na ferramenta *MySuite*, mas são identificadas por colaboradores com um alto conhecimento de negócio e repassadas para o time de P&D através de conversa face a face. Após transmitido o desejo este fica internalizado no conhecimento de cada colaborador do time de P&D, não havendo combinação deste conhecimento em outro meio eletrônico ou papel.

A próxima etapa foi identificada como sendo uma análise da solicitação realizada, porém houve algumas visões diferenciadas em relação ao time que realiza essa etapa de análise. Alguns desenvolvedores indicaram que essa etapa é

realizada pelo time de Suporte e outros desenvolvedores indicaram que essa etapa é realizada pelo time de Suporte em conjunto com algum integrante do time de Customização e/ou P&D. Essa diferença pode ser causada pelo motivo que alguns desenvolvedores não ficam envolvidos diretamente com a análise, normalmente desenvolvedores que possuem pouco tempo de empresa, consequentemente pouco conhecimento no negócio da empresa.

Após a solicitação ser analisada os entrevistados indicaram que a etapa de triagem de solicitações acontece. Esta etapa, acontece semanalmente normalmente nas segundas-feiras, e se trata de um processo de priorização de solicitações realizado por algumas pessoas. Neste encontro semanal para realizar esse processo, participam o responsável pelo time de Suporte e o responsável pelo time de Customizações. Porem foi identificado que colaboradores que não participam desse encontro não conseguem explicar o funcionamento desse encontro, ou seja, na ausência dos responsáveis pelas áreas o encontro para realizar a triagem semanal fica a “deriva”. Neste ponto o time de Customizações possui uma lista de demandas a serem trabalhadas na semana, tal lista é quebrada em itens e cada item é alocado em um quadro branco, conforme a data que deverá ser entregue.

O time de P&D recebe algumas demandas desta lista elaborada pelo time de Customizações, porém foi identificado que o time de P&D não está trabalhando em funcionalidades para os produtos, identificadas no *MySuite* e sim em projetos exclusivos para clientes. A escolha de alocar o time de P&D para estes projetos foi a natureza de urgência dos mesmos e a necessidade de uma equipe dedicada, assim o perfil do time de Customizações não se encaixava para realizar tais projetos, ficando a encargo do time de P&D.

O responsável da área de Customizações de posse dos itens mais importantes, ordenados pela data de entrega, aloca qual colaborador poderá desenvolver cada demanda. Neste ponto demandas que devem serem desenvolvidas por um desenvolvedor em especifico são sinalizadas na solicitação dentro do sistema *MySuite*, indicando o responsável pela solicitação.

Foi identificado através das entrevistas e observação que a etapa de desenvolvimento e testes são realizadas pelo desenvolvedores e a homologação final de uma solicitação deve ser realizada pelo time de Suporte, onde tal processo, vale para os dois times, Customização e P&D. Após concluído o desenvolvimento e

testes unitários o desenvolvedor responsável pela demanda altera o status da solicitação, no *MySuite*, alertando o colaborador do time de suporte que abriu a solicitação. Neste ponto em diante o colaborador do time de suporte homologa a solução desenvolvida, realiza a entrega para o cliente e faz o treinamento necessário.

Como citado anteriormente, desenvolvedores do time de P&D recebem algumas funcionalidades para desenvolvimento que não seguem o processo descrito acima. Algumas funcionalidades são levadas para o time através de conversas face a face e pela autonomia dos times tal funcionalidade já entra em produção. Outro ponto importante referente ao processo de desenvolvimento do time de P&D é que não identificado um processo padronizado para priorização de funcionalidades ou um local para manter um *backlog* dos produtos mantidos pelo time de P&D. No caso de demandas dos projetos exclusivos sendo executados pelo time de P&D as mesmas são desenvolvidas e entregue para os clientes pelo próprio time, havendo o registro da demanda, no *MySuite*, realizado apenas depois de entregue.

O processo de liberação de *release* de produtos não possui um mapa de datas previsto, ou seja, a liberação de versões dos produtos ocorre conforme a necessidade dos clientes em receber as correções e novas funcionalidades desenvolvidas nos produtos. Outro ponto levantando é que o desenvolvimento de novas funcionalidades é feito junto com a resolução de defeitos (*bugs*), ou seja, não existe uma diferenciação clara dos itens.

4.2 SITUAÇÕES PROBLEMÁTICAS NO PROCESSO UTILIZADO

Através da análise de dados das entrevistas e observação participante no processo de desenvolvimento de *software* inicialmente foi possível constatar que os times envolvidos no processo não possuíam uma visão sistêmica do processo, afetando a habilidade de colaboração na melhoria do processo.

Durante as etapas iniciais do processo, onde ocorrem a análise e seleção de prioridades foi identificado que este processo possui o envolvimento de poucas pessoas, normalmente as pessoas com mais experiência dos times. Ou seja, quando a demanda chega nas mãos dos desenvolvedores, acredita-se que a demanda esteja analisada e pronta para o desenvolvimento, fazendo com que estes

desenvolvedores não desenvolvam a capacidade de resolução de problemas de forma conjunta. Conforme foi relatado na situação problemática deste trabalho o processo de priorização deveria ser realizado por uma comissão de pessoas de diversos times, mas foi identificado que tal processo não é mais utilizado, sendo substituído pelo processo de triagem realizado pelos responsáveis do time de Suporte e Customização.

Ainda no processo de análise e seleção de prioridades não foi identificado um processo de estimar o trabalho a ser realizado, assim os times acabam alocando o trabalho de forma inconsistente não produzindo um ritmo sustentável. O time de P&D por não possuir um *roadmap* dos produtos bem definido, um *backlog* de funcionalidades e a falta de estimativas perde totalmente a capacidade de planejamento de médio e longo prazo, afetando diretamente o setor Comercial, que não consegue preparar campanhas mais especificas sobre novas funcionalidades ou até mesmo perdendo colocação de novos produtos em clientes. O planejamento do *backlog* semanal, por ter foco na solução de problemas voltados aos times de Suporte e Customização, não envolve pessoas do time de P&D, assim grande parte das demandas são analisadas e resolvidas como uma customização do produto e não como uma correção ou nova funcionalidade para o produto. Ou seja a oportunidade de agregar valor aos produtos é perdida, pois funcionalidades que poderiam ser agregadas ao núcleo dos produtos são realizadas em forma de customizações especificas para os clientes.

Foi identificado várias percepções referente a qual seria o índice de dívida técnica que os times trabalham, mas em geral foi identificado uma média acima de 50% entre os entrevistados. A falta de *feedback* na homologação de funcionalidades entregues ao time de suporte, aliado a falta de padronização de testes são descritos como principais motivos desta percepção de um índice alto.

Devido ao *core business* da STI ser muito especifico, valoriza-se muito o aprendizado do negócio. Assim percebe-se que durante todo o processo de desenvolvimento de *software* cria-se ilhas de conhecimento, prejudicando muito a aprendizagem organizacional. Através da observação do processo de desenvolvimento de *software* no time de P&D foi identificado que a passagem de conhecimento do time de P&D para os outros times é pouca, ou seja, quando uma melhoria no *framework* dos produtos, ou uma nova funcionalidade é desenvolvida

este conhecimento não é externalizado e combinado com outros times. Isso faz com que a liberação de versões dos produtos sejam visto como uma “caixa preta” dificultando o trabalho dos times. Um fator que impacta muito no compartilhamento de conhecimento é a cultura de trabalho individual que a empresa traz desde sua fundação.

Por fim, foi identificado que o processo de desenvolvimento utilizado pela empresa foi sofrendo alterações durante os últimos anos, a fim de, corrigir e melhorar o processo todo. Porém como o processo não possui indicadores que possam auxiliar a visualização dos problemas, as melhorias são realizadas com base na percepção dos colaboradores envolvidos. Além disso não é possível, pela falta de métricas, dizer se tal melhoria realmente impactou positivamente o processo.

4.3 PROPOSTA DE NOVO PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE* BASEADO EM MÉTODOS ÁGEIS.

Para realizar a proposta de um novo processo de desenvolvimento de *software* para os times de Customização e P&D foi levado em conta aspectos culturais da empresa objeto de estudo, aspectos culturais dos times envolvidos e tamanho dos times.

Inicialmente é pretendido a implantação desse novo processo no time de P&D, para posteriormente aplica-lo ou adapta-lo para o time de Customizações. A escolha pelo time de P&D como piloto do novo processo, se deve pelo fato de que as situações problemáticas mais graves afetam diretamente este time e também pelo resultado da análise de dados indicar que o time de Customizações possui um processo de desenvolvimento de *software* mais funcional no momento.

Conforme estudado no referencial teórico, mudanças no processo atual, vindas de forma “*top-down*” não tendem a ser duradouras, assim, o pesquisador deseja atuar nas crenças, valores e pressupostos dos colaboradores para criar uma cultura organizacional forte[13] facilitando a aceitação e padronização de um novo processo de desenvolvimento de *software.*

[13] HOUSSER; LIKER 2009

O pesquisador propõem um projeto inicial para um novo processo de desenvolvimento de *software* com seis fases, onde cada fase tem um período de tempo para ser executada. As fases propostas pelo pesquisador são:

- Preparação das lideranças: Realizar uma conscientização das lideranças sobre a Metodologia Ágil (1 semana);
- Conscientização dos times sobre os problemas encontrados: Criar uma visão compartilhada dos problemas encontrados (1 semana);
- Preparação dos times para adoção de práticas ágeis: Divulgar e ensinar os times sobre vantagens das práticas ágeis (2 semanas);
- Padronização de papeis e artefatos: Padronizar a forma que o *Product Backlog* é armazenado*,* quais estimativas de funcionalidades utilziar e quais papeis das metodologias ágeis utilizar (1 semana);
- Execução de 5 *Sprint's*: Executar com o time de P&D 5 interações onde possam ser executados e melhorados os processos de: planejamento, desenvolvimento, *review* diário, retrospectiva e apresentação do resultado da interação (75 dias);
- Apresentar resultados do novo processo: Através de métricas realizadas apresentar a evolução do processo no decorrer da fase de execução (1 semana);

Conforme citado anteriormente este projeto tem como planejamento ser executado no time de P&D, porém as quatro primeiras etapas podem ser executadas com todos os times da organização pesquisada, pois se tratam de fases que não tendem a alterar processos atuais.

A primeira fase diz respeito a preparar as lideranças da empresa em questão, conscientizando a aplicação de metodologias ágeis, abordando os prós e contras, a fim de solicitar o apoio destes no que tange a mudança da cultura da empresa e dos times envolvidos. O pesquisar, planeja, realizar seminários de 2 horas de duração durante 1 semana, onde participarão membros da diretoria e principais líderes de cada time.

Apresentar para os times as situações problemáticas encontradas no processo de desenvolvimento atual e principalmente o impacto destes problemas nos clientes, assim busca-se criar uma visão compartilhada da situação e começar a criar um conjunto de crenças e valores compartilhado, afim de, aumentar o

engajamento de todos nesta mudança de processo. O pesquisador planeja realizar dois seminários com espaço de 5 dias entre o primeiro e o segundo, onde, no primeiro seminário, o pesquisador, irá apresentar os problemas encontrados através deste trabalho e solicitar que todos reflitam sobre os impactos que estão sendo gerados para o cliente e para os times, para então, em um segundo seminário seja possível debater sobre os problemas, contando com os times para identificar onde os problemas acontecem.

Após os times conscientizados do problema, é de suma importância apresentar aos times o Manifesto Ágil e seu conjunto de valores e princípios. Após a apresentação do manifesto, será dado um prazo de 3 dias para que cada time possa se preparar e identificar qual ou quais princípios do Manifesto Ágil podem ser utilizados pelo time e de que forma. Estas apresentações tem previsão de ocorrer na semana subsequente a apresentação dos problemas do processo atual.

O pesquisador prevê, utilizar uma semana de seminários rápidos para ensinar os times sobre a metodologia *Scrum*, os papéis que a metodologia prevê, as principais práticas da metodologia XP e principalmente assuntos ligados a liderança, aspecto fundamental, segundo Farias (2013) para adoção de metodologias ágeis. Essa semana de apresentações tem foco na preparação de uma base de conhecimentos para a implantação de um novo processo de desenvolvimento de *software* ágil.

Na fase de padronização, o pesquisador, sugere a criação de um comitê ou um grupo de colaboradores que serão responsáveis pela implantação da metodologia ágil. Este comitê seria responsável pela nomeação do *Product Owner, Scrum Master* e servir de apoio ao time, interferindo caso o *Scrum Master*, por exemplo não consiga representar o papel que o cabe. O pesquisar sugere inicialmente que este comitê seja formado pelos responsáveis dos times de Suporte, Customização e P&D. Após o encerramento deste projeto, proposto pelo pesquisador, sugere-se que o papel de *Scrum Master* seja ensinado e alternado entre alguns colaboradores, afim de incentivar que outros colaboradores pratiquem aspectos de liderança característicos do *Scrum Master*. Deste ponto em diante, o pesquisador, sugere que as etapas posteriores sejam aplicadas apenas ao time de P&D, conforme justificado no início do capitulo.

Ainda na fase de padronização, o comitê, irá definir a forma e o local onde registra os desejos de novas funcionalidades, cabendo ao *Scrum Master* e *Product Owner* realizar a manutenção destes artefatos. Sugere-se o uso de histórias, a qual é uma prática apresentada pela metodologia XP (BECK, 2000 e FARIAS 2013). Importante que o time utilize uma forma padronizada para estimar suas histórias, tanto em grandeza como o processo de estimar. Cohn (2005) sugere que para times que estão iniciando em práticas ágeis comecem estimando em dias de trabalho, ou seja, para história deverá ser indicado uma estimativa de quantos dias de trabalho para concluir a história. Também é importante definir a forma que as estimativas serão realizadas, neste caso, pode-se utilizar o *planning poker* sugerido por Beck (2000). Não menos importante, durante a fase de padronização, o time deve buscar o entendimento de quando uma funcionalidade está realmente “pronta”, para isso Farias (2013) sugere que o time escreva junto com a história quais testes deve ser realizados para que a funcionalidade esteja “pronta”.

Durante a fase de execução, o pesquisador, prevê executar algumas práticas ágeis durante cinco interações (*Sprint*), com previsão de 15 dias cada. A primeira prática seria o planejamento da *Sprint* onde o time se reúne como o *Product Owner* e planeja o escopo e objetivo da *Sprint* atual. Durante a execução da *Sprint* é proposto utilizar a prática do *review* diário através de uma *stand-up meeting* para que cada desenvolvedor possa fornecer um rápido *feedback* sobre o seu trabalho. Segundo Audy (2013), em times ágeis que não ficam “blindados” durante uma *Sprint*, ou seja, times que possuem volume alto de troca de prioridades e interrupções por manutenção dos produtos durante o *review* diário pode se perder detalhes do que aconteceu no dia passado. Para isso, Audy (2013) sugere o uso de uma ferramenta visual chamada *Daily Tracking* que proporciona ao time uma forma de rastrear o tempo utilizado durante a *Sprint.* Devido ao fato, que o time de P&D, não possa ficar “blindado” durante a *Sprint* o pesquisador, sugere que o time em questão utilize-se desta ferramenta para monitorar o tempo gasto na *Sprint.*

Ao final de cada *Sprint*, o pesquisador, sugere que o time realize a apresentação do que foi produzido ao *Product Owner* promovendo um *feedback* do trabalho bem como a aceitação ou não das funcionalidades desenvolvidas. Após a apresentação de resultados, o pesquisador, sugere a execução da prática de *Sprint review*, ou seja, uma retrospectiva do que aconteceu de bom e o que pode ser

melhorado durante a próxima *Sprint.* Neste ponto, o pesquisador, sugere que o comitê formado na fase de padronização possa avaliar o *review* e o andamento da *Sprint,* repassando esse aprendizado para os demais times.

Ao final da execução de cinco *Sprint's* o pesquisador, junto com o comitê irá apresentar os resultados obtidos através das seguintes métricas, explicadas no capítulo 2.5.4:

- Entregas na data (MCHUGH, 2012);
- Aceleração do time (FARIAS, 2013);
- *Cycle Time* (FARIAS, 2013);
- Tempo de vida (FARIAS, 2013);
- Quantidade de histórias impedidas (FARIAS, 2013);
- *Happiness Index* do Time (FARIAS, 2013);
- WIP (FARIAS, 2013).

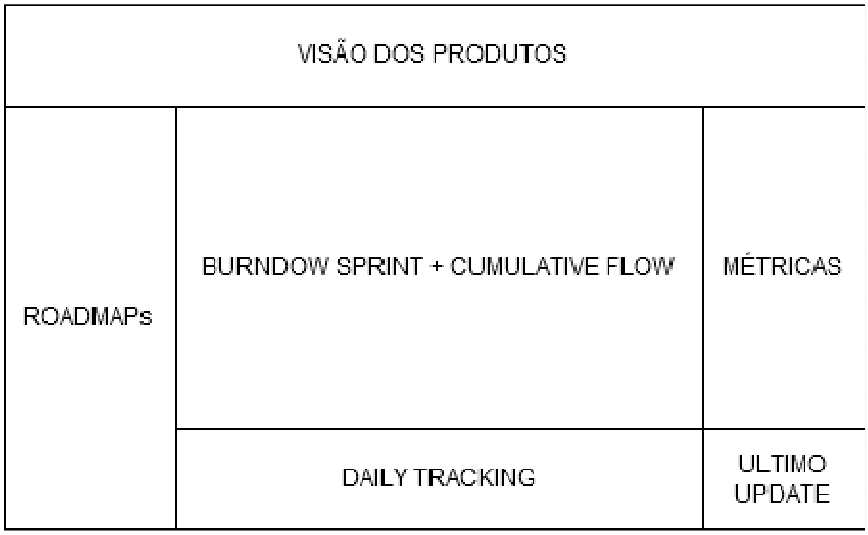
Após a apresentação de resultados para os demais times, o pesquisador, propõem a execução e adaptação do processo criado para o time de Customizações, visando criar uma padronização da metodologia de desenvolvimento utilizada pela empresa STI.

4.4 MONITORAMENTO DO PROCESSO ATRÁVES DA GESTÃO VISUAL

O pesquisador, sugere, que o time de P&D utilize uma ferramenta visual para compartilhar toda informação relevante sobre o processo de desenvolvimento e principalmente sobre o que está acontecendo durante a *Sprint* e com os produtos sendo desenvolvidos*.* Reolon (2013) descreve a ferramenta *Situation wall* onde o time pode compartilhar informações importantes do projeto que estão trabalhando.

Inicialmente sugere-se que o time de P&D possua um quadro branco de no mínimo 2 metros por 1 metro e meio onde será desenhado as separações conforme a Figura 12 ilustra.

Figura 12: Proposta *Situation Wall* para P&D da STI



Adaptado: Rebelo (2013).

A parte superior do *Situation Wall* sugere-se, adicionar a visão estratégica dos produtos que o time de P&D está trabalhando. Segundo Reolon (2013) essa informação seria a resposta da pergunta: “Porque esse projeto existe?”, porém tal resposta tem que ser materializável para que o time possa usar essa informação para tomar uma decisão durante o projeto.

No lado esquerdo do *Situation Wall* o time deve adicionar uma visão geral do *roadmap* previsto dos produtos que o time está trabalhando. Este *roadmap* pode ser em forma de gráfico, ressaltando os módulos a serem desenvolvidos e os que já foram desenvolvidos conforme sugere Reolon (2013).

Destaca-se o lado direito do *Situation Wall,* que possui um espaço para o time adicionar suas métricas, as quais podem ser inicialmente as sugeridas no capítulo 4.3. Reolon (2013) destaca que é importante ressaltar os problemas através destas métricas, alertando o time que, por exemplo sua produtividade não está adequada. Inclusive, Segundo Farias (2013) o time pode adicionar neste espaço novas métricas que acompanhem um objetivo firmando em uma *Sprint Review*.

Ao fim do *Situation Wall* fica localizado uma área para o time atualizar seu *Daily tracking* conforme Audy (2013) sugere e comentando no capítulo 4.3.

No centro do *Situation Wall*, sugere-se, que o time adicione o gráfico de *Burndown* da *Sprint* corrente para todos possam acompanhar a evolução do trabalho no decorrer do tempo. Ao lado do gráfico de *burndown*, o pesquisador sugere

adicionar um gráfico de *Cumulative Flow* para informar ao time o trabalho ao decorrer de todas as *Sprints*.

Por fim no ao canto inferior direito do *Situation Wall* o time deve indicar quando e quem fez a última atualização do *Situation Wall*, garantindo assim que a informação esteja sempre mais recente possível.

5 CONSIDERAÇÕES FINAIS

Através da triangulação de dados, método escolhido para a análise dos dados coletados neste estudo de caso, o pesquisador, conseguiu levantar informações importantes sobre o processo de desenvolvimento de *software* da empresa estudada. Sendo que algumas informações, até o momento, não eram consideradas importantes para o pesquisador e para a empresa.

O estudo, no referencial teórico, sobre cultura organizacional e conhecimento organizacional foi de grande importância para entender os impactos causados ao alterar processos, sem o correto envolvimento das pessoas que executam tais processos e/ou pessoas que são afetadas por eles. Da mesma forma, o pesquisador, não tinha percepção do quão vasto é este tema e principalmente as formas que a gestão do conhecimento impacta na cultura da organização.

Pesquisar sobre metodologias ágeis, permitiu ao pesquisador perceber que para um time ser considerado ágil, não basta executar um processo de desenvolvimento de *software* baseado em uma metodologia ágil. Segundo Farias (2013, p.89) “não é o quadro na parede que faz da sua equipe uma equipe ágil.”. No entanto é preciso observar alguns pontos que podem impactar e servem de premissa para a implantação de métodos ágeis (MELO, 2010): apoio gerencial; escolha do método e das práticas; realização de projeto piloto e educação e suporte ao time.

O aprendizado obtido ao atingir os objetivos deste trabalho permitiu ao pesquisador compreender a importância de um processo de desenvolvimento de *software* baseado em métodos ágeis na gestão do conhecimento organizacional. Através de práticas ágeis como jogo do planejamento, *review* diário, *Sprint review* e outras também citadas no trabalho, o pesquisador, pode perceber como “ilhas de conhecimento” podem ser quebradas através do compartilhamento de informações e troca de experiências, salientando as etapas da espiral do conhecimento de Nonaka e Takeuchi (2008).

O time de P&D da empresa estudada, bem como as lideranças da empresa, aceitaram a execução da proposta do novo processo de desenvolvimento de *software* realizado no presente trabalho visando balancear a autonomia que a equipe possui hoje com a responsabilidade de entregar as funcionalidades

assumidas no início de cada iteração, prática a ser implementada pelo processo proposto neste trabalho.

No entanto, ocorreram dificuldades para a realização deste trabalho. Realizar a proposta de um novo processo de desenvolvimento de *software* para o time de P&D, inicialmente para o pesquisador, seria realizar a adaptação de uma metodologia ágil existe à cultura da empresa estudada. Porém ao decorrer do trabalho, ao realizar a pesquisa no referencial teórico, o pesquisador, percebeu que tal proposta, anteriormente, idealizada não deveria ser apenas uma adaptação de uma metodologia existente e sim um alinhamento do time de P&D e da empresa estudada com os princípios ágeis. Além de um ambiente organizacional propicio, o pesquisador, percebeu que a melhor proposta, de um novo processo de desenvolvimento de *software*, seria utilizar práticas de diferentes metodologias ágeis para formar novo processo que inicialmente irá atender os objetivos do time e da empresa, porém existindo um ambiente organizacional preparado para essa mudança este novo processo poderia ser aperfeiçoado com mais facilidade.

Mesmo com a aplicabilidade restrita ao time de P&D, esta proposta de um novo processo de desenvolvimento de *software*, poderá ser adaptada ao time de Customizações, sendo até foco de um trabalho futuro dando continuidade ao presente estudo, medindo a evolução do processo atual e comparando com o processo executado por outros times.

REFERÊNCIAS

ANGELONI, Maria Terezinha et al. **Organizações do conhecimento:** infra-estrutura, pessoas e tecnologia; 2ª edição. São Paulo: Saraiva, 2008.

ANSI/IEEE Std 610.12-1990, **IEEE Standard Glossary of Software Engineering Terminology**, February 1991.

AUDY, José. *Daily Tracking?*. [S.I], 2013. (10 min 58 s). Disponível em: < http://www.infoq.com/br/presentations/dailytracking-comprometimento>. Acesso em: 27 agosto 2013.

BECK, Kent. ***Extreme programming explained:*** *embrace change*. Upper Saddle River: Addison-Wesley, 2000.

BECK, Kent et al. *Manifesto for Agile Software Development*, 2001. Disponível em: <http://agilemanifesto.org/>. Acesso em: 15 de agosto de 2013.

BOEHM, Barry. A View of 20th and 21st Century Software Engineering. **International Conference on Software Engineering (ICSE)**, Shanghai: 2006.

CABRAL, Anderson Ricardo Yanzer. Uma metodologia para aquisição de conhecimento em reuniões de projetos de desenvolvimento de *software*. 2012. 174 f. Tese (Doutor em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Pontifica Universidade Católica do Rio Grande do Sul – PUCRS, Porto Alegre, RS, 2012.

CAPARETO, João Luiz Xavier. ***Design*** **estratégico e** ***Scrum***: Suas relações para processos de projeto de *Websites* em agências de comunicação. 2012. 173 f. Dissertação (Mestrado em *Design*) – Programa de Pós-Graduação em *Design*, Universidade do Vale do Rio dos Sinos – UNISINOS, São Leopoldo, RS, 2012.

CARVALHO, Fábio Câmara Araújo. Gestão do Conhecimento;1ª edição. São Paulo: Pearson, 2012.

CARVALHO, Bernardo Vasconcelos; MELLO, Carlos Henrique Pereira. Revisão, Análise e Classificação da Literatura sobre o Método de Desenvolvimento de Produtos Ágil SRUM. **Simpósio de Administração da Produção, Logística e Operações Internacionais (SIMPOI)**, São Paulo: 2009.

COCKBURN, Alistair. *Agile Software Development: The Cooperative Game*; 2ª edição. Boston: Addison-Wesley Professional, 2006.

CONFORTO, Edivandro Carlos; AMARAL, Daniel Capaldo. Escritório de Projetos e Gerenciamento Ágil: Um Novo enfoque para a estrutura de apoio à gestão de projetos ágeis. **Encontro Nacional de Engenharia de Produção (ENEGEP)**, Foz do Iguaçu: 2007.

COHN, Mike. *Agile Estimating and Planning*; 1ª edição. Prentice Hall, 2005

CUNHA, Virginia Silva. Uma abordagem Orientada a Serviços para a Captura de Métricas de Processo de Desenvolvimento de *Software*. 2006. 137 f. Dissertação (Mestrado em

Computação) – Programa de Pós-Graduação em Ciência da Computação, Pontifica Universidade Católica do Rio Grande do Sul – PUCRS, Porto Alegre, RS, 2006.

FARIAS, André G. **Agile:** Desenvolvimento de Software com entregas frequentes e foco no valor de negócio;1ª edição. São Paulo: Casa do Código, 2013.

JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. **The Unified Software Development Process.** Reading: Addison Wesley, 1999.

JUNGES, Fábio M. **A gestão do conhecimento para a promoção do desempenho organizacional**: Um estudo no setor de TI do Rio Grande do Sul. 2011. 221 f. Dissertação (Mestrado em Administração de Empresas) – Programa de Pós-Graduação em Administração de Empresas, Universidade do Vale do Rio dos Sinos – UNISINOS, São Leopoldo, RS, 2011.

LARMAN, Craig. **Utilizando UML e padrões:** uma introdução à ánalise e ao projeto orientados a objetos e ao desenvolvimento iterativo; tradução Rosana Vaccare Braga ..[et al.]; 3ª edição. Porto Alegre: Bookman, 2007.

LERMEN, Michele. **Molps**: Uma Ontologia para definição de Linha de Produto de Software para Gerência de Projeto com Metodologias Ágeis. 2012. 182 f. Dissertação (Mestrado em Computação) – Programa de Pós-Graduação em Computação, Universidade do Vale do Rio dos Sinos – UNISINOS, São Leopoldo, RS, 2012.

LIKER, Jeffrey K.; HOSEUS, Michael. **A Cultura Toyota:** a alma do Modelo Toyota; tradução Francisco Araújo da Costa; Porto Alegre: Bookman, 2009.

MALLMANN, Paulo Roberto. Um Modelo Abstrato de Gerência de *Software* para Metodologias Ágeis. 2011. 132 f. Dissertação (Mestrado em Computação) – Programa de Pós-Graduação em Computação Aplicada, Universidade do Vale do Rio dos Sinos – UNISINOS, São Leopoldo, RS, 2011.

MARÇAL, Ana Sofia Cysneiros. SCRUMMI: Um processo de gestão ágil baseado no SCRUM e aderente ao CMMI. 2009. 205 f. Dissertação (Mestrado em Informática Aplicada) – Programa de Pós-Graduação em Informática Aplicada, Universidade de Fortaleza – UNIFOR, Fortaleza, CE, 2009.

MARTINS, José Carlos Cordeiro. Gerenciando projetos de desenvolvimento de *software* com PMI, RUP e UML; 4ª edição. Rio de Janeiro: Brasport, 2007.

MCHUGH, Sean. How To Not Destroy your Agile Team with Metrics. 22 outubro 2012. Disponível em: < http://www.infoq.com/articles/not-destroy-team-metrics>. Acesso em: 22 agosto 2013. Texto postado no Portal INFOQ, no link Artigos.

NONAKA, Ikujiro; TAKEUCHI Hirotaka. Gestão do Conhecimento; Tradução Ana Thorell; Porto Alegre: Bookman, 2008.

NOVELLO, Taisa C. Uma abordagem de *Data Warehouse* para Análise de Processos de Desenvolvimento de Software. 2006. 152 f. Dissertação (Mestrado em Computação) – Programa de Pós-Graduação em Computação, Pontifica Universidade Católica do Rio Grande do Sul – PUCRS, Porto Alegre, RS, 2006.

PRESSMAN, Roger S. **Software Engineering: a practitioner's approach**. Boston: McGraw-Hill; 5ª edição, 2001.

REBELO, Paulo. Sinais vitais de um projeto ágil: saúde através de indicadores. 17 dezembro 2011. Disponível em: < http://www.infoq.com/br/articles/sinais-vitais-projeto-agil>. Acesso em: 22 agosto 2013. Texto postado no Portal INFOQ, no link Artigos.

REOLON, Luiz. **Construindo muros para derrubar barreiras:** Gestão Visual Ágil . [S.I], 2013. (43 min 19 s). Disponível em: < http://www.infoq.com/br/presentations/gestao-visual-agil>. Acesso em: 27 agosto 2013.

ROSITO, Maurício C. Um Modelo de Integração entre a Gerência de Projetos e o Processo de Desenvolvimento de Software. 2008. 143 f. Dissertação (Mestrado em Computação) – Programa de Pós-Graduação em Ciência da Computação, Pontifica Universidade do Rio Grande do Sul - PUCRS, Porto Alegra, RS, 2008.

SCHWABER, Ken. *Agile Project Management with Scrum*. Redmond: Microsoft Press; 1ª edição, 2004

SENGE, Peter M. **A Quinta Disciplina:** arte e prática da organização que aprende; tradução: Gabriel Zide Neto e OP traduções; 26 edição. Rio de Janeiro: Best*Seller*, 2010.

SHORE, James; WARDEN, Shane. *The Art of Agile Development*; 1 edição. Sebastopol: O,Relly, 2007.

SILVA, Fernando Selleri. Uso de Representação do Conhecimento para Documentação em Metodologias Ágeis. 2009. 149 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Pontifica Universidade do Rio Grande do Sul - PUCRS, Porto Alegra, RS, 2009.

SILVA, Lúcio J. A contribuição do *BLOG* nas etapas do processo de Gestão do Conhecimento. 2011. 140 f. Dissertação (Mestrado em Administração e Negócios) – Programa de Pós-Graduação em Administração e Negócios, Pontifica Universidade do Rio Grande do Sul - PUCRS, Porto Alegra, RS, 2011.

SOARES, Michel S. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de *Software*. Novembro 2004. Disponível em: < http://www.dcc.ufla.br/infocomp/index.php?option=com_content&view=article&id=219&Itemid=73 >. Acesso em: 22 agosto 2013. Artigo postado no Portal INFOCOMP *Journal of Computer Science*, no link Volume 2.

SOMMERVILLE, Ian. **Engenharia de Software**; tradução Ivan Bosnic e KalinaG. De O. Gonçalvez; revisão técnica Kechi Hirama; 9 edição. São Paulo: Pearson Prentice Hall, 2011.

SUZIN, Juliana B. Análise das competências organizacionais para internacionalização: o caso VULCABRAS/AZALEIA. 2010. 140 f. Dissertação (Mestrado em Administração) – Programa de Pós-Graduação em Administração, Universidade do Vale do Rio dos Sinos – UNISINOS, São Leopoldo, RS, 2010.

TELES, Vinícius Manhães. *Extreme Programming* : aprenda como encantar seus usuários desenvolvendo *software* com agilidade e alta qualidade; 1 edição. São Paulo: Novatec Editora, 2006.

YIN, R.K. **Estudo de caso: planejamento e métodos.** Porto Alegre: Bookman, 2010.

VERGARA, S.C. **Métodos de pesquisa em administração.** São Paulo: Altas: 2010.