ÉRICO LUIZ ENCARNAÇÃO ROCHA

UTILIZAÇÃO DE FDD (*FEATURE DRIVEN DEVELOPMENT*) E SMARTRe (REQUISITOS REUTILIZÁVEIS) NA GESTÃO DE PROJETOS SCRUM

São Paulo
2013

ÉRICO LUIZ ENCARNAÇÃO ROCHA

UTILIZAÇÃO DE FDD (*FEATURE DRIVEN DEVELOPMENT*) E SMARTRe (REQUISITOS REUTILIZÁVEIS) NA GESTÃO DE PROJETOS SCRUM

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Especialista em Tecnologia da Informação

Orientador: Prof. Eduardo de Oliveira

São Paulo
2013

AGRADECIMENTOS

Ao professor Eduardo de Oliveira, pela orientação, por todo apoio e por ter me ajudado e me guiado durante todo o processo.

Agradeço aos amigos, familiares e companheiros de trabalho pelo suporte, apoio, paciência durante esta fase de minha vida.

Por último agradeço a Deus por todas as alegrias, pela saúde e força que me concedeu, para que eu conseguisse concluir mais este desafio.

Um bom começo é a metade.

(Aristóteles)

RESUMO

Segundo Marçal (2007) o Scrum é um método que aceita que o desenvolvimento de software é imprevisível e formaliza a abstração, sendo aplicável a ambientes voláteis. Ele se destaca dos demais métodos ágeis pela maior ênfase dada ao gerenciamento do projeto. No Scrum os requisitos são definidos pela *User Story,* que é uma pequena e simples descrição de uma funcionalidade dita da perspectiva da pessoa que deseja a nova capacidade, usualmente um usuário ou cliente do sistema. *SMARTRe* é um guia para a escrita de requisitos reutilizáveis desenvolvido por Keepence (1995) que consiste em uma categorização dos requisitos em não reusáveis, diretamente reusáveis ou baseado em parâmetros. A reutilização de requisitos proposta pelo *SMARTRe* conduz a uma melhora substancial na qualidade do processo de engenharia de requisitos reduzindo o tempo de construção, aumentando a qualidade do produto (PEREDNIKAS, 2008). FDD *(Feature Driven Development)* é uma metodologia ágil para gerenciamento e desenvolvimento de software. Ela combina as melhores práticas do gerenciamento ágil com uma abordagem completa para engenharia de software. O alvo de estudo é uma pequena empresa de desenvolvimento de softwares (o nome será mantido em sigilo por questão de confidencialidade) que utiliza a metodologia de gestão ágil Scrum. Nesta empresa alguns prazos não são cumpridos (Atrasos e/ou Não entregas) e o time de desenvolvimento reclama da falta de especificação de alguns requisitos. As *User Stories* não estão sendo suficientes para especificar tecnicamente os requisitos quando algumas funcionalidades são muito complexas ou necessitam de dados técnicos para que o desenvolvedor compreenda corretamente o que deve ser feito. A ênfase no gerenciamento da empresa de desenvolvimento de software, em conjunto com a falta de um levantamento de requisitos mais completo e a falta de uma documentação mais técnica, gerou atrasos, impossibilidade de entrega de algumas funcionalidades novas, incompreensão das tarefas por parte dos desenvolvedores e retrabalho. A utilização da escrita de requisitos *SMARTRe* e FDD possibilita a ênfase também em engenharia de software, além de um nível de maior entendimento dos requisitos por parte do time de desenvolvedores. Com uma especificação mais técnica e bem organizada, os desenvolvedores compreendem melhor as tarefas, diminuindo retrabalho e atrasos, sem perder a agilidade proposta pelo Scrum.

Palavras-Chave: Scrum. FDD. *SmartRe.* Engenharia de Software. Desenvolvimento de software.

ABSTRACT

According Marcal (2007) Scrum is a method that accepts that software development is unpredictable and formalizes the abstraction and applies to volatile environments. Scrum stands out among agile methods by greater emphasis on project management. In Scrum requirements are defined by the user story that is a short and simple description of a feature told from the perspective of the person who wants the new capacity, usually a system user or client. SMARTRe is a guide for writing reusable requirements developed by Keepence et al. (1995) which consists of a categorization of requirements in is not reusable, reusable or directly based on parameters. The reuse of requirements proposed by SMARTRe leads to a substantial improvement in the quality of requirements engineering process reduces construction time and increase product quality (PEREDNIKAS, 2008). FDD (Feature Driven Development) is an agile methodology for software development and management. It combines the best practices of Agile management with a comprehensive approach to software engineering. The aim of the study is a small software development company (the name will be kept secret for reasons of confidentiality) which uses the Scrum agile management methodology. In this company some deadlines are not met (delays and / or not deliveries) and the development team complains about the lack of specification of certain requirements. The user stories are not sufficient to specify technical requirements when certain features are very complex and require technical data for the developer to understand correctly what should be done. The emphasis in the management of software development company, in combination with the lack of a more complete requirements elicitation and the lack of a more technical documentation, generated delays, inability to deliver some new features, incomprehension of the tasks by developers and rework. The use of writing requirements SMARTRe FDD and also allows the emphasis in software engineering, as well as a greater level of understanding of the requirements by the development team. With a more technical specification and well organized, the developers better understand the tasks, reducing rework and delays, without losing the agility proposed by Scrum.

Keywords: Scrum. FDD. SmartRe. Software Engeneering. Software Development.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FDD	Feature Driven Development
PO	Product Owner
SM	Scrum Master
TDD	Test Driven Develpment
TI	Tecnologia da Informação
UML	Unified Modeling Language
XP	Extreme Programming

SUMÁRIO

1 INTRODUÇÃO

1.1 Considerações iniciais

O contexto utilizado para o desenvolvimento desta monografia é o de uma pequena empresa de desenvolvimento de softwares, que atua no mercado nacional, tem aproximadamente 30 funcionários e utiliza a metodologia de gestão ágil Scrum (o nome da empresa será mantido em sigilo por questão de confidencialidade).

Esta empresa atua no contexto de fábrica de software, onde é necessário rever constantemente decisões estratégicas para conseguir atender às necessidades de diversos clientes em diferentes áreas. A redução de custos operacionais e também a adaptação de processos e serviços são outros desafios que a organização enfrenta a cada dia. Segundo Kruger (2012) a metodologia ágil Scrum é baseada no desenvolvimento iterativo e incremental de software e entre os benefícios da sua aplicação estão o aumento do retorno sobre o investimento, maior flexibilidade em relação às mudanças no mercado e diminuição do tempo entre a concepção e o projeto do produto até a disposição do mesmo para os clientes.

Esta empresa de desenvolvimento de software, mesmo seguindo as especificações da metodologia Scrum, verificou que alguns prazos não são cumpridos (Atrasos e/ou Não entregas).

Além desses problemas com os prazos, o time de desenvolvimento reclama da falta de especificação de alguns requisitos e da dificuldade em estimar as horas que serão utilizadas para a realização das novas funcionalidades. Neste caso da empresa de desenvolvimento de software, as *user stories* definidas pelo Scrum, uma pequena e simples descrição de uma funcionalidade dita da perspectiva da pessoa que deseja a nova capacidade, não estão sendo suficientes para esta função quando algumas funcionalidades são muito complexas ou necessitam de dados técnicos para que o desenvolvedor compreenda corretamente o que deve ser feito.

1.2 Objetivo

O objetivo desta monografia é utilizar práticas de engenharia de software FDD (Feature Driven Development) e escrita de requisitos SmartRe (requisitos reutilizáveis) na gestão de projetos Scrum, visando mitigar os atrasos, criar uma documentação mais completa que inclui especificações técnicas, fornecer informações mais precisas para o time de desenvolvimento iniciar suas tarefas e evitar retrabalho. O estudo será realizado em uma das *Sprints* de um projeto de uma ferramenta web para criação e gerenciamento de formulários, desenvolvido pela empresa analisada por esta presente monografia.

1.3 Justificativa

Segundo Marçal (2007) o Scrum é um método que aceita que o desenvolvimento de software é imprevisível e formaliza a abstração, sendo aplicável a ambientes voláteis. Ele se destaca dos demais métodos ágeis, como por exemplo, extreme programming (XP) e Crystal, pela maior ênfase dada ao gerenciamento do projeto.

Essa ênfase no gerenciamento ocorre na empresa analisada e evidencia que a falta de um levantamento de requisitos mais completo e a falta de uma documentação mais técnica gerou atrasos, impossibilidade de entrega de algumas funcionalidades novas, incompreensão das tarefas por parte dos desenvolvedores e retrabalho.

Ao utilizar a escrita de requisitos SMARTRe e FDD é possível dar ênfase também em engenharia de software, além de detalhar melhor os requisitos para o time de desenvolvedores. Com uma especificação mais técnica e bem organizada, os desenvolvedores compreendem melhor as tarefas, diminuindo retrabalho e atrasos, sem perder a agilidade proposta pelo Scrum.

1.4 Abrangência

Esta monografia abrange a metodologia ágil de desenvolvimento de software Scrum utilizada por uma empresa que desenvolve softwares e faz consultoria em tecnologia da informação, técnicas de engenharia de software FDD (Feature Driven Development) por ser uma metodologia ágil com uma abordagem completa no quesito engenharia de software, e escrita de requisitos reutilizáveis SmartRe que visa uma economia de recursos com a reusabilidade de requisitos. O projeto de uma ferramenta web para criação e gerenciamento de formulários foi selecionado para analise, pois existe um histórico recente com a metodologia Scrum. Os processos de desenvolvimento de software e gestão Scrum deste projeto serão utilizados como base para o estudo que segue.

1.5 Metodologia

Estudo preliminar e diversas pesquisas exploratórias sobre os temas scrum, FDD (Feature Driven Development) e SMARTRe, Engenharia de Software e Engenharia de Requisitos.

Estudo de caso em uma empresa de desenvolvimento de software, aplicando as técnicas pesquisadas, durante um ciclo de desenvolvimento Scrum com duração de duas semanas.

Análise dos resultados obtidos com a proposta desta monografia e comparação com resultados anteriores, da mesma empresa de desenvolvimento de software, sem a utilização das práticas propostas.

1.6 Estrutura da Monografia

O capítulo 1 apresenta as considerações iniciais a respeito desta monografia, os objetivos que serão alcançados, justificativa, abrangência, metodologia utilizada e estrutura geral da monografia.

O capítulo 2 descreve as teorias necessárias para o entendimento da monografia, e que sustentam a solução proposta pela presente monografia.

O capitulo 3 apresenta uma forma de utilização de técnicas de engenharia de software e de engenharia de requisitos em conjunto com a metodologia ágil Scrum com a finalidade de complementar a metodologia e chegar aos resultados esperados.

O capitulo 4 contém as considerações finais a respeito da monografia e a apresentação dos resultados obtidos com as pesquisas. Além de tais considerações este capitulo contém as contribuições dadas pela monografia e sugere temas para evolução do tema.

2 REFERENCIAL TEÓRICO

2.1 SCRUM

Segundo Martin Ota (2010) Scrum é um método de gerenciamento do desenvolvimento de produtos e organização de trabalho. Definitivamente é adequado para conduzir uma equipe a desenvolver um produto. Pode ser definido também como uma forma de iteração contínua, com o cliente presente durante o desenvolvimento de seu produto e um jeito de comunicação no time de desenvolvimento. Ainda na concepção do autor, Scrum é uma ferramenta para melhoria contínua dos métodos utilizados, um processo exatamente e estritamente determinado.

O termo Scrum vem do Rúgbi, um esporte originário da Inglaterra, e é uma jogada onde oito jogadores da equipe se unem em um bloco e atuam juntos para conseguir ganhar a bola. O time trabalha integrado, cada membro tem seu papel bem definido e o time todo tem o foco em uma única meta. No time de desenvolvimento os membros devem entender bem o seu papel e as tarefas de cada incremento. O time todo deve ter apenas um foco e as prioridades devem ser muito bem definidas (RISING; JANOFF, 2000).

Hayata e Han (2011) dizem que Scrum é um método ágil geral, que tem o foco em gerenciar o desenvolvimento de software iterativo em vez de abordagens técnicas específicas de desenvolvimento ágil de software e projetos de TI. O processo Scrum consiste em 3 fases. A primeira é um esboço para estabelecer os objetivos gerais do projeto e a arquitetura de software. A segunda fase é uma série de ciclos (*sprints*), onde cada ciclo desenvolve uma nova parte ou funcionalidade do sistema. A terceira fase é o encerramento do projeto, onde é gerada a documentação necessária e é feita uma avaliação das lições aprendidas com o projeto.

O *Scrum Guide* (2012) define que o Scrum é um *framework* estrutural que é utilizado para gerenciar o desenvolvimento de produtos complexos desde o início de

1990. Scrum não é um processo ou uma técnica para construir produtos, em vez disso, é um *framework* dentro do qual você pode empregar vários processos ou técnicas. O Scrum deixa visível a eficácia relativa das práticas de gerenciamento e desenvolvimento de produtos, de modo que você possa melhorá-las.

De acordo com Laurie Willians (2011) a metodologia Scrum é um processo ágil de desenvolvimento de software que funciona como um invólucro para as práticas de engenharia de software no desenvolvimento iterativo e incremental. Esta definição de Laurie Willians (2011) é a definição base para a metodologia Scrum adotada para o desenvolvimento desta monografia.

De acordo com o *Scrum Guide* (2012), esse *framework* se apoia em três pilares fundamentais:

1 Transparência: Aspectos significativos do processo devem estar visíveis aos responsáveis pelos resultados.
2 Inspeção: Os usuários Scrum devem, frequentemente, inspecionar os artefatos Scrum e o progresso em direção ao objetivo para detectar indesejáveis variações. Esta inspeção, não deve no entanto, ser tão frequente que atrapalhe a própria execução das tarefas.
3 Adaptação: Se um inspetor determina que um ou mais aspectos de um processo desviou para fora dos limites aceitáveis, e que o produto resultado será inaceitável, o processo ou o material que está sendo produzido deve ser ajustado. O ajuste deve ser realizado o mais breve possível para evitar mais desvios.

Segundo Emerson José Morgado Brito (2012), para inspeção e adaptação o Scrum estabelece quatro eventos formais:

> "A primeira reunião é de planejamento a *sprint planning*, que da início a um ciclo de desenvolvimento, onde são definidos quais requisitos do produto chamados de *backlog* do produto ou *product backlog*, serão selecionados para o *sprint* e chamados de *backlog* do *sprint* ou *Sprint Backlog*, a segunda oportunidade são as Reuniões diárias as *Daily Meetings*, que são realizadas diariamente durante o ciclo, a terceira oportunidade é a reunião de Revisão *Spint Review* realizada ao término do ciclo, com a finalidade de apresentar o incremento de produto ao cliente, a quarta oportunidade é a reunião de retrospectiva *Sprint Retrospective*, onde os aspectos relevantes do ciclo são discutidos, analisados e melhorias são propostas, encerrando assim um ciclo de desenvolvimento." (2012, p. 21).

O Scrum é composto por equipes associadas a seus papéis, eventos, artefatos e regras. Cada componente dentro do Scrum serve a um propósito específico e é essencial para o uso e o sucesso do Scrum. O time Scrum é formado pelo P.O.

(*Product Owner*), *Scrum Master*, e a equipe de desenvolvimento (SCHWABER; SUTHERLAND; 2010).

Segundo Laurie Willians (2011) o P.O. cria os requisitos, prioriza esses requisitos e também documenta-os no chamado *Product Backlog* (visão geral do produto) durante o plano da *Release* (Versão do produto). No Scrum esses requisitos são chamados de características. A autora descreve que as equipes trabalham em iterações curtas de duas a quatro semanas, e cada iteração tem o nome de Sprint. Depois que a *Sprint* foi iniciada não é permitido incluir novas características no planejamento, e tais características entram no planejamento da próxima *Sprint* a ocorrer.

A figura 1 demonstra o ciclo de uma *Sprint Scrum*.

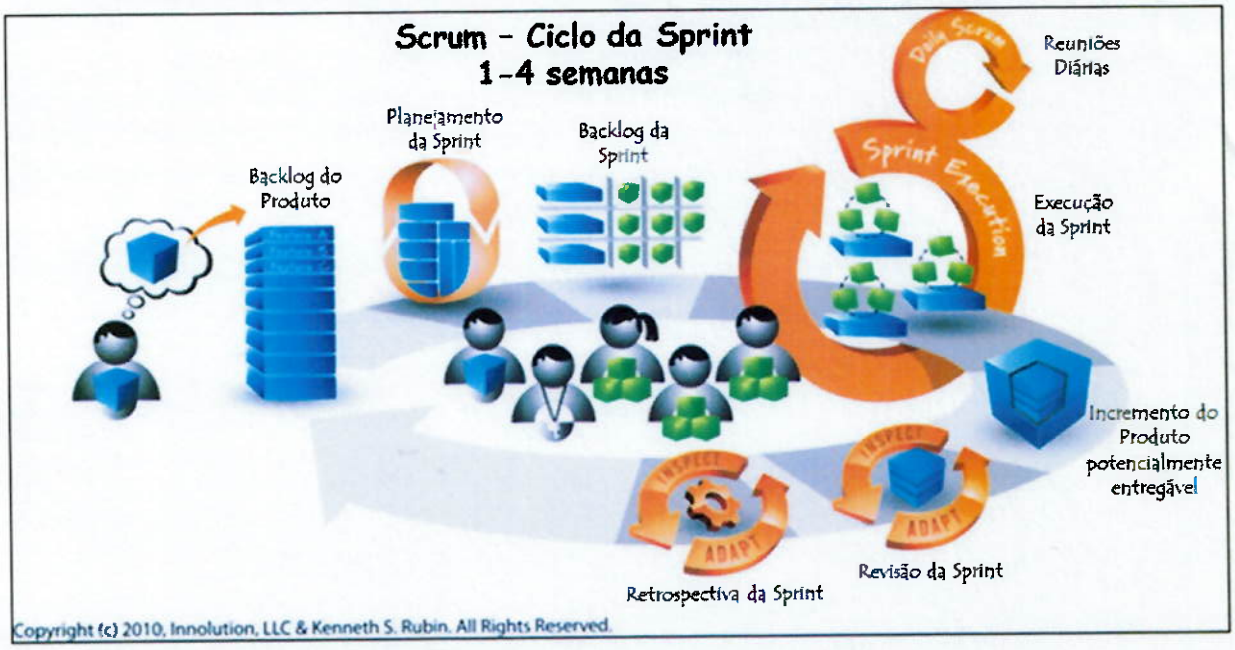

Figura 1 – Ciclo de uma *Sprint Scrum*.

Todos os dias durante a *Sprint* é realizada uma reunião com a equipe de desenvolvimento, em que todos falam o que fizeram no ultimo dia, as dificuldades encontradas, e o plano do que farão no dia atual. Essas reuniões costumam durar entre dez e quinze minutos e são feitas em pé com o propósito de não se estenderem além desse tempo.

O *Scrum Guide* (2012) sugere que após cada *Sprint* seja feita uma revisão do que houve durante a iteração e que seja feita também uma reunião de retrospectiva,

onde a equipe de desenvolvimento levanta os pontos positivos e os pontos a melhorar para a próxima iteração. Dessa forma o *Scrum* leva a equipe a um aprendizado e uma melhora contínua em seus processos de desenvolvimento.

O *Framework Scrum* é simples de entender, mas muito difícil de seguir (KRISHNA; BASU, 2011). Martin Ota (2010) demonstra que o Scrum descreve a visão, fixa um quadro (geralmente orçamento e tempo) e entrega a visão passo-a-passo, desde as partes mais importantes para as menos importantes do projeto. Ele declara a mudança como um evento bem vindo, mas tem uma gestão rigorosa de mudança, que afeta o time apenas no início de cada *Sprint* (tipicamente um dia por duas semanas). O Scrum trabalha com listas de prioridades, que podem ser vistas como as listas de requisitos.

Os requisitos ou características no Scrum são definidos por *Users Stories*. Segundo Mike Cohn:

> "User story é uma pequena e simples descrição de uma funcionalidade dita da perspectiva da pessoa que deseja a nova capacidade, usualmente um usuário ou um cliente do sistema. Essas pequenas documentações fazem parte do Backlog do Produto, principalmente porque no backlog do produto deve conter as necessidades dos clientes e não as funcionalidades do software a ser desenvolvido". (Apud ETTINGER, 2011).

Colin Doyle (2011) diz que às vezes as *user stories* não são o bastante no sentido de documentação de requisitos. A *user story* é uma peça muito importante de documentação no scrum, mas documenta apenas as necessidades dos clientes não abrangendo a parte técnica que é fundamental pra o time de desenvolvimento.

De acordo com Ken Schwaber e Jeff Sutherlan (2010) o Scrum é adequado para a gestão de projetos de software, pois evidencia, de forma transparente, todas as perspectivas do projeto, desde o andamento até os produtos gerados a cada iteração. Também é possível concluir que o Scrum é um *framework* preparado para lidar com as mudanças no decorrer do projeto, pois já é de sua natureza a convivência com mudanças constantes. Em relação à parte técnica e Engenharia de Software, o Scrum possui uma necessidade de integração com outras ferramentas e processos para que o projeto de desenvolvimento de software seja bem sucedido.

2.2 FDD (Feature Driven Development)

Jeff De Luca e Peter Coad criaram FDD em 1997 quando Jeff De Luca era gerente de projeto de um grande projeto de desenvolvimento de software em Singapura. O projeto era muito complexo e Jeff percebeu que a missão em suas mãos não poderia ser concluída no tempo determinado, com os recursos disponíveis e usando a estratégia tradicional de desenvolvimento de software. Ele, portanto, com a ajuda de Peter Coad e outros, criaram a técnica de modelagem em cor e do conceito de desenvolvimento orientado a característica. A primeira impressão disso foi publicada no livro "Modelagem Java em cores com UML", escrito por Peter Coad em 1999.

Segundo Sadhna Goyal (2007) FDD (*Feature Driven Development* – Desenvolvimento Orientado a Características) é um processo de desenvolvimento de software ágil e altamente adaptativo que é curto e altamente iterativo, com muita ênfase em qualidade em todos os passos, entrega frequentemente resultados tangíveis em todos os passos, fornece informações significativas de progresso e status do projeto com o mínimo de interrupções no time de desenvolvimento e é apreciado por clientes, gerentes e desenvolvedores.

Segundo Rychly e Tichá (2008), *Feature Driven Development* (FDD) é um processo incremental e iterativo de desenvolvimento de software. Embora seja um método ágil de desenvolvimento de software, ele é construído em torno de práticas tradicionais reconhecidas pela indústria e derivadas da engenharia de software, incluindo fases de planejamento, design e documentação, com um refinado sistema de decomposição de funcionalidades e responsabilidades dos desenvolvedores, relatórios precisos de progresso, verificação frequente, etc.

Sadhna Goyal (2008) define *feature* como uma funcionalidade ou característica que faz parte de um projeto de software. Deve ser pequena o suficiente para ser implementada em uma iteração, além de oferecer valor ao cliente.

A aplicação do método FDD leva a uma melhor consistência do desenho do software, implementação, e documentação, pois em alguns processos são criados

modelos que auxiliam a implementação e são parte da documentação (PALMER; FELSING, 2002)

FDD começa com a criação de um modelo abrangente em colaboração com especialistas do domínio. Usando informações da atividade de modelagem e de quaisquer outras atividades de requisitos que ocorreram, os desenvolvedores passam a criar uma lista de características. Em seguida, um plano é elaborado e as responsabilidades são atribuídas (GOYAL, 2008).

Em seu livro *"A Practical Guide to Feature-Driven Development"* Palmer e Felsing (2002), definem que para cada processo do FDD é necessário um critério para entrada e um critério para saída. Isso significa que para um processo iniciar é necessário que o processo anterior tenha sido finalizado e que as saídas previstas para o processo estejam de acordo com o planejado, ou seja, as tarefas do processo devem estar concluídas.

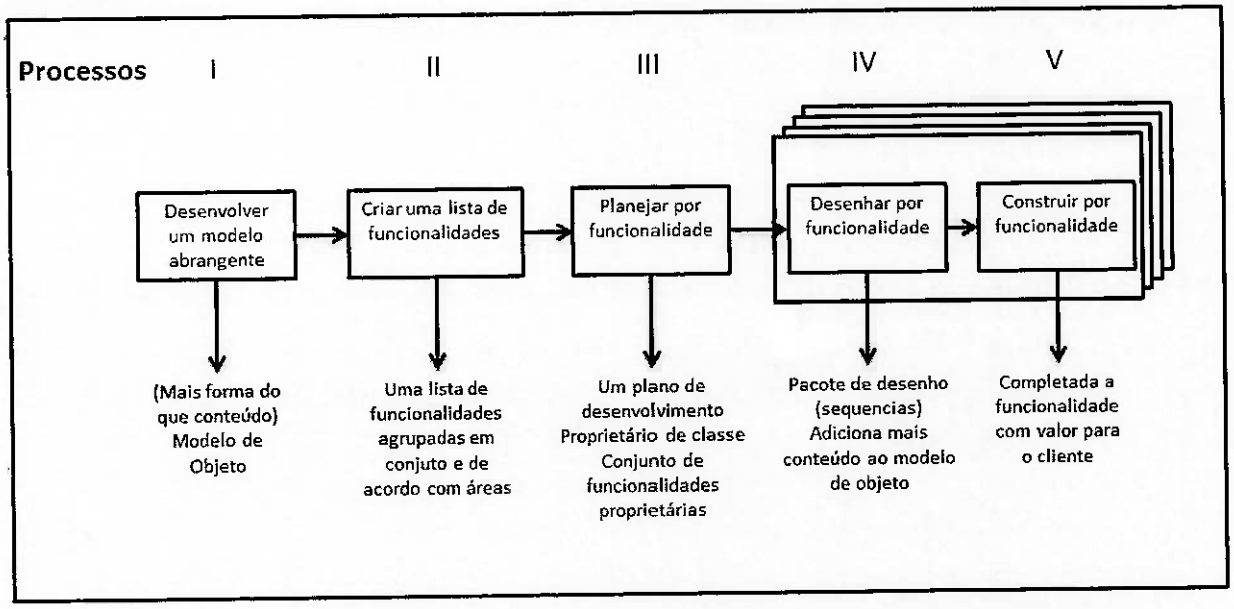

Figura 2 – Figura adaptada de 'Os 5 processos do FDD e suas saídas' (PALMER, SR., FELSING, apud GOYAL 2008).

FDD é composto por cinco fases conforme mostra a figura 2 e Kanwal, Junaid e Fahiem (2010) definem estas fases da seguinte maneira:

A primeira fase é desenvolver um modelo abrangente (*Develop an Overall Model*). A maior ênfase desta fase é na coleta e análise de requisitos, principalmente os

requisitos funcionais. A formação da equipe, e os documentos de requisitos funcionais são a principal saída desta fase.

A segunda fase é criar uma lista de Funcionalidades (*Build a Feature List*). A fase de criação da lista de funcionalidades especifica que a extração de características e funcionalidades é a base sob a qual o plano de desenvolvimento de software é preparado. Esta fase é dedicada à construção de uma lista composta por essas características ou funcionalidades. O domínio é decomposto em áreas (*major feature sets*) abrangendo atividades (*feature sets*) que contem cada funcionalidade, o que representa um passo em uma atividade.

A terceira fase é planejar de acordo com as Funcionalidades (*Plan By Feature*). A principal ênfase dessa fase está nas atribuições de tarefas para os membros da equipe. Nessa fase também é incluído um cronograma de projeto de acordo com as características ou funcionalidades.

A quarta fase é desenhar de acordo com as Funcionalidades (*Design By Feature*). Esta fase tem foco no projeto detalhado de requisitos funcionais do projeto. O projeto construído nas fases 1 e 2 é refinado e finalizado. A fase inclui também a formalização das especificações de projeto, na forma de classes. As classes criadas nesta fase são os principais resultados produzidos.

A quinta e ultima fase é construir de acordo com as funcionalidades (*Build By Feature*). A orientação principal desta fase é a implementação das especificações de design produzidos na fase anterior. O que determina se o item está concluído são os testes das funcionalidades, realizados ao final do desenvolvimento de cada item. Se os itens não estiverem em conformidade com o que foi especificado na quarta fase, estes itens devem ser corrigidos para que estejam de acordo com a especificação.

O método FDD é muito completo no quesito engenharia de software e sua especificação de requisitos é bem detalhada (GOYAL, 2008). Seu modelo de gestão de projeto é um pouco mais complexo que o do Scrum no que diz respeito a papéis. No Scrum temos o Scrum Master, o time de desenvolvimento e o *Product Owner* (SCHWABER; SUTHERLAND, 2010) já no FDD temos o Administrador do projeto, o chefe de arquitetura, o responsável pelas funcionalidades, o responsável pelas classes,

membro do projeto, administrador do domínio, entre outros. (PALMER; FELSING, 2002).

O FDD foi originalmente desenhado para um time com diferentes habilidades misturadas, diferentes níveis de experiência, diversas raças (chineses, indianos, americanos, australianos e europeus) e idades. O modelo de gestão do FDD é criado para organizar projetos de uma forma que as forças individuais dos integrantes do time de desenvolvimento são completamente utilizadas e ofereçam suporte para possíveis áreas de fraquezas, por isso um número maior de papéis e papéis mais específicos em relação à suas atribuições (PALMER; FELSING, 2002).

2.3 SMARTRe

De acordo com Keepence (1995), SmartRe é uma técnica para escrita de requisitos em que é possível identificar e escrever requisitos pensando em sua reusabilidade em projetos futuros.

Keepence (1995) diz que a engenharia de requisitos consome muito tempo, é cara, mas é uma fase crítica no desenvolvimento de software. O autor afirma também que ao mesmo tempo, a demanda por software continua a exceder a oferta e o reuso de software está no topo da lista da indústria de computação. SMARTRe sugere uma forma para a escrita de requisitos e descreve algumas técnicas e práticas para escrever requisitos reutilizáveis (KEEPENCE;MANNION; SMITH, 1995).

Mannion e Keepence (1995) argumentaram que, apesar de ter melhorado nossa compreensão da necessidade de produzir especificações claras, completas e consistentes de requisitos, na prática, ainda há espaço para melhorias.

A técnica derivou da definição de objetivos em gestão de psicologia e foi adaptada para o desenvolvimento de requisitos inteligentes e adaptada para fornecer orientações para os autores de especificações de requisitos. A sigla SMART (*Specific, Measurable, Attainable, Realisable, Timebounded* – em português: Específico,

Mensurável, Atingível, Realizável, Estimável) é usado para ajudar as pessoas no estabelecimento de bons objetivos (KEEPENCE; MANNION; SMITH, 1995).

Na especificação de requisitos de software, Keepence, Mannion e Smith (1995) definem SMART para ser:

> *Specifc (Específico)*
> O requisito deve expressar exatamente a necessidade do cliente. Especificidade compreende: clareza, coerência, simplicidade e um nível adequado de detalhe.
> *Measurable (Mensurável)*
> No contexto da engenharia de requisitos, mensurável quer dizer que é possível verificar que a exigência do requisito foi cumprida.
> *Attainable (Atingível)*
> Um requisito atingível é aquele em que é possível para o sistema expor/mostrar, sob certas condições e de maneira real. Alguns requisitos podem exigir um conhecimento muito amplo, é como se estivesse além da compreensão humana. A consequência em tentar responder a este tipo de requisito é que o sistema terá um custo proibitivo, ou nunca será aceito ou ambos.
> *Realisable (Realizável)*
> Por realizável queremos dizer é que é possível alcançar os objetivos, dado que se sabe sobre as restrições previstas pelo cliente sob a qual o sistema e o projeto deve ser desenvolvido.
> *Traceable (Rastreável)*
> A rastreabilidade de requisitos é a habilidade/capacidade de se saber a origem de um requisito desde sua concepção, especificação, design, implementação e teste.

Mannion, Keepence (1995) afirmam que os engenheiros de requisitos, muitas vezes reconhecem semelhanças em sistemas que já foram construídos e tentam identificar oportunidades de reutilização de requisitos. Devido a considerações de orçamento, no entanto, eles geralmente não têm motivação para se preocupar em fazer os requisitos reutilizáveis. Por outro lado, um engenheiro de requisitos especialista no domínio pode ter uma visão mais ampla.

Mannion, Keepence (1995) identificam as seguintes classes de requisitos:

- ✓ Não Reutilizável;
- ✓ Diretamente Reutilizável (composição);
- ✓ Parâmetro Base (geração).

> *Não reutilizáveis*
> Alguns requisitos são diretamente relacionados ao sistema sendo construídos apenas para aquele sistema. São incluídos nesta categoria requisitos temporais (por exemplo, prazos). Em um novo documento estes tipos de requisitos serão escritos essencialmente do mesmo jeito que eles foram sem reutilizar esforço.
> *Diretamente reutilizável (composição)*
> Dentro de um domínio particular, haverá algum requisito que se aplica a todos os sistemas que são desenvolvidos. Estes podem ser atributos naturais dos sistemas e/ou normas da indústria ou empresa. Em um domínio em que os

> sistemas desempenham funções muito semelhantes em ambientes quase idênticos é provável que haja uma quantidade razoavelmente grande de reutilização de requisitos.
>
> *Parâmetro ou Requisitos baseado em modelo (geração)*
>
> Há muitos requisitos que especificam algum nível de desempenho ou uma lista de funcionalidades necessária. Além da medida real de desempenho ou os itens da lista, o requisito deve ser reutilizável. Quando os requisitos são copiados e editados, eles têm seus valores alterados ou a lista de elementos modificada. Um requisito baseado em parâmetro tem elementos variáveis a ele. O elemento variável pode ser algo muito simples, tais como o número de terminais necessários num sistema ou uma lista de descrições funcionais.

O desenvolvimento e gestão de requisitos reutilizáveis é sempre uma questão difícil, pois há o problema adicional de muitas vezes serem escritos em linguagem natural.

Keepence (1995) diz em seu artigo que a reutilização de requisitos ocorreu predominantemente através de um método "copiar-e-editar" e que este método de reutilização raramente é benéfico porque o requisito construído é essencialmente novo e, portanto, precisa de validação. As diretrizes de classificação e associados apresentados permitem um maior rigor, aumentando assim o potencial de reutilização de requisitos. O alvo do SmartRe segundo Keepence (1995) é gerar requisitos diretamente reutilizáveis, por isso, cada requisito que não seja marcado como tal deve ser analisado para determinar o por que não foi considerado reutilizável.

As diretrizes a seguir, após a análise dos requisitos não reutilizáveis, ajudam a torná-los reutilizáveis (KEEPENCE, 1995):

> 1. Remoção de referências específicas: É muito comum um assunto ser constantemente referido em um requisito. Removendo referências específicas aumentará significativamente o numero de requisitos reutilizáveis.
> 2. Derivando termos comuns: Algumas vezes é comum utilizar diferentes termos para um mesmo elemento. A característica importante é que a uniformização de termos nos requisitos aumenta muito o nível de reutilização e que uniformização de termos não significa necessariamente semelhança de definição.
> 3. Divisão de partes específicas e partes genéricas: Alguns requisitos contém partes que são específicas e partes genéricas, isso pode reduzir o nível de reusabilidade do requisito. Na maioria dos casos a separação das partes específicas das genéricas aumentam a reusabilidade do requisito e transforma um requisito não-reutilizável em dois requisitos, um parcialmente reutilizável e outro diretamente reutilizável.

3 PRÁTICAS DE ENGENHARIA DE SOFTWARE FDD E ESCRITA DE REQUISITOS SMARTRE NA GESTÃO SCRUM

3.1 Histórico da experiência da empresa com Scrum

Segundo Krishna e Basu (2011) Scrum é uma das metodologias para desenvolvimento iterativo e incremental mais utilizadas. Os autores definem Scrum como um esqueleto de processo ou estrutura que contem um conjunto de práticas com papéis pré-definidos em que as partes interessadas estão envolvidas no desenvolvimento. Estes foram alguns dos fatores que influenciaram a empresa analisada a escolher Scrum como metodologia de gestão.

Outros fatores que também influenciaram a empresa analisada a implementar Scrum como metodologia de gestão para desenvolvimento de softwares foram o fato de ser uma metodologia ágil com a presença dos interessados durante o processo de desenvolvimento e o caso de ser transparente e adaptável (SCRUM GUIDE, 2012).

As primeiras ações da empresa em relação à adoção do Scrum como metodologia de gestão foram definir os papéis e treinar os funcionários para que a metodologia fosse realizada de forma original sem perder suas características. Esse período de treinamentos e definições de responsabilidades durou por volta de três meses, durante os quais foram realizadas várias turmas organizadas em ciclos. Todos os ciclos de treinamento foram teóricos e práticos, para todos os papeis necessários à metodologia. Na primeira etapa da aplicação da metodologia ágil a empresa definiu e selecionou uma equipe composta por cinco desenvolvedores, um *Product Owner* e um *Scrum Master*.

Inicialmente foram definidas as principais características do produto em reunião conduzida pelo *Scrum Master* onde o *Product Owner* explicou para todo o time, do ponto de vista do negócio, os detalhes relevantes de cada uma das características fundamentais. Após esta definição, foi criado o *Product Backlog* e o *PO* definiu as prioridades para desenvolvimento.

Com a definição das prioridades, foi realizada a reunião do time de desenvolvimento para detalhar as características do produto desejado em um conjunto de requisitos. Nesta etapa, além do *Scrum Master,* o *PO* teve participação importante esclarecendo algumas dúvidas em relação às características que permitiu dar mais consistência aos requisitos para todo o time. Na medida em que cada característica dava origem a um ou mais requisitos, o time passou estimar o tempo de desenvolvimento para cada requisito e seu respectivo esforço estimado com base na complexidade do requisito e na experiência da equipe.

O número de desenvolvedores que estariam envolvidos no projeto foi exatamente o número de desenvolvedores disponíveis na empresa no momento em que esta fase do projeto aconteceu. O Esforço foi calculado pelas horas estimadas multiplicadas pelo numero de desenvolvedores envolvidos na execução, por exemplo: se um requisito tinha a duração de uma hora e dois desenvolvedores fossem trabalhar juntos nesta hora, então a tarefa estaria estimada com duas horas de esforço (1 hora X 2 funcionários = 2 horas de esforço).

Com o *Product Backlog* desenvolvido e os requisitos estimados o time Scrum realizou a reunião de planejamento da *Sprint*. Nesta reunião o time, de posse dos requisitos e das estimativas, passou a criar o conjunto de tarefas que permitiriam entregar o requisito, no tempo estimado. O resultado desta reunião foi o *Sprint Backlog*.

Com o time e suas respectivas responsabilidades definidos, o *Product Backlog* definido, os requisitos estimados e a *Sprint* planejada chegou a hora de iniciar a primeira *Sprint*.

Todos os procedimentos e artefatos do Scrum foram respeitados como, por exemplo, as reuniões diárias de quinze minutos de duração que ocorriam todos os dias às nove horas e trinta minutos no período da manhã com todo o time presente. Foi feito o quadro de acompanhamento com as tarefas pendentes, em andamento e concluídas, além dos itens não planejados e impedimentos ocorridos, para que a gestão estivesse à vista de todos os envolvidos no projeto.

Durante as reuniões diárias e de acordo com o *Scrum Guide*, os desenvolvedores demonstravam o que foi feito desde a ultima reunião até o momento,

o que seria feito até a próxima reunião e quais foram as dificuldades encontradas no decorrer do caminho.

Os principais requisitos da primeira *Sprint* foram: criar o ambiente, fazer o modelo de dados, criar o banco de dados, criar um cadastro para os clientes e criar o modelo do construtor de pesquisas. O total de horas estimados nos requisitos foi de 448 horas. A velocidade do time composto por 5 desenvolvedores era de 30 horas/dia, pois o combinado era que cada desenvolvedor deveria entregar 6 horas de trabalho por dia. Ao final de 15 dias de *Sprint* (3 semanas), o time deveria ser capaz de entregar 450 horas. Nota-se que havia consistência entre a estimativa de tempo para os requisitos e a capacidade do time para entregar a *Sprint* completa.

O resultado desta primeira *Sprint* foi o seguinte:

1. Nem todos os itens planejados foram entregues;
2. A estimativa de tempo não foi feita corretamente para alguns itens;
3. A documentação foi insuficiente para o entendimento do trabalho por parte dos desenvolvedores;
4. A equipe não estava totalmente adaptada à metodologia Scrum;
5. Faltaram alguns conhecimentos técnicos para o desenvolvimento das tarefas;
6. Ainda se fazia necessário uma melhor definição dos padrões a serem utilizados pela empresa.
7. Houve problemas de comunicação entre os integrantes do time.

Todos os pontos acima foram observados pelo *Scrum Master*, ou pelo time durante a reunião de retrospectiva. Quanto à entrega, que é um dos pilares do *Scrum Guide*, a inspeção foi realizada pelo *PO* e o time, tendo havido consenso sobre o que foi entregue e o que foi rejeitado.

No gráfico de *Burndown* da primeira Sprint (ver figura 3) alguns itens mencionados acima ficam bem claros e explícitos, como por exemplo, o atraso nas entregas, e a estimativa incorreta de horas para os requisitos.

Dias do Projeto	Estimativa	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Plano	448	418	388	358	328	298	268	238	208	178	148	118	88	58	28	-2
Horas Planejadas Entregues		23	27,5	35	25	25	12,5	14,5	18	22	15	25	28,5	26	22	23
Adamento das Entregas	448	425	397,5	362,5	337,5	312,5	300	285,5	267,5	245,5	230,5	205,5	177	151	129	106
Velociadade do Time	30															

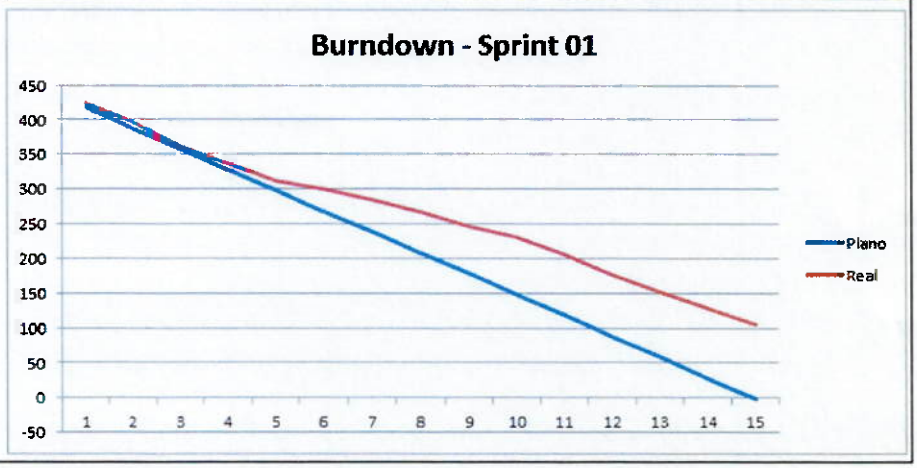

Figura 3 – Gráfico *burndown* da primeira *Sprint* da Empresa.

Ao final da *Sprint* o time de desenvolvimento deixou de entregar 106 horas, e o desvio iniciou a partir do quinto dia de *Sprint*, aumentando a cada dia até o final da iteração.

A metodologia Scrum é simples para compreender, porém extremamente difícil de aplicar na prática (KRISHNA; BASU, 2011). Essa primeira *Sprint* realizada pela empresa de desenvolvimento de software demonstra o quão verdadeira é esta observação feita por Krishna e Basu (2011).

Após a retrospectiva da primeira *Sprint* a empresa chegou a algumas conclusões. As primeiras medidas a tomar seriam reforçar o conhecimento da equipe na metodologia Scrum e definir os padrões que seriam utilizados pela empresa, como por exemplo, padrões de documentação de código e de modelo de dados. O objetivo dessas medidas seria evitar surpresas com as mudanças constantes que ocorreram durante andamento do projeto.

De acordo com Linda Rising e Norman Janoff (2000) atualmente em desenvolvimento de software os requisitos mudam constantemente e muitas vezes essas mudanças ocorrem durante o ciclo de vida do desenvolvimento do produto para atender requisitos de negócios, criando uma dor de cabeça constante para as equipes de desenvolvimento.

A primeira *Release* do produto da empresa de desenvolvimento de software entrou em produção em abril de 2012 e para a conclusão desta *release* ocorreram 7 *Sprints*. A cada *Sprint* realizada a empresa executou um breve treinamento de reciclagem na metodologia Scrum com o intuito de fortalecer os conhecimentos e aprimorar o comportamento da equipe ao longo do projeto.

O projeto teve sua segunda release em produção no mês de outubro de 2012, e esta *release* contou com 5 *Sprints*.

Nas reuniões de retrospectiva das *Sprints* o time de desenvolvimento não deixava de mencionar os pontos negativos e os pontos positivos, gerando um amadurecimento da equipe ao longo do tempo. Contudo, alguns pontos negativos estavam sendo repetidamente levantados pelos integrantes do time, como o atraso nas entregas e a falta de uma especificação mais completa para entendimento do desenvolvedor.

Apesar de existir as *User Stories* que deixavam claros os atores, as ações e as funcionalidades, ainda faltava uma especificação técnica para melhor compreensão dos desenvolvedores. Um exemplo dessa necessidade de uma especificação técnica foi no desenvolvimento do mecanismo central da ferramenta que estava sendo desenvolvida, além de conhecer os atores, ações e funcionalidades, os desenvolvedores precisaram de algumas horas de estudo do funcionamento, uma especificação de classes e um detalhamento maior da funcionalidade para não alterar a essência do mecanismo e principalmente não impedir que novas funcionalidades pudessem ser integradas ao mecanismo.

O Scrum foi seguido conforme os criadores Ken Schwaber e Jeff Sutherland propõem, sem perder o rigor e sua característica principal, artefatos e cerimônias *time-boxed* (com o tempo pré-definido). Ainda sim havia a necessidade de uma técnica de engenharia de software para dar suporte aos desenvolvedores em casos como o do mecanismo central que precisava de um estudo e de uma especificação das classes para ser desenvolvido. O fato desta especificação técnica não ter sido elaborada com detalhes causou significativo atraso na *Sprint* e no projeto.

3.2 Proposta para utilizar FDD e SmartRe em conjunto com Scrum

Segundo Krishna e Basu (2010), algumas empresas alteram características fundamentais do Scrum, como, por exemplo, aumentar o tempo das reuniões diárias que são de 15 minutos ou a duração de uma *Sprint* para um prazo diferente do proposto pela metodologia (de duas a quatro semanas). O Scrum propõe essas reuniões de curta duração e *Sprints* de tempo previamente definido exatamente para dar agilidade ao desenvolvimento. Os autores definem essas alterações de características como o "*ScrumBUT*". A palavra "*but*" no final da palavra Scrum é com o sentido de "mas", justificando o porquê de não seguir o proposto pelo Scrum.

Neste mesmo trabalho, Krishna e Basu (2010) também citam empresas e times que utilizam técnicas para incrementar o processo do Scrum, sem alterar suas características fundamentais, eles chamam de "*ScrumAND*". Esta é a proposta desta presente monografia, adicionar técnicas de engenharia de software ao Scrum, caracterizando "*ScrumAND*", para atender a necessidade de negócios da empresa de desenvolvimento de software e garantir a agilidade com consistência.

No Scrum *Guide* (2010) Ken Schwaber e Jeff Sutherland dizem que o *Product Backlog* é uma lista ordenada de tudo que deve ser necessário no produto, e é uma origem única dos requisitos para qualquer mudança a ser feita no produto. Quem fica responsável por essa parte é o *Product Owner*.

Paula Nascimento (2012) afirma que o primeiro passo da metodologia Scrum é criar a visão do produto, que após algumas reuniões com o time e o *PO* formam o *Product Backlog*. Na etapa após a definição do *Product Backlog* é realizada a criação das *User Stories* como forma de documentação e referência para os desenvolvedores.

A criação do *Product Backlog* e das *User Stories* é seguida da reunião de planejamento da *Sprint*, onde o time e o *PO* definem o *Sprint Backlog*, ou seja, os itens do *Product Backlog* que farão parte da *Sprint*. A *Sprint* inicia e tem duração de 2 a 4 semanas, e ao final é realizada uma reunião de revisão da *Sprint,* que é a entrega dos requisitos ao *PO*, e uma reunião de retrospectiva pra avaliar os pontos positivos, negativos e dificuldades enfrentadas. Estas definições do Scrum, confirmando a teoria

descrita no segundo capítulo desta monografia, não devem ser alteradas para a realização da proposta.

Esta monografia tem como proposta incluir algumas técnicas de engenharia de software em conjunto com o *framework* Scrum sem que o Scrum perca suas características. A utilização de *Feature Driven Development* e da técnica de requisitos reutilizáveis *SmartRe*, tem o objetivo de entregar aos desenvolvedores um material mais rico para que eles possam exercer melhor o seu trabalho e principalmente garantir que as entregas do Scrum sejam feitas com qualidade e no prazo correto.

A figura 4 mostra como seria a proposta do *framework* Scrum com a utilização de FDD e *SmartRe*. Figura adaptada da apresentação de Rildo F. Santos (2010).

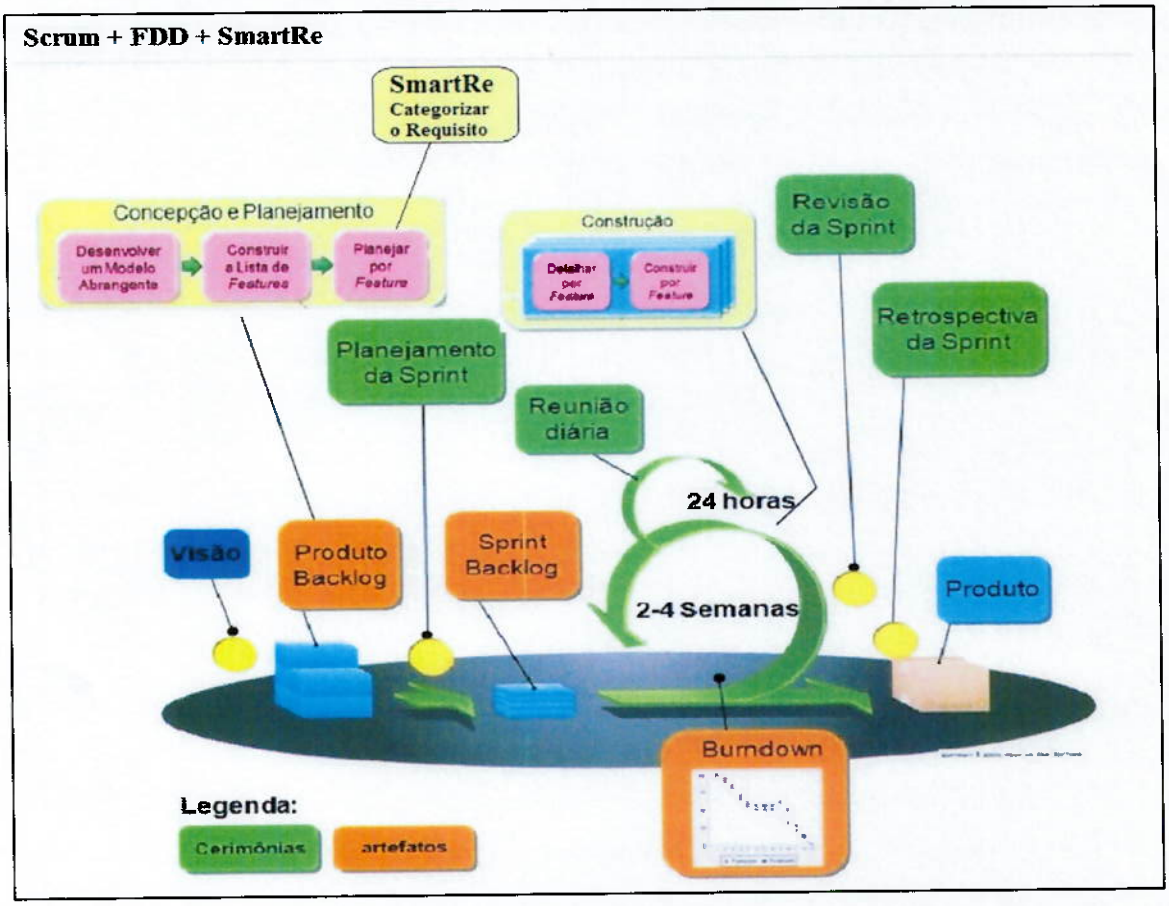

Figura 4 – *Framework* Scrum com as técnicas de FDD e *SmartRe*.

O momento da criação do *Product Backlog* seria integrado com os 3 primeiros processos do FDD, "Desenvolver um modelo abrangente", "Construir lista de Funcionalidades" e "Planejar por Funcionalidades", pois são os processos que definem

e detalham quais os requisitos e funcionalidades farão parte do produto a ser desenvolvido, ou seja, do *Product Backlog*. O projeto inicia com o desenvolvimento de um modelo abrangente do produto (primeiro processo da FDD), e logo depois se constrói uma lista de funcionalidades (segundo processo da FDD). Cada funcionalidade levantada é planejada e neste instante, é verificado as dependências entre os recursos, a complexidade das funcionalidades a serem implementadas e feita a antecipação de possíveis riscos, consideração de quaisquer marcos externos como pontos de checagem e *feedback*.

Quando é realizado o terceiro processo da FDD, "Planejar por Funcionalidade", identifica-se a ocasião ideal para incluir a categorização de requisitos de acordo com a proposta de Keepence (1995), pois é a etapa em que os requisitos são definidos, estimados e suas relações de dependência aparecem. Os requisitos são categorizados por não-reutilizáveis, diretamente reutilizáveis e parcialmente reutilizáveis (modelo de requisito).

O planejamento da *Sprint* é realizado após essa definição das funcionalidades, dos requisitos e do plano das funcionalidades com a categorização dos requisitos. No plano da *Sprint* existem tarefas para detalhar as funcionalidades (quarto processo da FDD) e para a construção das funcionalidades (quinto processo da FDD). Nos dois últimos processos da FDD os requisitos são modelados em classes e diagramas de sequencia e após este processo são implementadas as classes e os métodos, além da inspeção de código, testes de unidades desenvolvidas, e da construção da funcionalidade. A metodologia FDD sugere a utilização de UML para a modelagem.

Ao final de cada processo, na empresa de desenvolvimento de software, será realizada uma verificação das saídas previstas para que o próximo processo possa ser iniciado.

As características do Scrum não são alteradas em nenhuma situação, apesar da utilização de outras técnicas em conjunto e o Scrum não perde sua essência. Seus eventos de duração pré-definida continuam da mesma forma e seus artefatos não são alterados, além disso sua proposta de agilidade é mantida, permanecendo um gerenciamento eficaz. As técnicas aparecem como complemento de algumas áreas do desenvolvimento de software em que o Scrum não abrange.

3.3 Prática das técnicas utilizadas em conjunto

Na empresa de desenvolvimento de software em que este estudo se baseia, um de seus produtos é uma ferramenta web para criação e gerenciamento de formulários. As técnicas de FDD e *SmartRe* foram utilizadas em conjunto com o *framework* Scrum, motivadas por esta presente monografia e como teste durante um mês, abrangendo o planejamento e a duração de uma *Sprint*. Devido ao sigilo das informações, somente alguns passos do processo realizado foram disponibilizados e foi autorizada a divulgação de apenas um dos requisitos do produto. O requisito selecionado para a demonstração de parte das técnicas aplicadas é o requisito "Relatórios", que fez parte da primeira *Sprint* da terceira *Release* do produto.

Os tópicos utilizados do primeiro processo do FDD, "construir um modelo abrangente", foram criar uma explicação do domínio (a *user story* foi utilizada com essa função), modelagem lógica dos dados, criar um diagrama de classe, e escrever observações do por que o modelo foi escolhido e quais alternativas foram levadas em consideração. Ao final o resultado foi um modelo de objetos de alto nível, que serviu de referência para os desenvolvedores.

A *user Story* definida para este requisito foi: "Como um **usuário da ferramenta para criação e gestão de formulários** eu quero **gerar relatórios com os itens que eu selecionar e com a totalização de respostas e seus respectivos percentuais** para que seja possível **fazer análises especificas**".

Com a *user story* definida, com os primeiros diagramas prontos e com as observações a respeito dos diagramas produzidos, foi o momento em que iniciou o segundo processo da FDD "criar uma lista de funcionalidades".

De acordo com Palmer e Felsing (2002) e a partir dos itens gerados no processo anterior o domínio foi decomposto em áreas neste segundo processo (*major feature sets*), e cada área foi separada por atividades de negócios, e cada atividade com sua lista de funcionalidades. Cada item da lista de funcionalidades foi composto por "ação",

"resultado", e "objeto". O resultado deste processo foi uma hierarquia de funcionalidades a serem construídas, conhecido no Scrum como *Product Backlog*. Parte da lista de funcionalidades gerada para esta iteração no requisito "Relatórios" está na figura 5.

Esta lista é a saída do segundo processo do FDD e entrada para a terceiro processo do FDD "planejar por funcionalidade", aplicando a teoria de Palmer e Felsing (2002).

F05 - Relatórios	
F051	*Calcular* o *total* das *respostas coletadas*
F052	*Calcular* o *percentual* das *respostas coletadas em relação aos participantes*
F053	*Calcular* o *percentual* das *respostas coletadas em relação aos concluídos*
F054	*Calcular* o *número* de *não respostas para cada questão*
...	...
...	...

Tabela 1 – Lista de funcionalidades dos Relatórios

Planejar por funcionalidade, de acordo com a ideia de Palmer e Felsing (2002), foi planejar a ordem que as funcionalidades seriam implementadas, baseado na dependência e na complexidade das funcionalidades a serem implementadas. As tarefas básicas desse processo não tem uma sequencia exata. Determinar uma sequencia para o desenvolvimento de acordo com o valor de negócio para o cliente e estimar as horas que serão gastas em cada etapa do processo de desenvolvimento foram as principais atividades deste processo.

No projeto do software de criação e gerenciamento de formulários web os itens da lista de funcionalidades além de terem sido sequenciados, estimados e classificados de acordo com o valor de negócio, foram também classificados em não reutilizáveis, reutilizáveis, e parcialmente reutilizáveis, de acordo com Keepence (1995), como mostra a tabela 2 abaixo.

Business Value	Área de Negócio	Código	Item	Dependência	Estimativa em horas
100	Relatórios	F051	Calcular o total de respostas coletadas	N/A	3,0
50	Relatórios	F052	Calcular o percentual de respostas em relação aos participantes	N/A	3,0
50	Relatórios	F053	Calcular o percentual de respostas em relação aos concluídos	N/A	3,0
80	Relatórios	F054	Calcular o total de não respostas para cada questão	N/A	3,0
100	Relatórios	F055	Selecionar as questões do relatório	N/A	6,0
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...

Tabela 2 – Estimativa após o processo de planejamento por funcionalidade

O valor de negócio de cada item foi estipulado pelo *Product Owner*, e tem um valor que varia de 10 a 100. Este intervalo de valores foi definido por todos os envolvidos no projeto. O valor mínimo escolhido foi 10 porque os gestores quiseram evitar que funcionalidades com valor de negócio 0 fossem deixadas de lado pelo time de desenvolvimento.

A categorização dos requisitos foi efetuada no processo de planejamento das funcionalidades. A Tabela abaixo demonstra como ficaram os requisitos do "Relatório" categorizados de acordo com Keepence (1995)

Área de Negócio	Código	Item	Categoria do Requisito
Relatórios	F051	Calcular o total de respostas coletadas	Não Reutilizável
Relatórios	F052	Calcular o percentual de respostas em relação aos participantes	Não Reutilizável
Relatórios	F053	Calcular o percentual de respostas em relação aos concluídos	Não Reutilizável
Relatórios	F054	Calcular o total de não respostas para cada questão	Não Reutilizável
Relatórios	F055	Selecionar as questões do relatório	Diretamente Reutilizável
...	...	...	...
...	...	...	...
...	...	...	...

Tabela 3 – Categorização dos requisitos no projeto da ferramenta de formulários

Os requisitos foram classificados em "não reutilizáveis" quando eram funcionalidades específicas do projeto em questão, foram classificados em "diretamente reutilizáveis" quando havia possibilidade da funcionalidade ser utilizada sem alterações em outras partes do projeto e/ou outros projetos, e foram classificados em "parcialmente reutilizáveis" os requisitos que poderiam ser utilizados em outras partes do projeto e/ou em outros projetos contanto que alguns parâmetros fossem alterados.

Além do requisito "Relatórios", outro requisito, "Gráficos", fazia parte do *Product Backlog*. Este outro requisito é o responsável pela geração de gráficos dos relatórios desenvolvidos. A figura acima demonstra que a funcionalidade "F055 – Selecionar questões do relatório" foi classificada como "diretamente reutilizável", pois esta mesma

funcionalidade faz parte do requisito "Gráficos" do projeto, onde o usuário seleciona a questão em que ele quer ver os gráficos.

O principal motivo da classificação dos requisitos em relação a sua reutilização é simplesmente por gerar economias futuras com o processo de desenvolvimento e analise de requisitos. Keepence (1995) afirma que esta tarefa de classificação dos requisitos pode tornar a primeira fase do projeto mais cara, porém a relação custo-benefício geralmente é significativa quando se trata de outros projetos que reutilizam requisitos existentes de projetos anteriores.

Os três primeiros processos FDD ("criar um modelo abrangente", "criar uma lista de funcionalidades" e "planejar por funcionalidade") e a categorização dos requisitos (*SmartRe*) tiveram a duração de uma semana no total. O *Product Owner,* o *Scrum Master* e o time de desenvolvimento foram os responsáveis pelo primeiro processo do FDD incluído no *framework* Scrum. O segundo processo, juntamente com o terceiro e a categorização de requisitos foi de responsabilidade do time de desenvolvimento, auxiliados pelo *Scrum Master*. A atribuição do valor de negócio foi feita pelo *Product Owner* durante o terceiro processo.

Com os três primeiros processos do FDD, que correspondem ao *Product Backlog*, finalizados, a próxima etapa efetuada foi o planejamento da Sprint. Como as tarefas já foram estimadas, classificadas, e ordenadas de acordo com a prioridade em relação ao valor dado pelo *Product Owner*, o procedimento de criar um *Sprint Backlog* se tornou mais simples, pois o trabalho se tornou apenas escolher os itens mais prioritários e que se encaixavam na disponibilidade de horas do time.

Após o planejamento da *Sprint* e após a criação do *Sprint Backlog* o ciclo da *Sprint* iniciou. A duração estimada para esta *Sprint* foi de duas semanas, ou dez dias úteis. O número de profissionais envolvidos foi de cinco desenvolvedores, um *Scrum Master* e um *Product Owner*. A duração da *Sprint* em horas estava planejada em 300 horas com uma velocidade de 30 horas/dia.

Durante o ciclo da *Sprint* aconteceram os dois últimos processos da *Feature Driven Development*, "detalhar por funcionalidade" e "construir por funcionalidade".

A etapa de "detalhar por funcionalidade" consistiu em "desenvolver diagramas de sequencia" para requisitos complexos, "refinar o modelo de objetos" criado no primeiro

processo FDD, "escrever classes e métodos" e "inspeção de *desing*" para as classes obedecerem ao padrão utilizado pela empresa. Estes passos executados estão de acordo com as tarefas propostas por Palmer e Felsing (2002).

Esta etapa é finalizada com a entrega dos diagramas de sequencia, modelos de classes. Após a finalização do quarto processo da FDD é iniciado o quinto e ultimo processo da *Feature Driven Development* que é a "construção por funcionalidade".

Na "construção por funcionalidade" todo material gerado no processo anterior é implementado e testado. A implementação e os testes são feitos por unidade, ou seja, cada funcionalidade é implementada e testada separadamente.

Ao final da *Sprint* foi realizada uma reunião de revisão onde a equipe de desenvolvimento entregou as funcionalidades planejadas ao *Product Owner*. Após testar e validar as entregas, o *Product Owner* comunicou o que estava de acordo com o combinado e o que não estava. Algumas tarefas que o PO não aceitou voltaram para a lista de pendências, mas elas não tiveram a necessidade de serem replanejadas. Faltaram apenas pequenos ajustes, como o alinhamento de *layout*.

Depois que os requisitos foram construídos e apresentados para o *Product Owner*, foi o momento de realizar a reunião de retrospectiva da *Sprint*. Os pontos positivos e negativos da *Sprint* realizada foram levantados e discutidos, e além disso o time propôs formas de melhorar o desempenho para o próximo ciclo.

3.4 Resultados da experiência realizada na empresa de desenvolvimento de software

Na reunião de entrega de funcionalidades da empresa de desenvolvimento de softwares o *Product Owner* aceitou 90% das funcionalidades planejadas, sendo que estes 10% não foram aceitos devido a acertos no *layout* e em um dos casos, acerto de cálculo da funcionalidade. O requisito "Relatórios" foi entregue com sucesso.

Ao final da reunião de revisão da *Sprint* é realizada a reunião de retrospectiva da *Sprint*. Durante esta reunião os membros da equipe ressaltaram que o andamento da *Sprint* teve uma dinâmica melhor e que a complicação com o entendimento das tarefas quase desapareceu, em alguns casos o *Scrum Master* teve de intervir para auxiliar a equipe de desenvolvimento. Outro ponto importante levantado na reunião é que, apesar de 10% do planejado não ter sido aceito pelo *Product Owner*, a equipe não atrasou em relação ao planejado.

O fato de produzir modelos e estudar as funcionalidades antes de começar o desenvolvimento, gerou um número maior de reuniões com os integrantes da equipe. No primeiro momento foi realizada apenas uma reunião antes do início do desenvolvimento, com esta nova proposta o número aumentou para 5 reuniões. Isso aumentou o numero de pessoas com conhecimento sobre as funcionalidades e melhorou a colaboração entre os participantes do projeto.

O gráfico *burndown* desta *Sprint* ficou conforme a figura 5, e comparado com o das *Sprints* anteriores fica claro a evolução do time.

Dias do Projeto	Estimativa	1	2	3	4	5	6	7	8	9	10
Plano	300,0	270,0	240,0	210,0	180,0	150,0	120,0	90,0	60,0	30,0	-
Horas Planejadas Entregues		38,0	29,0	28,0	31,0	26,0	30,0	29,0	27,0	29,0	33,0
Andamento das Entregas	300,0	262,0	233,0	205,0	174,0	148,0	118,0	89,0	62,0	33,0	-
Velociadade do Time	30,0										

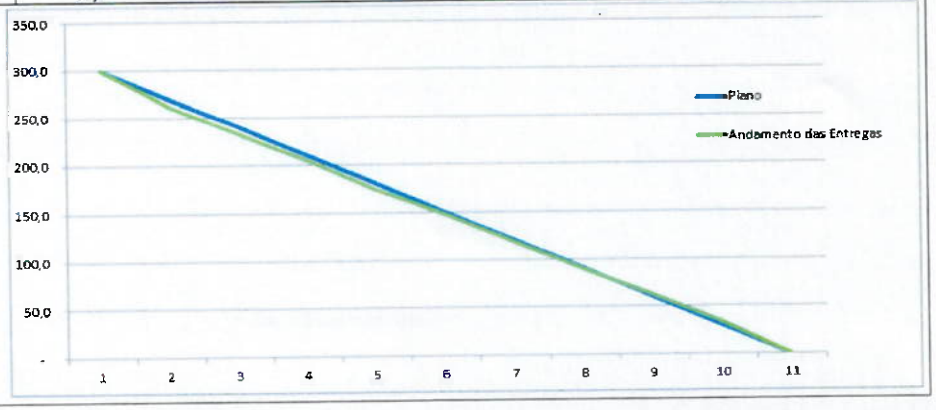

Figura 5 – Gráfico *Burndown Sprint* 2 *Release* 3 do projeto – com a utilização de FDD e *SmartRe* em conjunto com Scrum.

O primeiro ponto a ser observado é que mesmo com alguns dias com entregas abaixo do que o time propôs a entregar (velocidade do time), ao final da *Sprint* o time conseguiu recuperar o ritmo e finalizar com sucesso este ciclo. O segundo ponto a considerar é que o time não se distanciou muito do planejado em nenhum dia da *Sprint*, isso mostra que o time não acumulou atrasos. Este é um resultado muito positivo em comparação às *Sprints* anteriores, pois o acúmulo de atrasos e o distanciamento do que foi planejado era constante e impossível de conseguir recuperação (o time já esteve com 106 horas de atraso ao final de uma *Sprint* como mostrado no item 3.1 deste capítulo).

A dependência entre as tarefas também foi um fator que levou o time a não cumprir o tempo estabelecido para o dia da Sprint em alguns casos, mas os dias abaixo do planejado foram compensados por dias acima do esperado. Na visão geral da Sprint o planejamento foi bem realizado devido ao resultado obtido.

O time observou durante a reunião que esta foi a melhor *Sprint* realizada até o momento, levando em consideração as entregas para o *Product Owner* e o prazo estabelecido, que foi cumprido pela primeira vez apesar da necessidade de alguns ajustes na entrega final.

O fato de planejar por funcionalidades e classificar os requisitos deu uma maior segurança para os desenvolvedores no momento da implementação e principalmente no momento de estipular prazos.

4 Conclusão

O processo proposto foi realizado de acordo com as especificações da *Feature Driven Development* (FDD) e das especificações de requisitos *SmartRe* (requisitos reutilizáveis). O time de desenvolvimento esteve um pouco resistente no início do processo devido a um número maior de reuniões para a definição dos requisitos a serem trabalhados durante a *Sprint*. Após esta fase de definição o time de desenvolvimento esteve mais a vontade e seguro para realizar seu trabalho, pois houve uma significativa diminuição das dúvidas em relação às funcionalidades a serem implementadas.

O time conseguiu manter um ritmo interessante durante toda a duração da *Sprint* e isso foi primordial para o cumprimento dos prazos estabelecidos. O fato de estabelecer diretrizes concretas para o desenvolvimento do software em questão trouxe outros benefícios além do previsto, como o aumento da colaboração entre os membros da equipe, melhora no comprometimento com as entregas previstas, comunicação do time e maior facilidade na organização da equipe devido ao melhor sequenciamento das tarefas. A aplicação das novas técnicas concomitantemente com um melhor entendimento das práticas do Scrum levaram o time a ter um melhor desempenho no andamento do projeto.

Com esta monografia conclui-se que é possível utilizar práticas de engenharia de software em conjunto com a metodologia Scrum com o objetivo de aumentar a produtividade e mitigar atrasos e retrabalho. As técnicas de engenharia de software *Feature Driven Development* (FDD) juntamente com as técnicas de escrita de requisitos reutilizáveis *SmartRe* se mostraram capazes de diminuir consideravelmente a lacuna entre a gestão de projetos Scrum, existente na empresa de desenvolvimento de software, e as práticas de engenharia de software.

Para trabalhos futuros a sugestão é a combinação de técnicas para aprimorar os testes de software, como por exemplo, a utilização de *Test Driven Development* (TDD) em conjunto com as práticas desenvolvidas por esta monografia, com o objetivo de aperfeiçoar a qualidade dos softwares desenvolvidos.

REFERÊNCIAS BIBLIOGRÁFICAS

ANAND, Ashok; SEKAR, Vyas; AKELLA, Aditya. **SmartRE: An Architecture for Coordinated Network-wide Redundancy Elimination**. Barcelona: SIGCOMM, 2009.

BRITO, Emerson José Morgado. **Diretrizes para avaliação de ferramentas de gestão de projetos utilizando metodologias ágeis**. Maringá, 2012.

CHOWDHURY, Ashraf Ferdouse; HUDA, Mohammad Nazmul. **Comparison between Adaptive Software Development and Feature Driven Development**. IEEE, International Conference on Computer Science and Network Technology, 2011.

DE LUCA, Jeff. Disponível em: http://www.step-10.com/SoftwareProcess/FeatureDrivenDevelopment/. Acesso em: 27/12/2012

ETTINGER, Daniel. **O Backlog do Produto e a arte da User Story**. Disponível em: <http://danielettinger.com/2010/12/30/pb_userstory/>. Acesso em: 26/12/2012

GOYAL, Sadhna. **Agile Techniques for Project Management and Software Engineering**. Technical University Munich, 2007/08.

GUANG-YONG, Hu. **Study And Practice Of Import Scrum Agile Software Development**. Nanjing: IEEE, 2011.

HAYATA, Tomohiro; HAN, Jianchao. **A Hybrid Model for IT Project with Scrum**. IEEE, 2011.

KANWAL, Faria; JUNAID, Komal; FAHIEM, Muhammad Abuzar. **A Hybrid Software Architecture Evaluation Method for FDD – An Agile Process Model**. IEEE, 2010.

KEEPENCE, Barry; MANNION, Mike; SMITH, Stephen. **SMARTRe Requirements: Writing Reusable Requirements**. IEEE, 1995.

KRISHNA, Vinay; BASU, Anirban. **Scrum+ :: Is it "ScrumBut" or "ScrumAnd"**. IEEE, India Conference (INDICON), 2011.

KRUGER, Eduardo. **Por que o Scrum é melhor para a sua empresa e seus clientes?** Disponível em: < http://cio.uol.com.br/opiniao/2012/07/12/por-que-o-scrum-e-melhor-para-a-sua-empresa-e-seus-clientes/>. Acesso em: 15/03/2013

MANNION, Mike; KEEPENCE, Barry. **SMART Requirements**. Software Engineering Notes, Vol. 20, nº 2, 1995.

MELO, Claudia de; CRUZES, Daniela; KON, Fabio; CONRADI, Reidar. **Interpretative case studies on agile team productivity and management**. Elsevier, 2012.

MILLER, Granville. **Want a Better Software Development Process? Complement It**. IEEE, September, 2003.

MOE, Nils Brede; DINGSØYR, Torgeir; DYBÅ Tore. **A teamwork model for understanding an agile team: A case study of a Scrum project**. Elsevier, 2009.

NASCIMENTO, Paula. **Primeiro passo de um projeto Scrum: Visão**. Disponível em: <http://blog.myscrumhalf.com/2012/07/qual-o-primeiro-passo-de-um-projeto-scrum-visao>. Acesso em: 13/01/2013

OTA, Martin. **Scrum in Research**. Praga: Springer-Verlag Berlin Heidelberg, 2010. Y. Luo (Ed.): CDVE 2010, LNCS 6240, pp. 109 – 116.

PALMER, Stephen; FELSING, Mac. **A Practical Guide to Feature-Driven Development**. The Coad Series, 2002.

PRIES-HEJE, Lene; PRIES-HEJE, Jan. **Agile & Distributed Project Management: A Case Study Revealing Why Scrum Is Useful**. ECIS, 2011.

RISING, Linda; JANOFF, Norman. **The Scrum Software Development Process for Small Teams**. IEEE Software, Julho/Agosto 2000.

RYCHLÝ, Marek; TICHÁ, Pavlína. **A Tool for Supporting Feature-Driven Development**. IFIP International Federation for Information Processing, 2008.

SCRUM GUIDE. Disponível em: <http://www.scrum.org>. Acesso em: 18/12/2012

SIDDIQUI, Farheen; ALAM, M. Afshar. **Ontology Based Feature Driven Development Life Cycle**. International Journal of Computer Science Issues, Vol. 9, 2012.

VLAANDEREN Kevin; JANSEN, Slinger; BRINKKEMPER, Sjaak; JASPERS, Erik. **The agile requirements refinery: Applying SCRUM principles to software product management**. Utrecht: Elsevier, 2010.

WAGH, Ramrao. **Using Scrum for Software Engineering Class Projects**. Goa: IEEE – Computer Society, 2012.

WILLIAMS, Laurie; BROWN, Gabe; MELTZER, Adam; NAGAPPAN, Nachiappan. **Scrum + Engineering Practices: Experiences of Three Microsoft Teams**. ESEM, 2011.