UNIVERSIDADE ESTADUAL DO SUDOETE BAHIA – UESB
CURSO DE CIÊNCIAS DA COMPUTAÇÃO
MATHEUS NASCIMENTO SALES

ESTRATÉGIA PARA NORTEAR O PROCESSO DE ENGENHARIA DE REQUISITOS APLICADA À METODOLOGIA ÁGIL SCRUM (PER-SCRUM)

VITÓRIA DA CONQUISTA-BA
2020

MATHEUS NASCIMENTO SALES

ESTRATÉGIA PARA NORTEAR O PROCESSO DE ENGENHARIA DE REQUISITOS APLICADA À METODOLOGIA ÁGIL SCRUM (PER-SCRUM)

Trabalho de Conclusão de Curso apresentado ao curso de Ciência da Computação da Universidade Estadual do Sudoeste da Bahia (UESB) como parte dos requisitos necessários à obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Francisco dos Santos Carvalho.

VITÓRIA DA CONQUISTA-BA

2020

AGRADECIMENTOS

Gostaria de agradecer aos meus pais por proporcionarem toda condição para que eu pudesse chegar até aqui, aos meus irmãos por me darem o incentivo e exemplo para que eu prosperasse na graduação, a Sávylla por estar comigo desde meados da minha graduação e me incentivar em tudo para que eu chegasse nas minhas conquistas. Gostaria de agradecer tambem aos amigos que fiz no curso: Gustavo, Dalton, Lucas, Raphael, Pablo, Stefanie, Marcelo, Konai, Rodrigo, Badaró, a todos da turma de 2015.1 e todos os nossos veteranos (Iago, Yan, todos os Matheus) e calouros (em especial a Jorge e Jenifer) que tiveram a paciência de ajudar naqueles trabalhos cabeludos e fizeram parte da nossa resenha.

Aos professores e servidores técnico-administrativos lotados no Colegiado de Curso de Ciência da computação (CCCOMP) – Celina você é um anjo – e no Departamento de Ciências Exatas e Tecnológicas (DCET) da Universidade Estadual do Sudoeste da Bahia (UESB).

Ao professor Francisco que aceitou e me deu total apoio para que que este trabalho monográfico chegasse ao fim. Mesmo com a pandemia realizamos reuniões online, o que favoreceu para o adiantamento do TCC, deixo aqui meus agradecimentos.

RESUMO

As metodologias ágeis têm ganhado espaço nas organizações públicas e privadas, notadamente a partir da década de 1990. Trata-se de um modelo de desenvolvimento que valoriza um amplo conjunto de princípios e práticas que visam, por exemplo, gerar rapidez, integração entre os membros da equipe, envolvimento dos clientes e outras partes interessadas, buscando atender a um contexto produtivo caracterizado pelas mudanças constantes nos requisitos de produtos. Nessa perspectiva, são recomendáveis pesquisas acadêmicas que tenham como propósito aprimorar as fases que compõem o ciclo de vida de desenvolvimento de um software.Este trabalho monográfico teve como objetivo geralapresentar uma proposta para nortear o processo de engenharia de requisitos aplicada à metodologia ágil Scrum. Para tanto, foram estabelecidos os seguintes objetivos específicos: a) Descrever baseado na literatura especializada o processo de engenharia de requisitos utilizado em modelos tradicionais de desenvolvimento de software; b) Descrever a metodologia ágil Scrum, com ênfase nos seus princípios e práticas, na estruturação dos membros da equipe Scrum,nos eventos existentes enos artefatos gerados;c) Criar modelos no Draw.io para descrever os fluxos de atividades propostas para o processo de Engenharia de Requisitos aplicado ao Scrum; d)Propor critérios para priorização dos requisitos; e) Apresentar exemplo de pontuação das atividades constantes nos fluxos propostos. Quanto aos procedimentos metodológicos, utilizou-se de pesquisas do tipo básica, exploratória e descritiva, com abordagem qualitativa e realização de revisão bibliográfica, que serviram de base para conduzir à elaboração da proposta em questão. Ao final do trabalho, pôde-se concluir que a problemática que motivou este estudo foi analisada, e que os objetivos geral e específicos foram atingidos. Construiu-se uma proposta que poderá ser aplicada em trabalhos futuros em organizações públicas e/ou privadas.

PALAVRAS-CHAVE: Desenvolvimento ágil. Engenharia de Requisitos.Scrum. Modelos.

ABSTRACT

Agile methodologies have been getting space on public and private organizations, especially since the 1990s. It is a development model that valorize a wide set of principles and practices that aim, for example, to generate speed, integration between the team members, involvement of clientes and other stakeholders, seeking to meet a productive context characterized by changes in products requirements.In this perspective, academic research is recommended with the purpose of improving the phases that make up the software development life cycle.This monographic work had the general objective of presenting a proposal to guide the requirements engineering process applied to the Scrum methodology.This monographic work aimed to present a proposal to guide the requirements engineering process applied to the Scrum agile methodology. To this end, the following specific objectives were established:a) Describe based on the specialized literature the requirements engineering process used in traditional software development models; b) Describe the agile Scrum methodologie, with emphasis on its principles and practices, on the structuring of Scrum team members, on the existing events and on the artifacts generated; c) Criate models on Drow.io to describe proposed activity flows for the Requirements Engineering process applied to Scrum. d) Propose criteria for prioritizing requirements; e) Present an example of scoring the activities in the activity flows.As for the methodological procedures, basic, exploratory and descriptive research was used, with a qualitative approach and a bibliographic review, which served as a basis to lead to the preparation of the proposal in question. At the end of the work, it was possible to conclude that the problem that motivated this study was analyzed, and that the general and specific objectives were achieved. A proposal was built that can be applied in future works in public and/or private organizations.

KEY WORDS: Agile Development, Requirements Engineering, models.

LISTA DE SIGLAS E ABREVIATURAS

UESB – Universidade Estadual do Sudoeste da Bahia.

FDD – Feature Driven Development.

MSF – Microsoft Solution Framework.

XP – eXtreme Programming.

DSDM - Dynamic System Development Method.

LISTA DE FIGURAS

Figura 1 – Modelo espiral do processo de engenharia de requisitos. 19
Figura 2 – Diagrama de caso de uso para o sistema CasaSegura. 23
Figura 3 – Gerenciamento de mudanças de requisitos. 26
Figura 4 – Fluxo do processo Scrum. 29
Figura 5 – Fases e Artefatos do FDD. 30
Figura 6 – Uma representação do modelo do MSF. 31
Figura 7 – Atividades do ciclo de vida do software. 38
Figura 8 – Processo de Engenharia de Requisitos. 39
Figura 9 – Equipe do SCRUM. 41
Figura 10 – Atuação do Proprietário do Produto. 43
Figura 11 – Visão analítica do papel do Proprietário do Produto. 44
Figura 12 – Atividades propostas para o Scrum Master. 47
Figura 13 – Eventos utilizando metodologia ágil. 48
Figura 14 – Fluxo proposto abrangendo detalhes dos eventos do Scrum. 51
Figura 15 – Atuação do proprietário do produto (resumido). 61

LISTA DE TABELAS

Tabela 1 –Etapas e atividades do Estudo de Caso 36
Tabela 2 - Matriz de risco 3x3 56
Tabela 3 – Matriz de risco 5x5 56
Tabela 4 – Tipos de riscos 57
Tabela 5 – Histórias e pontuações 58
Tabela 6 – Pontuação para as interações e histórias 59
Tabela 7 –Pontuação em 3 pontos 60
Tabela 8 – tabela de pontuação das atividades do fluxo 62

SUMÁRIO

1 INTRODUÇÃO 10

1.1 Contextualização e motivação 10
1.2 Problema de pesquisa 11
1.3 Objetivo geral 11
1.4 Objetivos específicos 12
1.5 Justificativa 12
1.6 Estrutura dos capítulos da monografia 12

2 ESTADO DA ARTE: TRABALHOS RELACIONADOS 14

3 REVISÃO DA LITERATURA 17

3.1 Processo de Engenharia de Requisitos: uma visão das etapas que normalmente são aplicadas ao processo de desenvolvimento de software 17
3.1.1 Elicitação de Requisitos 20
3.1.2 Análise de Requisitos 23
3.1.3 Especificação de Requisitos 24
3.1.4 Validação de Requisitos 25
3.1.5 Gerenciamento de Requisitos 25

3.2 Desenvolvimento ágil de software 26

3.2.1 XP (Extreme Programming) 27
3.2.2 SCRUM 28
3.2.3 FDD (Feature Driven Development) 29
3.2.4 MSF (Microsoft Solution Framework) 30
3.2.5 DSDM (Dynamic System Development Method) 31
3.2.6 TDD (Test Driven Development) 32
3.2.7 BDD (Behavior Driven Development) 33

4 METODOLOGIA 35

4.1 Tipos de pesquisas quanto aos objetivos 35
4.2 Tipos de pesquisa quanto a abordagem 35
4.3 Tipos de pesquisas quanto aos procedimentos 36
4.4 Etapas de pesquisa 36
4.5 Instrumentos de pesquisa 36
4.6 Coleta de análise de informações 37

5 APRESENTAÇÃO DA PROPOSTA PER-SCRUM 38

5.1 Framework proposto: PER-SCRUM 38
5.2 Ciclo de vida do Software 38
5.3 Composição da Equipe Scrum 41
5.4 Eventos da metodologia Scrum 48
5.5 Estratégias de pontuações para o cálculo de riscos 53
5.5.1 Primeira estratégia 56
5.5.2 Segunda estratégia 58
5.5.3 Terceira estratégia 60
5.6 Exemplo de aplicação: Pontuação dos fluxos 61

6 CONCLUSÃO 62

CAPÍTULO 1 – INTRODUÇÃO

Neste capítulo é apresentado uma breve contextualização sobre o motivo que levou ao problema de pesquisa, os objetivos geral e específicos, justificativa e a forma como foi estruturada os capítulos desta monografia.

1.1 Contextualização e motivação

Atualmente, as empresas e organizações de desenvolvimento de software estão cada vez mais interessadas em acelerar a produção dos projetos sem perder de vista a qualidade dos mesmos. A engenharia de requisitos tem forte impacto sobre a qualidade final da construção de softwares. Muitos dos projetos são prejudicados quando a engenharia de requisitos não é cumprida da forma eficiente (SOMMERVILLE, 2007). Dessa forma, é importante que os engenheiros de softwares conheçam a real necessidade do cliente para a proposição e desenvolvimento com metodologias eficientes.

Pesquisas recentes indicam que, quanto mais tarde ocorra a detecção de erros no sistema, mais caro e mais complexo torna-se a resolução do problema (JALOTE, 2005). Por esse motivo, é importante que, durante o processo de desenvolvimento de software, cada fase seja seguida com cautela. No entanto, se um dos requisitos especificados não for cumprido, poderá haver prejuízos quantitativos ou qualitativos.

Na literatura especializada, constata-se que o processo de Engenharia de Requisitos envolve fases fundamentais. Na percepção de Pressman e Roger (2006), estão presentes nesse processo as fases de elicitação, análise e negociação, especificação e validação dos requisitos. Estas fases podem ocorrer de forma sequencial ou simultaneamente.

Geralmente, o processo de Engenharia de Requisitos, contém um número expressivo de atividades que poderão ser eliminadas em parte, com a proposição de uma metodologia que contemple as etapas eliminadas.

Observa-se na literatura relacionada ao objeto de estudo, a necessidade de pesquisas para integrar o processo de engenharia de requisitos aos princípios de metodologias ágeis para o desenvolvimento de software. Assim, é importante a

proposição de princípios metodológicos voltados para simplificar o tradicional processo de engenharia requisitos.

Entre as principais metodologias ágeis que podem ser aplicadas em um projeto de desenvolvimento de software, destacam-se: Programação extrema (*XP, eXtreme Programming*), Scrum, desenvolvimento dirigido a funcionalidades (*FDD*, *Feature Driven Development*), Framework de solução da Microsoft (MSF, *Microsoft Solution Framework*) e Método de desenvolvimento de sistemas dinâmicos (DSDM, *Dynamic System Development Method*). É importante lembrar que não há a melhor metodologia, mas a solução mais adequada dentro do contexto de uma organização.

Os princípios fundamentais dessas metodologias são: economicidade de tempo e recursos, simplicidade, interação com o cliente e projetos enxutos.

Os princípios da economicidade de tempo e recursos no desenvolvimento de software com uso das metodologias ágeis provocam a melhoria na participação de empresas nos processos competitivos que requeiram uma entrega dos produtos em menos tempo. Assim, torna-se pertinente aplicar essas metodologias na engenharia de requisitos, com o propósito de reduzir o tempo para o desenvolvimento do software.

Com o intuito de contribuir para a melhoria da etapa de engenharia de requisitos no desenvolvimento de software, este trabalho propõe uma estratégia para contribuir com o processo de Engenharia de Requisitos na Metodologia SCRUM.

1.2 Problema da pesquisa

Como propor uma estratégia capaz de representar e avaliar os fluxos de atividades da metodologia SCRUM, incorporando critérios para priorização dos requisitos?

1.3 Objetivo geral

Propor uma estratégia capaz representar e avaliar os fluxos de atividades da metodologia SCRUM, incorporando critérios para priorização dos requisitos.

1.4 Objetivos específicos

- Descrever à luz da literatura especializada o processo de engenharia de requisitos.
- Descrever a metodologia ágil Scrum, com ênfase nos seus princípios e práticas, na estruturação dos membros da equipe Scrum, nos eventos existentes e nos artefatos gerados.
- Criar modelos no Draw.io para descrever os fluxos de atividades propostos para o processo de Engenharia de Requisitos aplicado ao Scrum.
- Propor critérios para priorização dos requisitos.
- Apresentar exemplo de avaliação (pontuação) das atividades nos fluxos de atividades propostos na presente estratégia.

1.5 Justificativa

Este trabalho tem relevância acadêmica e econômica. No âmbito acadêmico, trata-se de uma proposta inédita que terá como base a realização e melhorias contínuas propostas com o escopo abrangente para cobrir todo o ciclo de vida do desenvolvimento de software, com foco na etapa de engenharia de requisitos. No âmbito econômico, a redução de tempo e custo ganha relevância em um contexto onde as organizações produtivas buscam redução das suas despesas relacionadas a criação de produtos de software.

1.6 Estrutura dos capítulos da monografia

Além deste Capítulo 1 – Introdução, este trabalho monográfico possui outros capítulos, a saber:

Capítulo 2 – Estado da Arte: Trabalhos Relacionados. Este capítulo apresenta a descrição de alguns trabalhos utilizados como objeto de pesquisa.

Capítulo 3 – Revisão da Literatura. Faz uma apresentação do processo de Engenharia de Requisitos e descreve algumas metodologias ágeis.

Capítulo 4 – Metodologia. Descreve os processos metodológicos utilizados para a realização da pesquisa desta monografia.

Capítulo 5 – Apresentação da proposta PER-SCRUM. Este capítulo descreve, com base nos conceitos das pesquisas, uma proposta de framework capaz de representar a avaliar os fluxos de atividades da metodologia SCRUM.

Capítulo 6 – Conclusão. Capítulo que descreve as conclusões tomadas com base nas pesquisas realizadas

CAPÍTULO 2 - ESTADO DA ARTE: TRABALHOS RELACIONADOS

Este Capítulo apresenta uma breve descrição de alguns trabalhos relacionados como objeto de pesquisa desta Monografia.

Um importante trabalho foi desenvolvido por Santos (2018), intitulado “REACT: uma abordagem ágil de apoio ao processo de desenvolvimento de requisitos de software baseada em evidências empíricas”. Considerou-se que a adoção e a criação de novas abordagens ágeis na indústria de software tem obtido significado crescimento nas últimas décadas em função da busca por minimizar alguns problemas das abordagens tradicionais para desenvolvimento de software, notadamente no que se refere ao processo de Engenharia de Requisitos. Nessa perspectiva, Santos (2018) apresentou na sua dissertação de Mestrado o método REACT (Requirements Evolution in Agile Context), uma proposta de métodologia ágil para o desenvolvimento dos requisitos de um produto de software, consdierando os modelos de qualidade MR-MPS-SW e CMMI-DEV.

Daneva et al. (2015 apud SANTOS, 2018) anotou que no Agile-RE as atividades do processo de Engenharia de Requisitos são executadas de modo simultâneo, ou seja, duas ou mais atividades do processo podem ocorrer ao mesmo tempo. A Agile-RE é diferente da Engenharia de Requisitos Tradicional quanto a: aceitação de mudanças nos requisitos durante o ciclo de desenvolvimento; produção simplificada de documentações; evolução e detalhamento dos requisitos ao longo de curtas iterações de desenvolvimento (QUSEF et al., 2010; PAETSH et al., 2003 apud SANTOS, (2018).

O trabalho de Santos (2018) apontou ainda a existência de desafios na Agile-RE, a exemplo dos identificados mediante mapeamentos ou revisões sistemáticas da literatura, realizadas por Medeiros et al. (2015), Inayat et al. (2014) e Jaqueira et al. (2012). Exemplos desses desafios: Pouca disponibilidade do cliente; Requisitos não-funcionais são negligenciados; Controle de mudanças nos requisitos ineficiente; Rastreabilidade de requisitos ineficiente; User stories são inadequadas para descrever aspectos técnicos; Consenso na negociação ou priorização dos requisitos; Documentação geralmente mínima; ineficiência na análise e inspeção dos requisitos; Comunicação entre o cliente e a equipe; Custos e estimativas do projeto devido às mudanças constantes nos requisitos.

Muitos métodos e práticas ágeis têm requerido pesquisas com o objetivo de gerar outras abordagens adaptadas e fundamentadas nos mesmos. Geralmente, a fim de preencher algum gap científico ou desafio deixado pelos próprios métodos ágeis atuais (MEDEIROS, 2017). Exemplos disto são as abordagens: ScrumS, uma adaptação do Scrum para área de Safe Agile Development (ESTEVES et al., 2015); SobA (Story based Agile software development), uma metodologia de desenvolvimento centrada em user stories (KUMAR et al., 2008); e a Brew Model, uma técnica ágil para a priorização de requisitos (NEGRÃO; GUERRA, 2011). Ademais, é importante lembrar, também, das abordagens criadas para o contexto de organizações de grande porte, principalmente, tendo o Scrum como fundamento, tais como: o SAFe (Scaled Agile Framework) (LEFFINGWELL et al., 2018) e o LeSS (Large-Scale Scrum) (LARMAN; VODDE, 2016).

O estudo conduzido por Medeiros (2017) ressalta que uma SRS (Software Requirements Specification) ruim age como catalizador de outros problemas ao longo do projeto, principalmente, em contextos ágeis. Os autores inferem que em ambientes ágeis as SRS são, geralmente, superficiais, insuficientes e inadequadas. Neste cenário, a autora propôs uma abordagem ágil para SRS, a qual se utiliza de um conjunto de boas práticas da Agile-RE como Agile Modeling, Protótipos de telas e Cenários de aceitação a fim de fornecer uma SRS ágil com foco na equipe de desenvolvimento. Segundo a autora, a proposta foi aplicada na indústria de software por meio de um estudo de caso.

Similarmente, o estudo conduzido por Boness e Harrison (2007) teve como objetivo propor uma técnica ágil para a elicitação de requisitos, a qual foi criada com base nos conceitos e práticas da GORE (Goal Oriented Requirements Engineering) e dos métodos ágeis – a Goal Sketching. A ideia é de gerar um diagrama composto de diferentes tipos de metas, as quais guiassem a elicitação de requisitos de forma iterativa e incremental. A técnica, segundo os autores, foi aplicada e validada em três projetos reais de desenvolvimento de software.

Por fim, o estudo conduzido por Racheva et al (2010) teve como objetivo aplicar um modelo conceitual para priorização ágil de requisitos. O modelo foi validado por meio de um estudo de caso, o qual contou com a participação de diversos profissionais de companhias de desenvolvimento de software da Holanda, Itália, Bulgária e Turquia. É importante dizer que todas estas abordagens ágeis, foram validadas empiricamente

na indústria de software. Igualmente, todas foram aplicadas em ambientes ágeis, ou seja, em projetos ou equipes que utilizam métodos ágeis para desenvolver software.

CAPÍTULO 3 - REVISÃO DA LITERATURA

Neste capítulo é detalhado o processo de Engenharia de Requisitos com sua estrutura e funcionamento de cada etapa. Logo após é descrito o desenvolvimento ágil de software e as metodologias que foram desenvolvidas baseadas em princípios e valores.

3.1 Processo de engenharia de requisitos: uma visão das etapas que normalmente são aplicadas ao processo de desenvolvimento de software

Mesmo com o avanço da tecnologia nos últimos anos, problemas quantitativos e qualitativos ainda persistem nos projetos de software. Os principais problemas durante o desenvolvimento de um software estão associados aos requisitos, cerca de 40% a 60% dos problemas que ocorrem nos projetos de software são causados por falhas na fase de levantamento destes (LEFFINGWELL, 1997). Constata-se quando as etapas que compõem o ciclo de vida do processo de engenharia de requisitos são bem realizadas, os requisitos implícitos são minimizados, consequentemente os desenvolvedores podem compreendê-los melhor (PAULA FILHO, 2003).

Segundo o Standish Group (2009) no Chaos Report, aproximadamente 36% dos principais fatores de falhas em projetos de software são derivados de erros associados aos requisitos, implicado em última análise na qualidade final do software.

Em pesquisa realizada por Schwaber (2007) em mais de cem projetos de desenvolvimento de software, identificou-se que 35% dos requisitos sofreram alterações, 65% das funcionalidades descritas pelos requisitos nunca ou raramente foram implementadas e que em torno de 50% do tempo do projeto foi usado no processo de Engenharia de Requisitos, definição e implantação de arquitetura de software e especificação do produto.

Quanto mais tarde se descobre um erro em requisitos, mais caro torna para fazer sua correção. Quando ele é descoberto depois do processo de implementação, seu custo de correção é de até 20 vezes mais do que se fosse descoberto em qualquer outra fase no processo de desenvolvimento do software (BOEHM et al. 2002).

Face ao exposto, é plenamente justificáveis pesquisas que associem a área de Engenharia de Requisitos com os métodos ágeis. No entendimento de Santos (2018),

essa associação supracitada, conhecida como Engenharia de Requisitos Ágil (Agile-RE - Agile Requirements Engineering), tem sido objetivo das investigações de diversos pesquisadores, a exemplo de Daneva et al. (2015), Heikkila et al. (2015), Inayat et al. (2014), Cao e Ramesh (2008), Leffingwell et al. (2018).

Ademais, ressalta-se que nem todos os métodos ágeis aplicados ao desenvolvimento de software possuem foco no Processo de Engenharia de Requisitos. Geralmente, implementar as atividades de Engenharia de Requisitos mediante métodos ágeis é grande desafio (INAYAT et al., 2014; BJARNASON et al., 2011).

Na literatura são encontrados vários termos que definem requisitos. Para Young (2004), requisito é um atributo necessário em um sistema que ajuda identificar a capacidade, características ou a qualidade deste para ter um valor ou utilidade para o usuário. É importante porque fornece a base para todo o trabalho de desenvolvimento que deve ser seguido.

Mesmo que muitas definições de requisitos destaquem as necessidades dos usuários finais, ao defini-los deve-se levar em consideração todos aqueles que participam do sistema, ou seja, aqueles que interagem direta ou indiretamente com o projeto de software: clientes, gerente de projeto, analistas de sistema, usuários finais, desenvolvedores, etc. (KOTONYA; SOMMERVILLE, 1998).

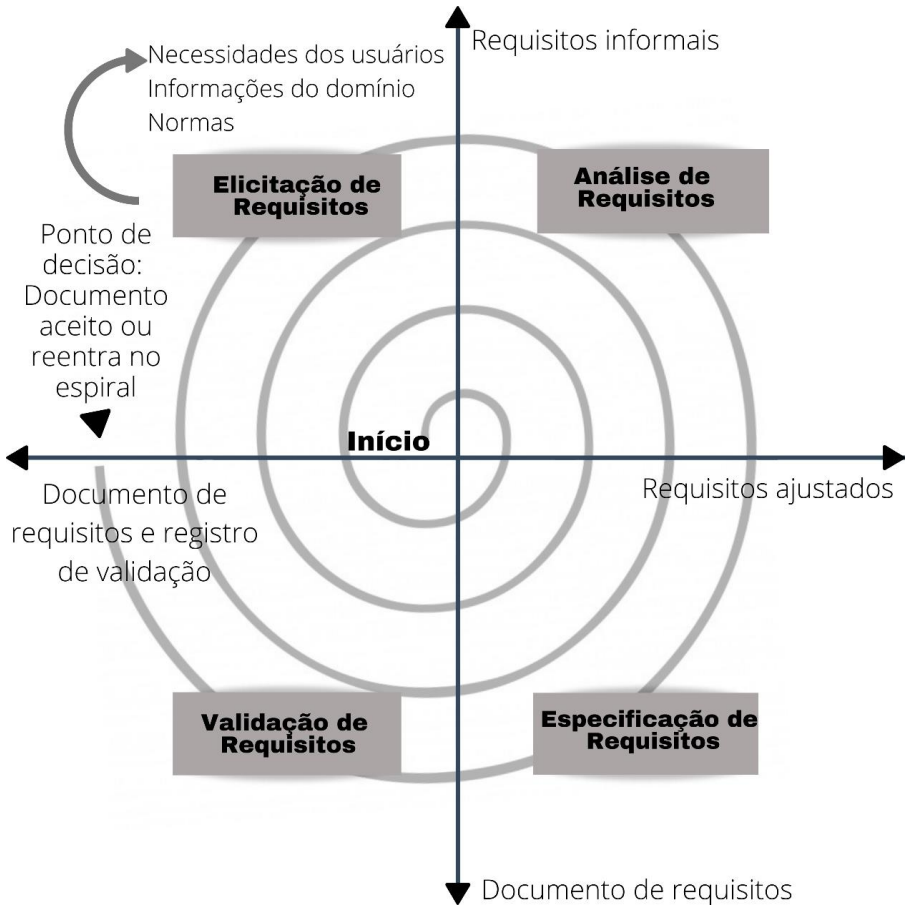
Para desenvolver um software é necessário um conjunto de ações para compreender as expectativas do cliente, analisar e especificar, gerenciar, verificar e validar os requisitos (YOUNG, 2004). Sommerville (2007) diz que essas atividades são chamadas de processo de engenharia de requisitos, no qual, as mesmas, durante o desenvolvimento do software, podem ser realizadas de maneira interativa.

Ressalta-se que há várias propostas para as atividades que podem compor o processo de Engenharia de Requisitos. Nesta parte da monografia, apresenta-se uma visão inicial deste processo.

Por exemplo, a Figura 1 ilustra a interação das atividades no processo de Engenharia de Requisitos. No começo do processo os engenheiros de software trabalham com os clientes e usuários finais para descobrirem suas necessidades. Finalizando essa atividade, alguns requisitos são levantados de maneira informal. Seguindo o espiral, começa a atividade de análise de requisitos, que é o processo onde os requisitos serão analisados e ajustados. Assim, é iniciada a atividade de

especificação dos requisitos onde os requisitos ajustados passam a compor um documento de requisitos mais elaborado. Na última atividade, este documento será verificado e possivelmente validado, pois, se ele não for aceito, será realizado um novo ciclo no espiral. Mudanças ou interrupções nos requisitos podem ocorrer desde a atividade de elicitação até durante a operação do sistema, a atividade de gerenciamento de requisitos fica responsável por essa atuação. Até que o engenheiro de requisitos e o cliente aceitem o documento de especificação de requisitos, as atividades serão repetidas em ciclos.

Figura 1- Modelo espiral do processo de engenharia de requisitos



Fonte: Adaptado de Sawyer e Kotonia, 2001.

Para Sommerville e Sawyer (1997), o termo “engenharia” acarreta na necessidade de utilizar técnicas, práticas e conceitos para garantir a completude, consistência e relevância dos requisitos. Desse modo. Carvalho e Chiossi (2001) definem engenharia de requisitos como um processo que transforma idéias de usuários (entrada) em um documento (saída), que fornece a todos os participantes uma descrição do que o produto deverá fazer sem, no entanto, informar como deve

ser feito. Isso pode ser desenvolvido por meio de modelos de análise ou especificação, cenários de usuários ou listas de funções e de características.

A engenharia de requisitos propicia aos engenheiros de software um método para entender e analisar as necessidades dos usuários para que assim possam proporcionar soluções para a construção do software (THAYER; DORFMAN, 1997).

3.1.1 Elicitação de Requisitos

Fase do processo que dedica-se à obtenção dos requisitos para o sistema. É quando os engenheiros de software e analistas de sistemas reúnem-se com os usuários finais e clientes, a fim de obter um entendimento do problema e conhecer as funcionalidades do sistema a ser desenvolvido (SOMMERVILLE, 2011).

Requisitos surgem da inter-relação entre usuários e analistas, e é dessa interação que nascem os requisitos para a construção do software (SIDDIQI,1997).

Além de perguntar ao usuário suas necessidades e definir um escopo do problema, esta atividade propõe um estudo da organização onde o software será implantado, uma análise dos processos de negócios onde o software será utilizado e do domínio da aplicação (KOTONYA; SOMMERVILLE, 1998).

Davis (1993) descreve essa atividade relacionando ao aprendizado do problema a ser resolvido, das restrições de hardware e software, identificando corretamente os usuários e suas necessidades com o propósito de decidir um custo concreto do ponto de vista do negócio.

Algumas técnicas para elicitação de requisitos podem ser usadas em combinação com o intuito de evitar impasses nessa atividade, por exemplo: Entrevistas, Análise de Documentos, Reunião de Brainstorming, Workshop, Protótipos, Diagrama de Caso de Uso, Análise de Interfaces, Desenvolvimento de Aplicação Articulada (JAD, *Joint Application Development*) e Análise de Desempenho e Capacidade (KOTONYA; SOMMERVILLE, 1998; SOMMERVILLE, 2011; YOUNG, 2004).

- *Entrevistas*

De acordo com Sommerville (2011), nessas entrevistas, a equipe de engenharia de requisitos coloca perguntas às partes interessadas sobre o sistema que eles usam atualmente e o sistema a ser desenvolvido. Os requisitos são derivados das respostas a essas perguntas. Para que a entrevista não fique dispersa e não se torne cansativa, é necessário um plano de entrevista. As entrevistas podem ser de dois tipos:

‘ - Entrevistas fechadas, nas quais a parte interessada responde a um conjunto pré-definido de perguntas.

- Entrevistas abertas, nas quais não há agenda predefinida. A equipe de engenharia de requisitos explora uma série de problemas com as partes interessadas do sistema e, portanto, desenvolve uma melhor compreensão de suas necessidades.

A vantagem da entrevista é que o desenvolvedor pode obter uma vasta informação sobre o sistema desejado. A desvantagem é que pode haver informações discrepantes entre as partes interessadas (MENDONÇA, 2014).

- *Brainstorming*

Essa técnica é utilizada como forma de obter uma “tempestade” de ideias, ou seja, o cliente e os engenheiros de requisitos fazem reuniões para coletarem e discutirem essas ideias para que o problema possa ser melhor compreendido, cooperando assim para atingir o objetivo (MENDONÇA, 2014).

- *Joint Application Desingn (JAD)*

Criada pela IBM, esta técnica baseia-se no trabalho em grupo que, através de reuniões, os engenheiros de software auxiliam os clientes a saberem formular algum problema e investigar uma possível solução. Através dessa colaboração, o JAD facilita o entendimento compartilhado do que o produto deve ser (MENDONÇA, 2014).

O JAD engloba quatro princípios básicos (VIANNA, 2019):

1. Ninguém é melhor para explicar um determinado processo do que as pessoas que trabalham com ele.

2. Os profissionais de TI são os mais preparados para identificar as possibilidades que a tecnologia oferece, assim como suas limitações.
3. Sistemas de informação e processos do negócio não são isolados.
4. Os melhores sistemas de informação são resultado do trabalho conjunto de todas as pessoas envolvidas: profissionais de TI, usuários, gestores, analistas de negócio, facilitadores etc.

Vantagens: facilita o reparo do entendimento de pontos controversos do sistema através da disseminação de diversos pontos de vista dos participantes do sistema (MARTINS, 2001).

Desvantagens: usuários operativos podem ter pontos de vista que geram conflitos com os da gerência. A sessão pode ser muito longa se os conflitos forem de difícil solução. O facilitador deve ser alguém experiente na resolução de conflitos (MARTINS, 2001).

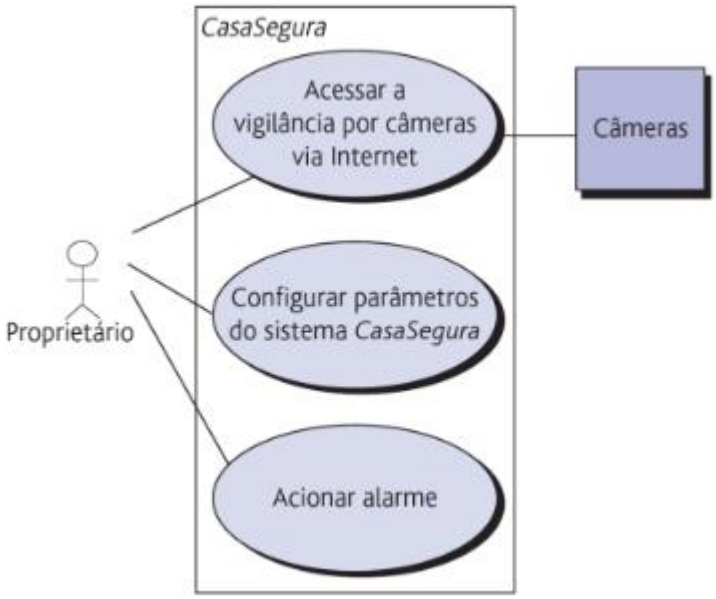
- Casos de Uso

Um caso de uso é uma imagem das ações que um sistema executa, representando os atores (SCHNEIDER et al, 2001). Deve ser acompanhado por uma descrição textual e não deve ser usado isoladamente de outras técnicas de coleta de requisitos. Os casos de uso devem sempre ser complementados com atributos de qualidade e outras informações, como características da interface. Muitos desenvolvedores acreditam que casos de uso facilitam a comunicação da equipe. Eles fornecem um contexto para os requisitos, expressando sequências de eventos e um idioma comum para os usuários finais e a equipe técnica. Os casos de uso por si só não fornecem informações suficientes para permitir atividades de desenvolvimento. Outras técnicas de elicitação de requisitos também devem ser usadas em conjunto.

A UML define o caso de uso como um conjunto de sequências de ações que será realizado pelo sistema, gerando um resultado observável de um ator específico (ERICSSON; PENKER, 2000).

A Figura 2 representa um diagrama de caso de uso preliminar para o produto *CasaSegura* (PRESSMAN; MAXIM, 2016).

Figura 2 - Diagrama de caso de uso para o sistema CasaSegura



Fonte: Pressman e Maxim, 2016.

Uma breve descrição deve ser feita ao caso de uso para um melhor entendimento da responsabilidade de cada ator com cada atividade do sistema (SOMMERVILLE, 2011). A seguir, mostra um exemplo da descrição do diagrama: O proprietário da casa tem a responsabilidade de poder interagir com todas as atividades disponíveis pelo sistema. A câmera tem a responsabilidade de gerar saída de vídeo para o usuário.

3.1.2 Análise de requisitos

Análise de requisitos é uma atividade que está ligada à elicitação, tendo em vista que, conforme os requisitos vão sendo descobertos, algum grau de análise sobre eles deve ser realizado. Alguns requisitos incompatíveis, vagos e faltantes são identificados, por esse motivo, esta atividade tem como objetivo encontrar possíveis problemas causados na atividade anterior (KOTONYA; SOMMERVILLE, 1998).

Durante a análise, alguns elementos sobre os requisitos auxiliam na detecção de problemas (MARTINS, 2001), a exemplo de:

- Verificação de Necessidade: quando é necessário a análise de um requisito obtido. Alguns requisitos propostos podem não contribuir para o problema específico.

- Verificação de consistência e completitude: consistência quer dizer que nenhum requisito deve entrar em contradição e completitude significa que nenhum serviço ou restrição que sejam necessários ao sistema tenha sido omitido.

- Verificação de Possibilidade: a verificação dos requisitos deve assegurar que eles são possíveis de serem implementados, tanto em custo temporal quanto financeiro.

3.1.3 Especificação de Requisitos

Essa é a atividade onde serão produzidos os documentos que organizarão os requisitos sistema a ser desenvolvido. Turine e Masiero (1996) afirma que, para evitar futuros problemas durante a implementação do software, é importante que este documento seja organizado de maneira que os requisitos sejam fáceis de ler e compreender. Requisitos e particularidades do sistema devem estar descritos de maneira precisa, completa e verificável (IEEE, 1990).

Kotonya e Sommerville (1998) relatam que a única forma de compreensão entre todos aqueles que participam do processo de requisitos é a linguagem natural. Porém, vale destacar que, apesar da vantagem de todos poderem compreender a documentação, esta pode apresentar um alto grau de ambiguidade, que favorece o surgimento de inconsistências (DAVIS, 1993).

O documento supracitado pode ser escrito através da combinação de: Modelos Matemáticos, Linguagem Natural, Diagramas, Tabelas, Modelos de interface e de Características (PRESSMAN, 2006).

Segundo Sommerville (2011), um documento de requisitos é constituído por um conjunto de usuários, que abrange desde a gerência sênior da organização que até os engenheiros responsáveis pelo desenvolvimento do software.

Isso posto, Sommerville (2011), classificou os usuários do documento de requisitos em:

- Clientes do sistema: lêem e especificam os requisitos do sistema para avaliar se os mesmos atendem suas necessidades.

- Gerentes de projeto: usam o documento de requisitos para planejar uma proposta para o sistema e planejam seu processo de desenvolvimento.

- Engenheiros do sistema: utilizam os requisitos para entenderem o sistema em desenvolvimento.

- Engenheiros de teste do sistema: usam os requisitos para desenvolverem testes de validação do sistema.

- Engenheiros de manutenção do sistema: usam os requisitos para entenderem o sistema e a relação entre as partes.

Damian (2000) alega que, pelo fato desse documento ser utilizado para informar a finalidade do software, aqueles que defendem a apresentação formal alegam que trazem vantagem de comunicação com os desenvolvedores do sistema, quando se trata de ausência de ambiguidade, bom entendimento do projeto e reconhecimento de abstrações fundamentais ao sistema.

Contudo, na literatura especializada identifica-se outras classificações, ora mais resumidas, ora mais abrangentes.

3.1.4 Validação dos Requisitos

Pressman (2006) afirma que a validação de requisitos envolve tanto clientes quanto desenvolvedores e é uma maneira de identificar se os requisitos gerados e documentados estejam plausíveis com as necessidades daqueles que irão fazer uso do sistema.

Essa atividade de validação está preocupada em evitar erros em um documento de requisitos, que podem gerar custos de retrabalho quando esses problemas são descobertos durante o desenvolvimento ou após o sistema estar em serviço (SOMMERVILLE, 2011).

A validação de requisitos pode ser aplicada em outras versões de documento geradas nas atividades intermediárias do processo de engenharia de requisitos e não somente na última versão (PRESSMAN, 2006).

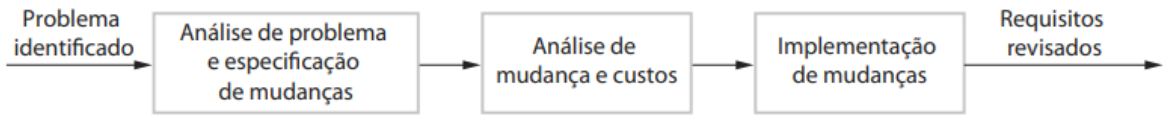
3.1.5 Gerenciamento de Requisitos

O gerenciamento de requisitos é o processo de entendimento e controle de alterações nos requisitos do sistema. É necessário acompanhar os requisitos individuais e manter os links entre os requisitos dependentes, para poder avaliar o

impacto das alterações nos requisitos. É preciso estabelecer um processo formal para fazer propostas de alteração e vinculá-las aos requisitos do sistema. O processo formal de gerenciamento de requisitos deve começar assim que uma versão preliminar do documento de requisitos estiver disponível. No entanto, você deve começar a planejar como gerenciar requisitos alterados durante o processo de obtenção de requisitos (SOMMERVILLE, 2011).

Os requisitos podem sofrer mudanças em qualquer atividade do processo de engenharia de requisitos. Kotonya e Sommerville (1998) afirmam que essas mudanças são inevitáveis e, mesmo assim, não significa que o processo escolhido tenha sido falho. As mudanças nos requisitos são feitas de maneira controlada, para isso, existem atividade como mostra a Figura 3.

Figura 3 - Gerenciamento de mudanças de requisitos



Fonte: Sommerville, 2011.

O processo tem início na identificação do problema. Problema identificado ou mudança requerida tem sua viabilidade analisada e eventuais dúvidas são tiradas junto ao seu solicitante. Logo após, é feita análise dos efeitos da mudança junto a outros requisitos e são estimados os custos dessa mudança no projeto.

Completadas essas análises ocorrerá a decisão se a alteração prosseguirá ou quais negociações serão necessárias para sua aprovação. Caso a decisão da modificação seja aceita é feita uma atualização nos documentos de requisitos e encaminhada para a equipe de desenvolvimento (SOMMERVILLE, 2007).

3.2 Desenvolvimento ágil de software

Desenvolvimento ágil é um conjunto de metodologias desenvolvidas baseadas em princípios e valores que tem como objetivo produzir softwares com qualidade em menor tempo com a finalidade de atender o mercado relacionado no desenvolvimento de software com rapidez (FOWLER, 2001).

Segundo o supracitado autor, a utilização dos métodos tradicionais, o sistema deve ser bem planejado e detalhado com muita antecedência, o que pode gerar dificuldades para mudanças nos seus requisitos após implementá-los. Como forma de flexibilizar e adaptar-se a mudanças nos requisitos durante o processo de desenvolvimento do softwawre, os métodos ágeis foram criados.

Fowler (2001) ainda afirmou que as metodologias ágeis são orientadas a pessoas, ou seja, requerem experiência e competência das pessoas e priorização das comunicações entre os membros da equipe de desenvolvimento, cliente e outras partes interessadas.

A seguir, são apresentadas algumas metodologias ágeis:

3.2.1 XP (Extreme Programming)

A programação extrema (XP) é talvez o método mais conhecido e amplamente utilizado dos métodos ágeis. Leva este nome porque a abordagem foi desenvolvida levando as boas práticas reconhecidas, como o desenvolvimento iterativo, a níveis "extremos". Por exemplo, no XP, várias novas versões de um sistema podem ser desenvolvidas por diferentes programadores, integradas e testadas em um dia (SOMMERVILLE, 2011).

De acordo com Mendonça (2014), o XP baseia-se em quatro princípios básicos:

1. Comunicação: que mantém uma melhor relação com o cliente através de conversas pessoais, evitando qualquer outro método menos eficiente.
2. Simplicidade: O sistema não deve possuir funções desnecessárias, visa a criação de códigos simples.
3. Feedback: Busca sempre atualizar o cliente quando alguma unidade do sistema fica pronto.
4. Coragem: Os desenvolvedores estão sempre pedindo mais detalhes do sistema para o cliente e determinam um esforço suficiente para desenvolvê-la.

3.2.2 SCRUM

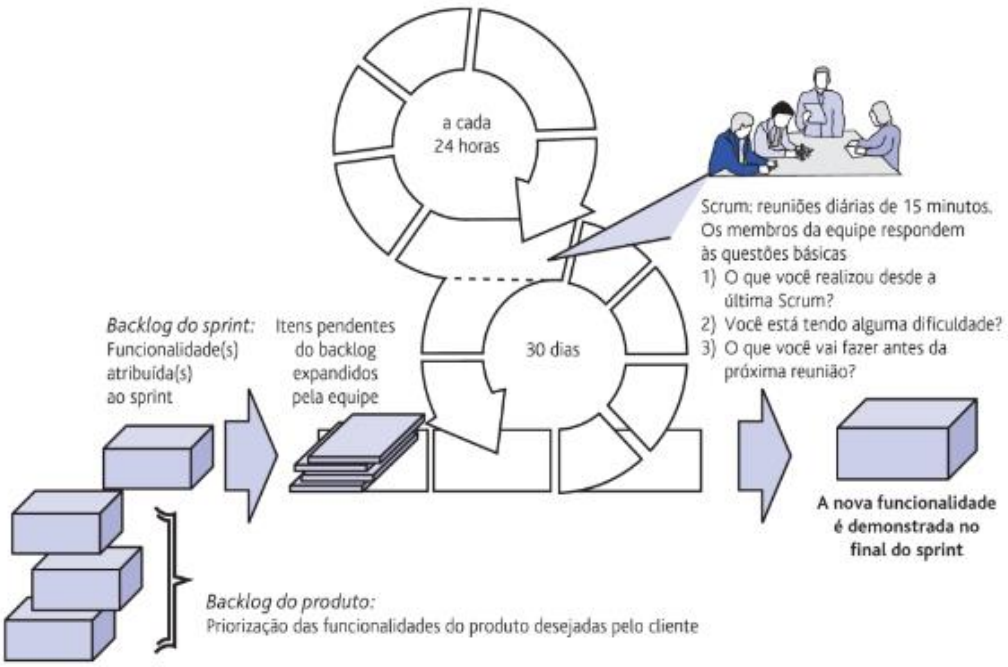
O Scrum é um processo iterativo e incremental cujos princípios são utilizados para orientar atividades com o intuito de produzir um serviço de qualidade em um ambiente passível de modificações (SOMMERVILLE, 2011)

Sommerville (2011) anotou três fases para o Scrum. A primeira é uma fase básica de planejamento, na qual você estabelece os objetivos gerais do projeto e projeta a arquitetura do software. Na sequência é feito uma série de ciclos de sprint, em que cada ciclo desenvolve uma funcionalidade do sistema. Finalmente, a fase de encerramento finaliza o projeto, completa a documentação necessária, como quadros de ajuda do sistema e manuais do usuário, e avalia as lições aprendidas com o projeto.

O recurso inovador do Scrum é sua fase central, ou seja, os ciclos de sprint. Um Scrum Sprint é uma unidade de planejamento na qual o trabalho a ser feito é avaliado, os recursos são selecionados para o desenvolvimento e o software é implementado. No final de um sprint, a funcionalidade concluída é entregue às partes interessadas (SOMMERVILLE, 2011).

Os três procedimentos principais do Scrum são backlog, Sprints e reuniões scrums. Todos os requisitos necessários são registrados no backlog e contém todas as funcionalidades melhorias e bugs do sistema. Um sprint tem duração de 30 dias de desenvolvimento, e é baseada nas informações do backlog. Durante uma Sprint a equipe passa por várias reuniões que servirão como base para o planejamento do próximo sprint. Se um sprint relacionado a um backlog está sendo desenvolvido, nenhum requisito desse mesmo backlog poderá ser modificado até que esse sprint acabe, porém ainda há flexibilidade para mudanças de requisitos do próximo sprint a ser desenvolvido (MARTINS, 2001). O fluxo do processo Scrum é mostrado na Figura 4.

Figura 4 - Fluxo do processo Scrum



Fonte: Pressman e Maxim, 2016.

3.2.3. FDD (Feature Driven Development)

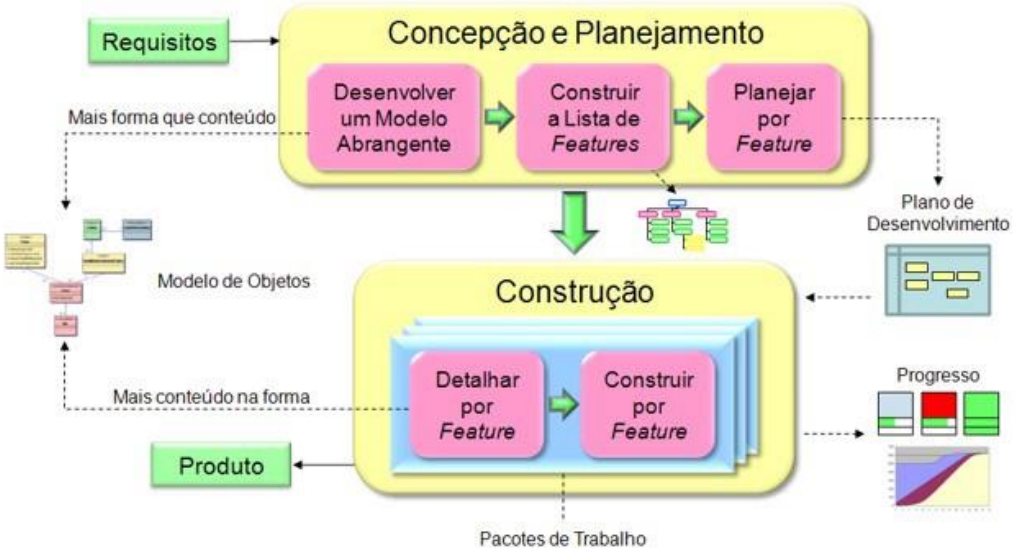
O Feature Driven Development é uma metodologia ágil voltada para a conquista das principais partes interessadas em um projeto de software: clientes, gerentes e desenvolvedores. É levantada uma lista de funcionalidades a serem planejadas de forma incremental, seguindo o planejamento é necessário fazer o refinamento, este tem que estar o mais de acordo possível com o que o cliente deseja, consequentemente mantém-se contato contínuo com o time de clientes (NOGUEIRA, 2016).

Essa metodologia surgiu em oposição aos métodos disponíveis na época, os quais focavam na entrega exaustiva de documentações (BECK et al.,2001). Por esse motivo, o FDD tem como princípio valorizar o software em funcionamento, pois isso é considerado mais importante do que produzir massantes documentações completas de um software (HIGHSMITH; COCKBURN, 2001).

Essa metodologia, assim como as outras, é muito concreta, possuindo somente duas fases que duram normalmente duas semanas. São elas: concepção & planejamento e construção. A primeira é composta por 3 etapas: desenvolver um

modelo abrangente, construir a lista de funcionalidade e planejar por funcionalidade. A segunda é composta de duas etapas: Detalhar a funcionalidade e Construir por funcionalidade (NOGUEIRA, 2016).

A figura 5 mostra as fases e os artefatos propostos pelo FDD:



Fonte: www.medium.com.

3.2.4 Framework de solução da Microsoft (MSF, *Microsoft Solution Framework*)

O MSF é uma abordagem eficaz de desenvolvimento de soluções de software. Esta metodologia foi introduzida pela primeira vez em 1994 como um conjunto de boas práticas para um efetivo desenvolvimento de produtos da Microsoft, bem como as atividades dos serviçoes de consultoria da Microsoft. Depois de apresentar o MSF, a Microsoft reuniu o feedback de seus grupos de desenvolvimento de software, parceiros e clientes em todo o mundo. Como resultado, o MSF foi aprimorado e evoluído e, depois de publicar várias versões, a Microsoft introduziu as personalizações do MSF (PAVLOV, 2007).

Para Wilson (2005), esta metodologia ágil fornece orientação para pessoas e processos para ajudar equipes e organizações a terem mais sucesso na entrega de soluções de tecnologia de negócios a seus clientes. É uma abordagem deliberada e disciplinada de projetos de tecnologiaa baseada em em um conjunto de princípios, modelos, disciplinas, conceitos, diretrizes e práticas comprovadas pela Microsoft.

A figura 6 uma representação do modelo do MSF.

Fonte: Adaptado de Practicesoftware, 2020.

No centro do MSF está o modelo de processo, o qual possui um ciclo de vida de cinco estágios para a solução. Entre cada estágio há requisitos claros para justificar o contínuo custo de desenvolvimento.

A visão deve ser uma única área de trabalho global, mas o escopo pode impor restrições a isso. Wilson (2005) afirma que, a visão é uma das áreas mais importantes de qualquer implantação de área de trabalhoe tambem uma das mais negligenciadas, geralmente porque o cliente não consegue ver o valor em um planejamento inicial e precisa ser visto para entregar algo ao negócio o mais rápido possível.

3.2.5 DSDM (Dynamic System Development Method)

O método de desenvolvimento de sistemas dinâmicos é uma metodologia de desenvolvimento ágil que é utilizada para construir e manter sistemas que, em um

ambiente de projeto controlado, cumpram limitações de prazo reduzido por meio de uso de prototipação incremental (PRESSMAN; MAXIM, 2016).

Essa metodologia baseia-se na regra dos 80%, isso significa que somente será desenvolvido o suficiente daquilo que é requisitado para que assim possa passar para o próximo incremento. Os outros 20% podem ser desenvolvidos mais tarde, quando todos os requisitos forem reconhecidos e suas possíveis alterações terem sido finalizadas (PRESSMAN; MAXIM, 2016).

O DSDM é composto por três princípios de iteração:

1. Iteração de modelos funcionais: um conjunto de protótipos que demonstram funcionalidades para o cliente são produzidos.
2. Iteração de projeto e desenvolvimento: os protótipos desenvolvidos durante a iteração anterior são revistos para garantir que sejam capazes de oferecer aos usuários valor de negócio em termos operacionais.
3. Implementação: coloca a última versão do incremento de software no ambiente operacional.

3.2.6 TDD (Test Driven Development)

De acordo com Beck (2002), o desenvolvimento dirigido por testes (TDD, Test Driven Development), é uma metodologia que que transmite segurança aos desenvolvedores por possuírem um agrupamento de técnicas que impulsionam o desenvolvimento de simples projetos e um agrupamento de teste que promove confiança. As companhias que aderem a esta metodologia se atentam a duas regras básicas: a primeira regra diz que, antes que algum código seja escrito, deve-se escrever algum teste automatizado que leva a falha e a segunda regra é que qualquer algoritmo duplicado deve ser removido. Estas regras simbolizam o TDD que possui três termos, Vermelho, Verde e Refatoração, chamados por Beck (2002) de filosofia da metodologia. O mesmo autor ainda define os termos: Vermelho – é necessário que seja escrito um pequeno teste que gere falha, Verde – faça com o que o reste funcione rapidamente e Refatoração – é necessária a remoção da duplicação para fazer o teste ser aprovado.

Dentre as vatagens de utilizar o FDD podemos citar a frequente entrega de feedbacks da fase em que se encontra o código, da convicção para mudar já que as

aplicações finalizadas possuem testes, caso uma alteração reflita em algo que já está pronto o teste irá retificar se aquilo que estava pronto continua funcionando e enquanto ainda há aprovação dos testes não há o dever de continuar desenvolvendo.

3.2.7 BDD (Behavior Driven Development)

O Desenvolvimento Dirigido a Comportamento (BDD, Behavior Driven Development) é uma metodologia desenvolvida para testar e elaborar uma aplicação de acordo como o cliente descreveu, sem que, de início, seja necessária a utilização de uma linguagem técnica. Dessa forma o foco se resume ao comportamento e não na forma como o projeto vai ser implementado.

A criação do BDD se deu ao fato do desenvolvimento de software costumar ter trabalho em excesso causada pela dificuldade de comunicação entre os engenheiros de software e o cliente (KUDRYASHOV, 2015). Muitas vezes os profissionais de desenvolvimento entendem mal o que o cliente realmente precisa, o que gera muita demanda até chegarem em um senso comum.

De acordo com Kudryashov (2015), pelo fato dos desenvolvedores utilizarem de termos técnicos para se comunicarem com os usuários, estes não conseguiam entender sobre as funcionalidades do sistema. Por esse motivo foi proposta uma metodologia que facilitasse essa comunicação para que qualquer informação pudesse ser compartilhada entre os indivíduos durante desenvolvimento do projeto.

Para Kudryashov (2015) estes são os principais benefícios do BDD:

1- Rastreamento direto do trabalho de desenvolvimento aos objetivos de negócio.
2- Usuários satisfeitos, resultando em bons negócios.
3- Uma ótima comunicação entre todas as partes envolvidas no projeto.
4- Código sofisticado com baixo custo de manutenção e menor risco de erro no desenvolvimento.

Neste capítulo foram discutidos referenciais teóricos no que diz respeito a Engenharia de Requisitos e as respectivas etapas pertencentes, além da descrição de algumas Metodologias Ágeis.

Esta pesquisa permitiu entender o quão são importantes as etapas da engenharia de requisitos em um projeto de software. É necessário aprimorar o entendimento dos

requisitos antes de dar início ao processo de desenvolvimento. Este entendimento a respeito das reais necessidades do cliente é exatamente o maior desafio na engenharia de requisitos.

Não existe uma metodologia única que possa ser utilizada por todas as empresas que produzem software. Cada entidade deve designar sua própria metodologia de trabalho, visando a adequação da mesma em relação aos tipos de softwares em desenvolvimento, cultura organizacional, experiência dos funcionários e assim por diante.

Este estudo tem como objetivo expandir a compreensão dos conceitos que envolvem o processo de Engenharia de Requisitos e Metodologias Ágeis de forma que permita que, as ponderações por essa monografia apresentadas, tenham todo embasamento técnico para que possa chegar na sua finalidade.

CAPÍTULO 4 – METODOLOGIA

Os procedimentos metodológicos escolhidos para nortear a realização da presente investigação, foram baseados no paradigma construtivista. Destacar-se-á a seguir, a natureza e o paradigma da pesquisa, os principais instrumentos para construção da estratégia proposta para o processo de engenharia de requisitos (PER-SCRUM).

4.1 Tipos de pesquisas quanto aos objetivos

Em uma primeira etapa da investigação monográfica, foi preciso fazer uso da pesquisa exploratória com o propósito de aproximar o pesquisador do objeto de estudo, considerando a problemática que considerada desafiadora, posto que se trata de um objeto de estudo bem recente na variante estratégica escolhida para compor o PER-SCRUM. Assim, foram necessárias consultas bibliográficas, livros, teses, dissertações, monografias, artigos e outras fontes secundárias.

Para descrever as características da estratégia proposta no PER-SCRUM, tornou-se necessária uma pesquisa descritiva. Segundo Gil (2010) esse tipo de pesquisa tem como finalidade descrever as características do objeto de estudo, especificando as características das variáveis nele envolvidas. Essa junto ora apresentada nos passos metadológicos, permite um planejamento bem flexível e, geralmente, se transforma num estudo de caso ou pesquisa bibliográfica. Por isso, segundo Granda (2015), é pertinente essa correlação de pesquisas, especialmente para trazer novas luzes sobre tema desafiador e ainda muito recente.

4.2 Tipos de pesquisas quanto à abordagem

Foi escolhida abordagem qualitativa, que serviu para analisar aspectos subjetivos relacionados ao objetivo de estudo. Segundo Chizzotti (1995, p. 79) afirmou:

> A abordagem qualitativa parte do fundamento de que há dinâmica entre o mundo real e o sujeito, uma interdependência entre o sujeito e o objeto, um vínculo indissociável entre o mundo objetivo e a subjetividade do sujeito. O conhecimento não se reduz a um rol de dados isolados, conectados por uma teoria explicativa; o sujeito–observador é parte integrante do processo de conhecimento e interpreta os fenômenos, atribuindo–lhes um significado.

4.3 Tipos de pesquisas quanto aos procedimentos

Construir a estratégia do PER-SCRUM requereu, portanto, uma abordagem qualitativa, com procedimentos que perpassaram pelas pesquisas bibliográfica, de levantamento e pelo estudo de caso com entrevista com membros do projeto GED-UESB.

4.4 Etapas da pesquisa

Tabela 1 – Etapas e atividades do Estudo de Caso

Etapas	Atividades
1 – Seleção do caso a ser estudado (Reflexão)	- Escolha do tema - Escolha das questões de pesquisa - Seleção de perspectivas paradigmáticas
2 – Planejamento do Estudo de Caso	- Seleção do contexto - Seleção de estratégia (qualitativa) - Realização de pesquisas preliminares - Escrita preliminar da metodologia (projeto)
3 – Condução do Estudo de Caso	- Seleção das fontes de pesquisa - Coleta de dados e informações – Modelagem usando BPMN na ferramenta Draw.io.
4 – Estabelecimento de conexões de dados e informações	- Análise preliminar de dados e informações
5 – Escrita do caso	- Análise intensiva dos dados e informações
6 – Apresentação dos resultados	- Redação final do caso

Fonte: Adaptado de Carvalho, 2012.

4.5 Instrumentos de pesquisa

Este trabalho monográfico utilizou dos seguintes instrumentos de pesquisa:

a) Software Draw.io modelagem organizacional.
b) Entrevista estruturada como os desenvolvedores do Sistema GED da UESB;
c) Uso de abordagem adaptada do SCAMPI.

4.6 Coleta e análise de informações

As etapas de coleta e análise de informações seguiram os requisitos propostos na Norma IEEE 830/98 que dispõe sobre a Especificação de Requisitos para Desenvolvimento de Software. Considerando-se o escopo do trabalho voltado apenas para a fase inicial de modelagem, as informações coletas foram baseadas na proposta do PER-SCRUM, visando aprimorar compreensão, visibilidade e construção do Processo de Engenharia de Requisitos no escolhido método ágil. Para tanto, fez-se uso das técnicas de observação e entrevista utilizadas que resultaram nos modelos gerados e na estratégia PER-SCRUM.

CAPÍTULO 5 – APRESENTAÇÃO DA PROPOSTA PER-SCRUM

Este capítulo descreve, com base nos conceitos sobre Engenharia de Requisitos e metodologias ágeis, uma proposta de um framework capaz de representar e avaliar fluxos de atividades da metodologia SCRUM, incorporando critérios para priorização dos requisitos. Esta proposta foi desenvolvida em consideração às características apontadas na problemática da pesquisa.

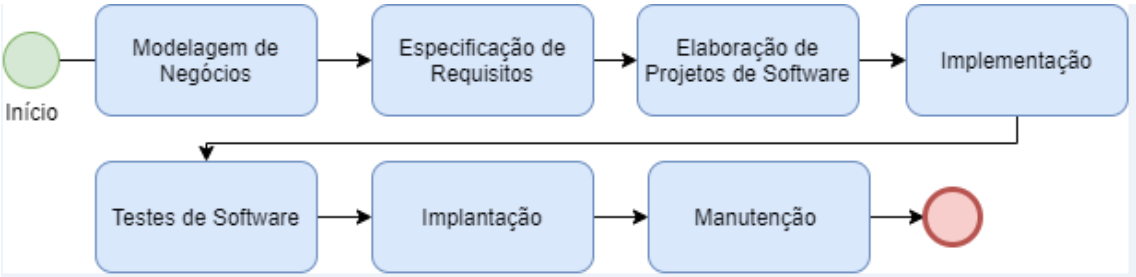
5.1 Framework proposto: PER-SCRUM

A proposta de Framework deste trabalho é composta por um amplo conjunto de fluxos de atividades com respectiva descrição. Desse modo, nas subseções seguintes, serão apresentadas visões (geral e analítica) desses fluxos, com o propósito dos leitores deste trabalho compreenderem o funcionamento e como poderão ser simplificadas as atividades do Processo de Engenharia de Requisitos no desenvolvimento de software baseado em metodologias ágeis.

5.2 Ciclo de vida do Software

Inicialmente, é exibida na Figura 7 o Ciclo de Vida do Software, composto por sete etapas.

Figura 7 – Atividades do ciclo de vida do software



Fonte: Autoria própria, 2020.

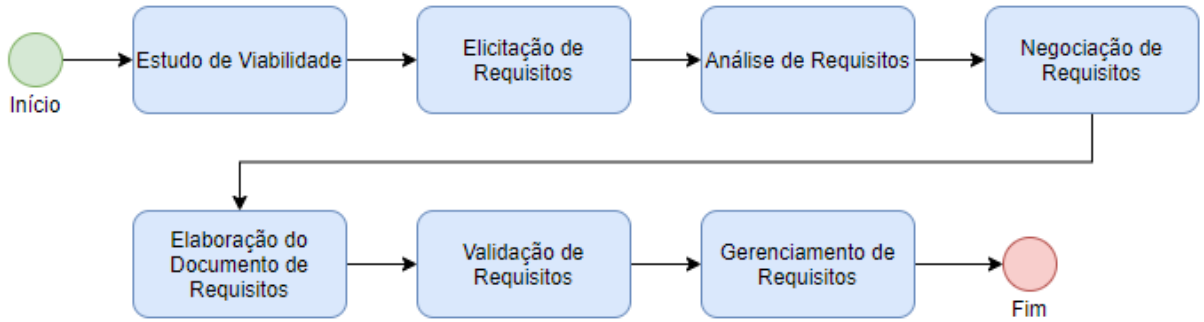
Observa-se na Figura 7 que esse fluxo é padrão, recomendado por institutos e pesquisadores da área de Engenharia de Software.

Segundo Sommerville (2011), a norma ISO/IEC 12207 apresenta o processo de ciclo de vida do software. Este modelo da ISO/IEC 12207 é composto por três categorias, quais sejam: processos fundamentais (aquisição, desenvolvimento e fornecimento), processos organizacionais (visam auxiliar na organização do desenvolvimento do software) e os processos de apoio (suporte ao desenvolvimento).

A Especificação de Requisitos é a segunda fase do Ciclo de Vida exibido na Figura 7, objeto deste presente.

Na literatura especializada, o processo de Engenharia de Requisitos, geralmente, é composto por um quantitativo médio entre 4 a 10 fases. No caso em questão, foi escolhido uma representação desse processo composta por sete fases, conforme mostra a Figura 8.

Figura 8 – Processo de Engenharia de Requisitos



Fonte: Autoria própria, 2020.

Estudo de viabilidade, como o próprio nome já diz, tem como objetivo avaliar se o sistema é viável sob o ponto de vista organizacional da empresa desenvolvedora, financeiro, técnico, entre outros. De acordo com Sommerville (2007), é a etapa que decide se o gasto com tempo e esforço para o sistema, valerá a pena ou não.

Elicitação de Requisitos é a fase do projeto em que o cliente informa o que realmente deseja que seja construído. Nas metodologias ágeis, a elicitação é feita com clientes que fazem parte do desenvolvimento. Para esse fim, são fornecidos, por exemplo, cartões para escreverem histórias de usuário (userstories), que servirão como artefatos de saída para a finalização dessa etapa (JAQUEIRA, 2013). Dessa maneira, os clientes poderão acompanhar o desenvolvimento do software adequando

os requisitos do sistema mais significativos em tempo de construção. Assim, o sistema final vai condizer ao que realmente era para ser desenvolvido (AMBLER, 2002).

Análise de Requisitos é uma atividade que visa o entendimento das necessidades do cliente ou usuário. Para Pressman e Maxim (2016, p.167) “[..] resulta na especificação das características operacionais do software, indica a interface do software com outros elementos do sistema e estabelece restrições a que o software deve atender.”. Ou seja, é uma atividade que visa fazer a ligação entre a alocação do software em nível de sistema, o projeto de software e as necessidades do usuário. Além disso, esta etapa tem a capacidade de ampliar os requisitos básicos levantados nas etapas anteriores da engenharia de requisitos.

Negociação de Requisitos trata de fazer um contrabalanço nas funcionalidades, desempenho ou outras características do sistema, em função do custo e prazo para a entrega do produto. Para Pressman e Maxim (2016), o objetivo da negociação é desenvolver uma estratégia de projeto que supra as necessidades do cliente e, simultaneamente, considere as restrições do mundo real para o desenvolvimento do sistema. Desse modo, é possível que os dois lados entrem em consenso: o cliente que vai ter a maioria das suas necessidades implementadas e o desenvolvedor que vai cumprir com prazos e custos estipulados anteriormente.

Elaboração de Documento de Requisitos dispõe de um artifício que serve como meio de comunicação entre o usuário e o projetista de software. Segundo Turine (1996, p. 7), “O documento de requisitos de um software contém todos os requisitos funcionais e de qualidade do software, incluindo as capacidades do produto, os recursos disponíveis, os benefícios e os critérios de aceitação.”. Sendo assim, é importante evitar que decisões de projeto sejam tomadas durante o progresso do documento de requisitos.

Validação de Requisitos tem o propósito de detectar inconsistências relacionadas às fases anteriores do processo. De acordo com Moretto (2018), esta etapa tem como objetivo identificar ambiguidades ou erros que não assegurem a qualidade no documento de requisitos e consequentemente uma má implementação. Toda equipe de revisão busca erros no conteúdo ou interpretação, falta de informação, requisitos conflitantes ou irreais que ainda existiam no documento (PRESSMAN; MAXIM, 2016).

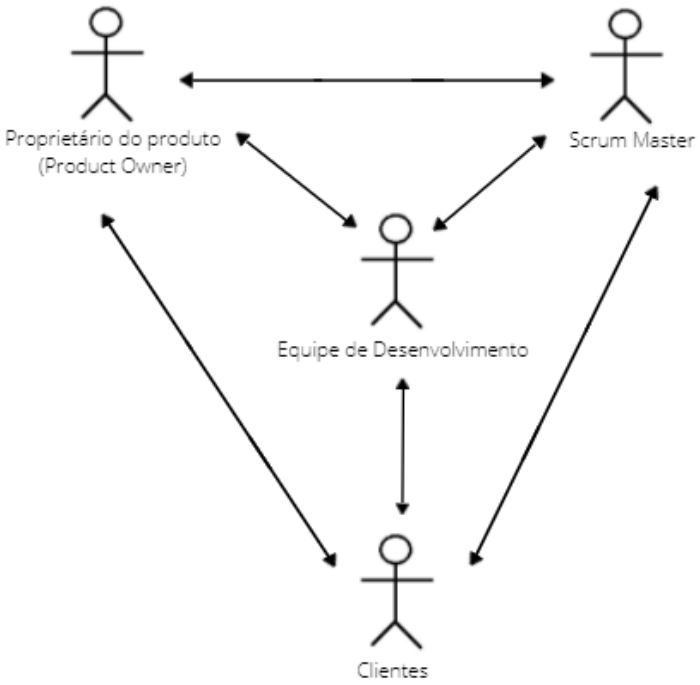
Para Pressman e Maxim (2016), o Gerenciamento de Requisitos tem a finalidade de ajudar a equipe de projeto a identificar, controlar e acompanhar o processo de mudança de requisitos na medida que o desenvolvimento do projeto prossegue. Essas mudanças ocorrem de acordo com que os clientes vão entendendo suas reais necessidades.

5.3 Composição da Equipe Scrum

A Figura 9 exibe uma representação sumarizada da composição da Equipe SCRUM.

Estão presentes o Proprietário do Produto, o Scrum Master, a Equipe de Desenvolvimento e os Clientes. Todavia, sabe-se que entre as Partes Interessadas, outros atores podem estar presentes, a exemplo de stakeholders que podem não atuar diretamente com o Sistema em desenvolvimento. Cita-se, por exemplo, gestores de outras unidades organizacionais e parceiros da organização que possam utilizar o sistema.

Figura 9 – Equipe do SCRUM



Fonte: Autoria própria, 2020

Papel do *ProductOwner*

Este componente da Equipe do Scrum, O Product Owner, é responsável pelo retorno dos investimentos (ROI, *Return on Investments*) do projeto, o que significa que ele escolhe qual funcionalidade do produto vai ser desenvolvida para resolver problemas críticos de negócios.

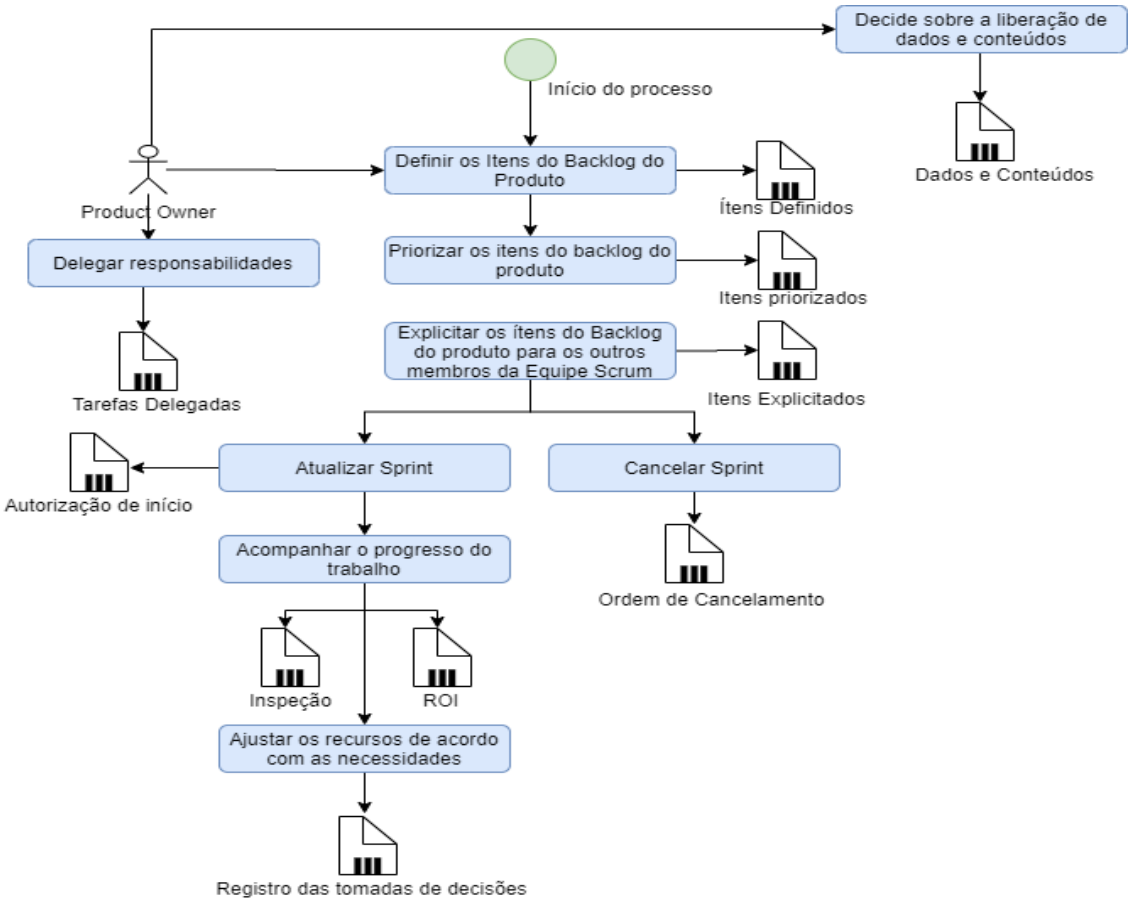
O Product Owner usa o product Backlog para dar a mais alta prioridade aos requisitos de maior valor para os negócios, para inserir requisitos não funcionais que levam a liberações e implementações oportunistas de funcionalidade e para ajustar constantemente o produto em resposta a mudanças nas condições de negócios, incluindo novas ofertas competitivas (SCHWABER, 2004).

Pichler (2010) ainda destaca a relevância do ProductOwner: O ProductOwner tem autoridade para definir um objetivo e modelar a visão. Ele não é apenas um gerente de produto que agora redige requisitos e define algumas prioridades, ele tem o comprometimento de gerar uma visão objetiva para qualquer envolvido a fim de assegurar que o time está no caminho certo.

O ProductOwner tem essas responsabilidades mencionadas, podendo executá-las totalmente ou parte delas, uma vez que o mesmo também poderá delegá-las para a equipe de desenvolvimento (SCHWABER, 2004)

Na figura 10, foi proposto uma representação contendo atividades que, geralmente, podem ser realizadas pelo Proprietário do Produto.

Figura 10 – Atuação do Proprietário do Produto

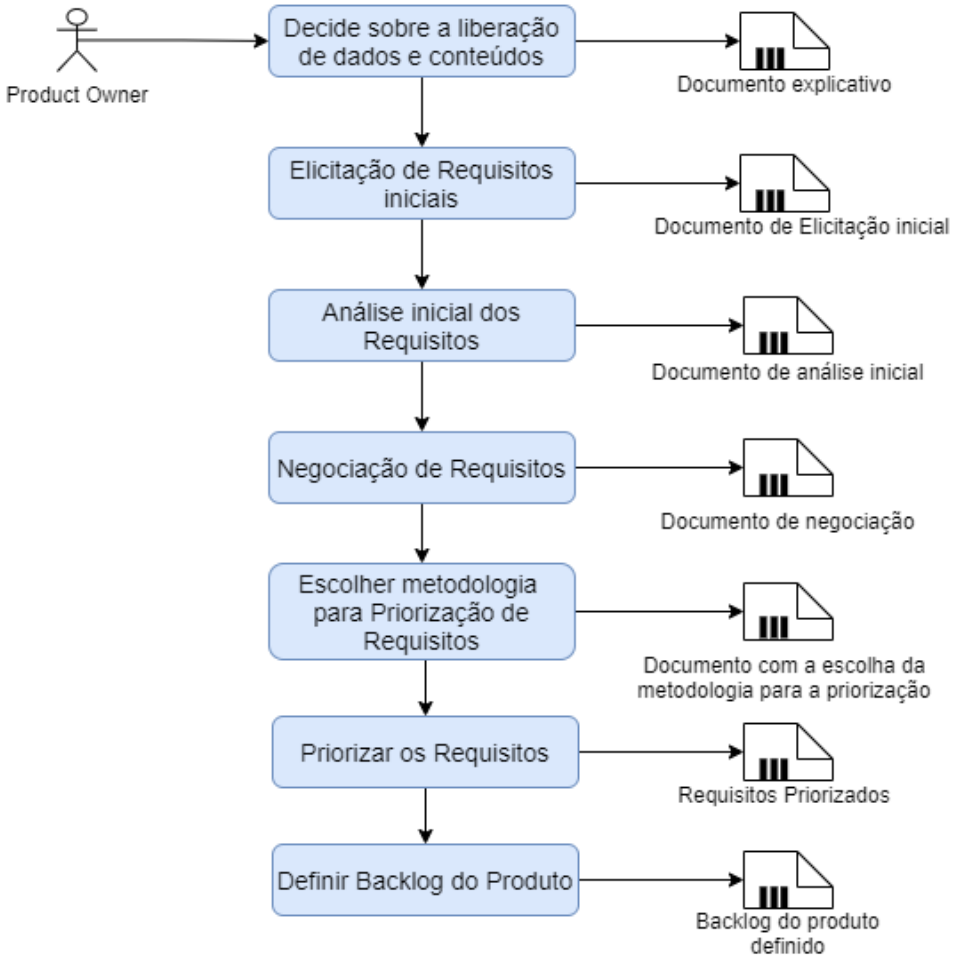


Fonte: Autoria própria, 2020

Em conformidade com a Figura 10, é válido exibir uma sequência lógica contendo atividades desenvolvidas pelo Proprietário do Produto. Entre as mesmas destacam-se: a definição dos itens e priorização da Backlogo do Produto, estas seguidas da explicitação para os membros da Equipe de Desenvolvimento. Ressalta-se, inclusive, a possibilidade do Proprietário do Produto delegar atividades, autorizar ou cancelar uma sprint.

Além disso, uma visão mais analítica do papel do Proprietário do Produto pode ser visualizada na figura 11.

Figura 11 – Visão analítica do papel do Proprietário do Produto



Fonte: Autoria própria, 2020.

Observa-se que na figura 11 foram acrescentadas as atividades iniciais que fazem parte do Processo de Engenharia de Requisitos, abrangendo o estudo de viabilidade, a elicitação, a análise e a negociação de requisitos. Na etapa da negociação de requisitos deve-se escolher uma metodologia para priorizar os requisitos.

Papel do *Scrum Master*

O Scrum Master, além de ter o embasamento técnico sobre o processo Scrum, é responsável pela sua correta efetivação e maximização das suas capacidades, ou seja, é ele quem vai fazer o Scrum funcionar.

O Scrum define práticas, reuniões, artefatos e terminologias. Para Schwaber (2004), o Scrum Master é responsável por esses conhecimentos e como aplicá-los corretamente. Pelo fato do Scrum ter a possibilidade de obter falhas, o Scrum Master poderá clarear o entendimento sobre seu funcionamento e guiar o projeto Scrum através do amontoado de complexidade.

Assim, Daneva et al (2013) afirmaram que a estrutura do Scrum possui vários eventos internos (rituais) que buscam garantir a priorização razoável de histórias e tarefas do usuário. A priorização de histórias de usuários e suas tarefas associadas devem ser realizadas de modo contínuo, visando garantir que a equipe Scrum trabalhe corretamente.

A atividade de priorização deve abranger todos os eventos Scrum anteriormente citados. De acordo com Daneva et al (2013) deve-se preparar a lista de pendências geradas no evento de preparação e iniciar o uso da metodologia Scrum, a partir da criação do Backlog do Produto. Na reunião de planejamento da Sprint, verificar-se-á as dependências entre as tarefas, incluindo, evidentemente, a construção e análise das histórias de usuários.

Durante a reunião diária, tanto as tarefas quanto os itens constantes na Backlog do produto devem ser operacionalizados de acordo com priorização realizada no evento anterior. Sabe-se que na reunião de revisão da Sprint novas análises são ralizadas para gerar entradas para a reunião de retrospectiva da Sprint, momento no qual a equipe de scrum poderá priorizar outros aspectos considerados essenciais.

Daneva et al (2013) apresentaram considerações importantes para a priorização dos requisitos de software. Defendeu seu ponto de vista sobre o assunto, afirmando que é preciso escrever os casos de uso essenciais e discutí-los os com os arquitetos, clientes, representantes e outras partes interessadas (equipe de desenvolvimento e stakeholders diretos e indiretos). Propôs que a estruturação dos casos de uso de alto nível e requisitos sejam escritos durante a Backlog do Produto para que em seguida ocorra a fase de estimativa e as sessões de priorização dos requisitos.

Recomenda-se que a equipe de desenvolvimento tenha uma correta compreensão dos novos conceitos associados ao produto que está sendo criado. Nesse sentido, o Dono do Produto, deverá classificar os requisitos e priorizá-los.

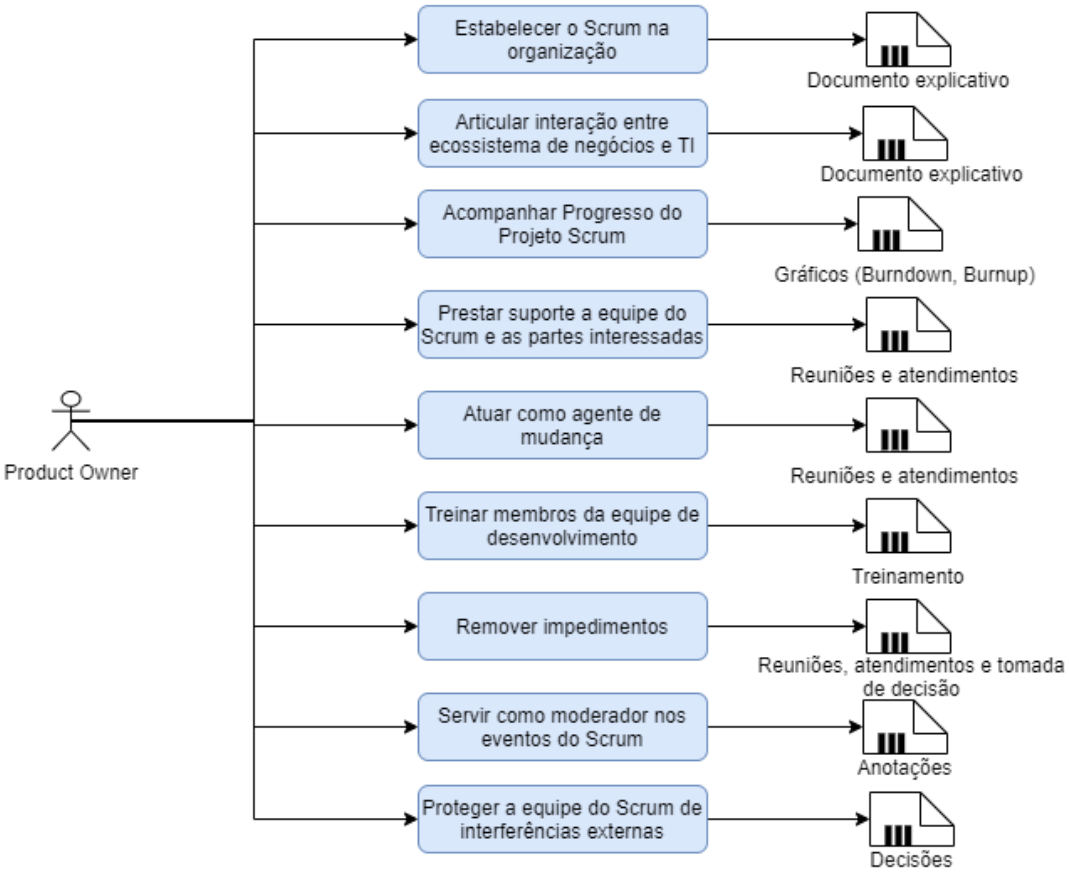
O Scrum Master atende a todos os participantes de um projeto Scrum e as partes interessadas externas para compreender e aplicar o Scrum Framework corretamente.

É um papel que apoia a equipe Scrum para executaro Scrum Framework com sucesso econtribui para melhorar sua produtividadee desempenho continuamente.

O papel do Scrum Master: uma visão mais detalhada

- Estabelecer o processo Scrum em sua organização, sendo uma nova maneira de pensar e agir; atua como agende de mudanças; treina a equipe para desenvolver novas normas e padrões de equipe; estabelece o Scrum Framework em seu ecossistema de negócios e TI; atua como agente de mudança e apoiar a adaptação dos processos existentes para maximizar produtividade da equipe Scrum; treina a equipe Scrum para entender e viver os valores do Scrum Framework; Garante colaboração eficiente e estreita entre o Dono do Produto Scrum e o Scrum Team; Remove impedimentos que dificultam a continuidade do trabalho; lidera o progresso do trabalho; modera os rituais do Scrum (eventos do Scrum); protege a equipe Scrum de ameaças externas, interferências e interrupções enquanto a equipe funciona.

Figura 12 – Atividades propostas para o Scrum Master



Fonte: Autoria própria, 2020.

Papel da Equipe de Desenvolvimento

A equipe de desenvolvimento, são pessoas que são responsáveis pelo desenvolvimento de tarefas realizáveis do produto. Eles são os encarregados de transformar os requisitos do ProductBacklog em algo passível de ser testado e utilizado ao final de cada Sprint.

Massari (2016) considera algumas características muito importantes para possuir uma boa equipe de desenvolvimento:

- Auto organização – a equipe sabe o que tem que fazer e como deve ser feito sem precisar de um chefe, mas sim um líder facilitador.
- Multifuncionalidade – a equipe possui capacidade necessária para a entrega do produto.

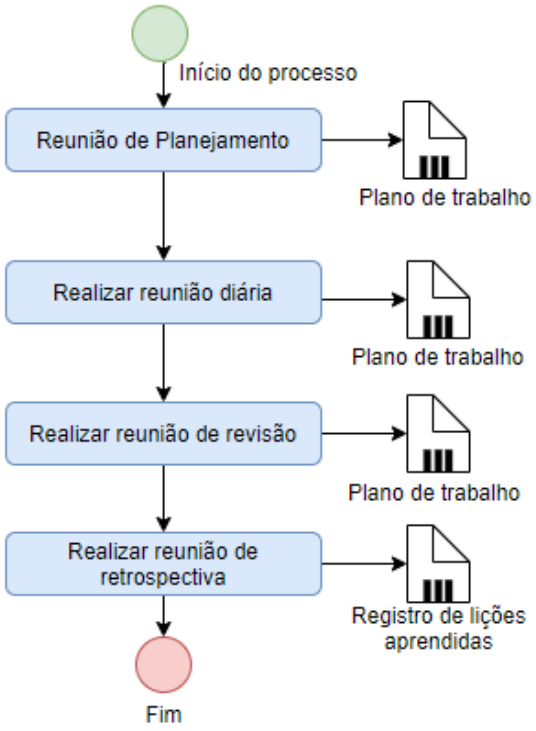
- Jamais negocia qualidade – a equipe de desenvolvimento não deve abrir mão de esforços que ponham em jogo a qualidade do produto.
- Recomendação de 5 a 9 integrantes – quantidade necessárias de pessoas com habilidades necessárias para manter sempre uma boa comunicação e organização para a entrega de cada Sprint.

Papel dos Clientes:

Participar ativamente dos eventos do Scrum e prestar as informações solicitadas pela equipe Scrum. Diferentemente das metodologias tradicionais, os clientes participam continuamente das etapas de construção do produto. Portanto, assumem um papel de coprodutores, com atitudes proativas e decisivas.

5.4 Eventos da metodologia Scrum

Figura 13 – Eventos utilizando metodologia ágil Scrum



Fonte: Autoria própria, 2020

O que é a Reunião de Planejamento?

É a fase inicial de cada Sprint. Nessa fase o ProductOwner, o Scrum Master e a equipe de desenvolvimento se reúnem para combinarem o que deverá ser arquitetado para a finalização da Sprint, cada reunião tem 8 horas de durabilidade.

De acordo com Cruz (2013), o planejamento é dividido em duas partes, cada uma com 4 horas de duração e com propósitos particulares:

Na primeira, o ProductOwner seleciona o ProductBacklog de maior prioridade, informa à equipe o que é desejado e esta informa as funcionalidades que poderão ser desenvolvidas na próxima Sprint.

Na segunda, a equipe planeja as Sprints. O objetivo é entender como as funcionalidades serão desenvolvidas. Como a equipe é responsável por gerenciar seu próprio trabalho, ela precisa de planos preliminares para iniciar a Sprint. As tarefas que compõem esse plano são colocadas em um Sprint Backlog. As tarefas na SprintBacklog surgem à medida que a Sprint é desenvolvida. No início do segundo período de quatro horas da reunião de planejamento da Sprint, a Sprint começou e o relógio está correndo para o seu desenvolvimento em torno de 30 dias (SCHWABER, 2004).

No geral, o objetivo é começar a trabalhar, e não pensar em como trabalhar (SCHWABER, 2004).

O que é a Reunião de Diária?

Reunião diária é o momento em que o time se junta por 15 min com o objetivo de sincronizar os trabalho dos membros do time diariamente e agendar qualquer reunião que a equipe precise para progredir. De acordo com Schwaber (2004), essa reunião é sustentada em 3 perguntas, são elas:

1. O que você finalizou no projeto desde a última reunião?
2. Quais os seus planos ou afazeres entre essa e a próxima reunião?
3. Algo impede que você cumpra seus compromissos com esta Sprint ou estes projetos?

O Scrum Master é quem garante que essa reunião seja feita respeitando sua duração e regras, e o time é encarregado a conduzi-la.

O que é a Reunião de Revisão?

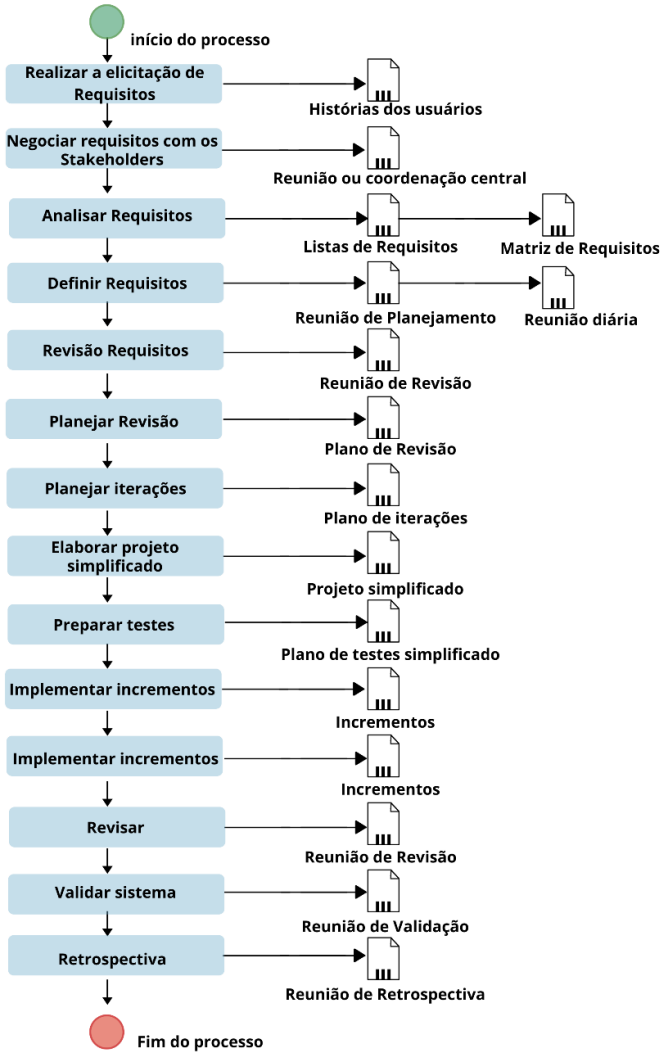
No final de cada Sprint, uma reunião de revisão é realizada. Essa reunião tem como objetivo apresentar para o ProductOwner e os stakeholders que querem participar, tudo aquilo que foi desenvolvido na Sprint.

É uma reunião informal, na qual as funcionalidades são apresentadas e visa reunir pessoas para determinar, em colaboração, o que a equipe deverá fazer e seguida (SCHWABER, 2004).

O que é a Reunião de Retrospectiva?

Nesta reunião, o ScrumMaster incentiva a equipe a revisar, dentro da estrutura e das práticas do processo scrum, seu processo de desenvolvimento para torná-lo mais eficaz e agradável para a próxima Sprint. A equipe se reúne, analisa o que deu certo e o que poderia ser feito melhor e aplica na Sprint seguinte (SUTHERLAND, 2016).

Figura 14 – Fluxo proposto abrangendo detalhes dos eventos do Scrum anteriormente descritos.



Fonte: Autoria própria, 2020.

Verifica-se que, de um modo geral, cada organização poderá adequar os fluxos dos eventos às suas necessidades organizacionais, ora acrescentando atividades, ora suprimindo. A figura acima é um exemplo de um fluxo mais detalhado que poderá ser aplicado a organizações mais experientes com a metodologia Scrum e possua um nível de maturidade mais elevada.

Todavia, deve-se, ainda, observar que o gerenciamento de requisitos é uma atividade integrante do processo de desenvolvimento de software. Os requisitos elicitados de software precisam ser tratados por meio de comunicação e integração adequadas entre as partes interessadas, tanto no âmbito do desenvolvimento tradicional quanto no ágil.

As metodologias ágeis são constituídas de processos inovadores e iterativos que suportam requisitos variáveis e ajudam a lidar com as mudanças ao longo do processo de desenvolvimento de software e que requer priorização. Os requisitos são elicitados no início de cada processo e projeto de desenvolvimento de software (produto) e estes são priorizados de acordo com sua importância para o mercado e para o próprio produto. Uma das etapas mais importantes e influentes ao criar um produto de software é a priorização de requisitos. A priorização de requisitos ajuda a equipe de software a entender a existência e a importância de um requisito específico, sua importância de uso e sua urgência de lançar no mercado.

Existem muitas técnicas de priorização de requisitos com seus pontos fortes e fracos. Caso contrário, muitos deles não consideram todos os fatores que devem ser considerados ao priorizar requisitos como custo, valor, risco, tempo de colocação no mercado, número de requisitos e efeito de requisitos não funcionais nos requisitos funcionais.

Há, portanto, várias metodologias de priorização de requisitos que auxiliam na tomada de decisões, mas o importante é que ainda falta explicar os fatores importantes que têm influência significativa na priorização de requisitos. Foi proposta uma metodologia de priorização de requisitos com base em vários fatores, como tempo de colocação no mercado, custo, risco, etc.

Os requisitos devem ser priorizados com base em fatores de influência, como custo, valor, risco, tempo de colocação no mercado etc., e através do efeito de requisitos não funcionais sobre os requisitos funcionais. Isso melhorará a qualidade geral do produto de software quando ele for incluído no processo de desenvolvimento do Scrum.

Em conformidade com a litetratura especializada, a exemplo das recomendações de Sutherland (2010), pode estar presentes no processo de desenvolvimento:

• Etapa 1 - Inspecionar: fazer o que for preciso para entender o projeto atual;

• Etapa 2 - Adaptar: indicar uma direção e visão sobre os próximos passos do projeto e, em seguida, criar estratégias e executar essa visão;

• Etapa 3 - Aprenda: Observar, aprender e ensinar um ao outro;

• Etapa 4 - Reiniciar: desenvolver novas iterações, caso necessário;

Geralmente, podem ser realizadas atividades de identificação de quais tarefas serão necessárias para cumprir os objetivos de cada Sprint, identificação e priorização do que é essencial desenvolver o melhores artefatos, produtos e serviços de software, em conformidade com as expectativas dos clientes.

5.5 Estratégias de pontuações para o cálculo de riscos

A partir da literatura especializada no assunto, a exemplo de considerações de Alkandari e Al-shammeri (2017), nesta monografia foi incluída um conjunto de critérios para priorização dos requisitos e tarefas, são eles:

1) Impacto no produto para o ambiente externo – não se deve confundir o supracitado impacto com valor agregado. O primeiro impacto se refere aos aspectos não organizacionais, enquanto o segundo aos organizacionais.

2) Restrições – devem ser avaliadas desde a concepção do projeto no estágio de planejamento da iteração. Estimar as restrições do projeto e da iteração e escolher os requisitos com base nelas no planejamento da iteração é o trabalho do gerente de projeto.

3) Escopo – atribuir uma pontuação para a abrangência dos requisitos do projeto.

4) Dependência para realização de outras atividades – Nos projetos de desenvolvimento de requisitos, podem estar presente as dependências entre os requisitos, estas têm impacto no ritmo de desenlvimento e complexidade do projeto. Há dependências cronológicas e arquitetônicas. Caberá a equipe de desenvolvimento e os membros das demais partes interessadas avaliar a importância das dependências e atribuir uma pontuação para as dependências encontradas.

5) Volatilidade - a tendência dos requisitos de mudar ao longo do tempo em resposta às necessidades em evolução dos clientes, stakeholders, organização e ambiente de trabalho deve ser incorporada ao cálculo da priorização dos requisitos. Essa tendência pode levar em consideração aspectos internos e externos. Ressalta-se, assim, que na

presente proposta de pontuação cada equipe de projeto poderá incorporar subcritério(s).

6) Risco – é o cálculo do impacto da propabilidade e impacto ou urgência dos requisitos ou tarefas. Os riscos de requisitos estão diretamente associados a requisitos específicos e podem ser riscos de impactos negativos ou positivos.

7) Esforço - deve-se utilizar de estimativas e métricas para cálcular o número de membros da equipe que estão responsáveis pelas atividades do processo de engenharia de requisitos.

8) Tempo – é preciso estimar o tempo é atribuir uma pontuação em relação à priorização dos requisistos e tarefas.

9) Débito técnico (technical debt) – dívida técnica (também conhecida como débito técnico ou “technical debt”) trata-se de uma metáfora associada às consequências do desenvolvimento de software deficiente, gerando um produto que precisará ser concluído. O termo foi introduzido em 1992 por Ward Cunningham.

10) Oportunididade de uso – se refere às ocorrências advindas do ambiente externo que requerem um reposicionamento do projeto e uma nova priorização de requisitos. Deve-se avaliar ocorrências associadas à concorrência, às mudanças no comportamento do consumidores, alterações legais etc. Caberá a equipe do Scrum incluir no cálculo de priorização essas ocorrências, pois podem variar no tempo e no espaço.

11) Valor agregrado– na literatura especializada há muitas informações sobre o conceito de valor agregado. Neste trabalho, destaca-se apenas que deve ser realizado uma pontuação para o cálculo do valor agregado. Não há uma receita pronta para os critérios que podem ser incorporados a esse critério de priorização. Por exemplo, há casos nos quais os gestores dos projetos podem incorporar ao cálculo o retorno financeiro, o retorno social, a valorização da marca ou do produto, o posicionamento da organização no mercado, a conquista de novos mercados etc.

12) Custo - trata-se de um dos fatores críticos de priorização, posto que influencia diretamente na selação dos requisitos acessíveis e poderão ser ser implementados.Os recursos financeiros devem ser bem dimensionados e incluídos no orçamento do projeto.

13) Conhecimento da equipe acerca daquilo que será realizado – a experiência associada aos conhecimentos práticos e teóricos são essenciais para priorizar os requisitos. Alguns tipos de requisitos poderão alcançar uma alta pontuação na priorização, caso não requeiramo desenvolvimento de conhecimentos. Porém, para os requisitos que demandem um tempo para desenvolver esses conhecimentos é recomendável pontuá-los com média ou baixa prioridade.

14) Gerenciamento de recursos disponíveis – deve-se pontuar o gerenciamento dos recursos do projeto, considerando a identificação, aquisição e gerenciamento dos recursos necessários para a conclusão bem-sucedida do projeto. Tal atividade engloba o planejamento do gerenciamento dos Recursos (informacionais, computacionais, materiais, equipamentos e humanos), a estimativa de uso dos recursos necessários ao desenvolvimento dos requisitos, a adquirir dos recursos, o desenvolvimento da equipe, o gerenciamento da equipe e o controle dos recursos.

Esta proposta de priorização de requisitos no Scrum abrange:

1 – Definição da visão do produto;
2 – Elicitação de requisitos com os clientes;
3 –Negociar e priorizar requisitos;
4 – Definir os requisitos funcionais e não funcionais;
5 - Verificar os requisitos;
6 – Validar os requisitos;
7 – Gerenciar os requisitos.

Considerando que a criação e a priorização dos requisitos podem ser realizadas com as histórias dos usuários, recomenda-se neste trabalho monográfico

utilizar três pontuações: alta prioridade (peso 3), média prioridade (peso 2) e baixa prioridade (peso 1).

Em projetos maiores e mais complexos, os requisitos contém muitos detalhes que devem ser observados na implementação. Recomenda-se o uso das histórias de entrega uma vez que contém mais detalhes adicionais inerentes aos tipos de requisitos, especificações de design, segurança, por exemplo. Destaca-se que os detalhes do projeto colaboram para uma melhor compreensão dos requisitos.

No Scrum, a colaboração cliente-desenvolvedor é primordial para estabelecimento de um processo de engenharia de requisitos, contribuindo para a (re)priorização de requisitos. Segundo Racheva et al. (2010), repriorizar os requisitos no momento da inter-iteração desempenha um papel fundamental em projetos ágeis, geralmente presentes em contextos de pequenas e médios projetos.

A priorização de requisitos é um processo de tomada de decisão (Alenjung e Persson 2008; Hermann e Daneva (2008). Para Pfleeger (2001), um processo é definido pelos papéis das pessoas, artefatos (usados ou produzidos) e recursos.

5.5.1 Primeira estratégia

Em relação ao cálculo dos riscos é recomendável que se construa a matriz de riscos, conforme proposta a seguir:

Tabela 2 – Matriz de risco 3x3

Fonte: autoria própria, 2020.

Tabela 3 – Matriz de risco 5x5

Fonte: autoria própria, 2020.

No caso explificado, foram apresentadas nas tabelas 2 e 3 modelos de Matriz de Riscos. A tabela 2 exibe uma Matriz de Risco com do tipo 3 x 3, ou seja, três pontuações para as probabilidades e três para os impactos. Resulta em apenas três tipos de riscos: Alto, Médio e Baixo.

Porém, na tabela 3, do tipo 5 x 5, há 5 pontuações para as probablilidades e cinco para os impactos. Resulta, no exemplo apresentado, em quatro tipos de riscos de acordo com a faixas resultantes da multiplicação em probablidade e impacto.

Tabela 4 – Tipos de riscos

16 a 20	Crítico
10 a 15	Alto
5 a 8	Moderado
1 a 4	Baixo

Fonte:Autoria própria, 2020.

As stories devem ser refinadas com suas regras de negócio, decompostas em stories menores (caso necessário). Assim, poderão ser definidos os cenários de uso, ou melhor, os comportamentos do produto na visão de seus usuários finais.

A proposta até aqui apresentada adiciona as contribuições de Darwish e Megahed (2016):

1) O Dono do Produto formula o backlog do produto. Partes interessadas podem participar do backlog do produto. Durante essa etapa, uma história de usuário será escrita em um cartão e descreverá um recurso que agregará valor ao cliente. Posteriormente, serão realizados os testes de aceitação. Realizar-se-á o planejamento das liberações e interações incluindo informações sobre o tempo e as funcionalidades do produto. O plano de liberação é construído atribuindo histórias às iterações.
2) Participam dessa etapa a equipe de desenvolvimento e a equipe do cliente. Em relação à priorização das histórias, deve-se ressaltar que os clientes exercem um papel bastante ativo conforme destacado anteriormente.

5.5.2 Segunda estratégia

A equipe de desenvolvimento estimará a velocidade para desenvolver o requisito que será concluído por iteração. Pode adotar pontuações em várias escalas. Por exemplo de 1 a 5 ou de 1 a 10 ou de 1 a 15 e, assim, sucessivamente. Cada equipe poderá escolher e implementar a escala mais apropriada tanto para a velocidade de desenvolver o requisito ou para avaliar o custo ou a complexidade. Todavia, deverá compreender que essa pontuação é específica para cada item, diferentemente da pontuação mais geral para critérios de análise geral. Neste projeto será apresentado mais adiante um exemplo com pontuação de NI – Não Implementado – Pontuação: 0; PI – Parcialmente Implementado – Pontuação: 1; LI – Largamente Implementado – Pontuação: 2 e FI – Totalmente Implementado – Pontuação: 3

Pode-se tambem utilizar outras pontuações, a exemplo da representada na figura abaixo. A tabela 5 representa as histórias e as pontuações em uma escala de 1 a 10.

Tabela 5 – Histórias e prontuações

Histórias	Pontuação
História A	4
História B	4
História C	4
História D	5
História E	5
História F	6
História G	9
História H	5
História I	2

Fonte: Autoria própria, 2020.

Por sua vez a tabela 6 exibe as pontuações para cada uma das interações, em conformidade com a pontuação de cada uma das histórias.

Tabela 6 – Pontuação para as interações e as histórias:

Interações	Histórias	Pontuação
Interação 1	A, B	4+4=8
Interação 2	C, D, E	4+5+5=14
Interação 3	F, G	6+9=15
Interação n	H,J	5+2=7

Fonte: Autoria própria, 2020.

Verificou-se que a 3 interação (F;G) alcançou a maior pontuação com 15 pontos, seguinda Interação 2 (C;D,E) com 14 pontos.

Assim, a equipe de desenvolvimento procederá à implementação do requisisto e realizará as atividades de testes. Para tanto, essa equipe poderá escrever os testes no verso do cartão de histórias, quer sejam testes com ou sem ferramentas automatizadas.

Ressalta-se que na metodologia aqui proposta, um cartão de história deverá conter uma breve descrição da funcionalidade descrita pelos usuáriso e/ou pelo cliente.

Devem ser consideradas algumas recomendações nessa etapa do processo de desenvolvimento do produto:

1) Criar um canal de comunicação entre a equipe do Scrum de modo que a equipe de membros indicados pelo cliente faça corretamente a descriação das histórias.

2) As histórias serão priorizadas;

3) Pode-se subdividir uma história, sempre que necessário;

4) Após, deve-se proceder aos testes de validação das histórias

5) Cada equipe Scrum deverá escolher a técnica ou as técnicas para obter as informações das histórias: entrevistas, questionários, observação, oficinas de redação de histórias. Ou seja, há liberdade para adequar da técnica a cada contexto de desenvolvimento;

6) A equipe de desenvolvimento e equipe do cliente devem atuar em conjunto na fase de testes. Recomenda-se realizar testes de interface do usuário, usabilidade, desempenho, teste de estresse e testes de aceitação.

5.5.3 Terceira estratégia

Uma outra alternativa é usar a pontuação de 14 critérios, conforme exemplo apresentado na tabela abaixo.

Tabela 7 – Pontuação em 14 critérios

Critérios	Baixa (1)	Média (2)	Alta (3)
Impacto no produto		2	
Restrições		2	
Escopo	1		
Dependência para realização de outras atividades			3
Volatilidade		2	
Risco		2	
Esforço(fórmula)			3
Tempo			3
Débito técnico (technical debt) – informação histórica de outros projetos ou análise realizda no projeto atual		2	
Oportunidade de uso			3
Valor agregado			3
Custo			3
Conhecimento da equipe acerca daquilo que será realizado		2	
Complexidade		2	
Total	1	14	18

Fonte: autoria própria, 2020.

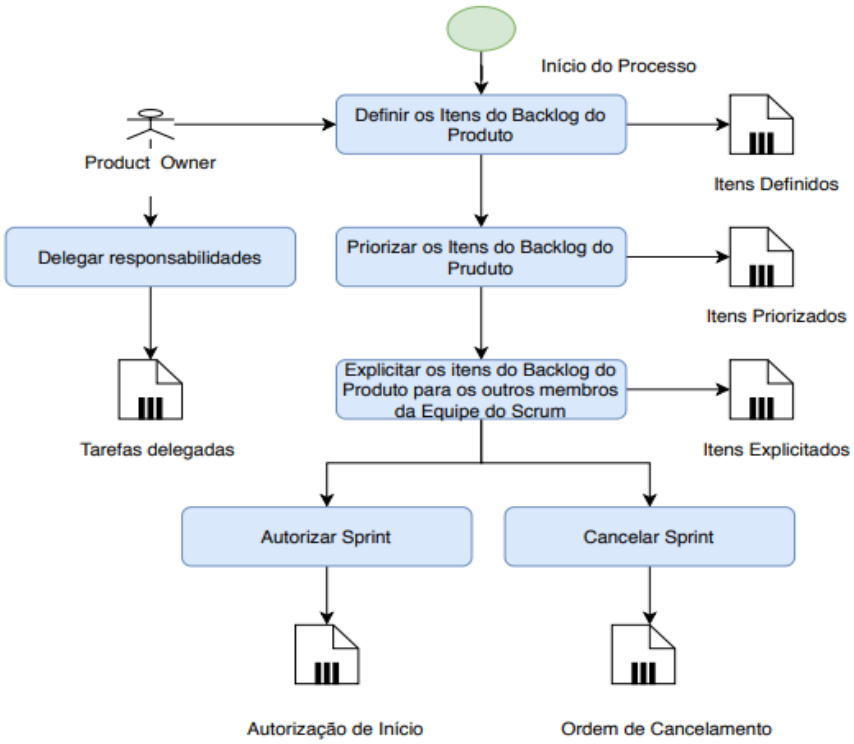
Verificou-se que o projeto em questão alcançou 18 pontos de alta prioridade, 14 de média e 1 de baixa prioridade. Cada um dos itens de maior prioridade devem ser devidamente acompanhados, pois são considerados essenciais para o sucesso do projeto. Essa avaliação deve se adequar a cada projeto e suas especificidades.

Porém, de acordo com a proposta adaptada do SCAMPI, apresenta-se a seguir um exemplo para pontuação da presença de atividades nos fluxos.

5.6 Exemplo de aplicação: Pontuação dos fluxos

Para concluir este trabalho monográfico, aparesenta-se a seguir apenas um exemplo do que poderá ser utilizado em todos os fluxos de um projeto de desenvolvimento de software na metodolgia SCRUM, segundo os templates aqui propostos.

Figura 15 – Atuação do proprietário do produto (resumido)



Fonte: Autoria própria, 2020.

NI – Não Implementado – Pontuação: 0 (0-25%)

PI – Parcialmente Implementado – Pontuação: 1 (26-50%)

LI – Largamente Implementado – Pontuação: 2 (51-75%)

FI – Totalmente Implementado – Pontuação: 3 (76-100%)

Tabela 8 – tabela de pontuação das atividades do fluxo

Itens	NI	PI	LI	FI	Total
1 Definir os Itens da Backlog		1			1
2 Delegar responsabilidade			2		2
3 Priorizar os Itens da Backlog do Produto			2		2
4 Explicitar os Itens da Backlog do Produto			2		2
5 Autorizar/cancelar Sprint				3	3
Total	0	1	6	3	10

Fonte: autoria própria, 2020

No exemplo apresentado, foram avaliados apenas 5 itens. A pontuação máxima segundo a metodologia adaptada do SCAMPI para esse caso seria: 5 itens x 3 (pontuação máxima se todos os itens estivessem totalmente implementados). Resultaria, no máximo, de 15 pontos.

Contudo, no exemplo, apresentado, a pontuação máxima alcançada foi de 10 pontos.

Usando um cálculo simples de percentual, chegou-se a seguinte conclusão: (10*100)/15 = 66,67%. Vale dizer, no exemplo utilizado, a organização foi avaliada na faixa de LI – Largamente Implementado – Pontuação: 2 (51-75%), pois alcançou uma pontuação entre 51 a 75%.

6 - CONCLUSÃO

Após as pesquisas realizadas é perminente apresentar as principais conclusões desta monografia que teve como propósito buscar uma resposta para a seguinte questão de pesquisa: Como propor uma estratégia capaz representar e avaliar os fluxos de atividades da metodologia SCRUM, incorporando critérios para priorização dos requisitos?

Conforme apresentado nos capítulo 4 de desenvolvimento, constatou-se que a estratégia proposta conseguiu responder à questão formulada, pois o objetivo geral da investigação foi alcançado. Ou seja, foi possível representar e avaliar os fluxos de atividades da metodologia SCRUM, incorporando critérios para priorização dos requisitos. Além disso, foi possível também a) Descrever à luz da literatura especializada o processo de engenharia de requisitos; b) Descrever a metodologia ágil Scrum, com ênfase nos seus princípios e práticas, na estruturação dos membros da equipe Scrum,nos eventos existentes e nos artefatos gerados;c) Criar modelos no Draw.io para descrever os fluxos de atividades propostas para o processo de Engenharia de Requisitos aplicada ao Scrum;d) Propor critérios para priorização dos requisitos; e) Apresentar exemplo de avaliação (pontuação) das atividades constantes nos fluxos de atividades propostos na presente estratégia.

Trata-se de um trabalho inédito que deverá ser continuado com outras pesquisas. Recomenda-se:

a) Aplicar a estratégia usando os métodos de estudo de casos.
b) Incorporar novas recomendações para tornar a estratégia cada vez mais adaptável aos diferentes tipos de organizações e projetos.

REFERÊNCIAS

ALENJUNG, B; PERSON, A. Portraying the practice of decision-making in requirements engineering: a case study of large scale bespoke development.**Requirements Engineering Journal**, 13, pp. 257-279, 2008.

ALKANDARI, Mohammad; AL-SHAMMERI, Asma. Enhancing the Process of Requirements Prioritization in Agile Software Development - A Proposed Model**. Journal of Software**, v. 12, n. 6, June 2017.

AMBLER, Scott. **Agile modeling:** effective practices for extreme programming and the unified process. New York: John Wiley & Sons, 2002

BECK, Kent. **Test driven development:** By Example Addison-Wesley. The Addison-Wesley Signature Series, 2002.

BECK, Kent et al. **Manifesto for agile software development**. Disponível em:http://AgileManifesto.org. Acesso em: 29 out. 2020.

BJARNASON, E.; WNUK, K.; REGNELL, B. A case study on benefits and side-effects of agile practices in large-scale requirements e ngineering. In:**Proceedings of the 1st Workshop on Agile Requirements Engineering**. Publisher: ACM Press. New York, July, 2011.

BOEHM, B. W et al. "Software engineering economics: background, current practices, and future directions. **Proceedings of the 24th International Conference on Software Engineering. ICSE 2002.** Florida: IEEE Computer Society, 2002.

BONESS, K.; HARRISON, R. Goal sketching: towards agile requirements engineering. **International Conference on Software Engineering Advances** (ICSEA 2007). DOI: 10.1109/ICSEA.2007.36. Cap Esterel, France, 2007.

CAO, Lan; RAMESH, Balasubramaniam. Agile requirements engineering practices: An empirical study. **IEEE software**, v. 25, n. 1, p. 60-67, 2008.

CARVALHO, A. M. B. R.; CHIOSSI, T. C. S. **Introdução à engenharia de software.** Campinas: Unicamp, 2001.

CHIZZOTTI, A. **Pesquisa em ciências humanas e sociais**. 2 ed. São Paulo: Cortez, 1995.

CRUZ, Fábio. **Scrum e PMBOK unidos no gerenciamento de projetos**. Rio de Janeiro: Brasport, 2013.

DAMIAN, D.E.H. **Challenges in requirements engineering**. Department of Computer Science, University of Calgary, 2000.

DANEVA, Maya et al. Agile requirements prioritization in large-scale outsourced system projects: An empirical study**. Journal of systems and software**, v. 86, n. 5, p. 1333-1353, 2013.

DANEVA, M. et al. A Reflection on Agile Requirements Engineering: Solutions Brought and Challenges Posed. In: **XP 2015 Workshops, Helsinki, Finland**. 2015.

DARWISH, Nagy Ramadan; MEGAHED, Salwa. Requirements Engineering in Scrum Framework. **International Journal of Computer Applications**. v. 149, n. 8, set 2016.

DAVIS, Alan M. **Software requirements: objects, functions, and states**. Prentice-Hall, Inc., 1993.

ERIKSSON, Hans-Erik; PENKER, Magnus. **Business modeling with UML**. New York, p. 1-12, 2000.

ESTEVES, R.; RODRIGUES, L.A.; PINTO, N.A. ScrumS: a model for safe agile development**. 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems**. DOI: 10.1145/2857218.2857225. Publisher: ACM. Caraguatatuba, Brazil, October 25 - 29, 2015.

FOWLER, Martin. **The new methodology**. Wuhan University Journal of Natural Sciences, v. 6, n. 1-2, p. 12-24, 2001.

GIL, Antonio Carlos. **Elaboração de projetos de pesquisa**. 5. ed. São Paulo: Atlas, 2010.

GRANDA, J. B**. Manual de metodología de la investigación científica**. 3. ed. Chimbote: Uladech, 2015.

HEIKKILÄ, Ville T. et al. A mapping study on requirements engineering in agile software development. In: **2015 41st Euromicro Conference on Software Engineering and Advanced Applications**. IEEE, 2015. p. 199-207.

HERRMANN, Andrea; DANEVA, Maya. Requirements prioritization based on benefit and cost prediction: an agenda for future research. In: **2008 16th IEEE International Requirements Engineering Conference**. IEEE, 2008. p. 125-134.

HIGHSMITH, Jim; COCKBURN, Alistair. Agile software development: The business of innovation**. Computer**, v. 34, n. 9, p. 120-127, 2001.

INAYAT, I.; MARCZAK, S.; SALIM, S.S.; DANEVA, M.; SHAHABODDIN, S. A systematic literature review on agile requirements engineering practices and challenges**.Journal Computer in Human Behavior**, v. 51, Part B, p. 915–929. 2014.

IEEE. IEEE Standards Glossary of Software Engineering Terminology.**IEEE Std 610.12-1990**, New York, 1990.

JALOTE, P. **An integrated approach to software engineering**. (D. Gries & F. B. Schneider, Eds.) (Third). New York: Springer, 2005.

JAQUEIRA, Aline de Oliveira Prata. **Uso de modelos i* para enriquecer requisitos em métodos ágeis**. Universidade Federal do Rio Grande do Norte, 2013.

JAQUEIRA, A., et al. Desafios de Requisitos em Métodos Ágeis: uma revisão sistemática. In: **3rd Brazilian Workshop onAgileMethods, São Paulo.** 2012.

KUDRYASHOV, Konstantin. The beginner's guide to BDD. **Dan North Q & A. https://inviqa.com/blog/bdd-guide**, 2015.

KUMAR, Chetan; NORRIS, John B. A new approach for a proxy-level web caching mechanism**. Decision SupportSystems**, v. 46, n. 1, p. 52-60, 2008.

KOTONYA, G.; SOMMERVILLE, I. **Requirements engineering:processes and techniques.** Chichester: John Wiley & Sons, 1998.

LARMAN, C.; VODDE, B. **Large-Scale Scrum:** More with LeSS**.** Addison-Wesley Professional, 1 edition, ISBN-10: 0321985710. USA, 2016.

LEFFINGWELL, D.; **Calculating the return on investment from more effective requirements management**. American Programmer v. 10, n. 4, p. 13-16; 1997.

LEFFINGWELL, Dean; KNASTER, Richard. **SAFe 4.5 Distilled: Applying the Scaled Agile Framework for Lean Enterprises**. Addison-Wesley Professional, 2018.

NEGRÃO, E.C; GUERRA, E.M.**A Case Study for Prioritizing Features in Environments with Multiple Stakeholders.** OOPSLA'11: ACM international conference companion on Object oriented programming systems languages and applications companion. DOI: 10.1145/2048147.2048187. Publisher: ACM. Portland, Oregon, USA, October 22 - 27, 2011.

MASSARI, Vitor. **Agile Scrum Master no gerenciamento avançado de projetos**. Rio de Janeiro, RJ: Brasport, 2016.

MARQUES, André Nóbrega. **Metodologias ágeis de desenvolvimento: Processos e Comparações**.Faculdade de Tecnologia de São Paulo. São Paulo, 2012.

MARTINS, L. E. G. **Uma metodologia de Elicitação de Requisitos de Software Baseada na Teoria da Atividade**. Diss. Tese Doutorado Universidade Estadual de Campinas, Faculdade Elétrica e de Computação. Campinas, SP, 2001.

MEDEIROS, J.D.R.V. **An approach based on design practices to specify requirements in Agile Software Development.** Tese de Doutorado. Universidade Federal de Pernambuco. Recife, PE, Brasil, 2017.

MEDEIROS, Juliana et al. Requirements Engineering in Agile Projects: A Systematic Mapping based in Evidences of Industry. In: **CibSE**. p. 460, 2015.

MENDONÇA, Ricardo Augusto Ribeiro de. Levantamento de requisitos no desenvolvimento ágil de software.**Semana da Ciência e Tecnologia da PUC Goiás**, p.12. Goiás, 2014.

MORETTO, Luis Augusto. **Mecanismos para validação de requisitos**. Morettic**,** 2018. Disponível em: https://morettic.com.br/wp2/mecanismos-validacao-de-requisitos/. Acesso em: 18 de mar. de 2020.

NOGUEIRA, Wagner Alves Gonçalves. **Metodologia Ágil**: Proposta e Avaliação de Método para Desenvolvimento de Software. 2016.

PAETSCH, Frauke; EBERLEIN, Armin; MAURER, Frank. Requirements engineering and agile software development. In: **WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.** IEEE, p. 308-313, 2003.

PAULA FILHO, Wilson de Pádua. **Engenharia de software**. v. 2. Rio de Janeiro: LTC, 2003.

PAVLOV, Vladimir L. et al. **An Experience of Integrating INTSPEI P-Modeling Framework with Microsoft Solutions Framework for Agile Software Development**. In Proceedings of the IASTED International Conference on Software Engineering SE. 2007.

PFLEEGER, S. What Good Are Metrics? The Views of Industry and Academia**.** In: **Software Metrics, IEEE International Symposium on.** IEEE Computer Society, 2001. p. 146-146.

PICHLER, Roman. **Agile product management withscrum: Creating products that customers love**. Pearson Education, India, 2010.

PRESSMAN, R. S. **Engenharia de software.** 6. ed. São Paulo: McGraw Hill, 2006.

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de software.** 8 ed. McGraw Hill Brasil, 2016.

RACHEVA, Zornitza et al. A conceptual model and process for client-driven agile requirements prioritization. In: **2010 Fourth International Conference on Research Challenges in Information Science (RCIS)**. IEEE, 2010. p. 287-298.

QUSEF, A.; DE LUCIA, A. Requirements engineering in agile software development. **Journal of emerging technologies in web intelligence**, v. 2, n. 3, p. 212-220, 2010.

SANTOS, Kleoson B. C.**React:** uma abordagem ágil de apoio ao processo de desenvolvimento de requisitos de software baseada em evidências empíricas. Universidade Federal do Paraná, Belém, 2018.

SAWYER, P.; KOTONYA, G. Software requirements. In: **Guide to the software engineering body of knowledge, SWEBOK,** 2001. Disponível em: http://www.swebok.org/stoneman/trial_1_00.htm. Acesso em: 27 nov. 2019.

SCHNEIDER, Geri; WINTERS, Jason P. **Applying use cases: a practical guide**. Pearson Education, 2001.

SCHWABER, Ken. **Agile project management with Scrum**. Microsoft press, Redmond, WA, 2004.

SCHWABER, Ken. **The Enterprise and Scrum**. Redmond, WA: Microsoft Press, 2007.

SIDDIQI, Jawed. **Requirements engineering**: the emerging wisdom in software requirements engineering, IEEE-CS Press, Second Edition, 1997, p.p. 36-40.

SOMMERVILLE, I. **Software engineering**. 8th edition, Pearson Education, 2007.

_____. **Engenharia de software**. 9.ed. São Paulo: Pearson Prentice Hall, 2011.

SOMMERVILLE, I; SAWYER, P. **Requirements engineering**: a good practice guide, Chichester: John Wiley, 1997.

STANDISH GROUP. **Extreme chaos Report.** The Standish Group International Inc. (2009). Disponível em:. Acesso em: jan. 2020.

SUTHERLAND, Jeff. **Scrum**: a arte de fazer o dobro do trabalho na metade do tempo. São Paulo: Leya, 2016.

THAYER, R. H.; DOORFMAN, M. **Software requirements engineering**. 2. ed. IEEE Computer Society Press, 1997. Disponível em: http://www.sei.cmu.edu/news-at-sei/features/1999/mar/Background.mar99.pdf. Acesso em: 27 nov. 2019.

TURINE, M.A.S.; MASIERO, P.C. **Especificação de requisitos:** uma introdução. Instituto de Ciências Matemáticas de São Carlos, USP. 1996.

VIANNA, Mauro. Linha de código, 2019. **Reuniões de levantamento, como torná-las produtivas?** Disponível em: http://www.linhadecodigo.com.br/artigo/159/reunioes-de-levantamento-como-torna-las-produtivas.aspx. Acesso em: 29 de nov de 2019.

WILSON, Mark. Markwilson, 2005. **The Microsoft solution accelerator for business desktop deployment, the Microsoft solutions framework and the Microsoft operations framework.** Disponível em: https://www.markwilson.co.uk/blog/2005/02/microsoft-solution-accelerator-for.htm. Acesso em: 26 de out de 2020.

YOUNG, Ralph Rowland. **The requirements engineering handbook.**Norwood, MA: Artech House, 2004.