UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

***CAMPUS* LUIZ MENEGHEL**

RENATA MARQUES BARROS

ESTUDO DAS SUBÁREAS DA ENGENHARIA DE REQUISITOS EM METODOLOGIAS ÁGEIS

Bandeirantes

2013

RENATA MARQUES BARROS

ESTUDO DAS SUBÁREAS DA ENGENHARIA DE REQUISITOS EM METODOLOGIAS ÁGEIS

Trabalho de Conclusão de Curso apresentado à Universidade Estadual do Norte do Paraná Campus Luiz Meneghel como requisito para obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Profº. Drº. André Luís Andrade Menolli

Bandeirantes

2013

RENATA MARQUES BARROS

ESTUDO DAS SUBÁREAS DA ENGENHARIA DE REQUISITOS EM METODOLOGIAS ÁGEIS

Trabalho de Conclusão de Curso apresentado à Universidade Estadual do Norte do Paraná Campus Luiz Meneghel como requisito para obtenção do grau de bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

_____
Profº. Drº. André Luís Andrade Menolli
UENP – *Campus* Luiz Meneghel

_____
Profº. Me. José Reinaldo Merlin
UENP – *Campus* Luiz Meneghel

_____
Profº. Me. Rodrigo Tomaz Pagno
UENP – *Campus* Luiz Meneghel

Bandeirantes, __ de_____ de 2013

AGRADECIMENTOS

Agradeço primeiramente a Deus, pois, sem Ele nada eu poderia fazer. Ele é a minha força e me abençoou para que eu chegasse até aqui. Aos meus pais Raimundo e Regina por toda dedicação, amor e apoio oferecidos a mim durante toda a minha vida. À minha irmã Jaqueline por ser um exemplo de pessoa dedicada e esforçada. Ao meu namorado Walmor por ser paciente, me incentivar e motivar a acreditar em mim mesma. Agradeço aos meus amigos e companheiros Camila, Dayanne, Jéssica, Nathália, Patrícia e Robson por estarem comigo desde o início do período acadêmico. Ao meu orientador André Menolli por ser atencioso, prestativo, paciente e sempre se dispor a me orientar durante todo o desenvolvimento desta pesquisa.

Ser sábio é melhor do que
ser forte; o conhecimento
é mais importante do que
a força.
(Provérbios 24:5)

RESUMO

O processo de desenvolvimento de software abrange várias áreas e etapas até ser desenvolvido o produto final. A engenharia de requisitos faz parte desse processo sendo considerada uma das áreas críticas e que contribui significantemente para o sucesso do projeto quando executada de maneira precisa. Quando, porém, as subáreas da engenharia de requisitos são executadas de maneira falha, o resultado final provavelmente não atenderá às necessidades e expectativas dos *stakeholders*. Nesta pesquisa é realizado um estudo profundo sobre as subáreas de requisitos em metodologias ágeis. Essas metodologias visam o rápido desenvolvimento de software e a sua utilização tem crescido ao longo dos últimos anos. Objetiva-se conhecer como são executadas ás subáreas da engenharia de requisitos nas principais metodologias ágeis e identificar quais artefatos são utilizados. Foram realizadas análises sobre os artefatos para indicar os mais adequados para cada subárea da engenharia de requisitos. Assim, por meio desta pesquisa é possível obter maior familiaridade com o assunto.

Palavras-chave: Engenharia de Requisitos, Metodologias Ágeis, Artefatos.

ABSTRACT

The process of software development in several areas and steps to the final product being developed . The requirements engineering is part of this process is considered one of the critical areas and contributes significantly to the success of the project when executed accurately . When, however , the sub-areas of requirements engineering are performed so fail, the end result probably will not meet the needs and expectations of stakeholders. In this research a thorough study of the sub- requirements in agile methodologies is performed . These methodologies allow the rapid development of software and its use has grown over the past years. The objective is to know how to run the sub- requirements engineering in major agile methodologies and identify which artifacts are used . Analysis of the artifacts were taken to indicate the most suitable for each area of requirements engineering . Thus, through this research it is possible to gain greater familiarity with the subject .

Keywords: Requirements Engineering, Agile Methodologies, Artifacts.

LISTA DE FIGURAS

Figura 1 - Dados sobre desempenho de projetos de software.....15
Figura 2 - Principais causas de fracasso em projetos de software.....16
Figura 3 - Visão geral do ciclo de vida do *scrum*.....24
Figura 4 – Ciclo de vida de XP.....31
Figura 5 – Ciclo de vida DSDM.....36
Figura 6 - Ciclo de vida FDD.....39
Figura 7 - Criticidade e número de pessoas envolvidas metodologias Crystal.....41
Figura 8 – Ciclo de vida ASD.....43
Figura 9 - Tela inicial do coordenador de monografias.....68
Figura 10 - Tela de avaliação de monografia.....69
Figura 11 - Teste no método cadastrarProfessor.....73

LISTA DE QUADROS

Quadro 1 – Engenharia de requisitos implementação em SCRUM.....24

Quadro 2 – Engenharia de requisitos implementação em XP.....32

Quadro 3 – Técnicas de Modelagem.....33

Quadro 4 - Práticas de Crystal Clear e Crystal Orange 41

Quadro 5 - Artefatos das metodologias ágeis na subárea de elicitação de requisitos.....54

Quadro 6 - Artefatos das metodologias ágeis na subárea de análise de requisitos.....55

Quadro **7-** Artefatos das metodologias ágeis na subárea de especificação de requisitos.....58

Quadro 8 - Artefatos das metodologias ágeis na subárea de validação de requisitos.....59

Quadro 9 - Cartão de História – *User Stories* 1.....63

Quadro 10 – Cartão de História – *User Stories* 2.....64

Quadro 11 – Cartão de História – *User Stories* 3.....64

Quadro 12 – Documento de requisitos para elicitação de requisitos.....64

Quadro 13. Cartão CRC 1.....67

Quadro 14. Cartão CRC 2.....68

Quadro 15. Product Backlog.....69

Quadro 16. Teste de aceitação para user story 1.....71

Quadro 17. Exemplo de um cenário a ser testado 1.....71

Quadro 18. Teste de aceitação para user story 2.....72

Quadro 19. Exemplo de um cenário a ser testado 2.....72

Quadro 20. Teste de aceitação para user story 3.....72

Quadro 21. Exemplo de um cenário a ser testado 3.....73

Quadro 22. Critérios para avaliação dos artefatos na subárea de elicitação de requisitos.....74

Quadro 23. Critérios para avaliação dos artefatos na subárea de análise de requisitos.....76

Quadro 24. Critérios para avaliação dos artefatos na subárea de especificação de requisitos..77

Quadro 25. Critérios para avaliação dos artefatos na subárea de validação de requisitos.....78

SUMÁRIO

1 INTRODUÇÃO 13
1.1 CONTEXTO E DELIMITAÇÃO DO TRABALHO 13
1.2 FORMULAÇÃO DO PROBLEMA 14
1.3 OBJETIVOS 16
1.3.1 Objetivo Geral 16
1.3.2 Objetivos Específicos 17
1.4 JUSTIFICATIVA 17
1.5 ORGANIZAÇÃO DO TRABALHO 17
2 MÉTODOS E MATERIAIS 19
3 METODOLOGIAS ÁGEIS 21
3.1 SCRUM 21
3.1.1 Ciclo de Vida SCRUM 23
3.1.2 SCRUM – Engenharia de Requisitos 24
3.2 EXTREME PROGRAMMING (XP) 25
3.2.1 Práticas do XP 26
3.2.1.1 Cliente Presente 26
3.2.1.2 Jogo do Planejamento 26
3.2.1.3 *Stand Up Meeting* 27
3.2.1.4 Programação em Par 27
3.2.1.5 Desenvolvimento Guiado pelos Testes 27
3.2.1.6 Refatoração 28
3.2.1.7 Código Coletivo 28
3.2.1.8 Código Padronizado 29
3.2.1.9 Design Simples 29
3.2.1.10 Metáfora 29
3.2.1.11 Ritmo Sustentável 29
3.2.1.12 Integração Contínua 30
3.2.1.13 Releases Curtos 30
3.2.2 Ciclo de Vida XP 30
3.2.3 Extreme Programming – Engenharia de Requisitos 32
3.3 DYNAMIC SYSTEM DEVELOPMENT METHOD – DSDM 34
3.3.1 Ciclo de Vida – DSDM 35

3.3.2 DSDM – ENGENHARIA DE REQUISITOS 36
3.4 Feature Driven Development (FDD) 37
3.4.1 FDD – Engenharia de Requisitos 39
3.5 Crystal Family of Methodologies 40
3.5.1 Crystal Family of Methodologies – Engenharia de requisitos 42
3.6 Adaptive Software Development (ASD) 42
3.6.1 Adaptive Software Development – Engenharia de Requisitos 44
4 DESENVOLVIMENTO 46
4.1 ELICITAÇÃO DE REQUISITOS 46
4.2 ANÁLISE DE REQUISITOS 48
4.2.1 Classificação de Requisitos 48
4.2.2 Modelagem Conceitual 49
4.2.3 Projeto Arquitetônico e Requisitos de Alocação 50
4.2.4 Negociação de Requisitos 50
4.3 ESPECIFICAÇÃO DE REQUISITOS 51
4.4 VALIDAÇÃO DE REQUISITOS 52
4.4.1 Revisão de requisitos 52
4.4.2 Prototipagem 53
4.4.3 Validação do Modelo 53
4.4.4 Teste de Aceitação 54
4.5 LEVANTE DOS ARTEFATOS 54
5 Análise dos Artefatos das Metodologias 62
5.1 DESCRIÇÃO DO SISTEMA 62
5.2 APLICAÇÃO DOS ARTEFATOS NA SUBÁREA DE ELICITAÇÃO DE REQUISITOS 63
5.3 APLICAÇÃO DOS ARTEFATOS NA SUBÁREA DE ANÁLISE DE REQUISITOS 67
5.4 APLICAÇÃO DOS ARTEFATOS NA SUBÁREA DE ESPECIFICAÇÃO DE REQUISITOS 69
5.5 APLICAÇÃO DOS ARTEFATOS NA SUBÁREA DE VALIDAÇÃO DE REQUISITOS 71
6 RESULTADOS 74
6.1 RESULTADOS NA ELICITAÇÃO DE REQUISITOS 74
6.2 RESULTADOS NA ANÁLISE DE REQUISITOS 75
6.3 RESULTADOS NA ESPECIFICAÇÃO DE REQUISITOS 77
6.4 RESULTADOS NA VALIDAÇÃO DE REQUISITOS 78
7 CONCLUSÃO 80
REFERÊNCIAS 82
APÊNDICE A - ARTEFATOS ELICITAÇÃO DE REQUISITOS 85

APÊNDICE B - ARTEFATOS ANÁLISE DE REQUISITOS.....89
APÊNCIDE C - ARTEFATOS VALIDAÇÃO DE REQUISITOS.....90

1 INTRODUÇÃO

1.1 CONTEXTO E DELIMITAÇÃO DO TRABALHO

O processo de desenvolvimento de software ainda enfrenta desafios para o alcance de resultados satisfatórios. Um dos aspectos falhos é a compreensão do que o cliente espera do sistema. Embora aparentemente simples, muitas vezes esse processo não é realizado de maneira eficiente, o que influência de forma negativa o resultado final do projeto. As metodologias voltadas para o desenvolvimento de *software* devem oferecer suporte para melhorar esse aspecto.

Atualmente, existem diversas metodologias voltadas para o desenvolvimento de *software*, as tradicionais como RUP, Cascata, Espiral e as ágeis como XP e SCRUM. Para Paetsch, Eberlein e Maurer (2003), as metodologias ágeis tornaram-se mais populares nos últimos anos e têm buscado alcançar o objetivo da entrega mais rápida de *software* e garantir que o *software* atenda às necessidades do cliente.

A principal diferença entre o desenvolvimento tradicional e ágil não é a execução ou não dos processos de engenharia de requisitos, mas quando executá-los (LUCIA e QUSEF, 2010, tradução nossa). A ideia de ser ágil pode levar a conclusões errôneas sobre como se desenvolve software em metodologias ágeis. Embora, os processos de engenharia de requisitos não sejam seguidos tão "formalmente" como em métodos tradicionais, nos processos ágeis, também são executados, porém, de maneira mais flexível. Outra diferença é a participação do cliente. Em métodos tradicionais o cliente participa, em geral, apenas nas primeiras fases do desenvolvimento. E em metodologias ágeis o almejado é a presença do cliente durante todo o desenvolvimento do projeto.

Em metodologias tradicionais e ágeis umas das primeiras atividades é a elicitação e análise de requisitos. Segundo Pressman (2006), aparentemente pode parecer simples identificar os requisitos necessários para o sistema, entender como o sistema será introduzido nas necessidades do negócio e entre outros. Porém, não é fácil e não é simples, é uma tarefa difícil.

Existem diversas técnicas nas metodologias tradicionais para elicitação e análise de requisitos como entrevistas, etnografia, prototipagem, brainstorming, priorização de requisitos e modelagem. As metodologias ágeis também utilizam algumas dessas técnicas e outras como *user stories*, cartões CRC (Class Responsability Collaborator – Classe Responsabilidade

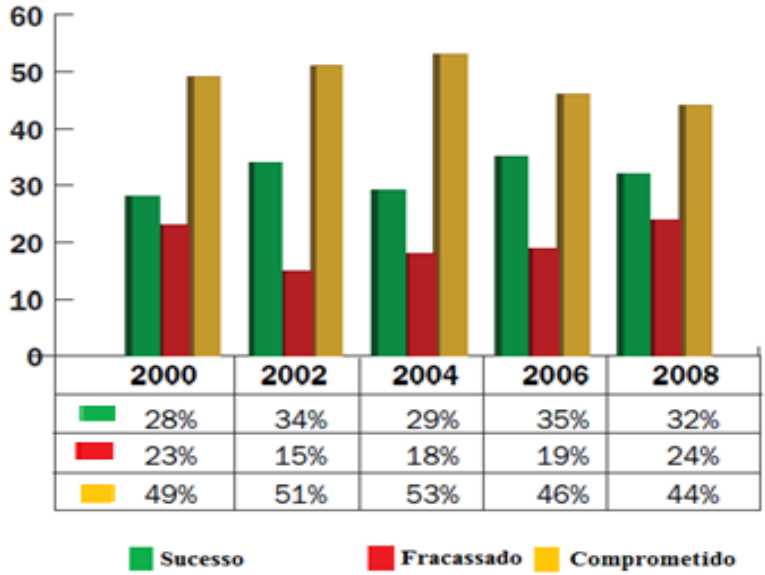
Colaborador) e lista de requisitos. Em geral, não existe uma regra para quais devam ser utilizadas.

Este trabalho é focado em metodologias ágeis e em como essas metodologias executam as subáreas da engenharia de requisitos. Na literatura são encontradas muitas metodologias ágeis, como por exemplo, o trabalho "*Requirements Engineering in Agile Software Development*" (Lucia e Qusef, 2010) diz que as metodologias Modelagem Ágil (MA), Feature Driven Development (FDD), Dynamic System Development Method (DSDM), Extreme Programming (XP) e SCRUM são as mais comuns, enquanto que o trabalho "Requirements Engineering and Agile Software Development" de Paetsh, Eberlein e Maurer (2003) cita XP, SCRUM, MA, The Crystal Methodologies, FDD, DSDM e Adaptive Software Development (ASD) como as mais comuns. Com o suporte desses trabalhos é proposto utilizar como base à nossa pesquisa essas metodologias.

1.2 FORMULAÇÃO DO PROBLEMA

Em Massachusetts, EUA, existe uma empresa conhecida como "The Standish Group", esse grupo realiza estudos que verificam o desempenho do processo de desenvolvimento de software das empresas e apontam as taxas de sucesso, fracasso, atrasos, entre outros, de projetos de TI. No documento Chaos Summary 2009 – The 10 Laws of Chaos é mostrado alguns resultados entre os anos 2000 e 20008 no que se refere a projetos classificados com sucesso, fracasso e comprometidos. Na Figura 1 são mostrados os resultados.

Figura 1. Dados sobre desempenho de projetos de software



Fonte: Chaos Summary 2009 – The 10 Laws of Chaos

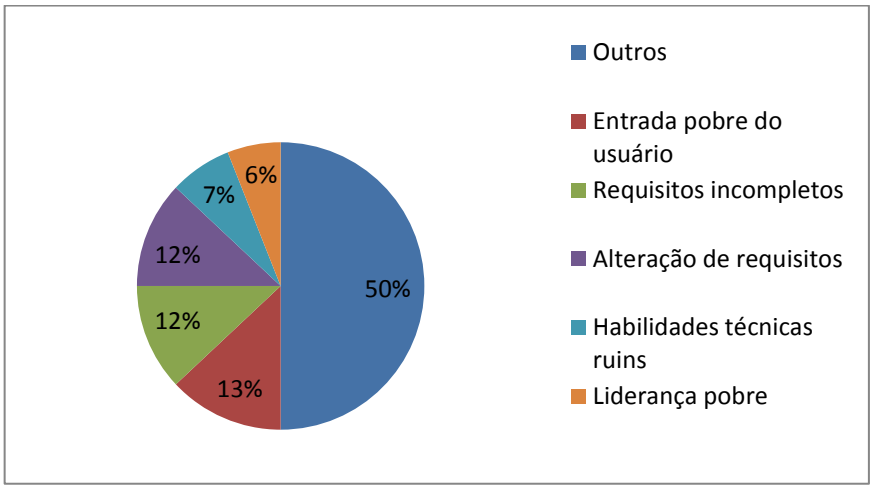
Os resultados apontam algumas variações nos três aspectos (sucesso, fracasso e comprometido), porém, pode-se observar que os projetos com sucesso tiveram variações de crescimento e queda a cada dois anos. Os dados mostram que em 2002 houve um índice de crescimento de 6% em relação a 2000, em 2004 houve uma queda de 5%, em 2006 novamente houve um crescimento de 6% e em 2008 uma queda de 3%. Portanto, o almejado sucesso em projetos na área de TI não tem crescido significantemente, esse é um desafio enfrentado por grande parte das empresas.

Um dos aspectos que o *software* precisa ter é qualidade. A qualidade de *software*, para Pressman (2006), é definida como: "Conformidade com requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas, que são esperadas em todo software desenvolvido profissionalmente". Vários estudos sobre projetos de software tem mostrado que uma das principais causas para projetos fracassados e/ou comprometidos é a falta de conformidade com os requisitos e funcionalidades esperados pelos clientes.

As fases de requisitos devem ser realizadas com atenção. Problemas inseridos no sistema durante as essas fases são os mais caros para remover. Estudos mostraram que aproximadamente 37% dos problemas encontrados no desenvolvimento de sistemas

desafiantes, estão relacionados com as fases de requisitos (POLINI, 2010, tradução nossa). Na Figura 2 são mostrados alguns resultados.

Figura 2. Principais causas de fracasso em projetos de software



Fonte: Adaptada de Polini (2007)

Portanto, mesmo com os avanços tecnológicos e grande quantidade de técnicas utilizadas nas subáreas de requisitos, especialmente levantamento e análise, ainda há muito que melhorar nessa área, pois influencia diretamente a qualidade e desempenho do software.

1.3 OBJETIVOS

Os objetivos desta pesquisa estão separados em objetivos gerais e objetivos específicos, descritos abaixo.

1.3.1 Objetivo Geral

O objetivo principal desta pesquisa é recomendar os artefatos mais indicados para as subáreas da engenharia de requisitos em metodologias ágeis. Com base nas principais metodologias ágeis e as atividades executadas na elicitação, análise, especificação e validação de requisitos, objetiva-se entender o que cada uma dessas metodologias oferece nas subáreas da engenharia de requisitos, identificar os artefatos utilizados, avaliá-los e finalmente recomendar os mais indicados para cada subárea.

1.3.2 Objetivos Específicos

Para atingir ao objetivo geral é necessário alcançar os seguintes objetivos específicos:

- Identificar as principais metodologias ágeis;
- Analisar as metodologias ágeis selecionadas;
- Analisar as subáreas da engenharia de requisitos nas metodologias ágeis;
- Levantar os artefatos;
- Classificar os artefatos quanto à subárea de requisitos;
- Aplicar os artefatos em um experimento;
- Avaliar os resultados dos artefatos no experimento.

1.4 JUSTIFICATIVA

A importância deste trabalho se reflete na necessidade de melhores práticas e resultados nas subáreas da engenharia de requisitos. Esta área é considerada uma das principais causas de fracasso de projetos, quando realizada de maneira ineficaz e ineficiente. Portanto, ao recomentar os artefatos mais indicados para cada subárea da engenharia de requisitos busca-se aperfeiçoar e alcançar melhor desempenho nessas subáreas para que as fases posteriores sejam realizadas com uma base sólida sobre requisitos claros e principalmente que atendam as necessidades dos *stakeholders*.

As metodologias ágeis dispõem de diferentes artefatos nas subáreas da engenharia de requisitos, em geral, cada uma utiliza um conjunto de artefatos. Não existe um consenso de quais artefatos são os melhores e mais indicados. Portanto, a relevância do trabalho está na necessidade de existir um consenso e indicação dos artefatos a serem utilizados para obtenção de resultados mais satisfatórios no desenvolvimento de *software*.

1.5 ORGANIZAÇÃO DO TRABALHO

O restante do trabalho está organizado da seguinte forma. O capítulo 2 apresenta a metodologia de desenvolvimento utilizada, bem como a definição da estrutura da pesquisa.

O capítulo 3 apresenta as metodologias ágeis analisadas nesse trabalho. São explicados os principais conceitos de cada metodologia ágil e como são executadas as subáreas da engenharia de requisitos nessas metodologias.

O capítulo 4 apresenta a definição das subáreas da engenharia de requisitos. Também apresenta o desenvolvimento do trabalho, o qual contém o levante dos artefatos das subáreas da engenharia de requisitos de cada metodologia ágil.

O capítulo 5 apresenta a análise dos artefatos levantados por meio da aplicação destes em um sistema real.

O capítulo 6 apresenta os resultados da avaliação dos artefatos no qual foram tomados por base critérios que caracterizam cada subárea da engenharia de requisitos.

O capítulo 7 apresenta as considerações finais do trabalho.

2 MÉTODOS E MATERIAIS

Nesta seção são expostos os métodos utilizados nesta pesquisa. A classificação das pesquisas quanto aos seus objetivos gerais pode ser classificada como exploratórias, descritivas e explicativas. As pesquisas exploratórias tem o intuito de explorar, estudar, analisar um determinado assunto para gerar maior familiaridade com o problema. As pesquisas descritivas têm por meta relatar as particularidades de uma população ou fenômeno ou, então, a determinação de relações entre variáveis e para isso utilizam-se algumas técnicas para coleta de dados como questionários e observação. Já as pesquisas explicativas objetivam reconhecer os fatores que determinam os "porquês" dos fenômenos (GIL, 2002).

Para se alcançar os objetivos propostos, esta pesquisa pode ser classificada como exploratória. Pesquisa exploratória porque foi necessário um estudo profundo sobre a engenharia de requisitos e os problemas que ocorrem nessa área, bem como, o estudo sobre as principais metodologias ágeis e como são executados os processos da engenharia de requisitos em cada uma dessas metodologias.

As pesquisas podem ser classificadas quanto ao procedimento técnico utilizado. A classificação quanto aos seus objetivos gerais contribui para familiarização com o segmento teórico. Mas, é necessário confrontar a parte teórica com os dados da realidade. Neste contexto as pesquisas podem ser classificadas quanto ao seu delineamento, que se refere ao desenvolvimento da pesquisa em termos de análise e interpretação da coleta de dados (GIL, 2002).

Sendo assim, esta pesquisa é classificada quanto ao procedimento técnico utilizado e seu delineamento como pesquisa bibliográfica. A pesquisa bibliográfica é desenvolvida a partir de materiais como livros, artigos científicos, jornais, revistas, são materiais que já foram publicados (GIL, 2002).

Levando em consideração a forma da interpretação e análise dos dados obtidos, caracteriza-se a pesquisa como qualitativa. Na pesquisa qualitativa a explicação dos fenômenos e o acrescentar de seus significados são simples, além de não necessitar que os dados obtidos sejam expressos estatisticamente. Geralmente os pesquisadores escolhem explorar seus dados indutivamente (GIL, 2002).

Para o desenvolvimento e alcance dos objetivos desse trabalho são necessários os seguintes passos metodológicos:

1. **Fundamentação Teórica:** para a realização do trabalho foi estudado o tema de metodologias ágeis para uma melhor compreensão de suas características, bem como para compreender como é a execução das subáreas da engenharia de requisitos em cada metodologia ágil.
2. **Estudo das Subáreas da Engenharia de Requisitos:** foram estudados os conceitos de elicitação, análise, especificação e validação de requisitos.
3. **Levante dos Artefatos:** foram levantados e analisados os artefatos que fazem parte da engenharia de requisitos de cada metodologia ágil. Com base nas características de cada artefato, estes foram classificados quanto sua subárea da engenharia de requisitos.
4. **Aplicação dos Artefatos em um Sistema Real:** foram selecionados alguns artefatos para serem aplicados em um sistema real. Cada artefato selecionado foi aplicado a uma ou mais subáreas da engenharia de requisitos no sistema proposto.
5. **Avaliação dos Artefatos:** os artefatos aplicados no sistema real foram confrontados com critérios de avaliação retraídos das definições de cada subárea da engenharia de requisitos. Com o resultado dessa avaliação foi possível recomendar os artefatos mais indicados para cada subárea da engenharia de requisitos em metodologias ágeis.

3 METODOLOGIAS ÁGEIS

O processo de desenvolvimento de software enfrenta vários desafios, especialmente o de entender quais são as reais necessidades dos clientes e entregar *software* no tempo e custo combinado. Aparentemente são aspectos simples de se resolver, mas na realidade grande parte das empresas e organizações que desenvolvem *software* enfrentam esses desafios.

O manifesto ágil iniciado no começo de 2001 (*Manifesto for Agile Sotware Development*) por profissionais que idealizavam melhorar o desempenho no desenvolvimento de *software*, trouxe o surgimento de novas metodologias, as então chamadas metodologias ágeis. Essas metodologias têm um conjunto de práticas e princípios em comum, e valorizam aspectos como:

- Indivíduos e interação ao invés de processos e ferramentas;
- Software em funcionamento mais que documentação abrangente;
- Colaboração do cliente mais que negociação de contratos;
- Responder a mudanças mais que seguir um plano.

A colaboração do cliente é um dos principais diferenciais das metodologias ágeis, pois, entender o que o cliente espera e precisa é um ponto essencial para o sucesso do projeto. A seguir serão descritas as metodologias ágeis mais comuns (PAETSH, EBERLEIN e MAURER, 2003; LUCIA e QUSEF, 2010) e como é a execução das subáreas da engenharia de requisitos nessas metodologias.

3.1 SCRUM

Scrum é um método para a gestão do processo de desenvolvimento de sistema por meio da aplicação de ideias sobre a flexibilidade, adaptabilidade e produtividade a partir da teoria de controle de processo industrial. Scrum se concentra em como a equipe deve trabalhar em conjunto para produzir um trabalho de qualidade em um ambiente em mudança. (PAETSH, EBERLEIN e MAURER, 2003, tradução nossa).

Para melhor compreensão desta metodologia é necessário conhecer os principais conceitos que fazem parte do Scrum. Esses conceitos incluem: os papéis (*Product Owner, Scrum Master, Team*), as cerimônias Scrum (*Sprint Planning Meeting, Daily Scrum, Sprint Reviews Meeting, Sprint Retrospectives Meeting*) e os artefatos (*Product Backlog, Sprint Backlog, Product Increment*).

O *Product Onwer* (proprietário do produto) é o profissional que representa a figura do cliente. Ele é responsável pelo *product backlog* e por gerenciá-lo, incluindo tarefas como expressar e priorizar claramente os itens contidos nesse artefato. O *Scrum Master* é aquele que conduzirá o desenvolvimento do projeto. Ele precisa ter conhecimento sobre os conceitos de *Scrum* (práticas, valores, artefatos) para saber como aplicá-los e assegurar que o time *scrum* tem colocado em prática esses conceitos (SCHWABER, SUTHERLAND, 2011). O terceiro papel em *Scrum* é o time *scrum*.

> [...] O Time Scrum é composto pelo *Product Owner*, a Equipe de Desenvolvimento e o Scrum Master. Times *Scrum* são auto-organizáveis e multifuncionais. Equipes auto-organizáveis escolhem qual a melhor forma para completarem seu trabalho, em vez de serem dirigidos por outros de fora da equipe. Equipes multifuncionais possuem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe. O modelo de equipe no *Scrum* é projetado para aperfeiçoar a flexibilidade, criatividade e produtividade [...] (SCHWABER; KEN e SUTHERLAND, 2011, p.5).

De acordo com Schwaber e Sutherland (2011), uma *sprint* é uma "caixa de tempo", isto é, um período para a realização de um trabalho/tarefa, geralmente, de 30 dias ou menos. Neste período é criada uma versão incremental e usável do produto. *Scrum* trabalha em ciclos de *sprint*, ao termino de uma *sprint* outra é iniciada.

A cerimônia *Sprint Plainning Meeting* (reunião de planejamento da Sprint) é uma reunião feita antes do início de uma *sprint*, pois, nela serão exibidas pelo *product owner* as funcionalidades de maior prioridade e quais deverão ser implementadas na *sprint*. Ao final da reunião, após definidos os objetivos da próxima *sprint* se tem como saída o artefato *sprint backlog* (SCHWABER, SUTHERLAND, 2011).

Daily scrum (reunião diária) é realizada todos os dias, geralmente no período matutino, com duração entre 15 – 30 minutos no máximo. O objetivo é que o time *scrum* fale sobre as atividades realizadas no dia anterior, se tiveram dificuldades para realizar determinada tarefa, quais são os impedimentos que surgiram, enfim, apesar dessas questões levantadas na reunião, o foco não é a resolução desses problemas, mas sim permitir que toda a equipe tenha uma visão abrangente de como está o andamento do projeto e estabelecer as tarefas prioritárias para o dia. O *Product Owner* é quem irá, após a reunião, buscar resolver junto com as pessoas envolvidas na tarefa, os problemas comentados na reunião diária (SCHWABER, SUTHERLAND, 2011).

Sprint Reviews Meeting (reunião de revisão da *sprint*) objetivamente é realizada ao termino de uma *sprint*, onde são exibidos os resultados e se o objetivo estabelecido na reunião de planejamento da *sprint* foi alcançado. Normalmente, logo após o termino dessa reunião, é

feita a *Sprint Restrospectives Meeting* (reunião de restrospectiva da *sprint*) pois, como o próprio nome indica, será feito uma "volta ao passado" da *sprint* para ser identificado pelo time *scrum* o que eles precisam começar ou parar de fazer, e/ou continuar fazendo. Isto é, esta reunião ajuda a identificar os pontos fortes e fracos para melhorar o desenvolvimento na próxima *Sprint* (SCHWABER, SUTHERLAND, 2011).

O *product backlog* é uma lista na qual contêm todos os requisitos necessários ao produto e, assim se torna como um núcleo no qual todas as mudanças que forem necessárias terão como base o *product backlog*. Esse artefato sofre mudanças durante todo o processo de desenvolvimento do produto, pois, os requisitos elicitados inicialmente não são, na maioria das vezes, todos os requisitos necessários para o projeto. Essas mudanças são bem-vindas, não são vistas como um atraso ou obstáculo. Normalmente, os itens do *product backlog* são ordenados por valor, prioridade e necessidade. Os itens de maior prioridade são os que serão desenvolvidos primeiro (SCHWABER, SUTHERLAND, 2011).

Sprint backlog é um artefato gerado a partir do *product backlog*. A diferença entre *product* e *sprint backlog*, é que no primeiro há as descrições de "todas" as funcionalidades do sistema, enquanto no segundo há apenas as descrições das funcionalidades que serão desenvolvidas na *sprint*. Assim, o *sprint backlog* é uma lista de tarefas que contém as tarefas priorizadas pelo time *scrum*, durante a reunião de planejamento da *Sprint* (SCHWABER, SUTHERLAND, 2011).

Todos os fundamentos citados fazem parte do ciclo de vida do *scrum*, descrito na próxima seção.

3.1.1 Ciclo de Vida SCRUM

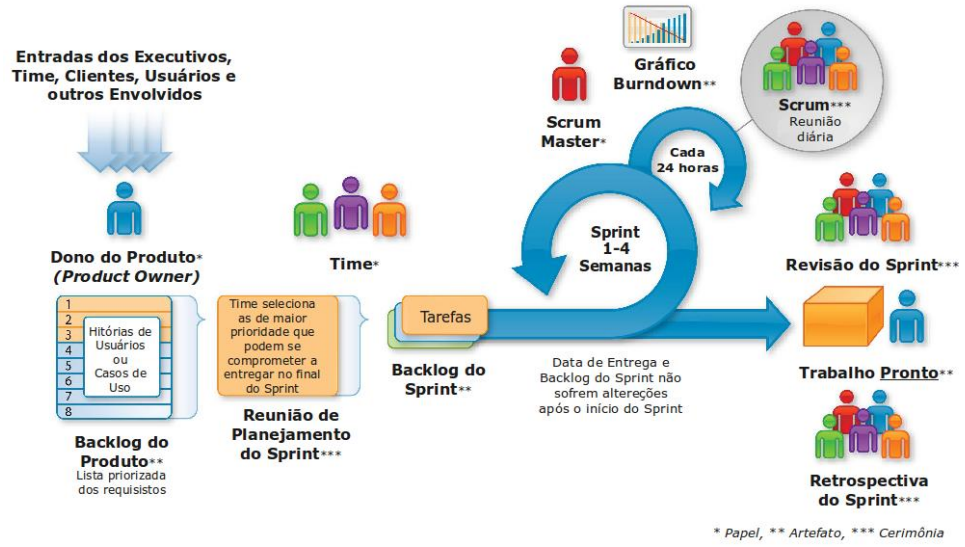
Planejamento, preparação, desenvolvimento e entrega compõe o ciclo de vida *scrum*. A fase de planejamento engloba a definição do time *scrum*, estimativas de custos, definição de datas, arquitetura de software, desenvolvimento do *product backlog* inicial, etc. É a fase que determina a visão do projeto.

Após o planejamento a próxima fase é a de preparação. Esta fase enfatiza algumas revisões e refinamentos. Por exemplo, revisar o *product backlog* e realizar mudanças caso preciso, refinar a arquitetura do sistema e realizar análise de domínio. A fase de preparação avalia vários aspectos do projeto.

A fase de desenvolvimento envolve as cerimônias *scrum*, citadas na seção anterior. É iniciado o desenvolvimento do sistema de maneira iterativa e incremental, por meio das *sprints*.

Finalizando o ciclo, a fase de entrega, como o próprio nome sugere, envolve a entrega do produto ao cliente. Na Figura 3 é apresentado o ciclo de vida do *scrum* abrangendo os principais papéis, cerimônias e artefatos.

Figura 3. Visão geral do ciclo de vida do *scrum*



Fonte: Braz (2011)

Como visto na figura 3, projetos *scrum* seguem um ciclo de vida com as tarefas realizadas, em geral, sequencialmente, porém, não necessariamente é seguida a mesma ordem, pois, muitas vezes é necessário voltar a tarefas realizadas anteriormente para dar continuidade a outras.

3.1.2 SCRUM – Engenharia de Requisitos

No Quadro 1, adaptado de Lucia e Qusef (2010), está resumido como as atividades de engenharia de requisitos são realmente implementadas no *Scrum*.

QUADRO 1. ENGENHARIA DE REQUISITOS IMPLEMENTAÇÃO EM SCRUM

Atividades de Engenharia de Requisitos	**Implementação Scrum**
Elicitação de requisitos	• *Product owner* formula o *product backlog*. • Todas as partes interessadas podem

	participar do *product backlog*.
Análise de requisitos	• Reunião de refinamento *backlog*. • *Product owner* prioriza o *product backlog*. • *Product owner* analisa a viabilidade dos requisitos.
Documentação de requisitos	• Comunicação face-a-face.
Validação de requisitos	• Reuniões de Revisão.
Gerência de requisitos	• Reunião de planejamento da *sprint*. • Os itens do *Product Backlog* para o rastreamento. • Mudanças de requisitos são adicionadas excluídos do/para o *Product Backlog.*

Fonte: Adaptada de LUCIA e QUSEF (2010)

Scrum não impõe regras quanto à execução das subáreas de engenharia de requisitos, mas por meio de suas cerimônias, artefatos e, entre outros conceitos, possibilita que essas subáreas sejam executadas. Como, por exemplo, o *product backlog*, embora não mencionado no quadro serve de base para a especificação de requisitos.

3.2 EXTREME PROGRAMMING (XP)

XP é uma das principais metodologias ágeis e será utilizada como base para esta pesquisa. Segundo Teles (2004), Extreme Programming ou XP, é um processo de desenvolvimento de software indicado para casos como:

- Projeto cujos requisitos não estão bem definidos e mudam frequentemente;
- Desenvolvimento de sistemas orientados a objeto;
- Pequenas equipes, de preferência até 12 desenvolvedores;
- Desenvolvimento incremental (ou iterativo), o sistema começa a ser desenvolvido logo no início do projeto e ao longo do tempo vão sendo adicionadas novas funcionalidades a ele.

Extreme Programming contém um conjunto de valores e práticas que moldam seu ciclo de vida. Os principais valores de XP são: *Feedback*, Comunicação, Simplicidade e Coragem.

- ***Feedback:*** Filho (2008), comenta "Quanto mais cedo impedimentos forem encontrados e removidos, menos tempo eles irão atrapalhar". O *feedback* ou retorno tem por finalidade auxiliar a equipe a identificar os erros, que são praticamente inevitáveis, o mais cedo quanto possível;

- **Comunicação:** De acordo com Ambler (2004), “A comunicação é uma via de duas mãos: ambas fornecem e obtêm informações como resultado”;
- **Simplicidade:** Teles (2004) afirma, “Também é necessário que a equipe compreenda e utilize o valor da simplicidade, que nos ensina a implementar apenas aquilo que é suficiente para atender a cada necessidade do cliente”;
- **Coragem:** Equipes XP precisam ter este valor. Coragem e ânimo diante dos possíveis problemas, mudanças e dificuldades que todo projeto pode encontrar.

3.2.1 Práticas do XP

Extreme Programming é direcionado seguindo um conjunto de práticas que são: Cliente Presente, Jogo do Planejamento, *Stand Up Meeting*, Programação em Par, Desenvolvimento Guiado por Testes, Refatoração, Código Coletivo, Código Padronizado, Design Simples, Metáfora, Ritmo Sustentável, Integração Contínua e Releases Curtos (TELES, 2004). Como muitas das práticas de XP são adotadas por outras metodologias, cada uma delas será descrita nas seguintes seções.

3.2.1.1 Cliente Presente

A prática do cliente presente é essencial para o sucesso do projeto. O cliente ao participar rotineiramente do desenvolvimento do projeto ajuda os desenvolvedores a compreenderem melhor o que já foi escrito nas *user stories*. Com esta prática erros podem ser identificados e corrigidos antecipadamente.

> Para que a troca de *feedback* possa ocorrer e o cliente possa obter o máximo de valor do projeto, é essencial que ele participe ativamente do processo de desenvolvimento. Além disso, a sua presença viabiliza a simplicidade do processo em diversos aspectos, especialmente na comunicação (TELES, 2004, p.24).

3.2.1.2 Jogo do Planejamento

De acordo com Filho (2008), o jogo do planejamento é realizado no início do projeto sendo uma reunião onde os clientes escrevem as *user stories* contendo os requisitos do sistema. Nesta reunião os cartões com maior valor agregado são priorizados pelos

programadores juntamente com os clientes. A responsabilidade dos programadores está em realizar estimativas para cada história (por exemplo, o cartão X será desenvolvido em duas semanas e custará R$ 500,00) e os clientes estabelecem o valor de negócio. No início da implementação de cada release e de cada iteração, é quando acontece o jogo do planejamento.

O XP envolve o planejamento e implementação dos releases que são lançamentos realizados geralmente de dois em dois meses. Já as iterações, normalmente tem duração de duas semanas. Os releases são subdivididos em várias iterações.

3.2.1.3 *Stand Up Meeting*

Segundo Teles (2004), *stand up meeting* ou reunião em pé é uma reunião feita todos os dias com a presença de todos os membros da equipe. O objetivo da reunião é verificar o trabalho que foi realizado no dia anterior e priorizar as tarefas a serem desenvolvidas no dia que se inicia. Deve ser uma reunião rápida e objetiva, normalmente de 10 minutos.

Stand up meeting é uma reunião similar a *daily scrum* que também é realizada diariamente em projetos que utilizam a metodologia *Scrum*.

3.2.1.4 Programação em Par

Programação em par é a prática de desenvolver código em pares. Isto é vantajoso em diversos pontos, como por exemplo, auxilia a identificação de erros antecipadamente, pois, apenas um programador não tende a ver os seus erros tão claramente como o outro que o auxilia.

> Todo o software de produção no XP é construído por dois programadores, sentados lado a lado, na mesma máquina. Esta prática assegura que todo o código de produção foi avaliado por pelo menos, outro programador, e resulta em uma melhor concepção, um melhor teste, e um código melhor(JEFFRIES, 2003, tradução nossa).

3.2.1.5 Desenvolvimento Guiado pelos Testes

Testar software após o se término ou testar apenas as suas funcionalidades durante o desenvolvimento não é uma das tarefas mais apreciadas pelos membros de uma equipe de desenvolvimento. No entanto, testar é preciso e quanto mais cedo testar, melhor.

> A equipe de programadores escreve testes de unidade para todos os componentes do sistema e os executa várias vezes ao dia para assegurar que

> funcionalidades adicionadas recentemente não tenham introduzido erros no código antigo. Os clientes escrevem testes de aceitação para assegurar que o sistema faz exatamente o que eles querem. Estes testes são executados sempre que uma nova funcionalidade é implementada para determinar se o cartão de história realmente está concluído (FILHO, 2008, p. 41).

O XP utiliza dois tipos de testes, os testes de unidade e testes de aceitação ou funcional. De acordo com Teles (2004), "os testes de unidade procuram verificar se os métodos das classes do sistema fornecem respostas corretas". Os testes unitários verificam se a funcionalidade testada corresponde ao esperado e se possuem erros ou não. Já os testes de aceitação são escritos pelo cliente, ou em alguns casos com o auxilio de um membro da equipe de desenvolvimento, e visam verificar se o sistema como um todo funciona de maneira correta e esperada pelos *stakeholders*.

3.2.1.6 Refatoração

Beck (2000) afirma que depois de implementada uma função do sistema os programadores ainda almejam em como poderiam alterar o sistema, para que adicionar novas funcionalidades seja algo simples de se realizar. Um dos valores de XP é a simplicidade, portanto os programadores objetivam simplificar o sistema mesmo após novas funcionalidades terem sido adicionadas. Isto é possível por meio da técnica de refatoração.

> O *refactoring* é o ato de alterar um código sem afetar a funcionalidade que ele implementa. É utilizado para tornar o software mais simples de ser manipulado e se utiliza fortemente os testes descritos anteriormente para garantir que as modificações não interrompam seu funcionamento (TELES, 2004, p. 25).

3.2.1.7 Código Coletivo

Segundo Beck (2000) se um membro da equipe verificar a necessidade de melhorar ou adicionar valor ao código, não somente pode, mas deve fazer. Todos os membros da equipe conhecem as funcionalidades do sistema, mesmo que não seja um conhecimento tão abrangente para todos, e são responsáveis pelo sistema inteiro.

De acordo com Teles (2004), a prática de código coletivo auxilia juntamente com a programação em par a evitar a formação de "ilhas de conhecimento", que geralmente é uma pessoa que possui conhecimento sobre o domínio ou área de negócio e no desenvolvimento somente ela tem conhecimento do código ou parte dele.

3.2.1.8 Código Padronizado

Teles (2004) afirma, “o XP recomenda que a equipe adote um padrão no início do projeto. Ela pode construir este padrão em conjunto ou pode adotar um que já exista e esteja documentado em algum lugar”. Esta prática complementa a programação em par e código coletivo, pois, se não existisse um padrão cada parte do código final estaria no estilo de um programador.

3.2.1.9 Design Simples

A simplicidade (um dos valores de XP) no design deve ser buscada e mantida. Em XP alcançar simplicidade é melhor que alcançar códigos complexos e que contenham funcionalidades a mais, o objetivo é desenvolver de maneira simples as funcionalidades exigidas pelos *stakeholders*.

Segundo Filho (2008) simplicidade é um ponto essencial para que o sistema adequa-se a modificações. Fazer o mais simples quanto possível de funcionar.

3.2.1.10 Metáfora

De acordo com Teles (2005) a equipe XP convive com o desafio de manter a integridade conceitual, ou seja, conservar as partes centrais do sistema trabalhando conjuntamente. Isto é um desafio, pois, geralmente, vários projetistas trabalham na arquitetura do sistema ao longo do desenvolvimento. Para ser alcançada a integridade conceitual e outros objetivos, é necessário um mecanismo que seja capaz de alinhar os pensamentos dos vários projetistas para que todos tenham uma visão única na maneira de adicionar, manter, alterar as funcionalidades do sistema. Para isto XP usa o conceito de metáfora.

Equipes de Extreme Programming desenvolvem uma visão comum de como o programa funciona, o que chamamos de "metáfora" (JEFFRIES, 2013).

3.2.1.11 Ritmo Sustentável

De acordo com Filho (2008) manter o ritmo sustentável significa evitar as horas-extras, ou seja, se o período de trabalho é de oito horas por dia, somente em situações muito específicas e realmente indispensáveis deve se recorrer a horas extras. É importante que o

ritmo de trabalho não afete a vida pessoal e saúde dos membros da equipe, pois, se houver um sobrecarga de serviço consequentemente também influenciaria na qualidade do código.

Ritmo sustentável visa manter o ritmo adequado de trabalho de uma equipe XP para que os colaboradores não se sobrecarreguem e, assim, trabalhem da melhor maneira possível para alcançarem um projeto de sucesso.

3.2.1.12 Integração Contínua

Como citado na prática de refatoração integrar novas funcionalidades ao sistema de maneira simples e sem alterar as outras funcionalidades ainda é um ponto almejado para os programadores.

Seria impossível integrar depois de apenas algumas horas de trabalho. Integração demora muito tempo e há muitos conflitos e chances de se estragar algo acidentalmente. A não ser que (BECK, 2000, p.77):

- Você possa executar testes rapidamente para certificar-se de que não estragou nada;
- Você programe em pares, diminuindo pela metade as quantidades de modificações que precisam ser integradas;
- Você refatore para que haja mais pedaços pequenos, reduzindo a probabilidade de conflito.

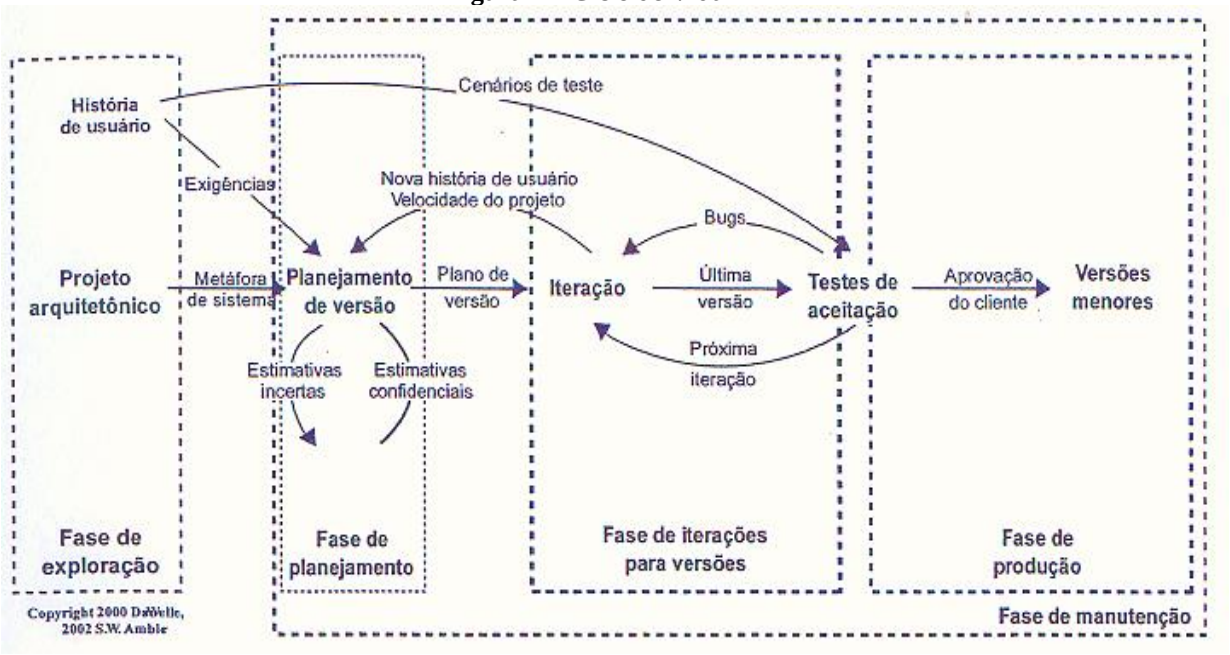
3.2.1.13 Releases Curtos

O XP considera que “um projeto de software é um investimento. O cliente investe uma certa quantidade de recursos na expectativa de obter um retorno dentro de certo prazo (TELES, 2004, p.185). Releases curtos são lançamentos de pequenas versões do sistema, normalmente, entre dois e três meses. É uma das características principais de XP que é vantajosa tanto para o cliente quanto para os desenvolvedores que podem verificar como está o desenvolvimento do sistema, e os clientes oferecer feedback.

3.2.2 Ciclo de Vida XP

Na Figura 4 está representado o ciclo de vida de um projeto XP.

Figura 4 – Ciclo de Vida XP



Fonte: Henrajani (2007)

Exploração é a primeira fase do ciclo de vida XP. Beck (2000) afirma sobre essa fase, "a fase do desenvolvimento em que o cliente informa o que o sistema todo poderia fazer de forma geral". Nessa fase são identificadas as primeiras histórias de usuários que de acordo com Ambler (2004), "as histórias de usuário são um guia básico da metodologia XP – elas fornecem requisitos de alto nível para o sistema e são subsídios cruciais para o processo de planejamento". Embora não exposto na figura estão incluídos nessa fase o conhecimento sobre o domínio do negócio, escopo do projeto, o primeiro contato do cliente com a equipe e a identificação das melhores tecnologias que poderão ser utilizadas no projeto. Busca-se definir também a base arquitetural do sistema.

Segundo Beck (2000), a fase de planejamento (fase posterior a de exploração) tem por objetivo que clientes e programadores decidam a data de entrega das primeiras histórias de usuários implementadas, que normalmente, são as de maior valor para o negócio. Nessa fase são utilizados os cartões de histórias para auxiliar os programadores a fazerem as estimativas. Geralmente, as funcionalidades de uma história de usuário são separadas em vários cartões. De acordo Filho (2008) nessa fase é realizado o planejamento das versões do produto escolhendo os cartões de acordo com funcionalidades que preferencialmente devem ser oferecidos juntos para o cliente.

A próxima fase é a de Iterações para Versões. Segundo Teles (2004), as iterações são subdivisões de um *release* com um determinado período de tempo para a implementação de um grupo de histórias. Na Figura 5 é mostrado ciclo de vida de uma iteração XP.

De acordo com Ambler (2004), o planejamento de iterações é semelhante ao de planejamento de versões a principal diferença está no foco que são as estórias de usuário escolhidas para a iteração corrente. Como mostrado na figura 5 nessa fase são realizados os testes de aceitação, plano de lançamento de história de usuários, plano de iteração e, entre outros, são identificadas novas estórias de usuários.

Beck (2000) afirma sobre a fase de produção, "tipicamente haverá algum processo para se certificar de que o software está pronto para entrar em produção. Esteja preparado para implementar novos testes para provar que ele está em boa forma."

E a fase de manutenção, como mostrado na figura 4, envolve as fases de planejamento, iterações para versões e produção. Abrange ações e práticas para conservar e manter o sistema de acordo com as funções exigidas.

3.2.3 Extreme Programming – Engenharia de Requisitos

No quadro 2 é mostrado como são realizadas em XP as atividades de engenharia de requisitos.

QUADRO 2. ENGENHARIA DE REQUISITOS IMPLEMENTAÇÃO EM XP

Engenharia de Requisitos	**Implementação XP**
Elicitação de Requisitos	• Requisitos elicitados como histórias. • Clientes escrevem histórias de usuários.
Análise de Requisitos	• Não é uma fase separada. • Análise durante o desenvolvimento. • Cliente prioriza as histórias de usuários.
Documentação de Requisitos	• Histórias de usuários e testes de aceitação como documentos de requisitos. • Produtos de software como informação de persistência.
Validação de Requisitos	• *Test Driven Development* (TDD). • Executar testes de aceitação. • *Feedback* frequente.
Gerência de Requisitos	• Planejamento de curtas iterações. • Histórias de usuários para rastreamento. • Refatore, conforme necessário

Fonte: LUCIA e QUSEF (2010)

Em XP a elicitação de requisitos é feita por meio das histórias de usuários. Segundo Pressman (2006), "cada história é escrita pelo cliente e é colocada em um cartão de indexação. O cliente atribui um valor (isto é, uma prioridade) para a história, com base no valor de negócio global da característica ou da função".

A fase de análise de requisitos, não é uma fase separada, isto é, pode acontecer em paralelo a outras. Assim como em *Scrum*, XP utiliza a técnica de priorização de requisitos. XP faz uso de cartões CRC (*Class Responsability Collaborator* – Classe Responsabilidade Colaborador) que são cartões que contem o nome da classe (por exemplo, Produto), a responsabilidade da mesma (por exemplo, conhecer a categoria do produto) e colaborador (classes que se comunicam entre si, por exemplo, a classe Fornecedor se comunicará com a classe Produto). No Quadro 3 são mostrados alguns pontos importantes de cartões CRC e Histórias de Usuários.

QUADRO 3. TÉCNICAS DE MODELAGEM

Artefato	**Descrição**	**Aplicações Comuns**	**Itere em**	**Mídia Sugerida**
Cartões CRC	São usados para explorar a estrutura, talvez para a modelagem conceitual, para entender o domínio do problema ou para o desenho, para trabalhar sobre a estrutura do software (Ambler, 2004).	Modelagem de domínio com clientes. Modelagem conceitual com clientes. Exploração da estrutura de software orientado a objetos.	• Conjunto de testes de aceitação. • Regra de negócio. • Caso de mudança. • Restrição. • Diagrama de classes. • Caso de uso essencial. • Glossário. • Diagrama da organização. • Código-fonte. • Caso de uso do sistema. • Cenário de uso. • Diagrama de caso de uso. • História de usuário.	1. Cartões de índice
Histórias de Usuários	As histórias de usuários fornecem uma visão de alto nível dos requisitos de um sistema (Ambler, 2004).	Exploração de requisitos. Lembrar de ter uma conversa com um cliente.	• Conjunto de testes de aceitação. • Diagrama de colaboração. • Diagrama de implantação. • Glossário. • Diagrama de robustez. • Diagrama de sequência. • Código-fonte.	1. Cartão de índice. 2. Processador de textos.

Fonte: Adaptado de Ambler (2002)

De acordo com Ambler (2004) um dos princípios básicos da Modelagem Ágil é "Modele com um propósito". Muitos desenvolvedores se preocupam se os artefatos gerados no desenvolvimento (como modelos, código-fonte ou documentos) estão bem detalhados, se contém informações suficientes ou se não estão muito complexos. Modelar com um propósito vai além desses fatores, o ponto chave está no porque criar certo artefato e para quem se esta criando.

Em XP os artefatos mais utilizados na elicitação e análise de requisitos, geralmente, são as histórias de usuários e cartões CRC. Isso não quer dizer que não são utilizados outros tipos de artefatos como diagramas de classe, casos de uso e diagramas de sequência. É necessário que os desenvolvedores modelem com um propósito e também em casos que os próprios clientes exijam algum artefato ou documentação específica. Para a validação de requisitos são utilizados os testes de aceitação, geralmente, escritos juntamente com as histórias de usuários e, os testes de unidade no qual é testado cada unidade do sistema separadamente.

3.3 DYNAMIC SYSTEM DEVELOPMENT METHOD – DSDM

Para Tomás (2009) a DSDM objetiva por meio de prototipagem incremental, construir e conservar sistemas que satisfaçam as exigências de prazo. Segue o princípio de Pareto em que 80% dos resultados de uma aplicação, no caso de desenvolvimento de software, estão diretamente relacionados a 20% de tempo que demoraria em entregar o sistema completo.

A DSDM assim como XP trabalha com um conjunto de princípios que segundo Filho (2008), são:

1. O envolvimento ativo dos usuários é imperativo;
2. A equipe precisa ter poder para tomar decisões;
3. Foco em entregas frequentes;
4. Alinhamento com os interesses de negócio;
5. Desenvolvimento iterativo e incremental para chegar a soluções apuradas;
6. Todas as mudanças de desenvolvimento devem ser reversíveis;
7. Requisitos são definidos em alto nível;
8. Testes são integrados durante o desenvolvimento.

A maior parte desses princípios é semelhante a princípios, práticas e valores de outras abordagens ágeis, como por exemplo, o princípio de envolvimento ativo dos usuários, no qual em XP é uma prática conhecida como cliente presente.

3.3.1 Ciclo de Vida – DSDM

Para Stapleton (2003) o ciclo de vida de DSDM possui sete fases. São elas:

1. Pré-Projeto;
2. Estudo de Viabilidade;
3. Estudo de Negócios;
4. Iteração do Modelo Funcional;
5. Iteração de Construção e Design do Sistema;
6. Implementação;
7. Pós-Projeto.

A fase de pré-projeto assegura que apenas os projetos adequados são iniciados e que eles estão configurados corretamente. Uma vez que tenha sido determinado que um projeto possa ir em frente, que o financiamento esteja disponível, etc, o planejamento do projeto inicial para o estudo de viabilidade é feito. Em seguida o projeto propriamente dito começa com o estudo de viabilidade (Manual DSDM, 2010, tradução nossa).

O estudo de viabilidade para Pressman (2006), "estabelece os requisitos básicos e restrições do negócio associados á aplicação em construção e depois avalia se a aplicação é uma candidata viável ao processo DSDM".

O estudo de negócios é realizado após o estudo de viabilidade. Nessa fase são identificados requisitos funcionais e não funcionais como também é realizada uma exploração sobre as propriedades do projeto. Para Pressman (2006), nessa fase também é determinada a arquitetura básica da aplicação e requisitos de manutenção são identificados. Bonow (2008) acrescenta que nessa fase são realizados *workshops* com todos os membros da organização que comporão a equipe de desenvolvimento do projeto, estimulando a comunicação entre eles e o conhecimento do projeto.

A fase de Iteração do modelo funcional é o início da criação de protótipos e consequentemente do desenvolvimento da aplicação.

> Produz um conjunto de protótipos incrementais que demonstram a funcionalidade para o cliente (note que todos os protótipos DSDM são destinados a evoluir para a aplicação a ser entregue). O intuito durante esse ciclo iterativo é obter requisitos adicionais pela provocação de feedback dos usuários á medida que eles exercitam o protótipo (PRESSMAN, 2006, p.68).

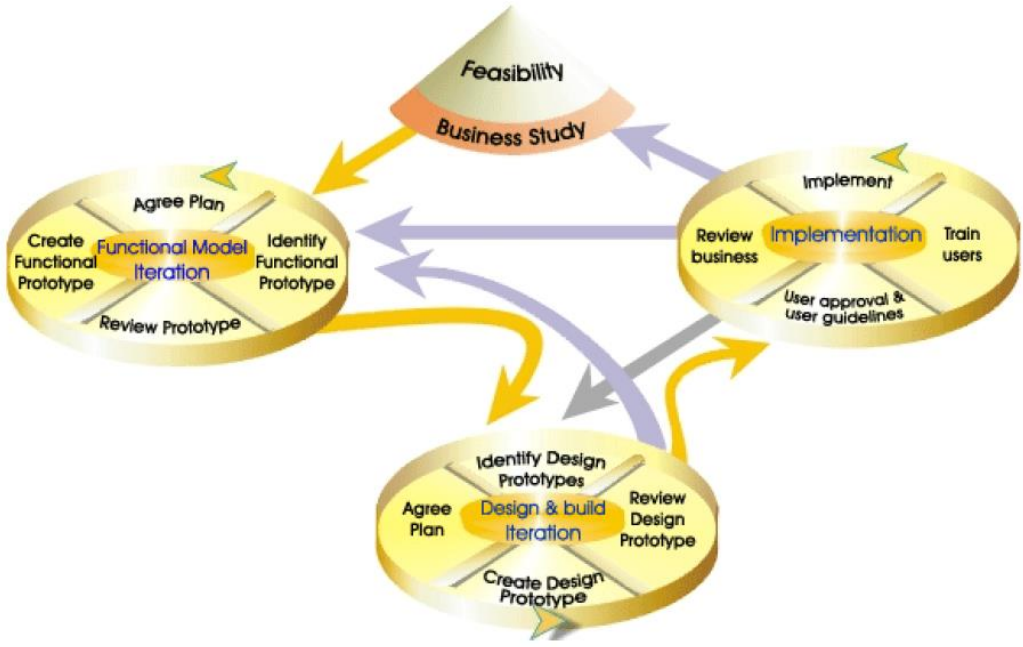
A fase de iteração de construção e *design* do sistema é a fase no qual o sistema é projetado para um nível de qualidade suficiente para ser colocado em segurança nas mãos dos usuários. O produto principal é o sistema testado (Manual DSDM, 2010, tradução nossa).

Assim, a tarefa de testar software é uma das mais importantes dessa fase, pois antes de entregar o protótipo para o cliente é necessário verificar se existem erros no sistema e analisar como está a sua qualidade.

Na fase de implementação a versão final do *software* é entregue ao cliente, bem como, há treinamento para os utilizadores. Quando preciso, após o termino das iterações de implementação pode-se retornar a qualquer uma das outras fases, até mesmo a de estudo de viabilidade (FILHO, 2008).

A última fase, Pós-Projeto tem o foco na manutenção do sistema. A natureza iterativa e incremental de DSDM significa que manutenção pode ser visto como desenvolvimento contínuo. A manutenção é uma parte esperada do ciclo de vida esperado de um sistema (DSDM Public Version 4.2 Manual, 2010, tradução nossa). Na figura 5 é mostrado o ciclo de vida DSDM.

Figura 5. Ciclo de vida DSDM



Fonte: DSDM Public Version 4.2 Manual (2010)

3.3.2 DSDM – ENGENHARIA DE REQUISITOS

Durante as fases de estudo de viabilidade e de negócios, os requisitos básicos são extraídos. Outros requisitos são extraídos durante o processo de desenvolvimento. DSDM não insiste em certas técnicas. Portanto, qualquer técnica de engenharia de requisitos pode ser

usada durante o processo de desenvolvimento (PAETSCH, EBERLEIN e MAURER, 2003, tradução nossa).

Embora não exista um padrão das técnicas a serem utilizadas em DSDM, normalmente, são usadas as técnicas de *workshops*, prototipagem, sessões JAD e MoSCOw (STAPLETON, 1995, tradução nossa). Com o uso dessas técnicas é oferecido suporte para as subáreas da engenharia de requisitos como, por exemplo, o uso de protótipos é muito eficiente tanto para a elicitação quanto validação de requisitos. Os *workshops* e sessões JAD colaboram para a elicitação de novos requisitos e também para a análise. E a técnica de MoSCow é utiliza para priorização dos requisitos.

DSDM gera vários artefatos durante todo o processo de desenvolvimento de *software*. Entre os variados artefatos há aqueles que abrangem os requisitos. Entre eles há a lista de requisitos priorizada, lista de requisitos não funcionais, modelo funcional (incluindo o protótipo funcional), *design* do protótipo e registro de revisão do *design* do protótipo. Cada um desses artefatos abrange ao menos uma subárea da engenharia de requisitos. Assim, percebe-se que DSDM oferece suporte a essa área da engenharia de *software*.

3.4 Feature Driven Development (FDD)

Feature Driven Development (Desenvolvimento Guiado por Funcionalidades) é uma metodologia que enfatiza um processo de curta duração orientado a modelos. É um processo que auxilia a equipe de desenvolvimento a atingir frequentemente, resultados de trabalho palpáveis. São utilizados blocos bem pequenos de funcionalidades que tenham valor para o cliente, denominados de funcionalidades. FDD organiza esses pequenos blocos em grupos de funcionalidades relacionadas ao negócio. Objetiva que sejam alcançados resultados de trabalho a cada duas semanas. Também inclui estratégias de planejamento (COAD, LEFEBVRE e DE LUCA, 1999, tradução nossa).

FDD é baseado em cinco processos, sendo eles, Desenvolver um Modelo Abrangente, Construir a Lista de Funcionalidades, Planejar por Funcionalidade, Detalhar por funcionalidade e Construir por Funcionalidade.

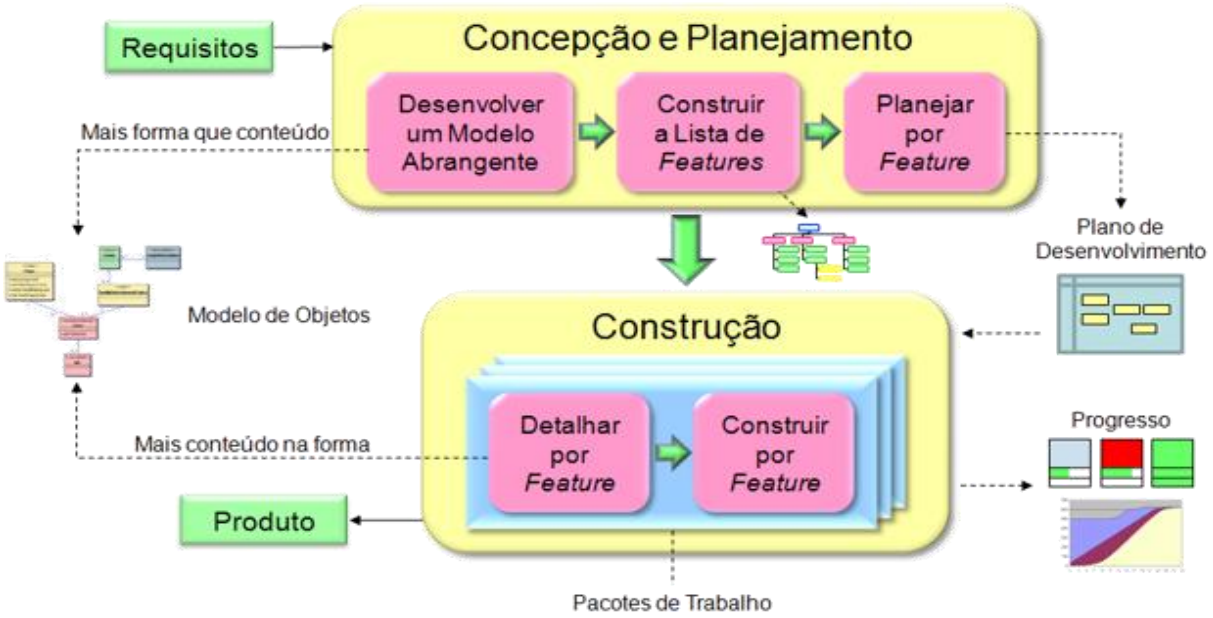
O primeiro processo, desenvolver um modelo abrangente, consiste em reunir os membros do domínio do negócio e desenvolvedores para que trabalhem juntos e seja desenvolvido um passo a passo de alto nível do sistema. Então os membros do domínio do negócio presentes, especificam mais detalhes para cada área do domínio do problema (ROSENBERG, STEPHENS e COLLINS-COPES, 2005, tradução nossa).

O segundo processo, construir a lista de funcionalidades, enfatiza a construção dessa lista por meio do conhecimento adquirido durante a tarefa inicial de modelagem. Uma funcionalidade é definida como uma pequena função que tem valor para o cliente. Geralmente, é expressa na forma de ação, resultado e objeto como, por exemplo, calcular o total da venda. Como entradas são utilizados os documentos de requisitos já existentes. Dentro de cada área do domínio as funcionalidades são reunidas em um grupo de funcionalidades que refletem uma atividade particular do negócio (ROSENBERG, STEPHENS e COLLINS-COPES, 2005, tradução nossa).

No terceiro processo, planejar por funcionalidade, os grupos de funcionalidades são sequenciados, isto é, haverá uma sequência para a implementação das funcionalidades em plano de alto nível atribuído ao líder de programadores. Ainda nesse processo são atribuídas a cada desenvolvedor as classes pelas quais devem ser responsáveis (ROSENBERG, STEPHENS e COLLINS-COPES, 2005, tradução nossa).

No quarto e quinto processos, detalhar e construir por funcionalidade, esses dois processos são bastante iterativos. O líder da equipe seleciona os grupos de funcionalidades a serem desenvolvidos na próxima iteração. São identificadas as classes e os desenvolvedores dessas classes, que provavelmente serão desenvolvidas na próxima iteração. A equipe cria detalhados diagramas de sequência para as funcionalidades e, escrevem as estruturas das classes e métodos. Posteriormente, a equipe conduz uma inspeção do projeto, isto é, verificar como está o andamento do projeto, detectar erros e resultados obtidos até o momento. Se a inspeção obtiver resultados satisfatórios, então os responsáveis pelas classes as prepararão para serem testadas, por meio de testes de unidade, inspeção e, entre outros. Após essas atividades, se forem obtidos os resultados esperados, as funcionalidades completadas são promovidas a versão atual (build) e, esses processos tornam a ser repetidos para os próximos grupos de funcionalidades (ROSENBERG, STEPHENS e COLLINS-COPES, 2005, tradução nossa). Na figura 6 é mostrado o ciclo de vida da FDD.

Figura 6. Ciclo de vida FDD



Disponível em: http://www.heptagon.com.br/fdd-estrutura
Acesso em: 15 nov. 2013

3.4.1 FDD – Engenharia de Requisitos

Na metodologia FDD não é muito evidenciado as subáreas da engenharia de requisitos, porém, é oferecido suporte para essas subáreas. Alguns requisitos devem ser elicitados gerando, por exemplo, um documento de requisitos para então ser iniciado os processos de FDD. De acordo com Silva, Hoentsch e Silva (2009), a lista de funcionalidades, quando gerada no segundo processo de FDD, é organizada hierarquicamente com requisitos funcionais. No processo de construção dessa lista, são levantadas todas as funcionalidades que sejam necessárias para cumprir as exigências dos *stakeholders.* No processo seguinte, os itens da lista são priorizados e identifica-se quais funcionalidades serão preferencialmente implementadas. Além de, as funcionalidades poderem ser classificadas quanto a funcionais ou não funcionais.

Vários artefatos são gerados em FDD que auxiliam a execução das subáreas da engenharia de requisitos. Por exemplo, os diagramas de sequência, diagramas de classes, testes de unidade e, entre outros citados anteriormente. Logo, mesmo que não sejam expressos com o conceito de engenharia de requisitos, FDD oferece suporte para essa área.

3.5 Crystal Family of Methodologies

The Crystal Family of Methodologies (Família de Metodologias Crystal) é um conjunto de diferentes metodologias que facilitam a escolha da metodologia mais adequada a cada projeto. Além disso, a abordagem de Crystal também inclui princípios para adaptação das metodologias com o objetivo de atender às diferentes circunstâncias de diferentes projetos (ABRAHAMSSON et al., 2002, tradução nossa).

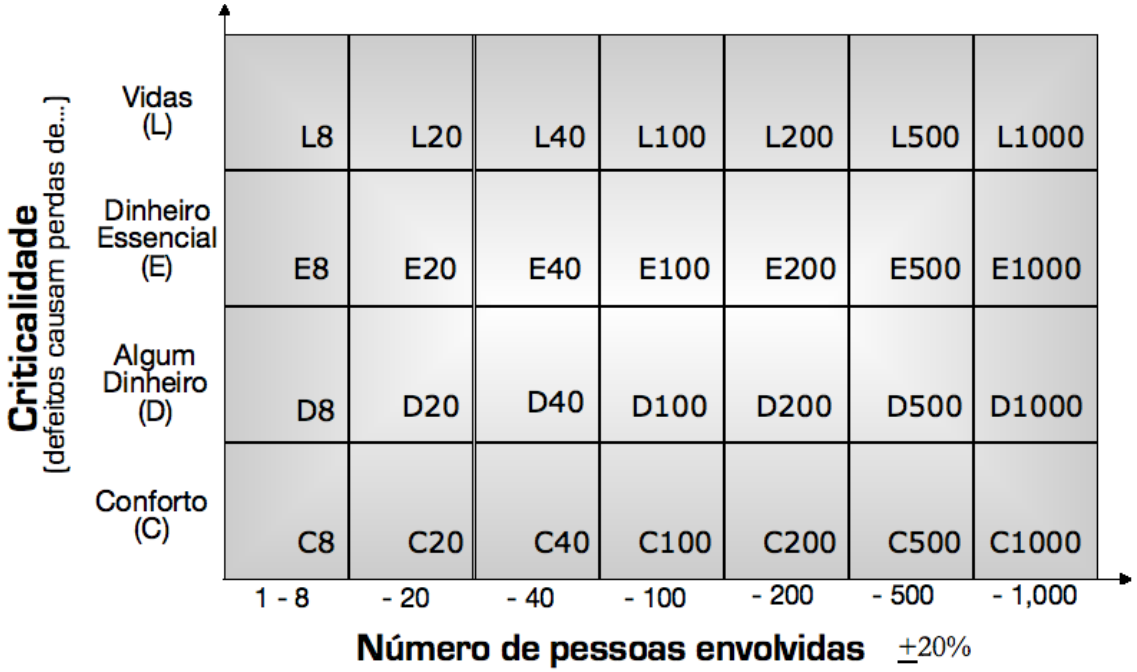
Alistair Cockburn é o autor de *Crystal*. Alistair e Crystal focalizam nos aspectos de colaboração e cooperação com as pessoas durante o desenvolvimento do projeto (HIGHSMITH, 2002). Cada metodologia é caracterizada como uma cor e dureza assim como os cristais, correspondendo ao tamanho do projeto e criticidade. São a *Crystal Clear, Orange, Orange Web, Red, Magenta, Blue* e, assim por diante. *Crystal Clear, Orange e Orange Web* são as únicas que realmente foram utilizadas em projetos reais. *Crystal Orange Web* difere da *Crystal Orange* porque ela não lida com apenas um projeto, mas com um fluxo consecutivo de iniciativas que demandam programação e, com os resultados de cada iniciativa que está sendo mesclado com o código base crescente que está sendo utilizado pelo público. (COCKBURN, 2006, tradução nossa). Segue a descrição das metodologias *Crystal Clear e Crystal Orange*.

A metodologia *Crystal Clear* é indicada para projetos pequenos que tenham no máximo 6 pessoas na equipe e seu risco seja classificado como D (baixo custo). Os incrementos produzidos nessa metodologia tem duração de, aproximadamente, três meses (JÚNIOR, 2008).

A metodologia *Crystal Orange* é indicada para projetos que tenham entre dez e quarenta pessoas na equipe. Geralmente, são subdivididas em mais de uma equipe as quais são dispostas de acordo com as responsabilidades determinadas. Uma característica importante é que essa metodologia não abrange risco de vida. Os incrementos produzidos duram entre dois e quatro meses (JÚNIOR, 2008).

Na figura 7 são mostrados os aspectos que auxiliam na escolha da metodologia mais apropriada para cada projeto, levando em conta a criticidade e número de pessoas envolvidas.

Figura 7. Criticidade e número de pessoas envolvidas metodologias Crystal



Fonte: Filho (2008)

As metodologias *Crystal* têm várias características em comum como, a centralização nas pessoas e comunicação e serem ajustáveis para adequação a determinadas configurações (COCKBURN, 2006, tradução nossa). Para Cockburn (2006), há duas técnicas básicas utilizadas em *Crystal*:

- A técnica de sintonização das metodologias: utilizando projetos de entrevistas e uma equipe de *workshop* para converter uma metodologia base para uma metodologia de partida do projeto;
- A técnica usada para manter o *workshop* de reflexão.

No quadro 4 é mostrado algumas práticas que são comuns entre *Crystal Clear* e *Crystal Orange.*

QUADRO 4. Práticas de Crystal Clear e Crystal Orange

Prática	Descrição
Preparação	Planejamento das iterações, ou seja, um cronograma para produzir uma versão utilizável. A equipe seleciona as funcionalidades a serem implementados na iteração e, sobre elas, montam o cronograma.
Monitoramento	O progresso do projeto é monitorado. Verifica-se o que a equipe foi capaz de realizar. O monitoramento é feito em pontos de medição (inicio, revisão1, revisão 2, teste, entrega).
Revisão e análise	Cada incremento ao projeto constitui um conjunto de iterações sendo que, nessas iterações, são realizadas as atividades de implementação, demonstração e revisão dos objetivos do incremento.

Fluxo e simultaneidade	No uso de várias equipes (Crystal Orange), elas podem proceder com máximo de paralelismo, permitindo o trabalho de várias equipes ao mesmo tempo.
Holistic Diversity Strategy	Consiste em elaborar estratégias para que a equipe seja multifuncional.
Refinamento de Metodologia	Uso de entrevistas e workshops para compreender e instanciar uma metodologia do crystal. A ideia central é que, a cada incremento, seja possível fixar e melhorar o processo de desenvolvimento.
Visões do usuário	Consiste em ter o usuário participando do desenvolvimento, analisando e validando cada entrega realizada.
***Workshops* de reflexão**	São realizados antes e depois do incremento

Fonte: Adaptado de Cockburn (2006) apud Júnior (2008)

3.5.1 Crystal Family of Methodologies – Engenharia de requisitos

As metodologias da família *Crystal*, assim como na FDD, não expõem tão claramente como são realizados os processos da engenharia de requisitos. Mas por meio de seus princípios, práticas, técnicas e estratégias pode-se identificar alguns aspectos dessa área.

São gerados vários artefatos tanto em *Crystal Clear* como *Crystal Orange*. Os chamados *work products (*produtos de trabalho) é uma das práticas dessas metodologias. Alguns dos artefatos gerados são: modelo de objetos, código-fonte, casos de teste, documento de requisitos, casos de uso e descrição das funcionalidades e, entre outros (COCKBURN, 2002, apud JÚNIOR, 2008). Muitos desses artefatos auxiliam nas subáreas de requisitos, como o documento de requisitos, que pode ser utilizado para elicitação, análise e especificação de requisitos e os casos de teste que são utilizados na validação de requisitos.

Além desses artefatos, existem práticas que também auxiliam nas subáreas de requisitos como, por exemplo, os *workshops* e entrevistas que possibilitam a elicitação e análise de requisitos.

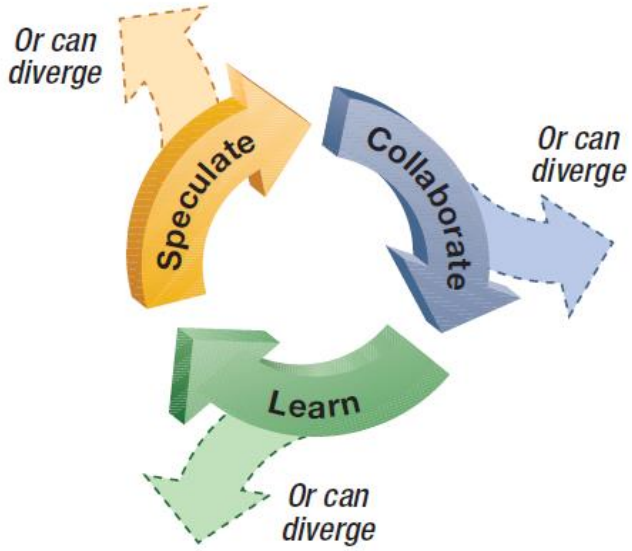
3.6 Adaptive Software Development (ASD)

A metodologia ASD atinge vários públicos. Entre eles, é destinada para equipes de projetos que têm lutado com projetos que enfrentam muitas mudanças e de alta velocidade e, estão procurando por caminhos para melhorar o desempenho e moderar o desgaste. Especialmente, pelos projetos que tendem a ficarem maiores e as equipes se tornarem distribuídas. Também para equipes de projetos que têm sido atribuídas a um projeto de alta velocidade e mudança, para suportar uma nova iniciativa de negócios críticos (HIGHSMITH, 2000, tradução nossa). As práticas de ASD são dirigidas por uma crença na contínua

adaptação a diferentes filosofias e um diferente ciclo de vida, uma abordagem para aceitar a mudança contínua como uma norma (HIGHSMITH, 2000, tradução nossa).

Em ASD, a estática *Plan-Design-Build* do ciclo de vida é substituída por um ciclo de vida dinâmico *Speculate-Collaborate-Learn* (Especulação, Colaboração e Aprendizado). Este é um ciclo de vida dedicado a um contínuo aprendizado e orientado para a mudança constante, re-avaliação, olhando para um futuro incerto e intensa colaboração entre os desenvolvedores, testadores e clientes (HIGHSMITH, 2000, tradução nossa). Na figura 8 é mostrado o ciclo de vida ASD.

Figura 8. Ciclo de vida ASD



Fonte: Highsmith (2000)

As fases do ciclo de vida ASD são designadas em um caminho para enfatizar o papel das alterações no processo. "Especulação" é usado em vez de "planejamento", como um "plano" é, geralmente, visto como algo no qual a incerteza é um ponto fraco e do qual os desvios indicam falhas. Da mesma forma, "Colaboração", destaca a importância do trabalho em equipe como meio de desenvolvimento de sistemas de alta mudança. "Aprendizado" salienta a necessidade de reconhecer e reagir a erros, e o fato de que os requisitos podem muito bem mudar durante o desenvolvimento (ABRAHAMSSON et al., 2002, tradução nossa).

Segundo Highsmith (2000) o ciclo de vida ASD possui seis características básicas. São elas:

- **Foco na Missão:** É definido o objetivo do projeto. As declarações de missão atuam como guias que encorajam a exploração no início, mas diminuem ao longo de um projeto. Uma missão fornece limites, mais que um destino fixo. Sem uma boa missão e um processo de refinamento constante na missão, os ciclos de vida iterativos tornam-se ciclos de vida-oscilantes, sem nenhum progresso;
- **Base nos Componentes:** ASD concentra-se em resultados e não tarefas e os resultados são identificados como componentes de aplicação. Componentes, neste contexto, define um conjunto de características (ou entregas) que estão a ser desenvolvidos durante um processo de ciclo iterativo;
- **Iteratividade:** Ciclos iterativos enfatizam o "refazer" tanto quanto "fazer". Refatoração e adaptação ás mudanças devem ser consideradas uma ótima sugestão para ambientes em que as mudanças são constantes;
- **Intervalo de Tempo Definido (time-boxed):** Ambiguidade em projetos de software complexos podem ser aliviadas através da fixação de prazos concretos em uma base regular. Essa técnica força os participantes do projeto a tomar difíceis e inevitáveis decisões no início do projeto;
- **Orientação a Riscos:** Planos de ciclos adaptativos são conduzidos através da análise da crítica riscos;
- **Tolerância a Mudanças:** Desenvolvimento adaptativo também é tolerante a mudanças, vendo a capacidade de incorporar a mudança como uma vantagem competitiva, não algo a ser sumariamente visto como um "problema".

3.6.1 Adaptive Software Development – Engenharia de Requisitos

ASD realiza sessões JAD (*Joint Application Development*) para apoiar a constante presença dos *stakeholders*. A sessão JAD é, essencialmente, um *workshop*, onde os desenvolvedores e representantes dos clientes se reúnem para discutir as características do produto desejado, e para melhorar a comunicação (ABRAHAMSSON et al., 2002, tradução nossa).

Diversos artefatos são gerados em ASD, entre os quais *The Product Specification Outline* auxilia na elicitação de requisitos. Segundo Highsmith (2000) *the product specification outline* contém as características, funções, objetos, dados de desempenho, operações e outras especificações relevantes do produto em um alto nível.

ASD não centraliza muito a área de requisitos, bem como a maior parte das metodologias ágeis, mas oferece algumas práticas, artefatos e técnicas que auxiliam na elicitação, análise, especificação e validação de requisitos.

4 DESENVOLVIMENTO

Cada artefato levantado foi classificado de acordo com as subáreas de engenharia de requisitos. Para esta classificação foi utilizado como base *The Guide to the Software Engineering Body of Knowledge* – SWEBOK (IEEE, 2004). Neste guia as subáreas de requisitos estão divididas em elicitação, análise, especificação e validação de requisitos.

4.1 ELICITAÇÃO DE REQUISITOS

A elicitação de requisitos é a fase que objetiva identificar de onde vêm os requisitos de *software* e como estes podem ser levantados pelos engenheiros. É o momento em que se inicia o processo de entendimento dos problemas que o *software*, a ser desenvolvido, precisará resolver. Os *stakeholders* são identificados e as relações entre a equipe e o cliente são instituídas (IEEE, 2004). Essa fase envolve a identificação das fontes de requisitos e técnicas de elicitação.

Existem várias fontes de requisitos, sendo imprescindível que todas sejam identificadas e seja avaliado o seu impacto sobre o *software*. Para ajudar a identificar as inúmeras fontes de requisitos e estrutura de gestão das mesms, são expostos os principais pontos sobre esse assunto (IEEE, 2004).

O primeiro ponto abordado são os objetivos. Estes devem ser vistos como motivadores para o *software*, visto que objetivos a longo prazo são, normalmente, considerados objetivos de alto nível e por isso devem ter atenção especial do engenheiro. O segundo ponto se refere ao conhecimento do domínio, pois, é essencial que o engenheiro de *software* tenha conhecimento sobre o domínio da aplicação. O terceiro ponto envolve os *stakeholders*, logo que, de fato vários *softwares* são considerados insatisfatórios por não atenderem as exigências de todas as partes interessadas. É necessário que o engenheiro de *software* reconheça, represente e administre os "pontos de vistas" de muitos *stakeholders.* O ambiente operacional (quarto ponto) precisa ser levado em consideração, pois, os requisitos provêm a partir do ambiente no qual o *software* será utilizado. E o quinto ponto, ambiente organizacional, mostra que o engenheiro de *software* precisa levar em consideração aspectos organizacionais como rotina e política interna da organização, pois, na maioria dos casos o *software* a ser desenvolvido será utilizado para oferecer suporte aos processos de negócios da empresa e,

portanto, deve estar de acordo com o ambiente para que não haja transtornos nos processos de negócios (IEEE, 2004).

Após a identificação das fontes de requisitos pode-se então começar o processo de elicitação por meio dessas fontes. Esta é considerada uma fase difícil, pois, aqueles que fornecem os requisitos, por exemplo, podem deixar de comunicar informações importantes, ou terem dificuldade em expressar como são realizadas suas tarefas, dificuldade em cooperar e mesmo em se expressar. O engenheiro de *software* deve estar atendo a questões como essas porque a elicitação não é uma tarefa passiva, sendo necessário que o engenheiro permaneça atento para a coleta das informações corretas. Existem inúmeras técnicas para a elicitação de requisitos, serão citadas as principais (IEEE, 2004):

- **Entrevistas:** é uma técnica bastante utilizada e suas vantagens e limitações devem ser bem conhecidas pelo engenheiro, para que seja utilizada em situações apropriadas e da melhor maneira;
- **Cenários:** é outra técnica bastante utilizada por trazer o contexto da elicitação dos requisitos de usuários. Uma de suas características é que por meio dessa técnica os engenheiros podem fazer perguntas como, por exemplo, "como isso é feito?" com relação às tarefas realizadas pelos usuários. O cenário mais comum utilizado são os casos de uso;
- **Protótipos:** Uma importante ferramenta para elicitação de requisitos são os protótipos que ajudam os *stakeholders* a visualizarem os requisitos semelhantemente como na técnica de cenários. Por meio dos protótipos os requisitos que não ficaram bem claros podem ser identificados e esclarecidos. Os protótipos possibilitam que os *stakeholders* compreendam de uma maneira mais clara as informações que eles precisam fornecer. Existem várias técnicas de prototipagem que oferecem subsídios tanto para a elicitação quanto para a validação de requisitos;
- **Reuniões facilitadas:** essa técnica objetiva proporcionar um ambiente no qual um grupo de pessoas possa compartilhar conhecimentos sobre seus requisitos de *software* de uma maneira mais fácil do que se fosse individualmente. É uma opção para a técnica de entrevista, pois, por meio dessa reunião são realizados debates que ajudam a aprimorar as ideias dos participantes, que são aspectos mais difíceis de surgirem por meio de entrevista. Deve ser realizada com cautela para que evitar situações conflitantes entre os participantes;

- **Observação:** essa técnica é bastante útil para que os engenheiros de *software* conheçam o ambiente organizacional dos usuários e assim saibam como eles interagem um com o outro e com outros *softwares*. O engenheiro conhece melhor o contexto do *software*, observando os usuários realizando suas tarefas.

4.2 ANÁLISE DE REQUISITOS

Esta seção abrange o processo de análise de requisitos que tem por objetivo encontrar e solucionar conflitos entre os requisitos, encontrar os limites do *software* e como deve ser a sua interação com o seu ambiente e preparar os requisitos de sistema para derivar os requisitos de *software* (IEEE, 2004).

A análise de requisitos tem sido reduzida à modelagem conceitual utilizando uma série de métodos para isto, por exemplo, a análise estruturada. A modelagem conceitual é bastante importante, mas a classificação de requisitos também deve ser incluída na análise para auxiliar a informar os *trade-offs* entre os requisitos (requisitos de classificação) e o processo de determinar estes *trade-offs* (requisitos de negociação) (IEEE, 2004).

4.2.1 Classificação de Requisitos

Os requisitos podem ser classificados de acordo com vários aspectos. Alguns exemplos de acordo com (IEEE, 2004) são:

- Classificação dos requisitos quanto às suas funcionalidades, restrições e especificações de qualidade (requisitos funcionas ou não funcionais);
- Classificação dos requisitos quanto à sua derivação, se são derivados de requisitos de alto-nível ou de propriedades emergentes, ou se estão sendo impostos diretamente no *software* por uma das partes interessadas ou por qualquer das outras fontes;
- Classificação do requisito para identificar se está sobre o produto ou o processo;
- Classificação do requisito quanto à sua prioridade. Normalmente, os requisitos classificados como de maior prioridade são os fundamentais e essenciais para que sejam alcançados os objetivos universais do *software*. Em geral, é utilizada uma escala para realizar a classificação. É importante que essa classificação considere os aspectos de custo, desenvolvimento e implementação;
- Classificação quanto ao escopo do requisito. O escopo do requisito refere-se à ampliação em que um requisito atinge o *software* e seus componentes. Um requisito

com escopo global tem muito mais influência sobre a arquitetura e componentes do *software*, podendo afetá-los significantemente. Já requisitos com escopo mais delimitado não oferecem tanto impacto a outros requisitos e concedem uma série de sugestões de *design*;

- Classificação do requisito quanto à sua volatilidade/estabilidade. Alguns requisitos podem, por razões de necessidades ou exigências, ser alterados e modificados durante todo o processo de desenvolvimento do *software.* Por isso é conveniente que haja certa estimativa da probabilidade de que requisitos possam sofrer essas modificações.

4.2.2 Modelagem Conceitual

A criação de modelos na área de análise de requisitos é muito importante e essencial pelo fato de trazer a representação do problema do mundo real. Seu objetivo é auxiliar na compreensão deste problema ao invés de iniciar a concepção da solução. Assim, os modelos conceituais abrangem modelos de entidades do domínio do problema que representam as relações do mundo real e suas dependências. Existem vários tipos de modelos como modelos de estado, modelos de objeto e de dados. São citados alguns dos fatores que inspiram a escolha do modelo em um projeto (IEEE, 2004).

- A natureza do problema. Alguns tipos de *software* demandam que certos aspectos sejam analisados, particularmente, de maneira rigorosa. Alguns modelos, como modelos de estados tendem a ser mais importantes para *softwares* em tempo real que para *softwares* de gerência da informação;
- A especialidade do engenheiro de *software*. Frequentemente é mais produtivo utilizar uma notação de modelagem ou métodos com os quais o engenheiro já esteja familiarizado;
- Os requisitos de processo do cliente. Os clientes têm a possibilidade de escolher uma notação ou método de modelagem no qual estejam mais familiarizados e mesmo proibir os que eles não conhecem. Este fator pode entrar em conflito com o anterior;
- A disponibilidade de métodos e ferramentas. Métodos e notações de modelagem podem acabar não sendo aceitos por não ser apoiado o suficiente por treinamentos e ferramentas, mesmo que sejam apropriados para determinados tipos de problemas.

É conveniente começar a modelagem a partir da construção do modelo de contexto do *software*. Compreender o contexto do *software* auxilia na ligação entre o seu ambiente

externo e o *software* pretendido. Isso é fundamental para compreender o contexto do *software* no seu ambiente operacional e identificar suas interfaces com o ambiente (IEEE, 2004).

4.2.3 Projeto Arquitetônico e Requisitos de Alocação

Projeto arquitetônico é o ponto em que os requisitos do processo sobrepõem-se com o *software* ou projeto de sistemas e explana como é impossível dissociar de maneira limpa essas duas tarefas. Muitas vezes o engenheiro de *software* atua também como arquiteto de *software*, pois, para a análise e elaboração dos requisitos é necessário que os componentes que satisfarão os requisitos sejam identificados. Esta é a alocação dos requisitos de atribuição, aos componentes, da responsabilidade para satisfazer aos requisitos (IEEE, 2004).

A alocação de requisitos é importante por permitir uma análise mais detalhada dos requisitos. Deste modo, por exemplo, uma vez que um grupo de requisitos tem sido alocado a um componente, os requisitos individuais podem ser analisados novamente para desvendar novos requisitos sobre como os componentes precisam interagir com outros componentes de modo a satisfazer os requisitos alocados. Em projetos considerados grandes, a alocação impulsiona uma nova rodada de análise para cada subsistema (IEEE, 2004).

Projeto arquitetônico está estreitamente identificado com a modelagem conceitual. O mapeamento de entidades de domínio do mundo real para componentes de *software* nem sempre é tão intuitivo de se identificar, de modo que o *design* da arquitetura é reconhecido como um tópico separado. Para a modelagem conceitual e projeto arquitetônico, os requisitos de notações e métodos são praticamente os mesmos (IEEE, 2004).

4.2.4 Negociação de Requisitos

Um termo comumente utilizado nessa área é “resolução de conflitos”. Isso acontece quando dois *stakeholders* entram em conflito por precisarem de características incompatíveis, entre requisitos e recursos, ou entre os requisitos funcionais e não funcionais, por exemplo. Esses problemas precisam ser claramente resolvidos e cabe ao engenheiro de *software* resolvê-los de forma a não ser uma decisão unilateral, mas ele deve consultar os *stakeholders* para chegar a um consenso sobre um *trade-off* apropriado. Frequentemente é importante, por razões contratuais, que decisões como essas sejam rastreadas de volta para o cliente. Este assunto tem sido classificado nessa área, porque os problemas surgem como resultado da

análise. Porém, um caso forte pode também ser considerado como um assunto na validação de requisitos (IEEE, 2004).

4.3 ESPECIFICAÇÃO DE REQUISITOS

Para outras áreas da engenharia o termo "especificação" refere-se à atribuição de valores numéricos ou aos limites das metas do *design* do produto. Para a engenharia de *software* o termo "especificação de requisitos de *software*" geralmente, refere-se à produção de documentos, ou equivalentes eletrônicos, que podem ser sistematicamente revistos, avaliados e validados. Sistemas complexos naturalmente exigem uma quantidade maior de documentação e, portanto podem ser produzidos até três tipos de documentos de especificação (definição do sistema, requisitos de sistema e requisitos de *software*). Para produtos de *softwares* menos complexos, somente o documento de requisitos de *software* é necessário. Estes três documentos são descritos aqui, e tem-se o entendimento que pode haver combinações entre eles, conforme as necessidades e exigências (IEEE, 2004).

- Documento de Definição do Sistema. Este documento, também chamado de documento de requisitos do usuário ou conceito de operações, objetiva registrar todos os requisitos do sistema. São definidos os requisitos de alto nível da perspectiva do domínio do sistema. As pessoas que tem acesso a este documento incluem os representantes do sistema cliente/usuários, por isso é necessário que o conteúdo do documento seja exposto em termos de domínio. São listados os requisitos do sistema conjuntamente com os objetivos universais do sistema, seu ambiente alvo e uma declaração das restrições, os pressupostos e requisitos não funcionais. Podem ser incluídos modelos conceituais para explanar o contexto do sistema, cenários de uso, as principais entidades do domínio e, entre outras;
- Documento de Especificação de Requisitos do Sistema. Os desenvolvedores de sistemas de importantes *softwares* e componentes que não incluem *software*, como por exemplo, um moderno avião, geralmente dividem a descrição dos requisitos do sistema a partir da descrição dos requisitos de *software*. Neste ponto de vista, os requisitos são especificados seguindo certa sequência. Primeiramente os requisitos de sistemas são especificados, após isto os requisitos de *software* derivam dos requisitos de sistemas e então os requisitos para os componentes de *software* são especificado;
- Documento de Especificação de Requisitos de *Software*. Este documento oferece base para um acordo entre clientes, contratantes ou fornecedores do que o produto de

software deve ou não fazer. Pode ser acompanhado do documento de definição dos requisitos de *software*, para auxiliar a leitura dos que não conhecem termos técnicos. Por meio deste documento é possível realizar uma avaliação rigorosa dos requisitos antes do *design* ser iniciado e como resultado evitar o redesenho posteriormente. Deve fornecer apoio para estimação de custos do produto, riscos e horários. As organizações podem utilizar este documento como base para desenvolver sua própria validação e verificação de planos de maneira mais produtiva. Fornece também base de informação para a transição de um produto de *software* para novos usuários ou novas máquinas e por fim para a melhoria do *software*. Os requisitos de *software* são frequentemente escritos em linguagem natural, porém, em especificação de requisitos de *software*, esta pode ser complementada com uso de descrições formais ou semi formais. É necessário que sejam escolhidas as notações mais propícias para que requisitos específicos e aspectos da arquitetura de *software* sejam escritos com mais precisão e de maneira mais sucinta do que a linguagem natural. A regra é que as notações escolhidas permitam aos requisitos serem escritos de maneira tão precisa quanto possível.

4.4 VALIDAÇÃO DE REQUISITOS

Os documentos de requisitos podem passar por procedimentos de validação e verificação. Por meio da validação é possível verificar e garantir se o engenheiro de *software* tem entendido os requisitos e também certificar se o documento está de acordo com os padrões da empresa e se são compreensíveis, consistentes e completos. Diferentes *stakeholders* devem analisar esse documento. É expressamente normal marcar um ou mais pontos no processo de requisitos, onde os requisitos são validados. O alvo é apanhar todos os problemas antes que os recursos sejam empenhados em resolver os requisitos. A validação de requisitos está preocupada com o processo de analisar o documento de requisitos para assegurar que se define o *software* corretamente (o *software* que os usuários aguardam) (IEEE, 2004).

4.4.1 Revisão de requisitos

Possivelmente a maneira mais corriqueira de se validar requisitos é por meio de inspeção ou revisão do documento de requisitos. Um grupo de revisores é identificado para uma breve procura por erros, premissas erradas, falta de clareza e desvio da prática padrão. A

formação o grupo de revisores é importante e isso pode auxiliar a prover orientações sobre o que procurar na forma de listas de verificação. As revisões podem ser estabelecidas logo após a conclusão do documento de definição do sistema, do documento de especificação do sistema, do documento de especificação de requisitos de *software*, a especificação de linha de base para uma nova versão ou qualquer outra etapa do processo (IEEE, 2004).

4.4.2 Prototipagem

A prototipagem normalmente permite que a interpretação do engenheiro de *software*, sobre os requisitos, seja validada e permite que novos requisitos sejam elicitados. Assim como para a elicitação, na validação de requisitos também existe uma gama de técnicas de prototipagem e vários pontos no processo em que a validação de um protótipo pode ser aperfeiçoada. Por meio do uso de protótipos é possível interpretar de forma mais fácil os pressupostos do engenheiro de *software* e, quando necessário, oferecer *feedback* do porque que eles estão errados, isto é visto como uma vantagem da prototipagem. Por exemplo, o comportamento dinâmico de uma interface, provavelmente, será mais bem entendido por meio de um protótipo animado do que por uma descrição textual ou modelos gráficos. Porém, existem também as desvantagens. Por exemplo, a atenção do cliente ou usuário pode ser desviada de pontos essenciais como observar as funcionalidades, e focar em questões de estética ou problemas de qualidade nos protótipos. Por este motivo, muitos aconselham que sejam evitados protótipos que usem *software* e como opção utilize, por exemplo, maquetes baseadas em *flip-chart*. Geralmente os protótipos são caros de se desenvolver, mas como podem evitar o desperdício de recursos por causa de tentativas de se desenvolver algo que não é o esperado pelo cliente, seu custo pode ser mais bem aceito e justificado, pois ajudará a evitar desperdícios maiores (IEEE, 2004).

4.4.3 Validação do Modelo

Este tipo de validação é normalmente necessário para validar os modelos produzidos durante a análise. Em modelos de objetos, por exemplo, é conveniente elaborar uma análise estática para conferir se existem vias de comunicação entre objetos que, no domínio dos *stakeholders*, trocam dados. Caso sejam utilizadas notações formais, é possível usar o raciocínio formal para comprovar as propriedades de especificação (IEEE, 2004).

4.4.4 Teste de Aceitação

Uma propriedade imprescindível de um requisito de *software* é que ele deve ser possível de validar que o produto final o satisfaça. Requisitos que não podem ser validados, não passam de apenas "desejos" e, logo não são essenciais. Portanto, uma tarefa muito importante é o planejamento de como conferir cada requisito. Projetar os testes de aceitação, muitas vezes já faz essa verificação. Para os requisitos não funcionais, no entanto, pode ser mais complicado projetar os testes de aceitação. Para esses requisitos serem validados é necessário que sejam analisados e cheguem ao ponto de ser expressos quantitativamente (IEEE, 2004).

4.5 LEVANTE DOS ARTEFATOS

Nesta seção são expostos os artefatos que foram levantados, bem como suas principais características e sua classificação às subáreas da engenharia de requisitos. Para levantar os artefatos foi necessário analisar como cada uma das metodologias ágeis selecionadas nesta pesquisa aborda a subárea da engenharia de requisitos. Após, os artefatos serem identificados foi realizada uma análise confrontando-os com as definições do guia SWEBOK (IEEE, 2004) quanto às subáreas de engenharia de requisitos. Nos quadros 5, 6, 7 e 8 são mostrados os artefatos levantados, a metodologia a qual pertencem, a subárea da engenharia de requisitos e as suas características.

O quadro 5 mostra os artefatos levantados na subárea de elicitação de requisitos, a metodologia a qual pertencem e suas principais características.

QUADRO 5. Artefatos das metodologias ágeis na subárea de elicitação de requisitos

(Continua)

Subárea da Engenharia de Requisitos	Metodologia	Artefato	Características
	XP	*User Stories*	- Descrição simples de partes de funcionalidade do sistema - São pequenas, histórias muito grandes devem ser fragmentadas - Capturam requisitos de alto nível - Escrita por *stakeholders* em uma entrevista com desenvolvedores - Proporciona uma comunicação clara e simples tanto entre os *stakeholders* e engenheiro de *software* quanto entre os membros da equipe

Elicitação de Requisitos			- São escritas em cartões
		Cartão de História	- Cartão no qual são escritas as *user stories* - Pequeno o suficiente para caber em uma iteração - Expõe as funcionalidades do usuário - Testável
	SCRUM	*Product Backlog*	- Lista principal de todas as funcionalidades desejadas no produto - Itens geralmente escritos como *user stories* - Dinâmico - Mutável
	FDD	Lista de Funcionalidades	- Os requisitos levantados são escritos nessa lista - Sua estrutura é na forma de ação, resultado e objeto, por exemplo, calcular o total de uma venda
	DSDM	Lista de Requisitos Priorizada	- Desenvolvida por meio de *workshops* ou reuniões facilitadas - Serve para identificar o subconjunto útil mínimo de funcionalidades que apoiam o negócio - Base para todas as decisões de planejamento do projeto
		Lista de Requisitos Não funcionais	- É um subconjunto da lista priorizada de requisitos - Lista que define os requisitos que especificam o quão bem o sistema deve operar e não o que ele deve fazer
	CRYSTAL	Documento de Requisitos	- Levanta e documenta os requisitos - Contém as descrições dos requisitos funcionais e não funcionais - Contem os objetivos do *software* - Considera o ponto de vista de vários *stakeholders* - Auxilia na comunicação entre o cliente e a equipe de desenvolvimento
	ASD	Esboço da Especificação do Produto	- Define as funcionalidades e características do produto

O quadro 6 mostra os artefatos levantados na área de análise de requisitos, a metodologia a qual pertencem e suas principais características.

QUADRO 6. Artefatos das metodologias ágeis na subárea de análise de requisitos

(Continua)

Subárea da Engenharia de Requisitos	Metodologia	Artefato	Características
	XP	Cartão de História	- Estimável ou fácil para estimar

Análise de Requisitos			- Cada cartão recebe uma prioridade
		Cartão de Tarefas	- Todos os cartões de tarefas estão relacionados a um cartão de história - Apresentam as responsabilidades dos desenvolvedores - Identificam detalhes da implementação da história - Mostram planos de *design* entre os desenvolvedores - Desenvolvedores estimam quanto tempo levarão para realizar a tarefa
		Cartão CRC	- Realiza a representação do problema - Usuários e desenvolvedores trabalham juntos para criar esse modelo - Semelhantes aos diagramas de classe - Proporciona interação do grupo - Incentiva um conhecimento intuitivo do modelo
	SCRUM	*Product Backlog*	- Itens possuem os atributos da descrição, ordem e estimativa - Priorizado pelo *Product Owner* - São analisados e revistos no processo de sua preparação
		Sprint Backlog	- Lista de tarefas - Derivada do *Product Backlog* - Os itens de maior prioridade no *Product Backlog*, geralmente, fazem parte das primeiras tarefas a serem executadas - Estimativa de tempo da execução de cada tarefa - São listadas as tarefas de uma *Sprint* com as determinadas funcionalidades que devem ser entregues ao final da *Sprint*
	FDD	Lista de Funcionalidades	- Organizada hierarquicamente com requisitos funcionais - Classificação dos requisitos quanto à sua prioridade - Representa todas as funcionalidades que satisfaçam aos requisitos - Tendo essa lista como base são gerados vários modelos - Abrange todas as funcionalidades necessárias para o software
		Diagramas de Casos de Uso	- Documenta o que o sistema faz do ponto de vista do usuário -Auxilia a comunicação entre os analistas e os clientes -Expõe as funcionalidades mais importantes do sistema -Mostra os relacionamentos entre atores e casos de uso e casos de uso

			entre si
		Diagramas de Classes	- Descreve as classes e seus relacionamentos - Elementos principais: classes, atributos e métodos - Representa a estrutura do sistema - Permite visualizar como o *software* interagirá com o seu ambiente - Mostra os relacionamentos existentes entre as classes que são abstraídas no projeto, e como esses relacionamentos colaboram para a execução de um processo específico - Modela o vocabulário do sistema
		Diagramas de Sequência	-Enfatiza a sequência das mensagens trocadas entre os objetos -Apresenta as interações entre atores e sistema - Identifica os métodos que devem ser disparados entre os atores e os objetos envolvidos
	DSDM	Lista de Requisitos Priorizada	- Todos os requisitos são priorizados usando as regras de MoSCow - Atualizada sempre que necessário
		Lista de Requisitos Não funcionais	- Os requisitos são priorizados usando a técnica de MoSCow
		Modelo Funcional (incluindo o Protótipo Funcional)	- Define a solução que será realizada sem entrar em detalhes de aspectos não funcionais - Evolui ao longo do projeto - O modelo funcional consiste em ambos: documentos e softwares (Protótipos Funcionais) - Tanto quanto possível, o modelo funcional está contido dentro do protótipo funcional, mas também deve ser documentado - Fornece uma demonstração coesiva das funcionalidades e dados dos requisitos a serem cumpridos - Demonstra a viabilidade de serem alcançados os requisitos não funcionais - Protótipos funcionais são partes do modelo funcional
		Documento de Requisitos	- O próprio cliente analisa o documento e oferece *feedback* - Delimita o escopo, limites do *software* - Integraliza e relaciona grupos de perspectivas dos *stakeholders*, evitando que haja conflitos

	CRYSTAL (CLEAR/ORANGE)	Casos de Uso ou Descrição das Funcionalidades	- Utilizado para demonstrar uma sequência de ações de um ator para a obtenção de algum resultado - Auxilia os *stakeholders* a compreender e a validar, ou não, o c comportamento do sistema - Delimita as reponsabilidades de cada ator
		Modelo de Objetos	- Ilustra o problema do mundo real - Representa a estrutura estática do sistema - Expõem classes, atributos, métodos e relacionamento entre as classes - Serve como documentação da estrutura do sistema - Bom para compreensão do funcionamento do sistema - Pode incluir vários diagramas
	ASD	Esboço da Especificação do Produto	- Proporciona um conhecimento inicial sobre os limites e escopo do *software* - Geralmente inclui uma lista de requisitos e modelos que demonstram as funcionalidades do sistema

O quadro 7 mostra os artefatos levantados na subárea de especificação de requisitos, a metodologia a qual pertencem e suas principais características.

QUADRO 7. Artefatos das metodologias ágeis na subárea de especificação de requisitos

(Continua)

Subárea da Engenharia de Requisitos	**Metodologia**	**Artefato**	**Características**
	SCRUM	*Product Backlog*	- Lista todos os requisitos - Expõe a tarefa prenunciada para o produto
	FDD	Lista de Funcionalidades	- Registra todos os requisitos, eventos e iterações possíveis ao projeto - Geralmente incluem modelos conceituais - Fornece apoio para a estimação do cronograma do projeto
		Diagramas de Casos de Uso	- Explana o contexto do sistema - Expõe o que o sistema deve fazer
	DSDM	Lista de Requisitos Priorizada	- Lista todos os requisitos do sistema - Serve para identificar o subconjunto útil mínimo de funcionalidades que apoiam o negócio

		Lista de Requisitos Não funcionais	- Lista os requisitos de desempenho e outros atributos de qualidade
	CRYSTAL (CLEAR/ORANGE)	Documento de Requisitos	- Registra todos os requisitos do sistema - Serve como base para a definição de um contrato - Documento utilizado por várias pessoas como cliente, gerente, engenheiros e desenvolvedores.
		Casos de Uso ou Descrição das Funcionalidades	- Demonstra o comportamento e funcionalidades do sistema - Explana o contexto do sistema - Visualizado por vários interessados
	ASD	Esboço da Especificação do Produto	- Objetiva definir todas as funcionalidades e características do produto - É utilizado para estimação e planejamento iniciais - Inclui uma lista de componentes que indica um grupo ou conjunto de características

O quadro 8 mostra os artefatos levantados na subárea de validação de requisitos, a metodologia a qual pertencem e suas principais características.

QUADRO 8. Artefatos das metodologias ágeis na subárea de validação de requisitos

(Continua)

Subárea da Engenharia de Requisitos	**Metodologia**	**Artefato**	**Características**
Validação de Requisitos	**XP**	Testes de Aceitação	- Verifica se os requisitos estão funcionando corretamente - É feita uma análise sobre o documento que contém os requisitos - É simulada a interação do usuário com o sistema - Indica o resultado esperado e não esperado de uma história (funcionalidade) - Associados às *user stories* - Escritos geralmente pelos clientes com a ajuda de um membro da equipe - Escritos juntamente com as *user stories*
		Testes de Unidade	- Os requisitos são revisados por

			meio das classes - Escritos pelos desenvolvedores enquanto codificam o sistema - Verificam se os métodos das classes do sistema fornecem respostas corretas - Atende aos objetivos de documentar e validar o funcionamento do sistema
	FDD	Testes de Unidade	- Escritos pelos desenvolvedores enquanto codificam o sistema - Realizado sobre cada classe do sistema - Procura-se verificar se os métodos das classes do sistema fornecem respostas corretas
	DSDM	Modelo Funcional (incluindo o Protótipo Funcional)	- O modelo funcional consiste em ambos: documentos e softwares (Protótipos Funcionais) - Protótipos funcionais são partes do modelo funcional - Modelo funcional e protótipo funcional, juntos, representam as funcionalidades que podem ser realizadas na iteração, prontas para serem testadas pelos utilizadores
		Design do Protótipo	- É um produto temporário que aborda questões não técnicas não abrangidas durante as atividades da iteração do Modelo Funcional - Os usuários podem comprovar ou não, se o desenvolvimento está na direção correta - Os *stakeholders* oferecem *feedback* aos desenvolvedores
		Registro de Revisão do *Design* do Protótipo	- Captura o *feedback* dos usuários sobre todos *Desgin* do Protótipo - Oferece a empresa a chance de rever o ao invés de testá-los - Registra os comentários positivos e negativos - Esse registro ajuda a evitar mais trabalho posteriormente
	CRYSTAL (CLEAR/ORANGE)	Casos de Teste	- Serve para executar testes de partes do sistema pré-definidas - Estabelece um conjunto de condições para o teste - Normalmente derivam de cenários de casos de uso - Verifica se o cenário ou requisitos testados respondem de acordo com o esperado
	ASD	Casos de Teste	- Serve para executar testes de partes do sistema pré-definidas - Estabelece um conjunto de condições para o teste - Normalmente derivam de

			cenários de casos de uso - Verifica se o cenário ou requisitos testados respondem de acordo com o esperado

5 Análise dos Artefatos das Metodologias

A fim de ser realizar a análise dos artefatos de requisitos das metodologias estudadas, foi efetuada a análise em um sistema real. Para tanto, para cada fase da engenharia de requisitos, alguns artefatos das diferentes metodologias foram selecionados e aplicados. A descrição do sistema encontra-se na próxima seção e a aplicação dos artefatos nas seções subsequentes.

5.1 DESCRIÇÃO DO SISTEMA

Deseja-se criar um sistema *on-line* para avaliação de Monografias. Por meio desse sistema, será controlado o envio e a correção de monografias. O sistema terá 3 tipos de usuários, o coordenador de monografia, professores e alunos. O professor poderá assumir dois papéis, orientador e membro de banca.

O coordenador é responsável por administrar todo o sistema, é ele quem cadastra os professores e alunos no sistema, além de associar um professor como sendo orientador de um aluno.

Quando chega o período de entrega de monografias, o coordenador abre o sistema para o envio de monografias, determinando uma data limite para o envio, e o sistema automaticamente envia e-mail para todos os orientadores informando-os dessa data. Após essa data, o sistema é fechado automaticamente e não é mais possível fazer o envio. O envio é responsabilidade do orientador. No envio, ele escolhe o nome do orientando, uma área de pesquisa, que está previamente cadastrada no sistema, descreve o título e resumo da monografia e anexa um arquivo pdf.

Após fechado o período de envio das monografias, o coordenador é responsável por associar a banca a cada monografia enviada, associando mais dois professores à banca, uma vez que o orientador está automaticamente na banca de avaliação. É então enviado um e-mail aos professores avisando-os que são membros da banca. Esse e-mail contém informações sobre o trabalho e data limite para avaliar o trabalho.

Quando o professor se loga no sistema aparece uma lista de trabalhos o qual este é membro, indicando o status da correção, ou seja, se o trabalho já foi corrigido ou não. O professor pode então baixar o pdf de cada trabalho e clicar no trabalho para começar a avaliá-lo.

Nota:

Após o período de avaliação finalizar, o sistema calcula a média de cada trabalho, mostrando uma lista de todos os alunos e suas notas ao coordenador e este autoriza o envio de um email, tanto para os alunos quanto para seus respectivos orientadores com a nota final e o comentário dos membros da banca.

5.2 APLICAÇÃO DOS ARTEFATOS NA SUBÁREA DE ELICITAÇÃO DE REQUISITOS

Para a fase de elicitação de requisitos foram escolhidos os artefatos ***user stories,*** **cartão de história e documento de requisitos**. O artefato ***Product Backlog*** não será utilizado, pois, é muito semelhante aos artefatos de XP além de, geralmente, descrever os requisitos no formato de ***user stories***. Os artefatos **lista de funcionalidades, lista de requisitos priorizada** e **lista de requisitos não funcionais** são listas com características semelhantes entre si e ao **documento de requisitos**. O **esboço da especificação do produto** é semelhante ao **documento de requisitos** e, embora sua estrutura seja organizada de maneira diferente e contenha alguns aspectos que não existem no **documento de requisitos**, ainda assim optou-se pela utilização deste por critérios de maior simplicidade e objetividade.

Foi utilizado, inicialmente os artefatos ***user stories*** e **cartão de história** para a elicitação de requisitos do sistema descrito acima. Nos quadros 9, 10 e 11 são mostrados três **cartões de história** com as ***user stories*** descritas e, os demais quadros encontram-se no Apêndice A - Elicitação de Requisitos.

QUADRO 9. Cartão de História – *User Stories* 1

US01	**Prioridade: Alta**
Como coordenador de monografia eu quero administrar o sistema	
para poder cadastrar professores e alunos e relacionar um professor	
como sendo orientador de um aluno.	

QUADRO 10. Cartão de História – *User Stories* 2

US02	**Prioridade: Alta**
Como coordenador de monografia eu preciso abrir o sistema para habilitar o envio de monografias e determinar uma data limite para o envio destas.	

QUADRO 11. Cartão de História – *User Stories* 3

US03	**Prioridade: Média**
Como professor orientador eu quero ser informado da data limite de envio da monografia para poder enviá-las dentro do prazo determinado.	

No quadro 12 é mostrado a elicitação de requisitos por meio do artefato documento de requisitos.

QUADRO 12. Documento de requisitos para elicitação de requisitos

Documento de Requisitos
Introdução
Este documento descreve um sistema *online* para avaliação de monografias. O propósito do sistema é possibilitar o controle sobre o envio e correção das monografias, permitindo que coordenador, professores e alunos tenham acesso ao sistema.
São estabelecidas prioridades para cada requisito, podendo variar entre prioridade baixa, média e alta.
Requisitos Funcionais

RF01 – Cadastro de usuários
Prioridade: Alta
Este requisito permite ao coordenador de monografias cadastrar professores e alunos.
RF02 – Relacionar professores com alunos
Prioridade: Alta
Este requisito permite relacionar um professor como sendo orientador de um ou mais alunos.
RF03 – Habilitar o envio de monografias
Prioridade: Alta
Este requisito permite ao coordenador abrir o sistema e habilitar o envio de monografias, bem como, determinar a data limite para envio.
RF04 – Enviar e-mail para os orientadores
Prioridade: Alta
Este requisito permite que o sistema envie email aos orientadores para informar a data limite de envio das monografias. E para avisá-los quando são membros de bancas. Esse e-mail contém informações sobre o trabalho e data limite para avaliar o trabalho.
RF05 – Encerrar o período de envio
Prioridade: Média
Este requisito permite que após a data limite para envio das monografias o sistema seja, automaticamente, fechado.
RF06 – Enviar monografia
Prioridade: Alta
Este requisito permite ao orientador escolher o nome do orientando, uma área de pesquisa, que está previamente cadastrada, descrever o título e resumo da monografia e anexar um arquivo pdf para enviar as monografias de seus orientandos.
RF07 – Associar banca
Prioridade: Alta

Este requisito permite ao coordenador associar dois professores, além do orientador, a cada banca de avaliação de monografias enviadas.
RF08 – Logar no sistema
Prioridade: Média
Este requisito permite ao coordenador, professores e alunos se conectarem ao sistema e visualizarem tarefas a eles designadas.
RF09 – Baixar monografia
Prioridade: Média
Este requisito permite ao professor baixar as monografias para avaliá-las
RF10 – Avaliar monografia
Prioridade: Média
Este requisito permite ao professor orientador avaliar, corrigir e comentar as monografias
RF11 – Visualizar correção
Prioridade: Média
Este requisito permite que os alunos e seus respectivos orientadores, recebam um email contendo uma lista que tenha as notas e comentários sobre as monografias
Requisitos Não funcionais
RNF01 – Acesso Online
Prioridade: Alta
Este requisito determina que o sistema seja web.
RNF02 – Segurança
Prioridade: Média
Este requisito estabelece que o sistema deve oferecer mecanismos de segurança quanto a autenticação de usuário, determinando que apenas pessoas previamente cadastradas tenham acesso ao sistema.

RNF03 - Usabilidade
Prioridade: Média
Este requisito estabelece que o sistema deve oferecer uma interface simples, intuitiva e amigável para que o acesso ao mesmo seja facilitado para os usuários

5.3 APLICAÇÃO DOS ARTEFATOS NA SUBÁREA DE ANÁLISE DE REQUISITOS

Os artefatos selecionados para a análise de requisitos são o **cartão de história**, **cartão CRC**, **modelo funcional (incluindo o Protótipo Funcional)** e **documento de requisitos**. Como o **cartão de história** e **documento de requisitos** já foram desenvolvidos, não são mostrados novamente.

A metodologia FDD utiliza alguns artefatos da UML como, **diagramas de casos de uso, de classes e de sequência** que por serem bastante conhecidos não foram desenvolvidos neste trabalho.

Os artefatos ***product backlog,*** **lista de funcionalidades, lista de requisitos priorizada, lista de requisitos não funcionais, e esboço da especificação do produto** não foram utilizados, pois, são muito similares. Optou-se pelo **cartão CRC** ao invés do **diagrama de classes e modelo de objetos** para visualizar as classes iniciais do sistema, pois os **cartões CRC** são mais simples e, geralmente, mais utilizados em metodologias ágeis para esse objetivo. Os artefatos **cartão de tarefas** e ***sprint backlog*** não foram utilizados, pois, são dedicados a divisão de tarefas que estão nos **cartões de história** e ***product backlog***.

Nos quadros abaixo são mostradas algumas das classes identificadas, suas responsabilidades e as classes que se comunicam. Os demais quadros encontram-se no Apêndice B – Análise de Requisitos.

QUADRO 13. Cartão CRC 1

Classe: Coordenador	
Responsabilidades	**Colaboração**
Cadastrar professores e alunos	Professor
Relacionar professor/aluno	Controladora
Habilitar o envio de monografias	
Determinar data limite para o envio	
Associar professor a uma banca	
Autorizar envio de email	

QUADRO 14. Cartão CRC 2

Classe: Professor (Orientador)	
Responsabilidades	**Colaboração**
Anexar monografia em pdf	Coordenador
Enviar monografia	Aluno
Avaliar monografia	Banca

O artefato **modelo funcional (incluindo o protótipo funcional)** inclui tanto documentação como protótipos. Como já foi criado um modelo de classes (**cartões CRC**), agora foram criados alguns protótipos do sistema abordado. Na figura 9 é mostrado o protótipo da tela inicial do coordenador de monografias. E na figura 10 é mostrado o protótipo da tela de avaliação de monografia.

Figura 9. Tela inicial do coordenador de monografias

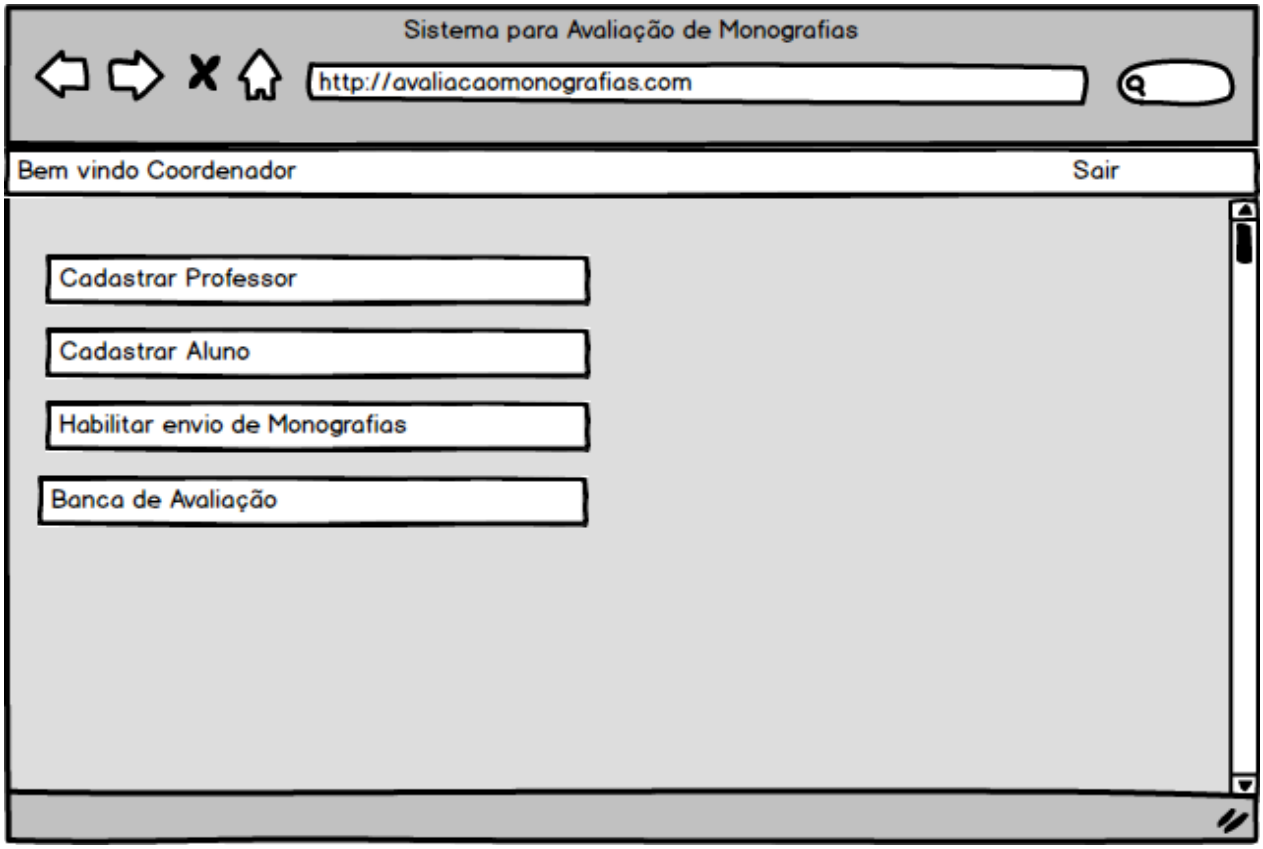
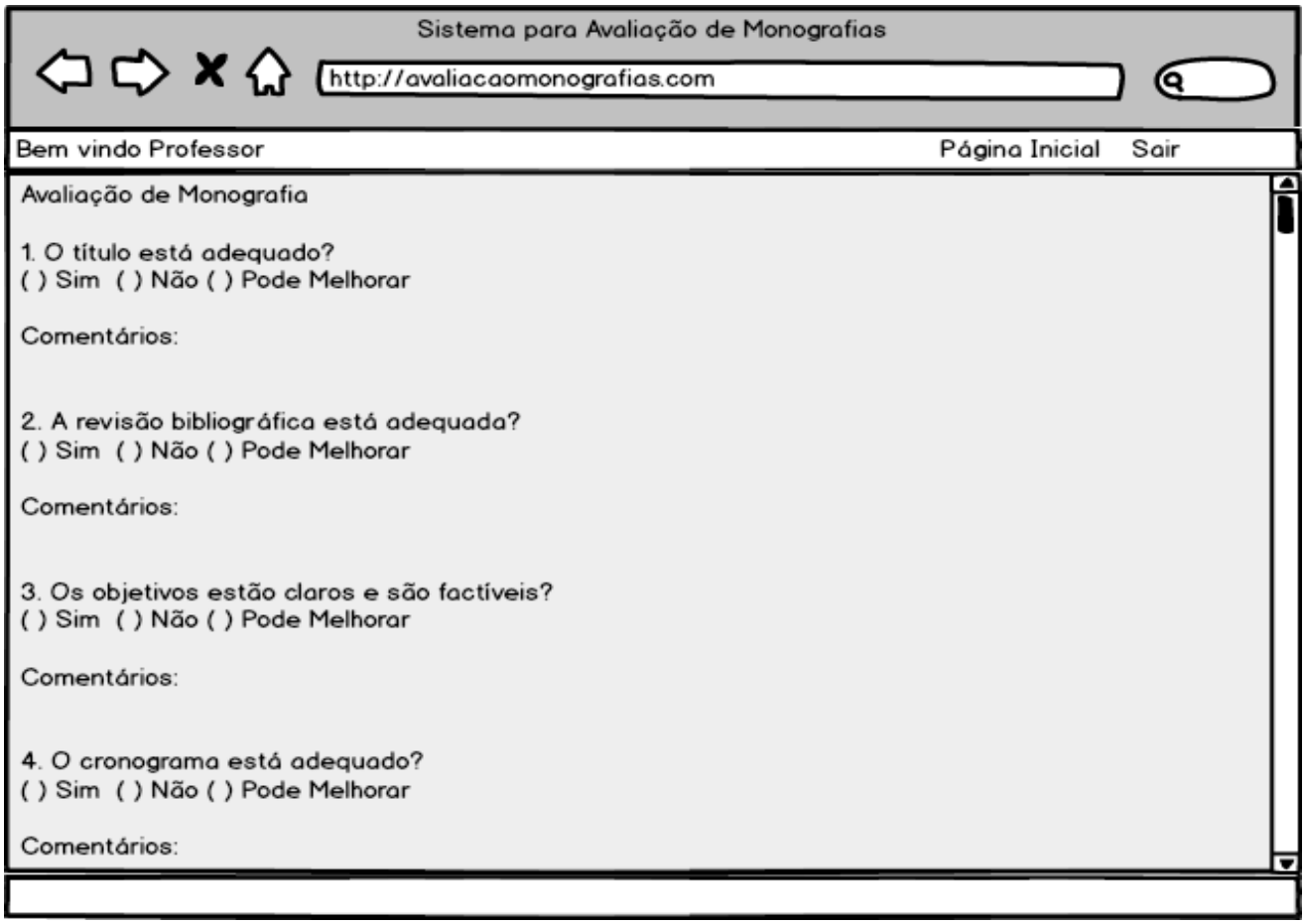


Figura 10. Tela de avaliação de monografia



5.4 APLICAÇÃO DOS ARTEFATOS NA SUBÁREA DE ESPECIFICAÇÃO DE REQUISITOS

Para a especificação de requisitos foi utilizado o ***product backlog*** e **documento de requisitos**. Os demais artefatos, **lista de funcionalidades, casos de uso e diagramas de casos de uso, lista de requisitos priorizada, lista de requisitos não funcionais e esboço da especificação do produto**, não foram utilizados por não contemplarem características significantemente diferentes dos outros artefatos.

QUADRO 15. Product Backlog

(Continua)

Id	Como um	Eu quero/preciso	Para	Notas	Prioridade
1	Coordenador	Administrar o sistema	Eu poder cadastrar professores e alunos e relacionar um professor	Necessário	Alta

			como sendo orientador de um aluno		
2	Coordenador	Abrir o sistema	Habilitar o envio de monografias e determinar a data limite para o envio delas		Alta
3	Professor	Ser informado da data limite de envio da monografia	Enviá-las dentro do prazo	O professor deve ser informado imediatamente após a habilitação de envio de monografias	Média
4	Coordenador	Sistema feche automaticamente após a data final de envio de monografias	Não ser mais possível fazer o envio após essa data		Média
5	Professor	Enviar as monografias dos meus orientados	Membros da banca possam visualizá-las e baixá-las		Alta
6	Coordenador	Associar os professores às monografias enviadas	Formar as bancas de avaliação		Média
7	Professor	Visualizar e baixar os trabalhos dos quais faço parte da banca	Poder avaliá-los	O professor pode visualizar outras monografias, mas não baixa-las	Alta
8	Coordenador	Após avaliar as monografias seja enviado um email aos professores e alunos	Visualizar as notas e comentários sobre as monografias		Média
9	Coordenador	Sistema seja web	Facilitar o envio de monografias	Requisito não-funcional	Alta
10	Coordenador	Sistema ofereça mecanismos de segurança para autenticação de usuário	O acesso ao sistema seja feito apenas por usuários cadastrados	Requisito não-funcional	Média
11	Coordenador	Sistema tenha a interface simples, amigável e intuitiva	O acesso ao sistema seja fácil	Requisito não-funcional	Média

5.5 APLICAÇÃO DOS ARTEFATOS NA SUBÁREA DE VALIDAÇÃO DE REQUISITOS

Para a validação de requisitos foram utilizados os seguintes artefatos: **testes de aceitação, testes de unidade e modelo funcional (incluindo o protótipo funcional)**. Quanto ao conteúdo, os **casos de teste** são muito semelhantes aos **testes de aceitação** e, por isso foi utilizado apenas **os testes de aceitação**. Os artefatos ***design* do protótipo** e **registro de revisão do *design* do protótipo** também não foram utilizados, pois, se assemelham com o artefato **modelo funcional (incluindo o protótipo funcional).**

Os **testes de aceitação**, normalmente são escritos no verso de cada **cartão de história**. Assim, são mostrados os **testes de aceitação** para cada ***user story***. Nos quadros 16 à 20 são mostrados os **testes de aceitação** para três ***user stories***. Os demais quadros encontram-se no Apêndice C – Validação de Requisitos.

QUADRO 16. Teste de aceitação para user story 1

US01 – Critérios para Aceitação
- O coordenador precisa estar logado no sistema para administrá-lo
- Para cadastrar professores e alunos é necessário que os mesmos façam parte da universidade
- No cadastro de cada professor deve haver um id único, username e senha, dados pessoais e profissionais e relação de seus orientandos
- Os alunos que podem ser cadastrados devem estar no último ano ou período do curso
- No cadastro de cada aluno deve haver um id único, username e senha, dados pessoais e profissionais e identificação de seu orientador

QUADRO 17. Exemplo de um cenário a ser testado 1

User Story	**Função que será testada**	**Estado inicial do sistema**	**Entrada**	**Saída esperada**
US01	Cadastro de professores	O sistema não tem professores cadastrados	Dados pessoais e profissionais a serem preenchidos no formulário	Cadastro realizado com sucesso

QUADRO 18. Teste de aceitação para user story 2

US02 – Critérios para Aceitação
- O coordenador deve habilitar o envio de monografias com no mínimo 15
dias de antecedência da data limite de envio
- A determinação da data limite de envio deve levar em consideração o
calendário acadêmico

QUADRO 19. Exemplo de um cenário a ser testado 2

User Story	**Função que será testada**	**Estado inicial do sistema**	**Entrada**	**Saída esperada**
US02	Habilitar o envio de monografia	O sistema esta desabilitado para o envio de monografias		Escolher uma data limite para envio

QUADRO 20. Teste de aceitação para user story 3

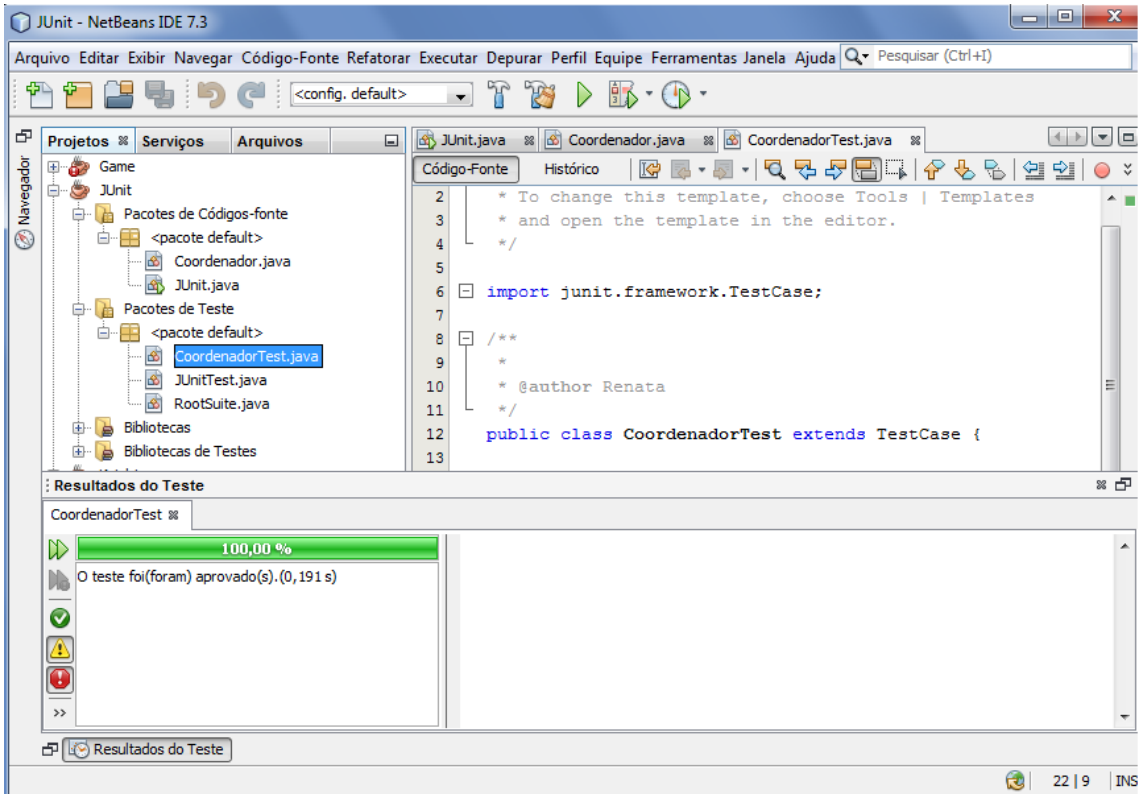
US03 – Critérios para Aceitação
- Os professores orientadores devem ser informados da data limite de
envio da monografia, por meio de um email

QUADRO 21. Exemplo de um cenário a ser testado 3

User Story	**Função que será testada**	**Estado inicial do sistema**	**Entrada**	**Saída esperada**
US03	Enviar email	O sistema possui email padrão		Orientadores receberem o email

Os **testes de unidade** objetivam testar isoladamente cada unidade do sistema. Foi gerado um exemplo simples de teste no método cadastrarProfessor da classe Coordenador. Esse teste objetiva verificar se o método corresponde aos resultados esperados e se existem erros. Na figura 11 é mostrado o resultado da execução do teste.

Figura 11. Teste no método cadastrarProfessor



Neste teste os resultados foram aprovados, o método de cadastro de professor não possui nenhum erro.

6 RESULTADOS

Para a avaliação dos artefatos utilizados foram tomados por base critérios que caracterizam cada subárea da engenharia de requisitos segundo o guia SWEBOK (IEEE, 2004). Foram selecionados os critérios e verificado se cada artefato atende a esses critérios. Nos quadros 22, 23, 24 e 25 foram demostrados o nível de adequação dos artefatos aos critérios, de três formas: não (o artefato não se encaixa nesse critério), parcialmente (o artefato contempla em partes ao critério, mas não o suficiente) e sim (o artefato se enquadra perfeitamente ao critério).

6.1 RESULTADOS NA ELICITAÇÃO DE REQUISITOS

Esta seção objetiva mostrar os resultados obtidos na subárea de elicitação de requisitos. No quadro 22 é mostrado os critérios de avaliação, os artefatos dessa subárea e o nível de adequação dos artefatos aos critérios de avaliação. Logo em seguida a explicação e comentários sobre os resultados e as recomendações dos artefatos mais indicados para a subárea de elicitação de requisitos.

QUADRO 22. Critérios para avaliação dos artefatos na subárea de elicitação de requisitos

	Artefatos					
Critérios de Avaliação	**Cartão de História**		***User Stories***		**Documento de Requisitos**	
Identificar Stakeholder	Não		Não		Não	
Auxilia na comunicação com o Stakeholder?	Sim		Sim		Sim	
Auxilia a delimitar o escopo do projeto?	Não		Parcialmente		Sim	
Auxilia a identificar as fontes de informações?	Não		Sim		Sim	
Técnicas de elicitação utilizada	X	Entrevista	X	Entrevista	X	Entrevista
		Cenários		Cenários		Cenários
		Protótipo		Protótipo		Protótipo
	X	Reuniões Facilitadas	X	Reuniões Facilitadas		Reuniões Facilitadas
		Observação		Observação		Observação
		Outras		Outras		Outras

Após a aplicação dos artefatos na subárea de elicitação de requisitos foram verificadas as suas características quanto aos critérios de avaliação estabelecidos. O cartão de história juntamente com as *user stories* oferecem como uma de suas principais vantagens a

simplicidade na comunicação com os *stakeholders*, pois, pelo fato de ser o próprio cliente quem as escreve e, consequentemente, ser escrito em uma linguagem não técnica, facilita a comunicação e entendimento dos requisitos. Auxilia também a comunicação entre os membros da equipe. Quanto a delimitação do escopo, as *user stories* oferecem certo subsídio para isto, porém, não o suficiente para projetos que exijam maior ênfase nesse aspecto. As *user stories* auxiliam na identificação das fontes de informações e requisitos, visto que, à medida que vão sendo escritas é possível identificar novas fontes conforme a necessidade de obtenção de novos requisitos. E em geral, a elicitação é feita por meio de entrevista e/ou reuniões facilitadas.

O documento de requisitos favorece a clareza na comunicação com os *stakeholders* e sua estrutura facilita a compreensão e visualização de todos os assuntos relacionados aos requisitos. Todo documento de requisitos introduz e delimita o escopo do sistema, esta é uma característica sempre presente nesse artefato. Assim como as *user stories* o documento de requisitos também auxilia na identificação de fontes de informação e a elicitação, geralmente, é realizada por meio de entrevista.

Portanto, para a subárea de elicitação de requisitos é recomendado a utilização das *user stories* para projetos simples e que não haja necessidade de muita documentação além de, necessitar da presença constante dos *stakeholders*. As *user stories* podem ser substituídas pelos casos de uso, para projetos com essa característica. O documento de requisitos é recomendado para os casos em que haja necessidade de maior detalhamento nos requisitos, definição do escopo, documentos complementares e, entre outros.

6.2 RESULTADOS NA ANÁLISE DE REQUISITOS

Esta seção objetiva mostrar os resultados obtidos na subárea de análise de requisitos. No quadro 23 é mostrado os critérios de avaliação, os artefatos dessa subárea e o nível de adequação dos artefatos aos critérios de avaliação. Logo em seguida a explicação e comentários sobre os resultados e as recomendações dos artefatos mais indicados para a subárea de análise de requisitos.

QUADRO 23. Critérios para avaliação dos artefatos na subárea de análise de requisitos

	Artefatos			
Critérios de Avaliação	**Cartão de história**	**Cartão CRC**	**Modelo funcional (incluindo o protótipo funcional)**	**Documento de Requisitos**
Auxilia a detectar e resolver conflitos entre os requisitos?	Parcialmente	Não	Sim	Sim
Auxilia a percepção de como deve ser a interação do sistema com o seu ambiente?	Sim	Parcialmente	Sim	Sim
Há a classificação dos requisitos quanto a funcionais e não funcionais?	Não	Não	Não	Sim
Os requisitos são classificados quanto a sua prioridade?	Sim	Não	Sim	Sim
É um modelo conceitual?	Não	Sim	Sim	Não

Após a aplicação dos artefatos na subárea de análise de requisitos foram verificadas as suas características quanto aos critérios de avaliação estabelecidos. O cartão de história oferece certo suporte para a detecção e correção de conflitos entre requisitos, mas, por ter uma estrutura muito simples não permite que seja tão perceptível e visível encontrar esses conflitos. O modelo funcional (incluindo o protótipo funcional) e documento de requisitos possibilitam a identificação desses conflitos com maior evidência, pela forma que são descritos e tratados os requisitos. Quanto à percepção de como deve ser a interação do sistema com seu ambiente, apenas o cartão CRC não oferece muito suporte, porém, os outros artefatos fornecem por meio de uma análise aprofundada ter essa percepção, pois, enquanto é feita a análise àqueles que estão envolvidos no ambiente que o sistema será inserido conseguem visualizar a interação do sistema nesse ambiente.

Em muitos artefatos não há a separação, nem ao menos classificação entre os variados tipos de requisitos. Um artefato de requisitos tende a estar mais bem organizado quando há uma separação lógica entre os requisitos. Nos artefatos utilizados somente o documento de requisitos mostrou essa característica com maior ênfase.

Quanto à priorização de requisitos, esta é uma característica bastante comum nos artefatos utilizados para a análise. Assim, apenas o cartão CRC não tem essa característica, embora eles derivem do cartão de história e *user stories* que possuem a priorização. Outro aspecto bastante característico na análise de requisitos é o uso de modelos. O cartão CRC e

modelo funcional (incluindo o protótipo funcional) são considerados modelos conceituais, pois, estabelecem o domínio do problema e expõe as informações que o sistema vai gerenciar.

Assim, para a análise de requisitos sugere-se a utilização do cartão de história para projetos que não exijam muito detalhamento e classificação de requisitos. O agrupamento dos artefatos cartão CRC, modelo funcional (incluindo o protótipo funcional) e documento de requisitos é uma ótima escolha para analisar por diferentes percepções, pois, complementam um ao outro. O cartão CRC, por exemplo, é útil e fácil para a compreensão dos *stakeholders* sobre como o sistema irá trabalhar, já o documento de requisitos auxilia bastante em aspectos como a detecção de conflitos, delimitação dos limites do *software* e, entre outros. O modelo funcional pode incluir outros modelos e diagramas para complementar a análise e documentação. Os protótipos são ótimos para o contato inicial dos *stakeholders* com o sistema e auxilia bastante a receber o *feedback* das partes interessadas. Portanto, recomenda-se a utilização desses três artefatos para uma análise completa.

6.3 RESULTADOS NA ESPECIFICAÇÃO DE REQUISITOS

Esta seção objetiva mostrar os resultados obtidos na subárea de especificação de requisitos. No quadro 24 é mostrado os critérios de avaliação, os artefatos dessa subárea e o nível de adequação dos artefatos aos critérios de avaliação. Logo em seguida a explicação e comentários sobre os resultados e as recomendações dos artefatos mais indicados para a subárea de especificação de requisitos.

QUADRO 24. Critérios para avaliação dos artefatos na subárea de especificação de requisitos

	Artefatos	
Critérios de Avaliação	***Product Backlog***	**Documento de requisitos**
É um documento que pode ser revisto, analisado e validado?	Sim	Sim
É considerado um documento de definição do sistema?	Parcialmente	Sim
Auxilia na identificação do que o sistema deve ou não fazer?	Parcialmente	Sim
Auxilia na estimação de custos do produto, riscos e horários?	Parcialmente	Sim
São utilizadas notações?	Não	Não

Após a aplicação dos artefatos na subárea de especificação de requisitos foram verificadas as suas características quanto aos critérios de avaliação estabelecidos. O *product*

backlog é um artefato que está em constante modificação durante todo o desenvolvimento do *software*, ele é atualizado (quando surgem novos requisitos), revisto, analisado e validado sempre que necessário. Em metodologias ágeis o documento de requisitos, geralmente, também tem essas características. O *product backlog*, quanto documento de definição do sistema, apesar de descrever todos os requisitos, não contempla outras características como, por exemplo, delimitar o escopo do sistema e organizar os requisitos quanto à classificação. Nesse aspecto o documento de requisitos é muito superior ao *product backlog*. Outro aspecto que o documento de requisitos é superior é quanto á identificação do que o sistema deve ou não fazer, pois, o *product backlog* enfatiza o que o sistema deve fazer quase não levando em consideração restrições quanto ao que não fazer. E também quanto à estimação de custos do produto, riscos e horários, o documento de requisitos é mais vantajoso, pois o *product backlog* oferece suporte para a estimação de custos e de tempo, mas não tanto para a estimação de riscos. Esses artefatos não utilizam notações, o que para a especificação de requisitos é um aspecto importante.

Portanto, para a especificação de requisitos é recomendada a utilização do artefato documento de requisitos por ser mais completo e servir de suporte para documentação. Não é um artefato complexo embora, o *product backlog* seja bem mais simples.

6.4 RESULTADOS NA VALIDAÇÃO DE REQUISITOS

Esta seção objetiva mostrar os resultados obtidos na subárea de validação de requisitos. No quadro 25 é mostrado os critérios de avaliação, os artefatos dessa subárea e o nível de adequação dos artefatos aos critérios de avaliação. Logo em seguida a explicação e comentários sobre os resultados e as recomendações dos artefatos mais indicados para a subárea de validação de requisitos.

QUADRO 25. Critérios para avaliação dos artefatos na subárea de validação de requisitos

	Artefatos		
Critérios de Avaliação	**Testes de aceitação**	**Testes de unidade**	**Modelo funcional (incluindo o protótipo funcional)**
Auxilia na revisão dos requisitos?	Sim	Sim	Sim
Utiliza prototipagem?	Não	Não	Sim
Identifica requisitos que não podem ser validados?	Sim	Sim	Sim
Há planejamento de verificar cada requisito?	Não	Parcialmente	Parcialmente

Após a aplicação dos artefatos na subárea de validação de requisitos foram verificadas as suas características quanto aos critérios de avaliação estabelecidos. Os três artefatos utilizados auxiliam na revisão e correção de requisitos. Quanto á prototipação, técnica muito comum nas metodologias ágeis, o artefato modelo funcional inclui a técnica de prototipagem e, assim auxilia na redução de riscos no projeto e possibilita aos *stakeholders* validarem ou não como será o novo sistema. Os três artefatos utilizados ajudam a identificar requisitos que não podem ser validados, isto é requisitos que não são importantes para o sistema e podem ser descartados. Embora, os três artefatos não ofereçam um planejamento detalhado para verificar cada requisito, geralmente, há essa verificação de uma maneira mais singela. Os testes de unidade, por exemplo, objetivam testar cada funcionalidade separadamente.

Assim, são recomendados para validação de requisitos os testes de aceitação quando forem utilizadas as *user stories*, pois são escritos conjuntamente. Os testes de unidade, dependendo da complexidade do projeto se tornam inviáveis pelo fato de testarem cada funcionalidade isoladamente, necessitando assim de bastante esforço e tempo. Portanto, sugere-se a sua utilização para projetos mais simples ou projetos que tenham uma equipe grande no qual essa tarefa possa ser compartilha. Os protótipos funcionais são recomendados para toda categoria de projeto, pois, embora tenham algumas desvantagens como o custo, são ideais tanto para a elicitação como validação de requisitos.

7 CONCLUSÃO

A necessidade de melhores resultados no processo de desenvolvimento de software é visível, pois, os índices de desempenho nessa área ainda são frustrantes. Atender as expectativas dos *stakeholders* tanto em cumprir o prazo de entrega, custo e funcionalidades esperadas são fatores que influenciam diretamente o desempenho de um projeto de software.

As metodologias ágeis investem fortemente no envolvimento do cliente durante todo o desenvolvimento do projeto para, entre outras vantagens, diminuir os riscos de entregar *software* que não atenda às suas expectativas. Para isto, as metodologias ágeis utilizam várias práticas, princípios e valores. Essa ênfase na participação ativa dos *stakeholders* é motivada pelo fato de que diminui os riscos de ser entregue um *software* que não atenda aos requisitos necessários, pois, durante todo o processo de desenvolvimento os *stakeholders* estão presentes e assim oferecem *feedback*. Essa é uma das características que diferenciam as metodologias ágeis das tradicionais. Nas metodologias tradicionais não há, normalmente, essa ênfase na participação ativa do cliente. Geralmente, os *stakeholders* participam mais nas primeiras fases do desenvolvimento.

As metodologias ágeis diferem também das tradicionais, quanto à execução dos processos de engenharia de requisitos. As metodologias tradicionais executam esses processos sequencialmente e geram bastante documentação. Já as metodologias ágeis são flexíveis, isto é, executam os processos de engenharia de requisitos, mas não tão sequencialmente como as tradicionais, além de não produzirem muita documentação. Nessas metodologias são utilizados diversos artefatos, técnicas, ferramentas que auxiliam na elicitação, análise, especificação e validação de requisitos. Mas o desenvolvimento de *software* ainda carece de melhorias nessa área.

A engenharia de requisitos em metodologias ágeis é um aspecto que deve ser levado em consideração. Inicialmente, isso pode parecer um aspecto conflitante, pois, a engenharia de requisitos tem vários passos a serem seguidos e metodologias ágeis objetivam desenvolver *software* com maior rapidez. Porém, como mencionado, as metodologias ágeis são flexíveis e assim a engenharia de requisitos se torna adaptável às metodologias ágeis e/ou vice-versa.

Com esta pesquisa foi possível compreender de uma maneira mais íntima como é essa adaptação da engenharia de requisitos às metodologias ágeis. Ficou constatado que embora a maior parte dessas metodologias não enfatize claramente a execução das subáreas da engenharia de requisitos, todas oferecem suporte para essa área. Como foi enfatizado o estudo

dos artefatos gerados nessa área, identificou-se que algumas metodologias não geram muitos artefatos para a subárea de requisitos. Outras possuem artefatos padrões, como XP com as *user stories*. Não existe um consenso de quais artefatos são os mais indicados para cada categoria de projeto. Por isso por meio do levante, análise e estudo dos artefatos dessas metodologias e, aplicação desses artefatos em um sistema real, foi possível verificar suas principais características e recomendar os artefatos mais indicados para cada situação e tipo de projeto. Com essa recomendação objetiva-se contribuir para a obtenção de melhores resultados nas subáreas da engenharia de requisitos e no desenvolvimento de *software*.

Conclui-se com base nessa pesquisa que para a obtenção de melhores resultados no desenvolvimento de *software* é necessário que haja uma melhora significativa na execução das subáreas da engenharia de requisitos, pois, de fato contribuem para o sucesso ou fracasso de um projeto. Conclui-se também que metodologias ágeis e engenharia de requisitos são aspectos que podem trabalhar conjuntamente. Assim, entende-se que embora existam muitos desafios na área de requisitos eles podem ser confrontados e vencidos por meio do uso de técnicas e práticas e por meio da participação ativa do cliente durante todo o desenvolvimento de *software* o que contribui significantemente para o sucesso do projeto.

REFERÊNCIAS

ABRAHAMSSON, P., SALO, O., RONKAINEN, J., and WARSTA, J. (2002). **Agile Software Development Methods**. Technical report, VTT Publications 478, VTT, Finland.

AMBLER, S. W. **Modelagem Ágil: Práticas eficazes para a Programação Extrema e o Processo Unificado**. Porto Alegre: Bookman, 2004. 351p.

BECK, K. **Programação Extrema (XP): Acolha as mudanças**. São Paulo: Bookman, 2000. 175p.

BONOW, R. **Sugestão e Modelagem de Práticas do Desenvolvimento Ágil para Aplicação em Desenvolvimento Distribuído Onshore Insourcing**. 2008. 87f. Monografia (Bacharelado em Ciência da Computação) - Universidade Federal de Pelotas, Pelotas, 2008.

BRAZ, A. **Metodo Ágil aplicado ao Desenvolvimento de SoftwareCOnfiável Baseado em Componentes**. 2011. 9f. Monografia (Tecnologia em Engenharia de Software) - Universidade Estadual de Campinas, Campinas, 2011. Disponível em: <http://alanbraz.files.wordpress.com/2011/05/wbma2011.pdf> Acesso em: 21 jun. 2013

Chaos Summary 2009: The 10 laws for chaos. Disponível em: <www.standishgroup.com> Acesso em: 3 jun. 2013

COAD, Peter; LUCA, Jeff de; LEFEBVRE, Eric. **Java Modeling Color with Uml: Enterprise Components and Process with Cdrom**. Prentice Hall PTR, 1999.

COCKBURN, Alistair. **Agile software development: the cooperative game**. Pearson Education, 2006.

DSDM Public Version 4.2 Manual. (n.d.). **DSDM Consortium - Enabling Business Agility**. Retrieved March 24, 2010, from http://www.dsdm.org/version4/2/public/default.asp

FILHO, D. L. B. **Experiências com desenvolvimento ágil**. 2008. 141f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 2008.

GIL, A. C. **Como Elaborar Projetos de Pesquisa**. 4ª ed. São Paulo: Editora Atlas S.A., 2002.

LUCIA, D. A., and QUSEF, A. **Requirements engineering in agile software development**. Journal of Emerging Technologies in Web Intelligence, 2(3):212–220, 2010.

HENRAJANI, Anil. **Desenvolvimento ágil em Java com Spring, Hibernate e Eclipse**. São Paulo: Pearson Prentice Hall, 2007.

HEPTAGON TI Ltda. **Heptagon Tecnologia da Inforção.**. Disponível em: <http://heptagon.com.br >. Acesso em: 15 novembro 2013.

HIGHSMITH, James A. **Adaptive software development**. New York: Dorset House, 2000.

HIGHSMITH, James A. **Agile software development ecosystems**. Addison-Wesley Professional, 2002.

HIGHSMITH, Jim. Retiring Lifecycle Dinosaurs. **Software Testing & Quality Engeneering (STQE)**, 2000. Disponível em: <http://stqemagazine.com>. Acesso em: 14 novembro 2013.

IEEE 2004. IEEE Computer Society. **Guide to the Software Engineering Body of Knowledge (SWEBOK).** Disponível em: http://www.computer.org/portal/web/swebok/htmlformat>. Acesso em 5 nov. 2013. New York: IEEE, 2004.

JEFFRIES, E. R. **What is Extreme Programming?**. Disponível em: < http://xprogramming.com/what-is-extreme-programming/ >. Acesso em: 31 mai. 2013

JÚNIOR, L. C. **AQUA – Atividades de Qualidade no Contexto Ágil**. 2008. 161p. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de São Carlos, São Paulo.

Manifesto for Agile Software Development, 2001. Disponível em: <http://agilemanifesto.org/> Acesso em: 31 mai. 2013

PAETSCH, F., EBERLEIN, A., and MAURER F. 2003. **Requirements Engineering and Agile Software Development**. In Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises(WETICE '03). IEEE Computer Society, Washington, DC, USA, 308-.

POLINI, A. "**Software Requirements**". Diponível em: < http://www1.isti.cnr.it/~polini/lucidiSE/Requirements1.pdf > Acesso em: 02 mai. 2013

ROSENBERG, Doug; COLLINS-COPE, Mark; STEPHENS, Matt. **Agile development with ICONIX process: people, process, and pragmatism**. Apress, 2005.

PRESSMAN, R. S. **Engenharia de Software**. sexta edição. ed. São Paulo: McGraw-Hill, 2006. 709p.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do Scrum:** Um guia definitivo para o Scrum: As regras do jogo. Brazil: 2011. Disponível em: <http://www.scrum.org/Scrum-Guides>. Acesso em: 21 maio 2013.

SILVA, F. G.; HOENTSCH, Sandra CP; SILVA, Leila. **Uma análise das Metodologias Ágeis FDD e Scrum sob a Perspectiva do Modelo de Qualidade MPS. BR**. Scientia Plena, v. 5, n. 12, 2011.

STAPLETON, J. **DSDM: Business Focused Development**. second . ed. Great Britain: Addison-Weley, 2003. 233p.

TELES, V. M. **Um estudo de caso da adoção das práticas e valores do Extreme Programming**. 2005. 173f. Dissertação (Mestrado em Informática) - Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

TELES, V. M. **Extreme Programming**. São Paulo: Novatec, 2004. 306p.

TOMÁS, M. R. S. **Métodos ágeis: características, pontos fortes e fracos e possibilidades de aplicação**. 2009. 17f. Dissertação (Mestrado em Fatores Sociais da Inovação) - Faculdade de Ciências e Tecnologia da Universidade Nova Lisboa, Portugal, 2009.

APÊNDICE A – ARTEFATOS ELICITAÇÃO DE REQUISITOS

US04	**Prioridade: Média**
Como coordenador eu desejo que o sistema seja automaticamente	
fechado após a data final de envio de monografias para que não seja	
mais possível fazer o envio.	

US05	**Prioridade: Alta**
Como professor orientador eu devo enviar as monografias dos meus	
orientandos para que os membros da banca possam visualizá-las e	
baixá-las.	

US06	**Prioridade: Média**
Como coordenador de monografia eu devo associar os professores às	
monografias enviadas para formar as bancas de avaliação.	

US07 **Prioridade: Alta**
Como professor eu quero visualizar e baixar os trabalhos dos quais faço
parte da banca de avaliação para poder avaliá-los.

US08 **Prioridade: Média**
Como coordenador de monografias eu quero que após a avaliação das
monografias seja enviado um email aos professores e alunos para que
possam visualizar as notas e comentários sobre as monografias

US09	**Prioridade: Alta**
Como coordenador de monografia eu desejo que o sistema seja	
web para facilitar o envio de monografias	

US10	**Prioridade: Média**
Como coordenador de monografias eu preciso que o sistema ofereça	
mecanismos de segurança para autenticação de usuário para que o	
acesso ao sistema seja feito apenas por usuários cadastrados	

US11	**Prioridade: Média**
Como coordenador de monografias eu quero que o sistema tenha uma	
interface simples, amigável e intuitiva para que o acesso ao mesmo seja	
facilitado	

APÊNDICE B – ARTEFATOS ANÁLISE DE REQUISITOS

Classe: Aluno	
Responsabilidades	**Colaboração**
Enviar monografia ao orientador	Professor

Classe: Banca	
Responsabilidades	**Colaboração**
Visualizar monografia	Professor
Baixar monografia	Coordenador

Classe: Controladora	
Responsabilidades	**Colaboração**
Enviar email	Professor
Fechar sistema	Coordenador
Calcular a média de cada monografia	
Exibir lista contendo o nome, nota e	
Comentários sobre as monografias	

APÊNDICE C – ARTEFATOS VALIDAÇÃO DE REQUISITOS

US04 – Critérios para Aceitação
- A opção de envio de monografias na tela do orientador deve ficar
desabilitada assim que acabar o período de envio

Exemplo de um cenário a ser testado

User Story	**Função que será testada**	**Estado inicial do sistema**	**Entrada**	**Saída esperada**
US04	Enviar monografia	Finalizado o período de envio de monografias	Username e senha do professor, menu enviar monografia	Após a data limite estabelecida não é possível enviar monografia

US05 – Critérios para Aceitação
- Para enviar monografias o professor deve escolher o nome do
orientando, uma área de pesquisa previamente cadastrada no sistema,
descrever o título e resumo da monografia e anexar um arquivo pdf.

Exemplo de um cenário a ser testado

User Story	**Função que será testada**	**Estado inicial do sistema**	**Entrada**	**Saída esperada**
US05	Anexar monografia	Função de enviar monografia foi selecionada e foi anexado um documento .doc	Nome do orientando, seleção da área de pesquisa, descrição do título e resumo da monografia	Aviso: “Não é possível anexar documentos que não sejam .pdf”

US06 – Critérios para Aceitação
- A banca de avaliação será composta por três professores, sendo um o orientador
- Para formar a banca de avaliação o coordenador deve escolher mais dois
professores que tenham conhecimento na área de pesquisa da
monografia
- Deve ser gerado um email para avisar os membros da banca de avaliação
sobre informações do trabalho e data limite de avaliação

Exemplo de um cenário a ser testado

User Story	**Função que será testada**	**Estado inicial do sistema**	**Entrada**	**Saída esperada**
US06	Formar banca de avaliação	Função de formar banca de avaliação foi selecionada	Título e área de pesquisa da monografia	Professores escolhidos para a banca são avisados

US07 – Critérios para Aceitação
- O professor só poderá baixar os trabalhos dos quais faça parte da
banca de avaliação e de seus orientandos

Exemplo de um cenário a ser testado

User Story	**Função que será testada**	**Estado inicial do sistema**	**Entrada**	**Saída esperada**
US07	Baixar monografia	Função de baixar monografia foi selecionada. Professor selecionou uma monografia a qual não faz parte da banca	Visualização da lista de monografias	Aviso: “Não é possível baixar monografia da qual não faça parte da banca de avaliação”