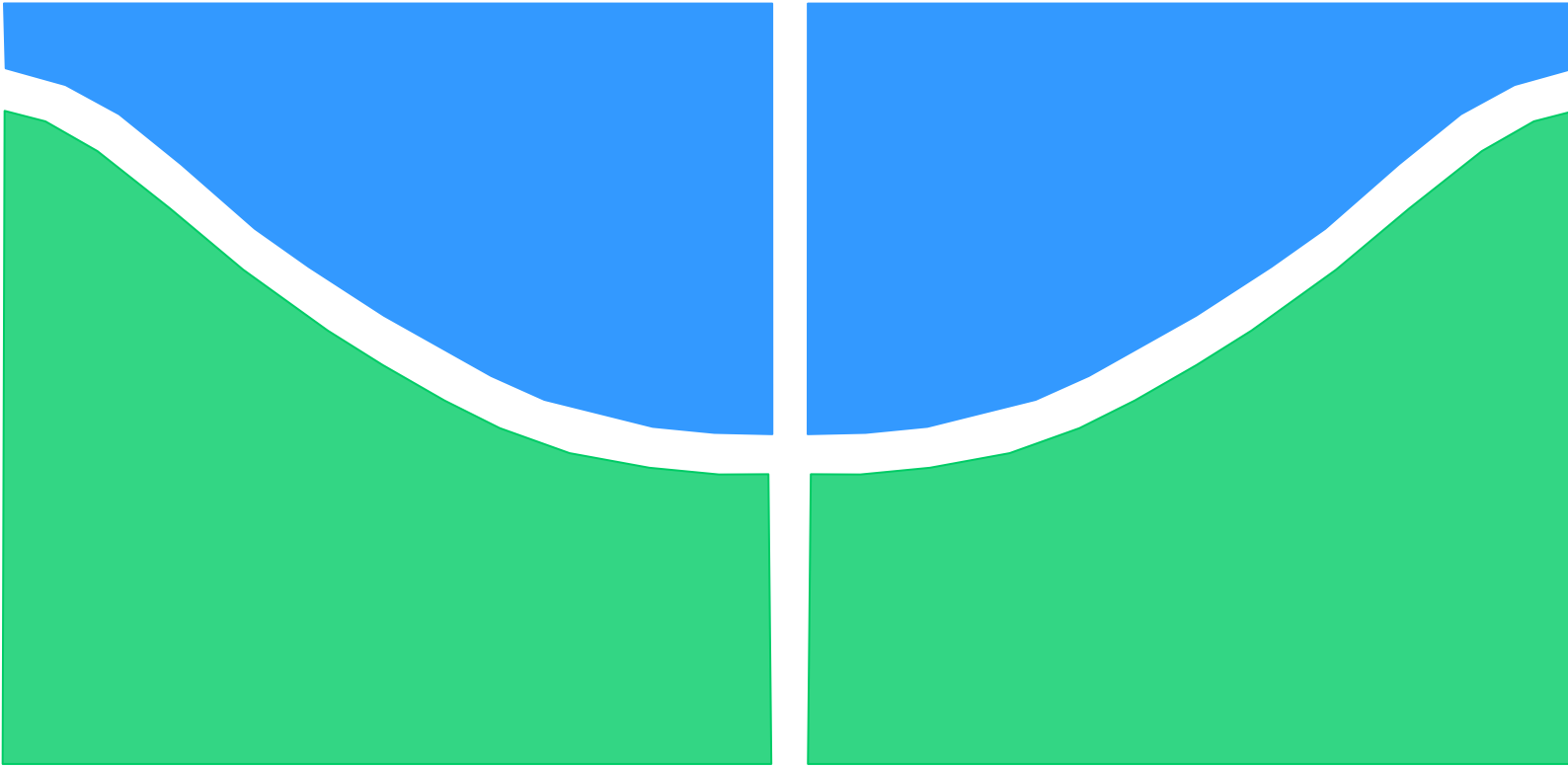

TRABALHO DE GRADUAÇÃO

Escalonando o Scrum Dentro de uma Empresa de Desenvolvimento de Software com Equipes Geograficamente Distribuídas.

João Paulo Rezende Possa

Brasília, Setembro de 2013

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

Escalonando o Scrum Dentro de uma Empresa de Desenvolvimento de Software com Equipes Geograficamente Distribuídas.

João Paulo Rezende Possa

Relatório submetido ao Departamento de Engenharia Elétrica
como requisito parcial para obtenção do grau de
Engenheiro de Redes de Comunicação

Banca Examinadora

Profª. Dra. Edna Dias Canedo, UnB/ UFGA
(Orientadora) _____

Prof. Dr. Rafael Timoteo de Souza, UnB/ ENE _____

Prof. Dr. Edgard Costa Oliveira, UnB/ FGA _____

Dedicatória(s)

Dedico esse trabalho aos meus pais, minhas irmãs e aos meus amigos que me apoiaram ao longo do curso de graduação.

João Paulo Rezende Possa

Agradecimentos

Agradeço aos meus colegas de universidade, por terem compartilhado comigo momentos bons e ruins ao longo desse período de formação, aos professores que me transmitiram o conhecimento necessário para me tornar um bom profissional, a minha professora e orientadora Edna Dias Canedo e principalmente a Deus por me dar forças para alcançar meus objetivos mesmo nas situações mais complicadas.

João Paulo Rezende Possa

RESUMO

Neste trabalho será apresentado um modelo de escalonamento do *Scrum* dentro de um ambiente com equipes distribuídas geograficamente. O *Scrum* é um método de desenvolvimento ágil de software que vem ganhando popularidade entre grandes empresas, que buscam novos meios de responder melhor e mais rápido às mudanças nos negócios. Muitas dessas empresas tem tentado implementar e usufruir dos benefícios do Scrum em projetos envolvendo equipes separadas por grandes distancias, esse novo contexto traz uma série de desafios para que se mantenha a coordenação e transparência do processo. Para desenvolver o modelo proposto, as práticas prescritas pelo *Scrum* foram complementadas com uma série de estratégias, reuniões adicionais e práticas de engenharia, que juntas formam um Framework para escalonar o *Scrum* dentro desse contexto menos favorável.

Palavras-chave: Métodos ágeis; Scrum; Escalonamento; Equipes distribuídas.

ABSTRACT

This work presents a model for scaling Scrum in an environment with geographically distributed teams. Scrum is an agile software development method that is gaining popularity within large companies searching new ways to respond quicker and better to business changes. Many companies have tried to implement and enjoy Scrum's benefits in projects involving teams separated by large distances, this new context brings a lot of challenges to maintain the coordination and transparency of the process. To develop this proposed model the practices prescript by Scrum were extended with some additional strategies, meetings and engineering practices that together form a framework to scale scrum in this less favorable context.

Keywords: Agile methods; Scrum; Scaling; Distributed teams.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

DII	Desenvolvimento Incremental e Iterativo
NASA	*National Aeronautics and Space Administration*
TDD	*Test Driven Development*
FSD	*Federal Systems Division*
RUP	*Rational Unified Process*
RAD	*Rapid Application Development*
DSDM	*Dynamic Systems Development Method*
XP	*Extreme Programming*
FDD	*Feature Driven Development*
AM	*Agile Modeling*
ROI	*Return on Investment*

1 INTRODUÇÃO

1.1 ASPECTOS GERAIS

Os métodos de desenvolvimento ágil de software vêm ganhando bastante popularidade ao longo das últimas duas décadas. A insatisfação com o modelo tradicional de desenvolvimento, com bastante planejamento e documentação na fase inicial do projeto, e pouca flexibilidade com relação às mudanças nos requisitos, é em parte responsável por essa popularidade.

O *Scrum* é o método ágil mais utilizado atualmente (VERSIONONE, 2012), existem inúmeros relatos de sucesso sobre a sua implementação em empresas desenvolvedoras de software (SCHWABER, 2004). Esse método, que originalmente foi projetado para ser aplicado em ambientes com pequenas equipes trabalhando em um mesmo local, hoje vem sendo utilizado por empresas multinacionais com diversas equipes trabalhando em locais geograficamente distribuídos.

A distância entre as equipes traz uma série de novos desafios. A comunicação, aspecto chave para o sucesso do *Scrum* (SCHWABER; BEEDLE, 2002) se torna mais complicada nesse tipo de situação. Coordenar e sincronizar o trabalho de todos envolvidos em um mesmo projeto exige práticas que complementem aquelas que já são prescritas pelo *Srum* tradicional.

Uma versão estendida é capaz de administrar as dificuldades que fazem parte desse tipo de ambiente, um conjunto de estratégias associado com práticas de engenharia e reuniões adicionais torna o processo mais transparente e a sincronização do trabalho das equipes muito mais prática.

Uma empresa que utiliza métodos ágeis e que enfrente pressões para rapidamente duplicar ou quadruplicar sua produtividade sob a restrição de um determinado orçamento pode abrir filiais em locais favoráveis ou terceirizar algumas equipes de desenvolvimento. Essa prática pode tornar possível que a empresa melhore seu tempo de resposta ao mercado e consequentemente obtenha ganhos com isso (SUTHERLAND, 2007).

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Esse projeto tem como objetivo propor um modelo de escalonamento do Scrum em empresas com equipes geograficamente distribuídas.

Para isso, serão apresentados um conjunto de estratégias, reuniões adicionais e práticas de engenharia, com a finalidade de tornar possível a implementação do Scrum dentro do ambiente proposto.

1.2.2 Objetivo Específico

Obter uma contextualização do que foi proposto através de um estudo de caso de uma empresa com equipes trabalhando em três cidades diferentes, mostrando os erros e os acertos que ocorreram durante o processo de escalonamento.

1.3 MÉTODO DE PESQUISA

O modelo sugerido foi baseado na análise e estudo qualitativo de diversos artigos e relatos práticos de experiências reais dentro desse contexto.

A base para o modelo proposto são os casos de sucesso no escalonamento do *Scrum* na empresa Universo Online (MARANZATO; NEUBERT; HERCULANO, 2012) e Microsoft (WILLIAMS, 2011) Uma série de modificações e novas proposições foram feitas visando adaptar as técnicas desenvolvidas para uma empresa com equipes trabalhando em locais diferentes, aliando boas práticas de engenharia com um método *Scrum* adaptado para esse tipo situação.

1.4 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado em cinco capítulos, da seguinte forma:

Capítulo 1: Apresenta os aspectos gerais do tema abordado, os objetivos do modelo de escalonamento proposto no trabalho, o método de pesquisa utilizado e a sua organização.

Capítulo 2: Expõe a definição de desenvolvimento ágil de software, suas origens e contexto histórico, a visão geral de alguns dos métodos mais populares, e como foi a recepção e adoção desses métodos.

Capítulo 3: Explica detalhadamente como funciona o *Scrum*, começando pelos conceitos por trás do método e terminando nos desafios que são encontrados ao implementá-lo.

Capítulo 4: Trata do escalonamento do *Scrum* em empresas com equipes geograficamente distribuídas. Nesse capítulo será proposto pelo autor um modelo estendido e modificado do *Scrum*, esse modelo será contextualizado em um estudo de caso de uma empresa.

Capítulo 5: É a conclusão do trabalho, faz considerações finais e fala sobre possíveis projetos futuros dentro dessa área de pesquisa.

2 DESENVOLVIMENTO ÁGIL DE SOFTWARE

2.1 DEFINIÇÃO

Definir o que é um método ágil não é uma tarefa simples, o próprio sentido do termo agilidade varia de acordo com a situação em que é empregado. Método ágil serve como um rótulo comum para um considerável número de procedimentos bem definidos que se auto denominam ágeis e que variam na prática.

Uma maneira de descrever os métodos ágeis é indicar quais são as principais características e práticas que a maioria deles compartilham. Barry Boehm (2003), afirma que os métodos ágeis são no geral processos muito leves, que empregam ciclos de iteração curtos - envolvem os usuários ativamente para estabelecer, priorizar e verificar os requisitos, e se baseiam em conhecimentos implícitos dentro da equipe a despeito de documentação. No mesmo texto, Boehm considera os métodos ágeis uma consequência da experiência de prototipagem e desenvolvimento rápido, assim como o ressurgimento de uma filosofia que considera programação um processo "artesanal" ao invés de um processo industrial.

De acordo com a definição de Alistar Cockburn (2001), agilidade implica ser efetivo e manobrável. Um processo ágil é leve e suficiente, onde a leveza é um meio de se manter manobrável e a suficiência é uma questão de continuar dentro do jogo.

De forma mais objetiva Larman (2004), descreve algumas das principais práticas que esses métodos compartilham, como pequenas iterações de duração pré-fixada com refinamentos de planos e objetivos, e evolucionário. Abrahamsson (2003) também elucida objetivamente algumas das principais características comuns aos métodos considerados ágeis, como desenvolvimento iterativo e incremental, capacidade de adaptação, cooperação e simplicidade. Os preponentes ágeis clamam que os aspectos chave de métodos leves e ágeis são a simplicidade e a velocidade.

Em janeiro de 2001 um grupo de dezessete pessoas especializadas em processos de desenvolvimento de software reuniram-se em um resort no estado de Utah nos EUA, representando diversos métodos como *Extremme Programming*(XP), *Scrum*, *Dynamic Systems Development Method*(DSDM), *Feature Driven Development*(FDD), e outros. Nesse encontro, os participantes publicaram um texto conhecido como o *Agile Manifesto*, e alguns

deles formaram a *Agile Alliance,* uma organização sem fins lucrativos que visa promover os métodos ágeis de desenvolvimento de software.

Algumas das abordagens comuns oriundas do processo de desenvolvimento ágil de software foram sugeridas por Ambler (2002):

- As pessoas são importantes;
- Comunicação é um aspecto crítico;
- Menos documentação é possível;

De acordo com Boehm (2003), todos os métodos ágeis seguem os quatro valores e os doze princípios básicos do *Agile Manifesto* que serão apresentados nas tabelas 2.1.1 e 2.1.2 respectivamente.

Tabela 2.1.1 Valores do *Agile Manifesto (AGILE ALLIANCE, 2001)*

Valores
Indivíduos e interações acima de processos e ferramentas.
O funcionamento do software acima de documentação abrangente
A colaboração dos clientes acima da negociação de contratos
A capacidade de responder a mudanças acima de um plano pré-estabelecido

Tabela 2.1.2 Princípios do *Agile Manifesto (AGILE ALLIANCE, 2001)*

Princípios
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.

Tabela 2.1.2 Princípios do *Agile Manifesto (AGILE ALLIANCE, 2001), continuação*

Princípios
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.

Levando em consideração todas as informações apresentadas é possível identificar os principais aspectos que caracterizam um método ágil. São eles:

- Iterativo e incremental;
- Adaptativo;

- Cooperativo;
- Simples.

Por ser iterativo e incremental, no final de cada iteração um pedaço do sistema é desenvolvido, testado e aprimorado, enquanto uma nova parte do sistema está sendo desenvolvida. As iterações permitem um aprimoramento contínuo do software, bem como uma maior transparência e facilidade de inspeção do processo de desenvolvimento. Ao fim de cada iteração uma funcionalidade nova do sistema pode ser integrada e lançada, tornando mais simples e frequente a obtenção do feedback dos clientes.

A capacidade de adaptação é uma das consequências diretas do modelo iterativo e incremental. Os métodos ágeis aceitam e permitem mudanças nos requisitos ao longo do processo de desenvolvimento, ou seja, em diversos casos os requisitos estão em constante modificação. A habilidade de se adaptar e administrar requisitos emergentes torna esses métodos mais versáteis em cenários onde muitas mudanças podem acontecer, em um intervalo de tempo relativamente pequeno.

Indivíduos e interações ao invés de ferramentas e processos são uma das principais particularidades dos métodos ágeis quando comparados aos métodos tradicionais. O desenvolvimento ágil de softwares depende muito da capacidade de comunicação e entrosamento entre equipe de desenvolvimento e os clientes, já a filosofia por trás desses métodos prioriza as pessoas e não os processos. Esse modelo enfatiza a importância da comunicação face-a-face entre os envolvidos e interessados no desenvolvimento do produto.

A simplicidade, no sentido de que o método em si é facilmente compreendido e que apenas a documentação necessária é produzida ao longo do processo, é um outro fator que os caracteriza. Muitos de seus defensores, como Beck (1999), argumentam a favor da redução dos esforços gastos com documentação, na maioria das vezes desnecessárias e subutilizadas ao longo do ciclo de vida de um software.

Desenvolver um software é uma atividade muito complexa, os meios que são utilizados para construir o produto final são extremamente voláteis. Requisitos de usuário, as pessoas envolvidas no processo e a interação do programa a ser desenvolvido com outros de um mesmo sistema acabam contribuindo muito para essa complexidade.

Problemas demasiadamente complexos costumam ser difíceis de prever ou são até mesmo imprevisíveis (SCHWABER, 2004), e desenvolver um software não é diferente. Para tentar contornar essa imprevisibilidade utiliza-se um processo adaptativo, como o

desenvolvimento incremental e iterativo, além de pessoas habilidosas e criativas envolvidas na ação, dado que o desenvolvimento de um software é uma atividade altamente intelectual.

A capacidade de "driblar" imprevistos, associada com a leveza do processo e foco na entrega periódica de um sistema em constante aprimoramento, torna os modelos ágeis bastante atraentes, tanto para as empresas quanto para seus clientes. Durante muito tempo o processo em cascata tradicional foi tido como o modelo ideal de desenvolvimento de software, entretanto sua incapacidade de administrar a complexidade e os imprevistos que envolvem esse tipo de atividade fizeram com que novos processos passassem a ser cogitados por pessoas que atuam na área, buscando uma significativa melhora de desempenho e, consequentemente, mais sucesso nos negócios.

2.2 ORIGENS E CONTEXTO HISTÓRICO

Embora muitas pessoas da área acreditem que boa parte dos fundamentos por trás dos métodos ágeis são relativamente recentes, existem relatos do uso de processos muito ligados ao desenvolvimento ágil que datam das décadas de 50, 60 e 70.

De fato, o conceito de desenvolvimento ágil de software é relativamente novo, mas muitas das técnicas inerentes a esse modelo já foram testadas e utilizadas em diversos projetos ao longo das últimas décadas. Em seu artigo sobre a história do desenvolvimento incremental e iterativo(DII), Larman e Basili (2003) descrevem uma série de projetos que fizeram o uso de processos incrementais e iterativos, ainda que muitos deles tivessem um alto nível de documentação e planejamento inicial.

Nos anos 60, o projeto Mercury da *NASA*(*National aeronautics and space administration*) utilizou o DII para o desenvolvimento de software. Ao longo do projeto foram utilizadas iterações de tempo pré-fixado com a duração da metade de um dia, além disso, algumas das técnicas utilizadas *no Extreme Programming*(XP) foram aplicadas, como *o test driven development*(TDD) que consiste em planejar e escrever testes antes de produzir um novo incremento.

Na década de 70 o DII foi aplicado no projeto da central de comando do submarino US Trident realizado pela *IBM Federal Systems Division*(FSD). Esse projeto de missão crítica, com mais de um milhão de linhas de código, fez o uso de iterações de tempo fixo com duração de seis meses, ainda que a duração das iterações fosse bem maior do que a recomendada pelos métodos atuais. Os responsáveis pelo projeto puderam notar uma boa

evolução dos requisitos devido ao feedback recebido ao fim de cada iteração, eles também constataram que o DII é eficiente no gerenciamento da complexidade de um projeto.

Também nos anos 70 a *TRW* fez uso do DII em um projeto milionário para o departamento de defesa dos Estados Unidos, o software desenvolvido seria utilizado em um sistema de defesa contra mísseis balísticos. Foram realizadas cinco iterações, e ao fim da primeira o sistema já possuía a capacidade de rastrear um objeto. Ao longo das iterações seguintes novas funcionalidades foram adicionadas, até que ao final da quinta e última iteração o sistema foi entregue por completo, com todas as suas funcionalidades previstas. Nesse projeto a duração das iterações não foi rigorosamente pré-fixado.

Outro caso em que o DII foi utilizado com sucesso foi o desenvolvimento do software para o ônibus espacial da *NASA*. A *IBM* FSD foi a empresa responsável pela construção do software, e ao longo do projeto foi aplicada uma série de dezessete iterações com a duração de aproximadamente oito semanas. No decorrer de um período de trinta e um meses, o DII foi utilizado para contornar eventuais problemas causados pelas mudanças de requisitos que ocorreram ao longo do tempo em que o software foi desenvolvido.

Já em meados da década de 80 a *TRW* realizou um projeto para construir um sistema de comando e controle chamado *Command and Control Processing and Display System Replacement*. A equipe de desenvolvimento realizou seis iterações com duração pré-fixada, cada uma com duração média de quatro meses. Oprocesso utilizado pelo projeto foi muito semelhante ao que mais tarde ficaria conhecido como *Rational Unified Process*(RUP).

No final dos anos 80, o departamento de defesa dos Estados Unidos teve que realizar modificações nos padrões exigidos para firmar contratos de desenvolvimento de software, de forma que o DII pudesse ser permitido na construção dos sistemas. Essas modificações foram consequência direta da alta taxa de falha na aquisição de softwares desenvolvidos através do processo estritamente cascateado e guiado por documentação.

Apenas nos anos 90 que os métodos DII passaram a ser largamente promovidos. Muitos artigos e livros passaram a discutir o tema e, consequentemente, novas ideias começaram a surgir, resultando nos primeiros métodos realmente considerados ágeis.

Ao longo dessas décadas alguns especialistas da área e estudiosos fizeram grandes contribuições para a evolução do DII. Winston Royce (1970), sugeriu a adição de 5 passos ao modelo sequencial tradicional para eliminar a maior parte dos riscos de desenvolvimento.

Esses passos sugeridos incentivaram a utilização do modelo incremental, além de um envolvimento mais formal por parte dos clientes no processo de desenvolvimento.

Gladden (1982), sugeriu uma nova visão do processo de desenvolvimento, para ele os objetivos do sistema são mais importantes do que os requisitos. Esse conceito coincide com a ideia ágil de que é mais importante ter um conhecimento geral do sistema, do que um detalhamento exagerado dos requisitos que provavelmente irão mudar ao longo do período de desenvolvimento, ou seja, é mais vantajoso trabalhar no software em si do que produzindo documentação compreensível.

McCracken e Jackson (1982), sugeriram dois cenários de processos de desenvolvimento de sistemas. No primeiro cenário eles recomendaram a prototipagem, que consistia em construir um protótipo no início do processo de desenvolvimento para obter uma resposta do cliente o mais cedo possível, e a partir dessas informações, novos protótipos seriam desenvolvidos modificando e incrementando o inicial, de acordo com as necessidades do cliente, até que se chegue ao produto final.

No segundo cenário, eles sugeriram um processo de desenvolvimento de sistemas feito pelo usuário final, e o desenvolvedor utilizando um modelo iterativo e incremental provendo ao usuário uma ferramenta de implementação semelhante às ferramentas *CASE* atuais.

Outro grande percursor de ideias, que mais tarde deu origem ao método ágil, foi Tom Gilb (1985), ele propôs um método alternativo ao tradicional método “cascata”, e que foi denominado EVO.

O EVO foi baseado em três princípios básicos:

- Entregar algo ao real usuário final;
- Medir o valor agregado ao usuário em todas as dimensões críticas;
- Ajustar e projetar os objetivos baseado em realidades observadas.

Nesse mesmo artigo, Gilb sugere que as primeiras iterações devem produzir as funcionalidades com maior custo benefício, em termo de valores agregados para o usuário final e custo de desenvolvimento. Outro conceito importante do método EVO é a realização de uma completa análise, projeto e teste em cada iteração. A vantagem do método evolucionário é a capacidade de responder bem a mudanças e contornar parte do problema causado pela complexidade da construção de um sistema. Outro aspecto do método proposto

por Gilb, que também coincide com o dos métodos ágeis atuais, é a participação ativa do cliente no processo de desenvolvimento do software.

James Martin (1991), lançou uma metodologia chamada *Rapid Aplication Development*(RAD). O RAD é constituído por quatro fases: planejamento de requisitos, projeto do usuário, construção e implementação. Nesse modelo, a fase de construção tem uma duração pré-fixada, onde o produto final de cada interação deve ser um sistema funcionando, que possa ser avaliado de modo que seja possível decidir se o produto está apto para entregue ou não. Martin recomenda iterações com 60 dias de duração, além de equipes compostas por uma a cinco pessoas.

2.3 VISÃO GERAL DE ALGUNS MÉTODOS ÁGEIS

Nessa seção será dada uma visão geral de alguns dos métodos ágeis que tem sido utilizados ultimamente (VERSIONONE, 2012) como o *Dynamic Systems Development Method(DSDM)*, *Extreme Programmin*g(XP), *Feature Driven Develeopment(FDD)* e *Agile Modeling(AM)*. O *Scrum* será descrito detalhadamente no capítulo 3.

2.3.1 DSDM

O DSDM é um framework para *Rapid Aplication Development* mantido pelo DSDM consortium. A filosofia por trás desse método prega que qualquer projeto pode ser guiado por objetivos estratégicos claramente definidos, além de focar na entrega rápida de benefícios reais aos negócios. (STAPLETON, 1997). O método em questão cobre todo o ciclo de vida de um projeto e provê um guia de boas práticas para a entrega de projetos dentro do orçamento e prazo devido. Algumas das técnicas utilizadas no DSDM são:

- Modelamento;
- DII;
- Workshops facilitados;
- Iterações com tempos pré-fixados.

Seu processo é dividido em cinco fases: Estudo de viabilidade, estudo de negócios, iteração de projeto e construção, e implementação. As últimas três fases funcionam de forma iterativa e incremental, onde o trabalho é realizado de fato. Como foi descrito anteriormente,

esse modelo faz uso de iterações com tempo pré-fixado, ou seja, a duração de cada iteração é planejada anteriormente.

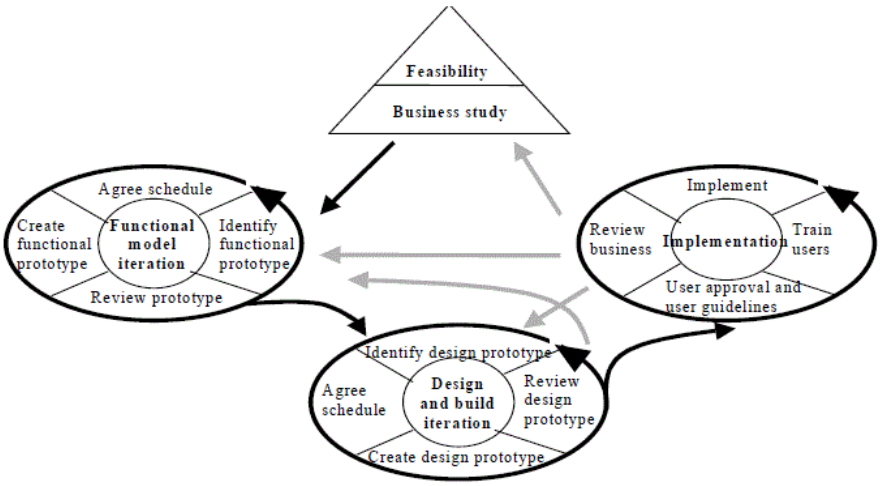

Figura 2.1 – Diagrama de processos DSDM (Stapleton 1997, p. 3)

No método DSDM são definidos 15 papéis para desenvolvedores e usuários. Dentro do grupo dos desenvolvedores existem somente duas categorias: Desenvolvedor e desenvolvedor sênior. O responsável pela qualidade do projeto e arquitetura do sistema é chamado de coordenador técnico.

Dentro do grupo de usuários o mais é importante é o chamado usuário embaixador que é a pessoa responsável por participar ativamente do projeto trazendo conhecimento da comunidade de usuários para os desenvolvedores e disseminando informações sobre o andamento do projeto para outros usuários. Já para os usuários que possam contribuir em algum aspecto específico do projeto, é dado o título de usuário conselheiro.

Existem ainda os papéis de *executive sponsor*, dado ao indivíduo da organização dos usuários que possui a responsabilidade e a autoridade financeira, além do *Visionary*, que é o usuário mais capacitado para perceber e indicar os objetivos de negócios do sistema e do projeto.

Entre os principais práticas do DSDM podemos citar (STAPLETON, 1997):

- Foco na entrega frequente de produtos;
- Participação ativa dos usuários no desenvolvimento;

- As equipes do método DSDM devem estar aptas para tomar decisões importantes;
- O DII é imprescindível para a convergência em uma solução de negócios precisa;
- Estar sempre testando e integrando ao longo do ciclo de vida do projeto;
- Todos os *stakeholders* devem ter uma postura colaborativa e cooperativa.

O tamanho das equipes no DSDM varia de duas a seis pessoas, e é permitido várias equipes trabalhando em um mesmo projeto. Esse método pode ser usado em projetos pequenos ou grandes. Nos projetos grandes seu uso só é recomendável caso o sistema possa ser dividido em componentes menores, que possam ser desenvolvidos por equipes pequenas. Para Stapleton (1997), o DSDM se encaixa melhor no desenvolvimento de sistemas voltados aos negócios do que sistemas relacionados a aplicações científicas ou de engenharia.

2.3.2 FDD

O FDD é outro conhecido método ágil, que busca efetividade e preza por aspectos de qualidade. Essa abordagem não cobre todo o processo de desenvolvimento de software, seu foco é nas fases de projeto e construção. Diferentemente de outras metodologias ágeis, o FDD clama ser aplicável em casos de desenvolvimento de sistemas críticos (PALMER; FELSING, 2002). O método é composto por cinco processos sequenciais, nos quais o sistema é projetado e construído:

- Desenvolvimento de um modelo geral;
- Construção de uma lista de funcionalidades;
- Planejar por funcionalidade;
- Projetar por funcionalidade;
- Construir por funcionalidade.

Os dois últimos processos, projetar e construir por funcionalidade, são a parte iterativa do FDD, um maior detalhamento de como funciona cada processo pode ser encontrado em (PALMER; FELSING, 2002), a figura 2 ilustra o sequenciamento dos processos.

O FDD possui 14 papéis divididos em três categorias, os principais papéis em um projeto são: Gerente de projeto, *chief-architect*, gerente de desenvolvimento, *chief programmer*, *class owner* e *domain expert*.

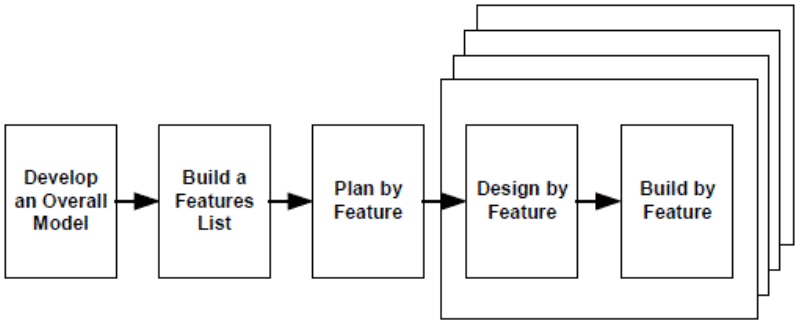

Figura 2.2 – Processos do FDD (Palmer e Felsing 2002).

Os papéis auxiliares são: *Release manager, language lawyer/language guru, build engineer, toolsmith e system administrator*. Os papéis adicionais são: *testers, developers* e *technical writers*. Nas equipes, um indivíduo pode possuir mais de um papel e um papel pode ser compartilhado por múltiplos indivíduos.

O FDD possui um conjunto de boas práticas que são recomendadas para o sucesso dos projetos:

- *Domain object modelling*: Exploração e explanação do domínio do problema;
- Desenvolvimento por funcionalidade;
- Posse individual de Classe(código): Cada classe possui um responsável por sua integridade, consistência e performance;
- Times pequenos e formados dinamicamente;
- Inspeção: usar os melhores mecanismos de detecção erros disponíveis;
- *Regular builds*: garantir que sempre haverá uma versão do sistema funcionando e que possa ser demonstrada;
- Relatar os progressos.

Gerenciamento de configuração: permite a localização e identificação das últimas versões completas de cada arquivo de código fonte.

Os criadores do método sugerem que o FDD seja adotado de forma gradual pelas organizações interessadas em utilizá-lo. Eles também recomendam seu uso para o desenvolvimento mais rápido de projetos voltados aos negócios, sem que ocorram perdas na qualidade.

2.3.3 XP

O *Extreme Programing* foi criado por Kent Beck nos anos 90, as ideias e práticas por trás do método foram maturadas ao longo do tempo que Beck fez parte do projeto C3 de *payroll*, da Chrysler. Em 1999, Beck lançou um livro descrevendo o seu método. Nesse livro Beck afirma que o XP atraiu bastante atenção de um público significativo graças a diversos fatores inerentes ao método proposto, como a ênfase nas comunicações, simplicidade, desenvolvimento guiado por testes, sustentáveis práticas orientadas ao desenvolvedor e devido ao nome interessante que ele deu ao método.

O processo do XP é dividido em cinco fases: *Explation, planning, iterations to release, productionizing, maintence* e *death.*

No decorrer da *exploration phase* a equipe de desenvolvimento se familiariza com as ferramentas, práticas e tecnologias que serão utilizadas ao longo do projeto, concomitantemente, os clientes escrevem os *story cards* descrevendo as funcionalidades que eles gostariam de ter na primeira versão do programa.

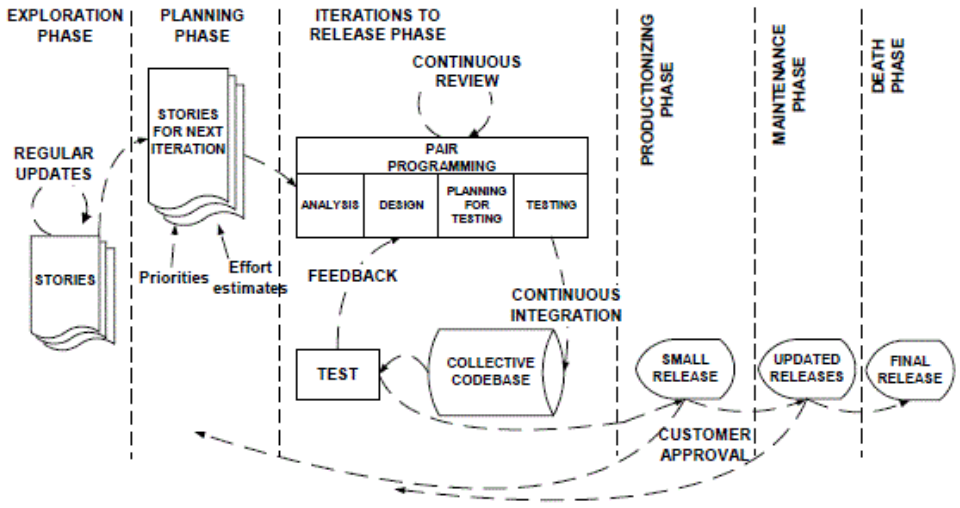

Figura 2.3- ciclo de vida do XP (Abrahamsson 2002)

A *planning phase* serve para priorizar as *stories* e para se chegar a um consenso sobre qual será o conteúdo do primeiro *release*. É durante essa fase que os programadores fazem a primeira estimativa de quanto esforço será gasto em cada *story*, e a partir dessas estimativas uma data de entrega para o primeiro lançamento é definida.

Na *iterations to release phase* o cronograma definido na *planning phase* é dividido em um determinado número de iterações, com duração de uma a quatro semanas. A primeira iteração deve criar um sistema que já possua a arquitetura do sistema como um todo, e isso é feito selecionando as *stories* que irão forçar a construção da estrutura do sistema global. Nas iterações seguintes, o cliente é quem decide quais as *stories* serão selecionadas para a iteração em questão. Ao final de cada iteração, testes funcionais criados pelos clientes são realizados, e já no término da última iteração o sistema está pronto para produção.

Durante a *productionizing phase* mais testes e checagens do sistema são realizados antes que ele possa ser lançado para os clientes. Após essa fase o sistema entregue tem que ser colocado para rodar. Na *maintence phase* novas iterações são realizadas para adicionar funcionalidades e corrigir erros do sistema.

Quando o sistema satisfaz as necessidades do cliente e ele já não possui mais *stories* para serem implementadas, o projeto chega na *death phase*. É nessa fase que a documentação necessária para o sistema é finalmente escrita, dado que não ocorrerão mais mudanças no seu código e arquitetura. Essa fase também é alcançada caso se torne caro demais continuar desenvolvendo o sistema, ou caso ele não esteja entregando as funcionalidades desejadas.

Sete papéis dentro do XP são apresentados por Beck (1999). São eles:

- Programador: Responsável por escrever testes e manter o código do programa o mais simples e conciso possível;
- Cliente: Responsável por escrever as *stories*, testes funcionais e por determinar a prioridade dos requisitos do sistema;
- *Tester*: ajuda o cliente a escrever testes funcionais e os põe para rodar regularmente, sempre divulgando os resultados. Também é responsável por manter as ferramentas de teste;
- *Tracker*: Acompanha as metas estipuladas pela equipe e dá o feedback de o quão precisas elas estão, visando melhorar as estimativas futuras. Ele também

acompanha o progresso de cada iteração e avalia se os objetivos serão alcançados dentro dos limites de prazos e de recursos disponíveis;

- *Coach*: É o responsável por fazer com que as outras pessoas sigam o processo e possui um alto nível de entendimento do XP. Éo responsável pelo processo como um todo;
- Consultor: Um membro de fora da equipe que com o seu conhecimento técnico presta apoio na resolução de problemas específicos;
- Gerente: Responsável por tomar decisões, deve estar sempre a par da situação e identificar eventuais deficiências e dificuldades no processo.

O conjunto de práticas recomendadas pelo XP é listado a seguir (BECK, 1999):

- *Releases* pequenos/curtos;
- Alta interação entre clientes e desenvolvedores;
- Projeto simples, sem complexidades desnecessárias;
- Desenvolvimento de software guiado por testes(TDD);
- Refatoramento contínuo do código;
- *Pair programming;*
- Propriedade coletiva: Qualquer desenvolvedor pode mudar o código a qualquer momento, quando achar necessário;
- Integração contínua, o sistema deve ser integrado várias vezes durante o dia, todos os testes devem ser rodados e novas mudanças só devem ser aceitas caso o sistema passe pelos testes;
- Jornada de trabalho com duração máxima de 40 horas por semana;
- Cliente por perto o tempo inteiro, acessível para a equipe de desenvolvimento sempre que for preciso;
- Espaço de trabalho aberto, para melhorar a comunicação entre os envolvidos no projeto;
- As regras e padrões de codificação devem ser seguidas pelos programadores.

A equipe possui suas próprias regras que são adotadas. Para que ocorram mudanças nessas regras deve haver um acordo dentro da equipe, e os impactos dessas mudanças devem ser levados em conta.

O XP não deve ser usado integralmente em qualquer tipo de projeto. Uma das ideias fundamentais por trás desse método é a de que não existe um processo que se adeque a todos os projetos existentes, pois as práticas devem ser adaptadas para se adequarem a um determinado projeto. Empresas ou organizações que possuam uma grande resistência a processos mais dinâmicos devem pensar duas vezes antes de pretender implementar o XP. Sistemas críticos ou que necessitem de uma vasta documentação formal também não se ajustam muito a esse modelo (SOMMERVILLE, 2009).

2.3.4 AM

Criada por Scott Ambler (2002), o *Agile Modelling* é uma metodologia baseada em conceitos práticos, voltada para a modelagem e documentação mais efetiva de sistemas baseados em software. A metodologia é composta por um conjunto de valores, princípios e boas práticas.

Os valores do AM são os quatro valores do XP acrescidos por mais um: Simplicidade, comunicação, feedback, coragem e humildade. Este último valor foi acrescentado por Ambler para enfatizar que pessoas diferentes possuem diferentes níveis de conhecimento, que devem ser explorados através da cooperação e seleção das pessoas mais indicadas para realizar um determinado trabalho.

Os princípios mais importantes do AM promovem a importância de um software funcional como principal objetivo do modelamento. Para tanto, é preciso aceitar as mudanças, assumir simplicidade, modelar com um propósito e promover mudanças de forma incremental. O AM possui um conjunto de boas práticas, são nelas que podemos observar a essência dessa metodologia:

O AM por si só é insuficiente para um projeto de desenvolvimento de software, ele necessita também de métodos auxiliares, tendo em vista que o seu foco é apenas no modelamento.

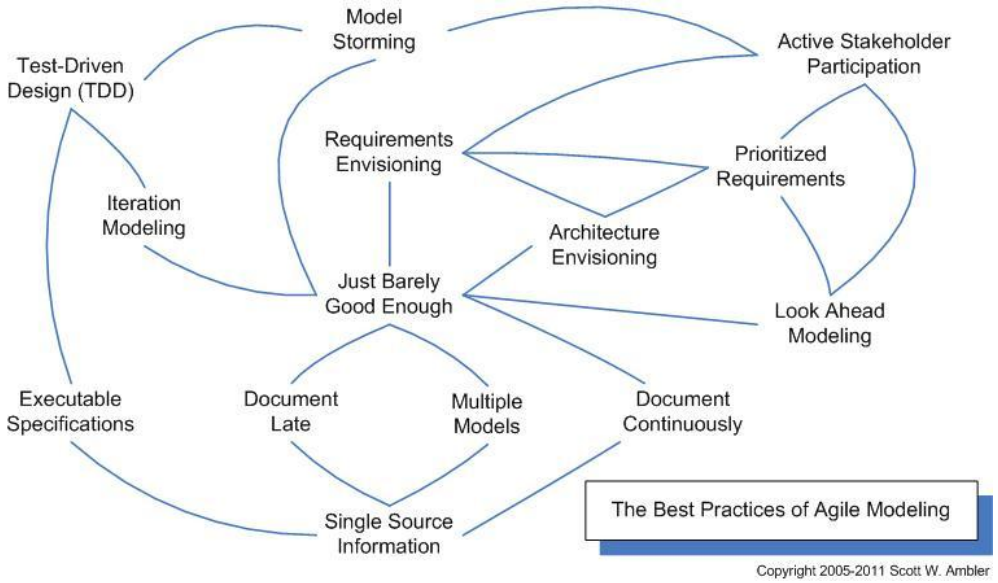

Figura 2.4 – Boas práticas recomendadas pelo AM (Ambler 2005)

2.4 ADOÇÃO

Durante um bom tempo muitos trabalhadores e estudiosos da área acreditaram que o modelo sequencial em cascata, guiado por uma vasta documentação, fosse o modelo ideal para o desenvolvimento de software.

O modelo em cascata, foi promovido e adotado muito tempo por diversos fatores como:

- É simples de lembrar e explicar;
- Dá a ilusão de um processo ordenável, responsabilizável e mensurável, com a divisão de suas etapas bem definida;
- Durante muito tempo foi promovido como o modelo mais apropriado por textos, cursos e empresas de consultoria da engenharia de software.

Embora muitas ideias por trás dos métodos ágeis remontem aos primórdios da história do desenvolvimento de software (LARMAN; BASILI, 2003), somente após meados dos anos 90 esses métodos passaram a ser amadurecidos e amplamente promovidos. Algumas mudanças no mercado, em especial o surgimento da indústria de aplicativos para internet e aparelhos móveis, fizeram crescer a demanda por processos de desenvolvimento cada vez mais rápidos, leves e eficientes.

A adoção ou não de métodos ágeis depende muito do contexto e da cultura da organização que planeja implementá-los. Diversas empresas, em especial as grandes, gastaram muitos anos tentando definir e aprimorar os processos a serem seguidos, muitos desses processos podem ir contra alguns princípios do desenvolvimento ágil impondo uma barreira para a implementação.

Empresas pequenas, onde os processos internos são mais flexíveis e seus funcionários possuem um maior grau de liberdade, costumam ter maior probabilidade de sucesso ao implementar o desenvolvimento ágil, porém muito tem se evoluído no campo da escalabilidade desses métodos, permitindo que empresas maiores também possam usufruir dos benefícios trazidos por eles.

Em um levantamento realizado no último ano pela VERSIONONE (2012), 84% dos 4048 indivíduos que participaram da pesquisa responderam que as suas organizações estavam usando na prática o desenvolvimento ágil, um crescimento de 4 pontos percentuais em comparação com 2011. No mesmo levantamento, 30% afirmaram que suas organizações possuem 10 ou mais equipes que adotaram métodos ágeis. Devido ao cenário competitivo e aos eventuais benefícios trazidos, diversas empresas grandes tem tentado escalonar e implementar o desenvolvimento ágil.

Conforme novos estudos forem realizados, e mais conhecimento for extraído das experiências práticas com a implementação desses métodos, é provável que um número cada vez maior de empresas passe a obter sucesso ao adotá-los.

3 SCRUM

A partir de ideias introduzidas por Takeuchi e Nonaka em um artigo publicado no ano de 1986, Jeff Sutherland e Ken Schwaber criaram ao longo da década de 90, o processo de desenvolvimento ágil de software que ficaria conhecido como *Scrum* (SUTHERLAND; SCHWABER, 1995). A abordagem criada por eles é focada no gerenciamento de desenvolvimento iterativo e incremental. Por ser a metodologia utilizada no modelo de escalonamento proposto por essa dissertação, nas seções a seguir, o *Scrum* será detalhadamente descrito.

3.1 CONCEITOS POR TRÁS DO SCRUM

Existem duas abordagens principais que são utilizadas no controle de um processo, o controle de processo definido e o controle de processo empírico. Para que o controle de processo definido seja bem sucedido é preciso que cada parte do trabalho a ser feito seja muito bem entendida. Dado um conjunto bem definido de entradas, as mesmas saídas devem ser produzidas ao fim do processo, sempre (SCHWABER; BEEDLE, 2002). Quando o controle de processo definido não pode ser utilizado devido à complexidade das atividades intermediárias, o controle de processo empírico deve ser usado.

Desenvolver um software é uma atividade extremamente complexa mas quando várias coisas interagem, o nível de complexidade pode aumentar assustadoramente. Os três principais fatores que tornam um software tão complexo são: Requisitos, tecnologia e as pessoas envolvidas no processo de desenvolvimento (SCHWABER, 2004).

Ao longo do processo de desenvolvimento existem diversos *stakeholders* envolvidos, e é bem comum que eles não saibam exatamente o que querem. Além disso, muitas das vezes eles possuem necessidades e opiniões diferentes, algumas delas inclusive antagônicas. Mudanças nos negócios e novas necessidades têm surgido em um ritmo cada vez mais acelerado, consequentemente, os requisitos costumam ser modificados repetidas vezes, o que os torna uma grande fonte de complexidade para o processo.

A maior parte dos projetos de desenvolvimento de software envolvem o uso de tecnologias avançadas, que são indispensáveis caso se queira ser competitivo dentro dessa indústria.

Pessoas também são uma enorme fonte de complexidade. Cada indivíduo apresenta nível de inteligência, habilidades, pontos de vista e atitudes diferentes, não bastasse isso ainda existem as complexas relações interpessoais.

Para enfrentar toda essa complexidade, deve-se usar um controle de processo empírico. Esse tipo de processo de controle se baseia em três princípios: Visibilidade, inspeção e adaptação.

Visibilidade implica que todos os fatores do processo que afetam o resultado devem ser visíveis àqueles que controlam o processo. Além de visíveis, esses fatores observados devem ser verdadeiros. No processo de controle empírico as aparências não podem enganar.

Todos os aspectos do processo devem ser inspecionados em uma frequência tal que as variâncias indesejadas possam ser identificadas. Essa frequência de inspeção deve levar em conta que processos podem mudar pelo simples fato de serem inspecionados. A pessoa que inspeciona deve possuir as habilidades necessárias para avaliar aquilo que ela está vistoriando.

O último princípio do controle de processo empírico é a adaptação. Se durante a inspeção é constatado que existem aspectos prejudiciais ao processo, e que o produto resultante será inaceitável para os padrões definidos, o inspetor deve ajustar o processo ou o material que está sendo processado o quanto antes.

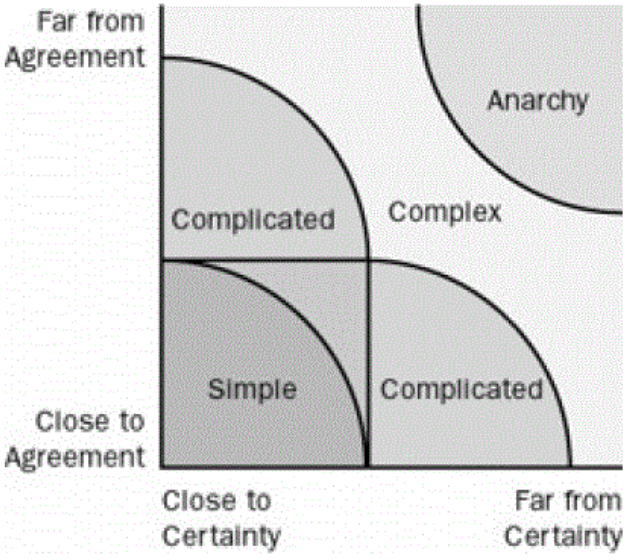

Figura 3.1- Gráfico do nível de complexidade (Schwaber 2004)

O *Scrum* é fundamentado no controle de processo empírico, o empirismo afirma que a sabedoria vem da experiência e que as decisões devem ser feitas a partir daquilo que se sabe (SCHWABER, 2004). Para melhorar a previsibilidade e controlar riscos, o *Scrum* aplica uma abordagem iterativa e incremental colocando em prática os três pilares do processo de controle empírico: Visibilidade, inspeção e adaptação.

3.2 O SCRUM

3.2.1 Visão Geral

Como foi apresentado anteriormente, o *Scrum* é uma abordagem empírica que aplica as ideias da teoria de controle de processos industriais no desenvolvimento de sistemas, resultando numa abordagem que reintroduz as ideias de produtividade, flexibilidade e adaptabilidade (SCHWABER; BEEDLE, 2006). Nessa abordagem nenhuma técnica específica de desenvolvimento de software é definida para a fase de implementação, esse método se concentra em como os membros da equipe devem proceder para produzir o sistema de forma flexível em um ambiente em constante modificação.

O *Scrum* fundamenta todas as suas práticas no desenvolvimento incremental e iterativo. Ao se iniciar uma iteração, a equipe deve avaliar o que deve ser feito e seleciona aquilo que acredita que ao fim da iteração pode resultar em um *increment of potentially shippable functionality*. A partir daí, a equipe se empenha em dar o melhor de si. Ao fim da iteração a equipe apresenta o incremento de funcionalidade construído para que os *stakeholders* possam inspecionar a funcionalidade e observar quais adaptações oportunas podem ser feitas no projeto.

O coração do *Scrum* se encontra na iteração (SCHWABER, 2004). A equipe olha os requisitos, considera a tecnologia disponível e avalia suas próprias capacidades e habilidades. Dessa forma determina de maneira coletiva como construir determinada funcionalidade. A equipe deve perceber o que deve ser feito e deve escolher a melhor maneira de fazê-lo, modificando sua abordagem diariamente, a medida em que novas dificuldades, surpresas e complexidades são encontradas. Todo esse processo criativo é a chave para a produtividade do *Scrum*.

Existem três papéis no *Scrum*: O *Product Owner*, o *Scrum Master* e a equipe de desenvolvimento. As responsabilidades de gerenciamento do projeto são divididas entre esses três papéis.

O *Scrum Master* é o responsável pelo processo, é ele quem orienta todos os envolvidos no desenvolvimento do projeto a seguir as práticas do *Scrum*. Ele é o responsável por ensinar e implementar o *Scrum* de tal forma que ele se encaixe na cultura organizacional da empresa e ainda assim entregue os resultados desejados. É ele também é o responsável por assegurar que todas as regras do processo sejam seguidas.

O *Product Owner* é o responsável por representar todos os interessados no projeto e seu sistema resultante. Ele assegura o financiamento inicial e contínuo do projeto criando os requisitos gerais iniciais do sistema, objetivos de retorno sobre o investimento(ROI) e planos de lançamento. A lista de requisitos é chamada *Product Backlog*, o *Product Owner* é o responsável por utilizar a *Product Backlog* para assegurar que as funcionalidades mais importantes sejam desenvolvidas primeiro, e é ele quem prioriza frequentemente a *Product Backlog* colocando sempre no topo dela os requisitos mais importantes para a próxima iteração.

A equipe de desenvolvimento é responsável por ampliar as funcionalidades. As equipes são multifuncionais, auto gerenciadas e auto organizadas. Os membros da equipe são conjuntamente responsáveis pelo sucesso de cada iteração e do projeto como um todo.

Na seção 3.2.2 cada papel, suas respectivas funções e responsabilidades serão apresentadas mais detalhadamente.

Um projeto *Scrum* começa com a visão do sistema que será desenvolvido. Geralmente é uma visão vaga no começo, mas à medida que o projeto vai se desenvolvendo ela vai se tornando mais clara. O *Product Owner* é o responsável por dar a visão de uma maneira que maximize o ROI daqueles que estão financiando o projeto, é ele quem formula um plano para alcançar esse objetivo e dentro desse plano está incluído o *Product Backlog*, que é uma lista de requisitos funcionais e não funcionais que quando transformados em funcionalidade irão entregar o que foi concebido na visão.

A *Product Backlog* é priorizada de tal forma que os itens que tem mais chance de acrescentar valor ao sistema possuem prioridade maior. A *Product Backlog* é o ponto de partida, seu conteúdo e respectivas prioridades começam a mudar assim que o projeto se inicia, as alterações experimentadas por ela são reflexo de mudanças de requisitos voltados

aos negócios e de quão rápido ou devagar a equipe consegue transformar *a Product Backlog* em funcionalidade.

O *Scrum* estrutura o desenvolvimento em ciclos de trabalho chamados *Sprints*. A duração dessas iterações não ultrapassa 4 semanas e assim que uma termina, outra é iniciada, sem pausa. As *Sprints* são de tempo pré-fixado, terminam em uma data específica independente do trabalho estipulado ter sido concluído ou não, elas nunca são estendidas. Geralmente as *Scrum teams* escolhem uma dada duração para as *Sprints* e a utilizam dali em diante, até que eles se aprimorem e possam utilizar um ciclo mais curto.

Cada *Sprint* é iniciada com uma reunião denominada *Sprint planning meeting*, na qual a equipe de desenvolvimento e o *Product Owner* se reúnem para definir o que será feito na *Sprint* que está por vir. Essa reunião é dividida em duas partes e será detalhada mais adiante. O resultado da *Sprint planning meeting* é uma *Sprint goal* e uma *Sprint Backlog*. A *Sprint goal* é um objetivo que deve ser alcançado ao término da Sprint através da implementação dos requisitos da *Product Backlog*. A *Sprint Backlog* é composta pelos itens da *Product Backlog* selecionados para aquela Sprint, e pelo plano de como entregar essas funcionalidades.

No decorrer da *Sprint* são realizadas reuniões diárias de 15 minutos chamadas *Daily Scrum*. Durante essas reuniões cada membro da equipe de desenvolvimento responde três perguntas: O que você fez nesse projeto desde a última reunião *Daily Scrum*?, O que você planeja fazer entre agora e a próxima reunião *Daily Scrum*?, Quais impedimentos surgiram no seu caminho que possam fazer com que você não cumpra aquilo que foi estipulado para essa Sprint e para esse projeto?

Ao final de cada *Sprint* ocorre uma reunião chamada *Sprint review*. Nessa reunião a equipe de desenvolvimento apresenta aquilo que foi desenvolvido durante a *Sprint* para o *Product Owner* e qualquer outro *stakeholder* que queira comparecer. No andamento dessa reunião a funcionalidade desenvolvida é apresentada de forma informal, para que as pessoas possam avaliar e colaborar na determinação daquilo que a equipe de desenvolvimento deve fazer em seguida.

Após a *Sprint review* e antes da próxima *Sprint planning meeting*, o *Scrum Master* promove em conjunto com a equipe de desenvolvimento uma reunião chamada *Sprint restrospective*, na qual o *Scrum Master* encoraja o grupo a revisar seu processo de desenvolvimento para torná-lo mais efetivo e satisfatório na próxima Sprint.

O *Sprint planning*, *Daily Scum*, *Sprint review* e *Sprint retrospective* constituem as práticas de inspeção e adaptação empíricas do *Scrum*.

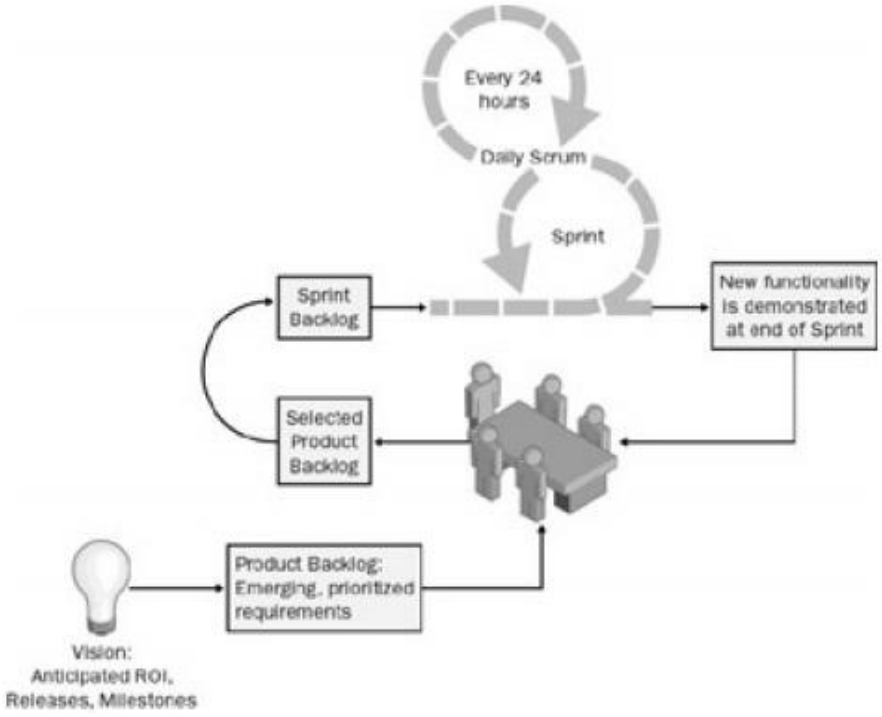

Figura 3.2 – Visão geral do processo Scrum (Schwaber 2004)

Os eventos descritos anteriormente serão explicados detalhadamente na seção 3.3.4.

3.2.2 Papéis

Como foi apresentado anteriormente, o *Scrum* possui três papéis: *Scrum Master*, equipe de desenvolvimento e *Product Owner*. Juntos, esses três papéis formam o que é conhecido como *Scrum Team*. Os *Scrum Teams* são multifuncionais e auto organizados. Equipes multifuncionais possuem as principais habilidades necessárias para cumprir o seu trabalho, sem precisar depender de pessoas de fora da equipe. O modelo de equipe do *Scrum* foi criado para otimizar a criatividade, flexibilidade e produtividade.

***Product Owner*:** Responsável por maximizar o valor do produto e o trabalho da equipe de desenvolvimento. Ele busca sempre alcançar o maior ROI possível identificando as funcionalidades do produto, convertendo essas funcionalidades em uma lista priorizada (*Product Backlog*) e decidindo o que deve ficar no topo dessa lista para a próxima *Sprint*.

Ele é a única pessoa responsável por gerenciar a *Product Backlog*, e esse gerenciamento inclui: expressar claramente os itens, ordená-los sempre buscando a melhor forma de atingir os objetivos, assegurar o valor do trabalho que a equipe de desenvolvimento

executa, garantir que a *Product Backlog* esteja sempre visível e clara para todos, e assegurar que a equipe de desenvolvimento compreenda no nível necessário os itens da *Product Backlog*.

No *Scrum* apenas uma pessoa pode assumir o papel de *Product Owner*. Qualquer um que deseje mudar as prioridades da *Product Backlog*, deve antes convencer o *Product Owner*, ele é a autoridade final no que diz respeito a priorização. Para que ele seja bem sucedido é preciso que suas decisões sejam respeitadas por todos dentro da organização, nenhuma outra pessoa está autorizada a mandar a equipe de desenvolvimento trabalhar em um conjunto diferente de requisitos.

Equipe de desenvolvimento: A equipe de desenvolvimento constrói o produto que o *Product Owner* indicar, ela decide quantos itens serão implementados em uma *Sprint*, e qual a melhor forma de cumprir seus objetivos.

Essas equipes são estruturadas e incentivadas pela organização a se auto gerenciarem e a organizarem o próprio trabalho. Cada membro da equipe de desenvolvimento recebe o título de desenvolvedor, o *Scrum* não adota nenhum título de especialista para os membros.

As equipes de desenvolvimento são multifuncionais e seus membros possuem todas as habilidades necessárias para a criação de um incremento do produto. Elas são auto organizadas e a responsabilidade pelo sucesso do trabalho é compartilhada de forma igual entre todos os seus membros.

Não existem subdivisões em tarefas específicas, como analise de negócios e testes. Cada membro da equipe tem seus pontos fortes, mas eles estão sempre aprendendo novas especialidades.

A equipe desenvolve o produto e ao mesmo tempo abastece o *Product Owner* com ideias de como melhorá-lo. Com relação ao tamanho, ela deve ser pequena o suficiente para permanecer ágil e grande o suficiente para conseguir completar um trabalho significável, seu tamanho pode variar de 5 a 9 membros.

***Scrum Master*:** O *Scrum Master* é o responsável por fazer com que todos aprendam e apliquem o *Scrum.* Ele faz isso assegurando que a *Scrum Team* siga as práticas, regras e teorias do *Scrum*. Ele não é o gerente dos membros da equipe, muito menos gerente de projeto, seu papel é servir a equipe. Ele ajuda a remover os impedimentos, protege a equipe de interferências de fora e os ajuda a adotarem práticas de desenvolvimento modernas.

Ele pode ajudar o *Product Owner* de várias maneiras: comunicando de forma clara a visão, objetivos e itens da *Product backlog* aos membros da equipe de desenvolvimento, encontrando técnicas efetivas para o gerenciamento da *Product backlog* e facilitando eventos *Scrum* quando for requisitado ou necessário.

O *Scrum Master* deve ensinar a equipe de desenvolvimento a se auto organizar e desenvolver sua multifuncionalidade, deve guiá-los para que eles sejam capazes de desenvolver produtos de alto valor, remover todos os entraves que estejam prejudicando-os, e treinar a equipe em ambientes organizacionais em que o *Scrum* ainda não tenha sido completamente compreendido e adotado.

No *Scrum,* definitivamente não existe o papel de gerente de projeto. As tradicionais responsabilidades do gerente de projeto são divididas entre os três papéis descritos anteriormente. As pessoas que preenchem esses três papéis são aquelas que têm compromisso com o projeto. Outras pessoas podem até ter interesse, mas elas não são diretamente responsáveis pelo projeto. O *Scrum* faz uma distinção clara entre esses dois grupos de pessoas e assegura que aqueles com responsabilidade direta tenham autoridade para fazer o que for necessário para o sucesso do projeto.

3.2.3 Artefatos

Os artefatos definidos pelo *Scrum* foram projetados para prover e maximizar a transparência de informações importantes, necessárias para que a *Scrum Team* seja bem sucedida no projeto.

***Product Backlog*:** É uma lista ordenada de tudo aquilo que possa ser necessário para o produto e é a única fonte de requisitos paras as modificações que serão feitas nele. Essa lista existe e evolui ao longo de todo o tempo de vida do produto. Apenas uma *Product Backlog* existe, o que significa que o *Product Owner* é responsável por tomar decisões de priorização considerando tudo aquilo que está contido nela, representando os interesses dos *stakeholders*.

A *Product Backlog* nunca está completa, ela evolui constantemente da mesma forma que o produto e o ambiente no qual ele será inserido evolui. Ela é composta por uma grande variedade de itens, principalmente novas funcionalidades do cliente. É também composta por importantes metas de melhoria do processo de engenharia, aprimoramento de objetivos, trabalho de pesquisa e detecção de erros. Os itens quem compõe a *Product Backlog* podem ser expressos por meio de *user stories, use cases* ou qualquer outra abordagem de requisitos que a equipe ache adequada.

Uma boa *Product Backlog* deve ser bem priorizada, os itens contidos nela são ordenados sequencialmente(1-N). Os itens do topo são os mais prioritários, esses itens devem trazer o maior *return on investment*(ROI) possível, devem ser mais refinados e detalhados do que os itens situados mais em baixo, tornando mais fácil para a equipe de desenvolvimento selecionar os itens que serão implementados na próxima *Sprint*.

Cada item deve possuir uma estimativa de esforço necessário para completá-lo. A equipe de desenvolvimento é a responsável por fazer as estimativas, a cada nova *Sprint* deve-se considerar reavaliar as estimativas, na medida em que novas informações vão surgindo e o aprendizado vai aumentando.

Constantemente, deve-se refinar a *Product Backlog*. A cada nova *Sprint* itens podem ser incluídos, retirados e repriorizados em resposta ao aprendizado e variabilidade do processo. O *Product Owner* deve estar sempre atento às novas demandas dos clientes, novas ideias, obstáculos técnicos que aparecerem e por aí vai.

Backlog Description	Initial Estimate	Adjust-ment Factor	Adjusted Estimate	work remaining until completion						
				1	2	3	4	5	6	7
Title Import				256	209	193	140	140	140	140
Project selection or new	3	0.2	3.6	3.6	0	0	0	0	0	0
Template backlog for new projects	2	0.2	2.4	2.4	0	0	0	0	0	0
Create product backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Create sprint backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Display tree view of product backlog, releases, sprints	2	0.2	2.4	2.4	0	0	0	0	0	0
Sprint-1	13	0.2	15.6	16	0	0	0	0	0	0
Create a new window containing product backlog template	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Create a new window containing sprint backlog template	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Burndown window of product backlog	5	0.2	6	6	6	0	0	0	0	0
Burndown window of sprint backlog	1	0.2	1.2	1.2	1.2	0	0	0	0	0
Display tree view of product backlog, releases, prints	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Display burndown for selected sprint or release	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Sprint-2	16	0.2	19.2	19	19	1.2	0	0	0	0
Automatic recalculating of values and totals	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
As changes are made to backlog in secondary window, update burndown graph on main page	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Hide/automatic redisplay of burndown window	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
Insert Sprint capability ... adds summing Sprint row	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Insert Release capability ... adds summary row for backlog in Sprint	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Owner/assigned capability and columns optional	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Print burndown graphs	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Sprint-3	14	0.2	16.8	17	17	17	0	0	0	0
Duplicate incomplete backlog without affecting totals	5	0.2	6	6	6	6	6	6	6	6
Note capability	6	0.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2
What-if release capability on burndown graph	15	0.2	18	18	18	18	18	18	18	18
Trend capability on burndown server	2	0.2	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4
Publish facility for entire project, publishing it as HTML web pages	11	0.2	13.2	0	0	13	13	13	13	13
Future Sprints	39	0.2	46.8	34	34	47	47	47	47	47
Release-1				86	70	65	47	47	47	47

Figura 3.3- Exemplo de *Product Backlog*(Schwaber 2004)

A figura 3.3 apresenta um exemplo de product backlog, com os itens ordenados de acordo a sua prioridade, cada item com a sua estimativa de esforço e quantitade de trabalho remanescente.

***Sprint Backlog*:** É composta pelos itens da *Product Backlog* selecionados para aquela *Sprint*, mais o plano de como entregar o incremento do produto e atingir os objetivos da *Sprint*. Os itens são analisados pela equipe de desenvolvimento e divididos em tarefas.

Ela é uma previsão feita pela equipe sobre quais funcionalidades serão acrescentadas ao produto no próximo incremento e o esforço necessário para conseguir entregar essas funcionalidades.

A *Sprint Backlog* é um plano com detalhamento suficiente para que seja possível observar os progressos durante a reunião *Daily Scrum*. Ela deixa visível para a equipe de desenvolvimento todo o trabalho que ainda deve ser feito para que o objetivo da *Sprint* seja alcançado.

A equipe de desenvolvimento é a única responsável pela *Sprint Backlog*, somente ela tem autoridade para acrescentar e remover itens nela. À medida que um determinado trabalho é completado, a estimativa do trabalho remanescente é atualizada. Quando elementos do plano são considerados desnecessários, eles são removidos.

Definição de "pronto": Quando um item da *Product Backlog* ou um incremento é descrito como "pronto", todos devem entender o que "pronto" significa. Antes que se inicie uma *Sprint*, o *Product owner*, a equipe de desenvolvimento e o *Scrum Master* devem definir o que é necessário para que um item da *Product Backlog* seja considerado "pronto". Dessa frma, todos devem compartilhar o mesmo entendimento de trabalho completo para assegurar a transparência.

Essa mesma definição orienta a equipe de desenvolvimento, a saber, quantos itens da *Product Backlog* ela é capaz de selecionar durante a *Sprint Planning*. O propósito da *Sprint* é entregar incrementos de *potentially releasable funcionality*, que sigam a definição de "pronto" da *Scrum team*. Cada incremento deve ser integrado ao produto e testado.

Uma boa definição de "pronto" diminui o perigo de atraso e risco no projeto. Uma *Scrum team* deve estar em constante evolução, o que se reflete em uma expansão e amadurecimento dessa definição.

3.2.4 Eventos

***Sprint planning meeting*:** É nessa reunião que a Sprint é planejada. Esse planejamento é feito através de um trabalho colaborativo que envolve toda a *Scrum team*. Essa reunião tem a duração pré-fixada de 8 horas para uma Sprint de 4 semanas, para *Sprints* menores a duração da reunião é proporcionalmente menor (2 horas por semana de *Sprint*).

A *Sprint planning meeting* é dividida em duas partes, o tempo disponível é dividido igualmente entre cada uma delas. A primeira parte serve para definir o que será feito ao longo da *Sprint* e a segunda para definir como realizar o trabalho.

***Sprint planning meeting* parte um:** Nessa parte a equipe de desenvolvimento trabalha para prever qual funcionalidade será desenvolvida ao longo da *Sprint*. O *Product Owner* e a equipe de desenvolvimento revisam os itens de maior prioridade contidos na *Product Backlog*. Normalmente esses itens prioritários já foram bem analisados na última *Sprint* durante o refinamento da *Product Backlog*.

A primeira parte é focada em compreender o que o *Product Owner* deseja, e por que aquilo que ele deseja é necessário. As decisões tomadas nessa parte são baseadas na *Product backlog*, no último incremento do produto, no desempenho da equipe de desenvolvimento na última *Sprint* e na capacidade de trabalho prevista para a próxima *Sprint*.

O número de itens da *Product Backlog* selecionados cabe apenas aos membros da equipe de desenvolvimento, apenas eles são capazes de avaliar o que pode ser completado durante uma *Sprint*.

Após a equipe de desenvolvimento fazer uma previsão de quais itens da *Product Backlog* serão transformados em funcionalidade, a *Scrum Team* cria uma *Sprint goal*. Essa meta é um resumo dos objetivos da *Sprint*, ela também dá a equipe de desenvolvimento uma certa flexibilidade com relação ao que eles devem entregar, embora eles possam ter que remover alguns itens durante a *Sprint*, deve haver o comprometimento em entregar algo tangível e "pronto" ao final, essa é a essência da meta da *Sprint*.

***Sprint planning* parte dois:** Tendo selecionado o trabalho para a *Sprint*, a equipe de desenvolvimento planeja como transformar essa funcionalidade em um incremento "pronto" do produto. Nessa etapa os itens da *Product Backlog* são decompostos e divididos em tarefas, a lista de trabalho a ser feito resulta na *Sprint Backlog*. O *Product Owner* não precisa estar presente durante essa parte, mas ele deve estar disponível pelo menos via telefone.

A equipe de desenvolvimento geralmente inicia projetando o sistema e planejando o trabalho que ela acredita conseguir completar ao longo de uma *Sprint*. Até o fim dessa reunião os trabalhos planejados para os primeiros dias da *Sprint* são decompostos em unidades menores, com duração estimada de um dia ou menos.

Ao fim da *Sprint planning meeting* a equipe de desenvolvimento estabelece um objetivo realista para aquilo que eles acreditam que podem entregar ao fim da *Sprint*, além disso, eles devem ser capazes de explicar para o *Product Owner* e para o *Scrum Master* como pretendem se auto organizar e trabalhar para conseguir cumprir a *Sprint goal*.

Um dos pilares do Scrum reside no fato de que a partir do momento que a equipe de desenvolvimento estabelece um objetivo para a *Sprint*, quaisquer acréscimos ou alterações devem ser adiados para a próxima *Sprint*. O *Product Owner* não tem autoridade para atribuir tarefas para a equipe de desenvolvimento no meio de uma *Sprint*, ele deve aguardar até a próxima *Sprint planning meeting* para fazer isso. O *Product Owner* tem autoridade para cancelar a *Sprint* caso ache necessário, mas isso raramente ocorre. Motivos como mudanças nos negócios ou outras circunstâncias externas que tornem os objetivos da *Sprint* obsoletos, podem levá-lo a tomar essa decisão.

***Daily Scrum*:** São reuniões diárias, com duração máxima de 15 minutos, que servem para que a equipe de desenvolvimento possa sincronizar o seu trabalho e estabelecer um plano para as próximas 24 horas. Ela é uma oportunidade para a equipe inspecionar e adaptar o seu trabalho ao longo da *Sprint*.

Para que a *Daily Scrum* não ultrapasse o tempo máximo, recomenda-se que as pessoas da equipe permaneçam em pé e que o local da reunião seja o mesmo todos os dias. O *Scrum Master* é responsável por assegurar que a *Daily Scrum* aconteça todos os dias. Durante essa reunião são respondidas três perguntas:

- O que você fez nesse projeto desde a última *Daily Scrum*?
- O que você planeja fazer entre agora e a próxima reunião *Daily Scrum*?
- Quais impedimentos surgiram no seu caminho?

Caso existam impedimentos, o *Scrum Master* é responsável por ajudar a equipe a resolvê-los, fazendo tudo o que estiver a seu alcance para tornar o trabalho da equipe mais efetivo.

A equipe de desenvolvimento usa a *Daily Scrum* para avaliar o progresso com relação a *Sprint goal*, e essa reunião otimiza a probabilidade da equipe conseguir alcançar a meta. Esse encontro melhora a comunicação, elimina o tempo que seria gasto em outras reuniões, remove impedimentos e promove a auto organização da equipe.

***Sprint Burndown Chart*:** Para poder avaliar como está indo o progresso do trabalho, cada membro da equipe de desenvolvimento está sempre atualizando na *Sprint Backlog* sua estimativa do tempo remanescente para que um determinado trabalho seja completado. É comum também que se atualize o trabalho remanescente para que a equipe como um todo atinja o objetivo da Sprint. O *Sprint Burndown chart* é ótima forma de ilustrar o progresso da equipe até a meta estabelecida. Esse gráfico é útil por mostrar o progresso, não em termos de quanto tempo foi gasto no passado, mas sim em termos de quanto trabalho ainda resta pela frente, o que ajuda na transparência do processo.

***Sprint review*:** É uma reunião que acontece ao término da *Sprint*, nela estão presentes o *Product Owner, Scrum Master*, equipe de desenvolvimento, clientes, usuários e quaisquer outras pessoas que tenha interesses no projeto. Ela tem duração pré-fixada de 4 horas para uma *Sprint* de 4 semanas, em *Sprints* menores o tempo é alocado de forma proporcional (1 hora por semana de *Sprint*).

Ela é uma reunião informal na qual a equipe *Scrum* e os *stakeholders* analisam o que foi feito durante a *Sprint*. Baseando-se nisso e em qualquer mudança na *Product Backlog* que tenha sido feita durante a *Sprint*, os participantes trabalham em conjunto para avaliar o que deve ser feito a seguir.

Durante essa reunião a equipe de desenvolvimento discute como os problemas que surgiram ao longo da *Sprint* foram resolvidos e quais experiências foram bem sucedidas. A equipe então demonstra o trabalho que foi feito para os que estiverem presentes na reunião e responde eventuais perguntas.

Na *Sprint review, o Product Owner* é capaz de identificar o que foi "feito" e o que não foi "feito", a partir daí o grupo discute quais serão os próximos passos. O resultado dessa discussão serve como base para a próxima *Sprint planning meeting*.

A *Sprint review* serve para inspecionar o incremento e adaptar o produto, o seu resultado é uma versão revisada da *Product Backlog* que define os itens que provavelmente serão trabalhados na próxima *Sprint*.

***Sprint retrospective*:** A *Sprint review* serve para a equipe *Scrum* inspecionar e adaptar de acordo com o produto, já *Sprint retrospective* envolve inspecionar e adaptar de acordo com o ambiente e o processo.

Ela tem duração pré-fixada de 3 horas para uma Sprint de 4 semanas, e em *Sprints* menores o tempo é alocado de forma proporcional. Essa reunião ocorre logo após a *Sprint review*. Nela estão presentes o *Scrum Master*, a equipe de desenvolvimento e o *Product Owner*, a presença desse último é opcional, porém, recomendável.

No decorrer da *Sprint retrospective* a equipe deve:

- Inspecionar como foi a última *Sprint* no que diz respeito a processos, ferramentas e pessoas;
- Identificar e ordenar os principais itens que foram bem e potenciais aprimoramentos;
- Criar um plano para implementar aprimoramentos que melhorem o trabalho da *Scrum team* nas próximas *Sprints*.

Embora aprimoramentos no processo possam ser implementados a qualquer momento no *Scrum*, a *Sprint retrospective* é uma oportunidade formal para isso, trazendo benefícios para o processo ao fim de cada iteração.

3.3 DESAFIOS

O *Scrum* é simples. Seus processos, práticas, regras e artefatos são poucos e fáceis de aprender, o problema é que essa simplicidade pode ser enganosa. O *Scrum* não é um processo prescritivo, ele não descreve como proceder em todas as circunstâncias (SCHWABER, 2004). Ele oferece um *framework* e um conjunto de práticas que mantêm tudo visível, permitindo que todos os envolvidos no processo possam observar e fazer as mudanças necessárias para melhorá-lo.

Muitas das pessoas responsáveis por gerenciar projetos, sempre utilizaram abordagens que fazem uso de planos detalhados, gráficos de Gantt e cronogramas de trabalho. Para muitas dessas pessoas o *Scrum* não é intuitivo, por outro lado, elas rapidamente captam o espírito do *Scrum* e entendem como ser bem sucedido fazendo o uso desse método (SCHWABER, 2004).

Um erro comum cometido por aqueles que tentam implementar o *Scrum* é tentar modificar princípios básicos, como por exemplo, estender a duração de uma *Sprint*. Se a Sprint nunca acaba, o tempo nunca se esgota. As previsões nunca serão aprimoradas!

Outro erro comum é assumir que determinada prática é desencorajada pelo Scrum apenas pelo simples fato do Scrum não descrevê-la. Esse método deixa a cargo dos indivíduos tomar a decisão específica mais apropriada em determinada circunstância.

Para que o *Scrum* dê certo é preciso que a equipe tenha o espaço e as ferramentas necessárias para se auto gerenciar. Muitas empresas possuem uma cultura na qual as decisões vêm sempre de cima para baixo. Querer impor o *Scrum* ou não permitir que a equipe tome suas próprias decisões é um determinante fator que contribui para o fracasso do processo.

As primeiras *Sprints* costumam ser as mais difíceis (SCHWABER, 2004), muitas das vezes a meta da *Sprint* não é alcançada, mas é justamente a inspeção e adaptação que vai tornando o *Scrum* um processo cada vez mais ajustado. Um resultado negativo no começo do processo não significa fracasso. Ao longo do tempo a equipe aprende a fazer estimativas melhores e é capaz de observar uma notável evolução do seu desempenho.

Quando o projeto é muito grande, uma só equipe não consegue dar conta da enorme quantidade de trabalho. Para resolver esse problema o *Scrum* deve ser escalonado, nesses casos múltiplas equipes devem trabalhar em paralelo. Quando múltiplas equipes trabalham em um único projeto, mecanismos especiais devem ser utilizados para manter a sincronia entre elas.

Cada projeto escalonado apresenta suas próprias complexidades, de acordo com o caso, a técnica utilizada para manter o funcionamento do processo pode variar. Nesse tipo de projeto é mais fácil haver uma perda na capacidade de comunicação dos envolvidos, além de uma diminuição da transparência do processo como um todo.

Uma técnica muito utilizada para minimizar esses problemas em projetos *Scrum* escalonados é o *Daily Scrum of Scrums*, uma reunião que ocorre logo após o *Daily Scrum* das equipes. Nesse encontro, um representante de cada uma se reúne com os outros e utiliza o tempo disponível para sincronizar o trabalho delas. Outras técnicas também são usadas, e a cada dia que passa novas práticas vem sendo criadas e implementadas no escalonamento do *Scrum* dentro de empresas brasileiras e de fora. O importante é analisar o caso em questão e encontrar a melhor forma de fazer com que os princípios do *Scrum* sejam colocados em prática.

No capítulo seguinte, o escalonamento do *Scrum* será tratado com mais detalhes, e um conjunto de procedimentos e boas práticas de engenharia serão propostos para que o processo tenha uma maior probabilidade de sucesso e de retorno sobre o investimento.

4. ESCALONANDO O SCRUM

4.1 RAZÕES PARA ESCALONAR O SCRUM

Os métodos ágeis foram originalmente projetados de modo a comportar pequenas e co-localizadas equipes, mas devido a sua grande popularidade, nos últimos tempos eles tem sido utilizados por grandes companhias em projetos de desenvolvimento de software que empregam múltiplas equipes distribuídas entre diversas localizações geográficas.

Para empresas grandes, contar com os serviços de pessoas trabalhando em diversos locais diferentes, seja através de terceirização ou da criação de filiais, é uma forma de aumentar o recrutamento de pessoal capacitado para determinado tipo de trabalho e reduzir custos na contratação de novos funcionários (SUTHERLAND, 2007).

Embora o escalonamento de métodos ágeis como o *Scrum* traga diversos desafios, a capacidade de resposta às mudanças de requisitos, o aumento na produtividade e a maior satisfação do pessoal da área de negócios, são fatores determinantes para que essas empresas se aventurem nessa empreitada.

De acordo com uma pesquisa (VERSIONONE, 2012), 35% dos entrevistados trabalham em uma empresa que possui equipes distribuídas utilizando métodos ágeis. Essa prática relativamente recente, tem ganhado cada vez mais popularidade dentro da indústria de desenvolvimento de software.

4.2 PRINCIPAIS DIFICULDADES

A separação física das equipes de desenvolvimento traz uma série de desafios, muitos dos aspectos chave do *Scrum*, como interação direta com o cliente e a comunicação face-a-face entre as equipes não se aplicam nesse caso.

Para que se continue obtendo os benefícios do *Scrum*, suas práticas devem ser estendidas e modificadas, sempre objetivando melhorar a comunicação e sincronizar o trabalho das equipes.

Uma das únicas práticas recomendadas pelo *Scrum* para a coordenação e sincronização das equipes é uma reunião chamada *Scrum of Scrums*, com formato muito parecido com o

Daily Scrum. Nessa reunião, representantes de cada uma das equipes respondem três perguntas:

- O que a sua equipe fez desde a última reunião que possa vir a ser importante para alguma outra equipe?
- O que a sua equipe irá fazer a seguir que possa ser relevante para alguma outra equipe?
- Existe algum impedimento no caminho da sua equipe que possa vir a atrapalhar alguma outra equipe ou que necessite de ajuda de uma delas?

Embora existam relatos do uso do *Daily Scrum* como forma de administrar as complexidades e problemas gerados pela diversidade de equipes e pela distância entre elas, em muitos casos essa reunião acaba sendo mal sucedida, sendo considerada por muitos envolvidos no projeto como inútil e até mesmo uma perda de tempo (PAASIVAARA; LASSENIUS; HEIKKILÄ, 2012).

À medida que o número de equipes aumenta, a duração da reunião tem que ser cada vez maior, para que cada representante seja capaz de responder as perguntas. Outro problema relatado é a falta do que reportar, muitas vezes os representantes estão mais preocupados com os problema internos da sua equipe e evitam expor ou participar de problemas que possam estar relacionados com outras equipes.

Só o *Scrum of Scrums* é insuficiente para que se consiga coordenar e manter o trabalho sincronizado em um projeto com equipes geograficamente distribuídas.

Outro grande desafio nesses casos é manter meios de comunicação confiáveis e eficazes, a comunicação e consequentemente a cooperação entre as equipes depende desses meios.

4.3 MODELO PROPOSTO

O desenvolvimento desse modelo de escalonamento do *Scrum*, em empresas com equipes geograficamente distribuídas, foi baseado na análise e estudo qualitativo de diversos artigos e relatos práticos de experiências reais dentro desse contexto.

A base para o modelo proposto são os casos de sucesso no escalonamento do *Scrum* na empresa Universo Online (MARANZATO; NEUBERT; HERCULANO, 2012) e Microsoft (WILLIAMS, 2011). Uma série de modificações e novas proposições foram feitas visando

adaptar as técnicas desenvolvidas para uma empresa com equipes trabalhando em locais diferentes, aliando boas práticas de engenharia com um método *Scrum* adaptado para esse tipo situação.

O resultado é uma versão do *Scrum* estendida e modificada, composta por um conjunto de estratégias, reuniões complementares e práticas de engenharia que tem como objetivo uma implementação bem sucedida no tipo de ambiente proposto.

4.3.1 Estratégias

Para que o *Scrum* possa ser implementado em um ambiente de trabalho com equipes trabalhando em diferentes locais, oito estratégias são propostas para minimizar as dificuldades que costumam surgir nesse tipo de situação:

- **Divisão de equipes por localidade**

As equipes devem ser formadas localmente, ou seja, todos os seus membros devem trabalhar no mesmo local e cada equipe deve ser responsável por uma funcionalidade, ao invés de um componente de software.

Isso ajuda a reduzir a dependência entre as equipes e diminui a complexidade da sincronização do trabalho de todas elas. Problemas como a falta de horas de trabalho sobrepostas entre os diversos locais e a questão da falta de comunicação em tempo real também são minimizados.

- **Múltiplos canais de comunicação**

Uma questão crucial para o sucesso do *Scrum* é uma boa comunicação entre os envolvidos no projeto, no caso de equipes geograficamente distribuídas, isso pode muitas vezes ser tornar um problema (PAASIVAARA; LASSENIUS; HEIKKILÄ, 2012).

O ideal é disponibilizar um rico ambiente de comunicação entre as equipes através de vídeo conferência, telefone, SMS, e-mail, chats e *wikis*. Essa diversidade é importante para que os envolvidos no projeto se sintam mais próximos e tenham uma maior facilidade para realizar as reuniões necessárias.

Deve-se evitar meios de comunicação lentos e pouco confiáveis, que possam vir a atrapalhar a coordenação, a colaboração e a resolução de problemas.

- **Ambiente de trabalho adequado**

Um ambiente de trabalho adequado promove uma melhor comunicação e cooperação, além de ser ideal para a realização das reuniões exigidas para o sucesso do projeto.

Caso a equipe não trabalhe em uma mesma sala, é importante que exista um ambiente de integração, para que seus membros possam estar sempre se comunicando e realizando as reuniões necessárias.

Uma sala de reuniões dedicada, com equipamentos para vídeo conferência, telefones e computadores, também é muito útil para que haja a comunicação com as equipes que estão em outros locais de trabalho.

- **Janelas de sobreposição nas jornadas de trabalho**

Embora a divisão de equipes por localidade reduza a dependência entre os diversos locais de trabalho, algumas reuniões, que visam à coordenação e a sincronização de todas as equipes, devem acontecer.

É importante que existam algumas janelas de sobreposição na jornada de trabalho dos diferentes locais para que essas reuniões possam ser efetuadas.

Em casos extremos de grandes diferenças de fusos horários, a jornada de trabalho das equipes de um determinado local deve ser modificada, para se estender um pouco mais durante a noite alguns dias na semana.

- ***Backlog* centralizada**

É importante que exista uma *backlog* centralizada com temas ainda não endereçados a nenhuma das equipes, essa *backlog* deve ter acesso global promovido através de uma ferramenta de gerenciamento de *backlogs* que tenha essa funcionalidade.

Essa *backlog* centralizada é um instrumento para manter a visão geral do projeto, e deve ser discutida constantemente com a área de negócios, pois pode levar a repriorizações ou a criação de novas equipes.

- **Treinamento adequado**

É recomendável que as equipes sejam treinadas adequadamente antes que a empresa implemente o Scrum. A partir do momento que as equipes estiverem bem treinadas, novas equipes poderão ser formadas com a divisão de equipes experientes que receberão novos membros.

Os membros mais experientes irão transmitir conhecimento para os membros novos, tornando mais rápido e fácil o aprendizado. É importante deixar a equipe imutável durante um período de tempo de adaptação, para que possa ocorrer a transferência de conhecimento. Mudanças muito frequentes podem trazer mais malefícios do que benefícios.

- **Visitas frequentes aos diversos locais de trabalho**

Sempre que necessário é importante que pessoas envolvidas no projeto, como *Product Owners*, viajem para os outros locais de trabalho, isso reduz as diferenças culturais, melhora a comunicação e aumenta a visão do projeto.

- **Adaptação contínua**

Cada projeto tem suas peculiaridades, e quando envolve diversos locais de trabalho separados por grandes distâncias, a complexidade aumenta bastante.

É importante estar sempre criando e adaptando novas soluções para que aquele projeto em particular seja bem sucedido. Esse é um processo contínuo que busca sempre uma melhora nos resultados.

4.3.2 Reuniões Adicionais

Além das tradicionais reuniões já prescritas pelo *Scrum*, para suportar o escalonamento e equipes distribuídas em diversas localidades, são necessárias novas reuniões que tem como objetivo coordenar e sincronizar o trabalho de todas elas.

Todas essas reuniões dependem da qualidade dos meios de comunicação disponíveis, a estratégia de múltiplos canais de comunicação é crucial para a realização delas.

- ***Sprint planning* geral**

Essa reunião ocorre após a *Sprint* planning de cada uma das equipes, um ou dois representantes de cada equipe são enviados para participar dela.

Nessa reunião é discutido o cronograma de lançamentos de cada equipe e suas *user stories*. Durante essa reunião cada representante de equipe explica suas *user stories* para os outros, permitindo que todas as equipes tenham ideia do que as outras estão fazendo.

- ***Scrum of Scrums***

Como foi explicado na seção 4.2, reuniões do tipo *Scrum of Scrums* costumam ser problemáticas e muitas vezes pouco efetivas.

Para evitar perda de tempo desnecessária a *Scrum of Scrums* deve ser realizada somente uma vez, no meio da Sprint para que cada equipe tenha ideia do que a outra está fazendo e possa sincronizar seu trabalho com as outras.

Durante essa reunião cada representante de uma equipe responde duas perguntas:

-O que sua equipe já fez até agora?

-Sua equipe possui algum obstáculo que possa vir a atrapalhar as outras equipes?

É recomendável que os participantes já tenham essas respostas escritas antes da reunião para que ela seja breve.

- ***Sprint restrospective* geral**

Participam dessa reunião os *Scrum Masters* e um membro de cada equipe. Nela, cada representante traz uma série de impedimentos locais organizados por prioridade, então uma lista global é formada e todos os impedimentos que são identificados como pertencentes a mais de uma equipe são discutidos.

A frequência com a qual essa reunião deve ocorrer varia de acordo com a natureza do projeto, de modo geral, a cada 3 *Sprints* é interessante que ela ocorra.

- ***Sprint review***

A *Sprint Review* continua basicamente a mesma que o Scrum prescreve, a diferença é que para o caso de várias equipes distribuídas, um link de vídeo conferência deve estar disponível para que representantes de outras equipes que estiverem interessados possam assistir a reunião e observar como está o andamento do trabalho como um todo.

- **Reunião semanal entre *Scrum Masters* e *Product Owners***

Nessa reunião participam todos os *Scrum Masters* e *Product Owners*. Nesse momento são discutidos impedimentos, as funcionalidades que estão sendo desenvolvidas e as mudanças na *backlog* que estão sendo planejadas para as próximas *Sprints*.

Essa reunião tem como objetivo aumentar a sincronização entre o trabalho das equipes e resolver pendências que muitas vezes influenciam o bom andamento do projeto.

4.3.3 Práticas de Engenharia

Para evitar problemas como bugs excessivos e códigos de baixa qualidade, algumas práticas de engenharia são recomendadas.

Projetos que envolvem várias equipes precisam de um cuidado especial, com o que diz respeito a qualidade e consistência das funcionalidades que são produzidas. Caso não se tome cuidado, ao longo prazo o produto pode se tornar problemático e insustentável.

- **Integração contínua**

Nessa prática os membros das equipes integram frequentemente o seu trabalho à versão principal do sistema que está em construção. Durante cada integração testes são realizados de forma automática por ferramentas, para que erros de integração sejam detectados o mais breve possível. Todos os testes devem passar para que a integração seja realizada com sucesso.

- ***Test driven development*(TDD)**

É uma prática que se baseia na repetição de um ciclo curto de desenvolvimento, na qual o desenvolvedor escreve unidades de testes automatizadas (inicialmente falhas) e depois

produz um código que passe no teste. Essa prática elimina diversos defeitos e torna a qualidade do código melhor, resultando muitas vezes em uma economia de tempo que seria gasto *debugando* código.

- **Cobertura do código**

É recomendável que 80% do código seja coberto por testes de unidade automatizados. Essa prática não elimina todos os bugs, mas erros de lógica podem ser facilmente detectados.

- **Revisão do código**

Quando o código é revisado por outro desenvolvedor, a quantidade de erros estruturais e de omissão diminui. Essa prática melhora a qualidade do sistema ao detectar erros que muitas vezes passariam despercebidos.

- **Suporte de ferramentas**

A implementação do *Scrum* em um projeto envolvendo diversas equipes distribuídas requer um conjunto de ferramentas que sirvam de suporte para melhorar a comunicação, gerenciamento de projeto, rastreamento de *bugs*, *backlog* acessível e etc.

4.4 RESULTADOS ESPERADOS

4.4.1 Melhorias

Ao estender o conjunto de práticas que o *Scrum* sugere, é possível usufruir dos benefícios que esse método de desenvolvimento ágil pode oferecer, entre esses benefícios podemos citar:

- Maior capacidade de administrar mudanças nos requisitos;
- Aumento de produtividade;
- Maior satisfação do cliente;
- Melhora na comunicação entre os profissionais envolvidos nos projetos;
- Aprimoramento constante das habilidades multifuncionais desses profissionais.
- Maior ROI.

Além dos benefícios obtidos com o *Scrum*, ao fazer uso das boas práticas de engenharia, a empresa consegue obter resultados melhores ainda.

Se seguidas corretamente, a qualidade do código e, consequentemente, do sistema produzido, melhora bastante, economizando tempo e dinheiro que seriam gastos corrigindo *bugs* e erros de programação.

O cuidado com a qualidade do código produzido também evita que os meios de comunicação, tão importantes no caso de equipes distribuídas, sejam usados em demasia para a resolução de problemas que poderiam ser evitados caso as recomendações fossem seguidas.

4.4.2 Desafios

Os principais desafios estão relacionados com a comunicação entre as equipes, as reuniões adicionais são imprescindíveis para a coordenação e sincronização do trabalho como um todo.

Caso os meios de comunicação falhem, as chances de ocorrerem prejuízos devido à falta de sincronização aumentam bastante, por isso é importante dar atenção especial a essa questão.

Diferenças culturais entre as equipes também podem tornar o trabalho mais complicado. É importante que ocorram visitas esporádicas de funcionários aos outros locais de trabalho para que haja a interação cultural e diminuição de barreiras causadas pela distância.

A diferença de fuso horário e, consequentemente, a dificuldade em estabelecer jornadas de trabalho com janelas sobrepostas, também é um fator a ser contornado, seja com jornadas de trabalho estendidas ou com reuniões em horários informais.

A falta de treinamento adequado e o descumprimento das práticas de engenharia recomendadas podem trazer malefícios durante o desenvolvimento de um produto, por isso é importante estar sempre atento a essas questões.

4.5 COLOCANDO EM PRÁTICA O ESCALONAMENTO

A empresa estudada nessa seção desenvolve sistemas de controle financeiro e cobranças. Originalmente sediada em São Paulo, recentemente a empresa expandiu os negócios abrindo uma filial em Brasília e outra no Rio de Janeiro.

O *Scrum* já vinha sendo utilizado na matriz a alguns anos, vários aspectos positivos foram observados em comparação com o antigo processo de desenvolvimento baseado em RUP que era utilizado.

A rápida resposta às mudanças nos requisitos entre uma *Sprint* e outra, fez com que a dinâmica da empresa mudasse, e muito tempo que antes era gasto em um longo processo de gerenciamento de requisitos passou a ser usado no desenvolvimento do produto em si.

Os desenvolvedores que antes trabalhavam somente de acordo com a sua especialidade passaram a desenvolver habilidades multifuncionais, e a sensação de responsabilidade coletiva trazida pelo *Scrum* fez com que o empenho aumentasse e o espírito de equipe crescesse no ambiente de trabalho.

No início da implementação do *Scrum* apenas uma equipe foi formada na matriz, sua missão era testar o *Scrum* em um projeto piloto relativamente menor, um dos gerentes da empresa fez um curso de certificação *Scrum Master* e posteriormente os sete membros da equipe de desenvolvimento receberam treinamento onde aprenderam o funcionamento e as regras do *Scrum*.

Conforme os envolvidos no projeto iam se adaptando ao novo processo, melhoras notáveis no desempenho da equipe e nos resultados do projeto eram observados, tanto a produtividade da equipe quanto a satisfação do cliente aumentaram.

Para expandir o *Scrum* dentro da empresa a primeira equipe, já experiente, teve seus membros distribuídos entre três novas equipes com seis membros cada. Conforme a demanda e a contratação de funcionários novas equipes iam sendo formadas, sempre usando a ideia de dividir para expandir, com três ou quatro funcionários experientes e já familiarizados com o *Scrum* removidos de suas respectivas equipes para a criação de uma nova.

Em meados de 2012 a matriz já contava com seis equipes e cada filial com duas, embora mais de uma equipe pudesse se envolver em um mesmo projeto, equipes de localidades diferentes ainda não haviam trabalhado juntas.

Com o surgimento cada vez mais frequente de projetos grandes e complexos, a diretoria da empresa decidiu que já era hora de tentar colocar várias equipes dos diferentes locais para trabalharem em um mesmo projeto.

Três equipes de São Paulo e as duas de Brasília fizeram parte desse projeto piloto com equipes distribuídas geograficamente, a estratégia dos gerentes era aplicar as mesmas práticas prescritas pelo *Scrum*, fazendo o uso de videoconferência, telefone e e-mails para manter a comunicação entre as equipes.

Inicialmente cada equipe ficou responsável por um componente do sistema que estava sendo desenvolvido, cada equipe possuía uma *backlog* própria e realizava suas reuniões conforme as regras do *Scrum*, o único evento utilizado para coordenação e sincronização das equipes era o *Scrum of Scrums*.

Após a primeira *Sprint* já era possível notar as dificuldades trazidas pela falta de coordenação, muitas dependências entre os componentes faziam com que os problemas

enfrentados por uma das equipes acabasse atrasando o trabalhos das outras, vários gargalos iam sendo formados.

Durante a *Scrum of Scrums*, poucas equipes reportavam problemas e quase nunca se discutia uma solução para os poucos que eram apresentados, a maior parte dos envolvidos no projeto achavam essas reuniões uma perda de tempo.

Observando os atrasos e a falta de coordenação a diretoria resolveu suspender o projeto por um tempo e pensar em maneiras de minimizar os problemas observados durante a primeira tentativa.

Ficou decidido que a empresa estenderia as práticas prescritas pelo *Scrum* com o conjunto de estratégias, reuniões adicionais e práticas de engenharia descritas na seção 4.3 para tentar escalonar de forma bem sucedida o *Scrum* em um projeto envolvendo todas as filiais.

Algumas das estratégias como ambiente de trabalho adequado, divisão de equipes por localidade, jornada de trabalho sincronizada e treinamento adequado já faziam parte da cultura da empresa.

Com as estratégias complementares, em especial a *backlog* centralizada e a disponibilização de meios de comunicação confiáveis e eficientes para as equipes, a transparência do processo e capacidade de coordenação das equipes aumentou bastante.

As reuniões adicionais trouxeram novas oportunidades para as equipes sincronizarem os trabalhos, já na *Sprint planning* geral era possível que cada equipe soubesse um pouco sobre o cronograma e o trabalho das outras, dando uma visão geral melhor do projeto como um todo.

A reunião semanal entre *Scrum Masters* e *Product Owners* acabou se mostrando ótima para sincronizar o trabalhos das equipes, de forma mais eficiente que a *Scrum of Scrums.*

Já nas primeiras *Sprints* após a implementação das modificações, foi possível notar a diminuição de problemas relacionados a dependências entre os componentes das equipes, e uma maior produtividade devido a diminuição dos gargalos.

Cada equipe agora tinha uma visão geral do que estava acontecendo, as *Sprint review* eram realizadas em uma sala com um link de videoconferência, para que membros de outras equipes e qualquer interessado pudesse acompanhar o que estava sendo feito pelas outras.

As práticas de engenharia, como integração contínua, cobertura de pelo menos 80% do código com testes de unidade automatizado, e a revisão por pares do código produzido reduziam muito o tempo gasto na configuração do sistema e na correção de bugs.

Embora a produtividade tenha diminuído um pouco nas primeiras *Sprints*, rapidamente as equipes se adaptaram e reverteram essa situação, à medida que todos iam se familiarizando com o novo processo menos tempo era gasto desnecessariamente nas reuniões, e mais assuntos relevantes eram colocados em pauta.

Atualmente a distância entre os locais de trabalho já não é um fator limitante, as melhoras feitas no processo permitiram que agora qualquer equipe possa fazer parte de um determinado projeto, independentemente da localidade.

O grau de satisfação dos funcionários aumentou bastante, a matriz que antes sofria com sobrecarga de trabalho agora tem o auxílio das filiais na maior parte dos projetos, frequentemente funcionários da matriz visitam a filial, para aumentar o contato direto e a troca de experiências, melhorando a integração e a difusão de conhecimento dentro da empresa.

A versão estendida do *Scrum*, associada com algumas estratégias e boas práticas de engenharia se mostrou bastante eficaz para contornar os problemas trazidos pelo escalonamento dentro de uma empresa com equipes distribuídas em locais de trabalho distantes.

Muitas empresas que passam por dificuldades na hora de coordenar equipes distribuídas podem fazer uso desse conjunto de práticas para tornar o processo de desenvolvimento mais produtivo, embora os ambientes de trabalho e os tipos de projetos desenvolvidos variem muito de empresa para empresa, parte dos problemas observados ao escalonar o *Scrum* podem ser semelhantes, e consequentemente compartilharem a mesma solução.

5. CONCLUSÃO

Os métodos ágeis estão sendo cada vez mais utilizados dentro da indústria de desenvolvimento de software.

O *Scrum* é o método ágil mais popular. Originalmente foi criado para ser utilizado em empresas com pequenas equipes trabalhando em um mesmo local de trabalho, e hoje em dia é usado por grandes empresas que possuem diversas equipes trabalhando em locais geograficamente distribuídos.

Esse novo contexto traz uma série de dificuldades. Para contornar os problemas e poder usufruir dos benefícios trazidos pelo *Scrum*, suas práticas devem ser estendidas e modificadas.

Um dos fatores cruciais no sucesso do *Scrum* é a comunicação. No caso de equipes separadas por grandes distâncias, ela pode ser tornar um problema. Felizmente hoje em dia contamos com diversos meios de entendimento eficientes, capazes de “diminuir” as distâncias e aproximar as partes envolvidas em determinado projeto.

A cada dia que passa novas técnicas e modelos de escalonamento do *Scrum* são propostos, e à medida que novas descobertas vão sendo feitas, os resultados obtidos pelas empresas que fazem uso destas técnicas vão melhorando (VERSIONONE, 2012).

As soluções propostas para enfrentar os problemas que surgem ao escalonar o Scrum, tem como objetivo minimizar as consequências trazidas pela distância entre as equipes envolvidas em um mesmo projeto, tornando o *Scrum* mais suscetível a esse tipo de situação.

As reuniões adicionais trazem a comunicação extra que é necessária para manter a sincronização entre as equipes, as estratégias visam diminuir a complexidade causada pela falta de comunicação em tempo real e separação física das pessoas envolvidas no projeto e as práticas de engenharia recomendadas diminuem o tempo gasto resolvendo problemas evitáveis.

5.1 ESTUDOS FUTUROS

Novos estudos de caso que possam obter resultados quantitativos através da aplicação do modelo proposto nesse trabalho esclareceriam muitas questões e trariam informações valiosas. Cada empresa possui suas particularidades e os projetos realizados por uma mesma empresa podem variar bastante ao longo do tempo, embora os resultados obtidos com o uso

do modelo proposto possam variar, sua a aplicação sistemática pode deixar claro seus pontos positivos e negativos. Embora existam relatos de versões escalonadas do *Scrum*, aliadas a práticas específicas de engenharia, poucos estudos foram feitos sobre esse tema específico.

REFERÊNCIAS BIBILIOGRÁFICAS

ABRAHAMSSON, P.; SALO, O.; RONKAINEN, J.; WARSTA, J. (2002). Agile Software Development Methods: Review and Analysis. VTT Publications 478, 2002.

ABRAHAMSSON, P. et al. (2003). *New Directions on Agile Methods: A Comparative Analysis.* Proceedings of ICSE'03, 244-254.

AMBLER, S. Agile Modeling: Effective Practices for EXtreme Programming and the Unified Process, Wiley; 1 edition (April 4, 2002), 400 pages, ISBN-10: 0471202827.

BECK, K. *Extreme Programming Explained: Embrace Change*. Boston, MA: [s.n.], 1999. 157 p. ISBN 0-321-27865-8.

BOEHM B.; TURNER R. (2003). *Balancing Agility and Discipline: A Guide for the Perplexed,* Addison-Wesley Longman Publishing Co., Inc.

COCKBURN A.; HIGHSMITH J. (2001a). *"Agile Software Development: The Business of Innovation."* Computer 34(9): 120-127.

COTTMEYER, M. "*The Good and Bad of Agile Offshore Development,"* in proceedings of the Conference on AGILE 2008, pp.362-367, 2008.

DANAIT, A. *"Agile offshore techniques- A case Study"* in proceedings of the Conference on Agile Development Conference, pp.214-217, 2005.

GLADDEN, G. R. *Stop the life-cycle, I want to get off* , ACM SIGSOFT Software Engineering Notes Volume 7 Issue 2, April 1982 Pages 35-39.

GILB, T. *"Evolutionary Delivery versus the "Waterfall" model",* ACM SIGSOFT Software Engineering Notes Volume 10 Issue 3, July 1985 Pages 49 – 61.

LARMAN, C. (2004). *Agile and Iterative Development: A Manager's Guide,* C. Alistair and H. Jim, Pearson Education, Inc.

LARMAN, C.; BASILI, V. R. *Iterative and Incremental Development: A Brief History,* IEEE Computer, June 2003.

MARANZATO, R. P.; NEUBERT, M.; HERCULANO, P. *Scaling Scrum Step by Step "The Mega Framework"* adc, pp.79-85, Agile Conference, 2012.

MCCRACKEN, D. D.; JACKSON, M. A., *Life cycle concept considered harmful*, ACM SIGSOFT Software Engineering Notes Volume 7 Issue 2, April 1982 Pages 29-32.

PAASIVAARA, M.; LASSENIUS, C.; VILLE, T. H. *Inter-team Coordination in Large-Scale Globally Distributed Scrum: Do Scrum-of-Scrums Really Work?*, ESEM '12 Proceedings of the ACM-IEEE ISESE Pages 235-238.

PAASIVAARA, M.; DURASIEWICZ, S.; LASSENIUS, C. *"Distributed Agile Development: Using Scrum in a Large Project,"* Software Process Improvement and Practice, Vol. 13, Issue 6, pp. 527-544, 2008.

ROYCE, W. *"Managing the Development of Large Software Systems*", Proceedings of IEEE Wescon (1970), pp. 382-338.

SCHWABER, K.; BEEDLE, M. (2002). *Agile Software Development with Scrum*, Upper Saddle River: Prentice Hall, ISBN 0-13-067634-9, 158 p.

SCHWABER, K. (2004). *Agile Project Management With Scrum*, Microsoft Press; 1 edition (March 10, 2004), ISBN-10**:** 073561993X 192p.

SCOTT, A. (2002). *Lessons in Agility From Internet-Based Development*, journal IEEE Software archive Volume 19 Issue 2, March 2002 Page 66-73.

SOMMERVILLE, I., *Software Engineering 9th Edition*, 792 pages pp, 2010 Pearson Higher Ed USA, ISBN-10 0137053460.

STAPLETON, J. *DSDM Dynamic Systems Development Method: The Method in Practice*, Addison Wesley Longman, 1997 ISBN 0201178893 163p.

STEPHEN, R.; PALMER, J.; FELSING, M., A Practical Guide to Feature-Driven Development , 2002 • Prentice Hall • Paper, 304 pp, ISBN-10: 0130676152.

SUMMERS, M. *"Insights into an Agile Adventure with Offshore Partners,"* in Proceedings of the Conference on Agile 2008, pp. 333-338, 2008.

SUTHERLAND, J.**;** SCHWABER, K. (1995). *Business object design and implementation*: OOPSLA '95 Workshop Proceedings. The University of Michigan. p. 118. ISBN 3-540-76096-2.

SUTHERLAND, J. et al. *Distributed Scrum: Agile Project Management with Outsourced Development*, Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07), 2007.

TAKEUCHI, H.; NONAKA, I. *"The New Product Development Game"* Jan 01, 1986, Harvard Business Review 10p.

VERSIONONE 7th survey (2012). Disponível em: <http://www.verionone.com>.

AGILE ALLIANCE, *Agile Manifesto* (2001). Disponível em: <http://www.agilemanifesto.org>.

WILLIAMS, L. et al. *Scrum + Engineering Practices: Experiences of Three Microsoft Teams*, ESEM '11 Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement Pages 463-471.