
**UNIVERSIDADE ESTADUAL DA PARAÍBA**
**CAMPUS I**
**CENTRO DE CIENCIAS E TECNOLOGIA**
**CURSO DE COMPUTAÇÃO**

**ALCINAEL FERNANDES PEREIRA**

# ANÁLISE DA IMPORTÂNCIA DOS REQUISITOS NO DESENVOLVIMENTO DE SOFTWARE UTILIZANDO ARTEFATOS DO SCRUM COM O APOIO DA ENGENHARIA DE REQUISITOS

**CAMPINA GRANDE**
**2018**

**ALCINAEL FERNANDES PEREIRA**

# ANÁLISE DA IMPORTÂNCIA DOS REQUISITOS NO DESENVOLVIMENTO DE SOFTWARE UTILIZANDO ARTEFATOS DO SCRUM COM O APOIO DA ENGENHARIA DE REQUISITOS

Trabalho de Conclusão de Curso de Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Computação.

**Área de concentração:** Engenharia de Software.

**Orientador:** Prof. Dr. Frederico Moreira Bublitz.

**CAMPINA GRANDE**
**2018**

P436a Pereira, Alcinael Fernandes.
Análise da importância dos requisitos no desenvolvimento de software utilizando artefatos do Scrum com o apoio da Engenharia de requisitos [manuscrito] : / Alcinael Fernandes Pereira. - 2018.
60 p. : il. colorido.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia , 2018.
"Orientação : Prof. Dr. Frederico Moreira Bublitz , Coordenação do Curso de Computação - CCT."

1. Requisitos. 2. Engenharia de software. 3. Framework Scrum. 4. Engenharia de requisitos.

21. ed. CDD 005.1

Elaborada por Giulianne Monteiro Pereira - CRB - 15/714 BC/UEPB

**ALCINAEL FERNANDES PEREIRA**

# ANÁLISE DA IMPORTÂNCIA DOS REQUISITOS NO DESENVOLVIMENTO DE SOFTWARE UTILIZANDO ARTEFATOS DO SCRUM COM O APOIO DA ENGENHARIA DE REQUISITOS

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 21 de Junho de 2018.

Prof. Dr. Frederico Moreira Bublitz (DC – UEPB)
Orientador

Prof. Me. Adrian Lívio Vasconcelos Guedes (DC – UEPB)
Examinador

Prof. Dr. Wellington Candeia de Araújo (DC – UEPB)
Examinador

## AGRADECIMENTOS

Agradeço primeiramente a meus pais que me ajudaram a chegar onde cheguei, principalmente a minha mãe Maria que me incentivou e esteve ao meu lado, a meus amigos que me auxiliaram nas dificuldades do caminho acadêmico e a meu namorado Emerson que sempre me apoiou e me deu forças tornando esse percurso possível.

Agradeço a meus professores que me mostraram novas perspectivas sobre a computação em especial ao professor Dr. Frederico Moreira Bublitz, meu orientador que me fez ver grandes possibilidades na computação e a professora Luciana que me fez ver que a engenharia de software é um campo vasto e importante na computação. Agradeço a minha instituição que me proporcionou os meios necessários para chegar ao fim deste ciclo de modo satisfatório.

## RESUMO

Os requisitos originam as funcionalidades e restrições de um sistema, é através deles que um sistema toma forma e pode ser construído, contudo dentro da engenharia de software existem metodologias distintas para desenvolvimento de sistemas. Dois dos processos que se destacam são a engenharia de requisitos, conhecida pela utilização de métodos documentacionais e o Scrum que por ser um *framework* baseado em processos ágeis torna o desenvolvimento mais flexível, pois permite que sejam assimiladas em seu escopo técnicas de outras metodologias. No entanto, embora sejam bastante difundidas e populares suas características específicas podem deixar lacunas que permitem possíveis falhas de projeto. Entretanto, ambas possuem a mesma finalidade, a criação de um requisito que segue os processos para efetivação de modo distinto. Objetivando a utilização de artefatos do Scrum direcionados para requisitos como referenciais neste processo, hipoteticamente inserir uma técnica de um modelo inflexível em um flexível, afim de tornar os requisitos produtos funcionais duráveis em qualquer fase de um software enquanto se manter ativo.



**Palavras-chave**: Requisitos, Engenharia de software, Framework Scrum, Engenharia de requisitos

## ABSTRACT

The requirements originate the functionalities and constraints of a system, it is through them that a system takes shape and can be constructed, however within the software engineering there are different methodologies for system development. Two of the processes that stand out are requirements engineering, known for the use of documentary methods and Scrum, which, because it is a framework based on agile processes, makes development more flexible, since it allows the assimilation of techniques from other methodologies. However, although they are fairly widespread and popular, their specific characteristics may leave gaps that allow for possible design flaws. However, both have the same purpose, the creation of a requirement which follows processes for effectiveness in a different way. Aiming to use Scrum artifacts targeted to requirements as benchmarks in this process, hypothetically insert a technique from an inflexible model into a flexible one, in order to make the functional product requirements durable at any stage of a software while remaining active.



**Keywords**: Requirements, Software Engineering, Scrum framework, Requirements Engineering

## LISTA DE FIGURAS

Figura 1 – Ciclo de Deming/ PDCA ..... 22
Figura 2 – Fluxo de processos do Scrum ..... 30
Figura 3 – Modelo de uma *user story*..... 40
Figura 4 – Modelo adaptado de uma *user story* ..... 40
Figura 5 – Modelo adaptado de um *theme*..... 41

## LISTA DE TABELAS

Tabela 1 – Etapas do ciclo de Deming/ PDCA…… 21

## LISTA DE GRÁFICOS

Gráfico 1 – Fatores de impacto sobre um projeto…….. 45
Gráfico 2 - Comparação de parâmetros sobre o gerenciamento de projetos…… 44
Gráfico 3 - Comparação do impacto sobre desenvolvimento …….. 48

## LISTA DE QUADROS

Quadro 1 – Modelos para um documento de requisitos..... 23
Quadro 2 – Modelo documentacional de uma *user story*..... 42
Quadro 3 – Modelo documentacional da *user story* [US002]..... 42
Quadro 4 – Definições dos campos da *user story* adaptada..... 43

# SUMÁRIO


1. INTRODUÇÃO ..... 11
2. ENGENHARIA DE SOFTWARE ..... 12
2.1. Métodos ágeis ..... 16
2.2. A engenharia de requisitos ..... 20
2.2.1. Requisitos de um sistema ..... 23
3. FRAMEWORK SCRUM ..... 27
4. ENGENHARIA DE REQUISITOS COM APOIO DA METODOLOGIA ÁGIL ..... 30
5. UTILIZAÇÃO DA ENGENHARIA DE REQUISITOS NO FRAMEWORK SCRUM ..... 33
5.1. Analise dos artefatos gerados no framework Scrum ..... 35
5.2. Aplicação das modificações propostas em artefatos geradores de requisitos do framework Scrum ..... 37
5.3. Impacto sobre as adaptações das users stories em desenvolvimento de software 43
6. METODOLOGIA ..... 48
7. CONCLUSÃO ..... 49
REFERÊNCIAS ..... 52
APÊNDICE A – MODELO PARA DOCUMENTO DE USERS STORIES ..... 57

# 1. INTRODUÇÃO

O desenvolvimento de sistemas é algo imprescindível na sociedade moderna, pois software está intrinsicamente ligado a equipamentos eletrônicos, ou seja, todo e qualquer eletrônico necessita de um sistema para manter-se ativo, desde os mais simples até os mais complexos, envolvendo as mais diversas áreas do conhecimento.

Podemos atribuir engenharia de software como a responsável pelos processos e técnicas que possibilitam a criação de um sistema com os recursos disponíveis e em um período de tempo relativamente compatível com o desenvolvimento. Para (PRESSMAN, 2011) a engenharia de software está relacionada a criação e o suporte de um software sendo amplamente utilizado pela indústria direta ou indiretamente. (SOMMERVILLE, 2011) complementa que a engenharia de software inclui técnicas que apoiam fases como especificações, projetos e evoluções, orientando assim a eficácia do desenvolvimento. Nos mostrando que constitui uma área ampla, onde podemos dividi-la em duas subáreas conhecidas como engenharia de requisitos e os métodos ágeis que trabalham de forma ligeiramente semelhante, porém são suas características especificas as tornam distintas.

Para (IEEE. *Recommended Practice for Software Requirements Specifications. New York*, 1998) a engenharia de requisitos é considerada um processo para obtenção, aprimoramento e verificação das necessidades de um cliente, tendo como meta uma especificação completa e correta dos requisitos de um software (NUSEIBEH; EASTERBROOK, 2000) ainda completa que esse processo é visto com finalidade de identificar o propósito através da identificação do *stakeholder*[1] e suas necessidades, documentar os requisitos de forma adequada para análise, comunicação e posterior implementação da solução. A engenharia de requisitos mostra como construir um sistema através das técnicas para levantamento e validação de requisitos, fornecendo artefatos de apoio em todas as fases do software.

Por outro lado, as metodologias ágeis trabalham em tempo de desenvolvimento, ou seja, permitem assimilar novos requisitos ao sistema sem comprometer o prazo, afinal não há preocupação em gerar documentação, tornando-a flexível e ágil em quaisquer fases do desenvolvimento. Para (DOS SANTOS SOARES, 2004) as metodologias ágeis são adaptativas em vez de preditivas. (DE REZENDE ALVES, 2010) menciona que o foco nas

[1] Pessoas, grupos ou entidades que tenham relações ou interesses diretos ou indiretos com ou na empresa. Cf. SILVA & GARCIA, 2011, p.03.

interações com pessoas cria novas condições para abordagem dos requisitos e algumas boas práticas são recomendadas para o sucesso das atividades.

O requisito é a base de um sistema, pois todos os processos da engenharia de software diretamente ou indiretamente levam a ele. Para (VENTURA, 2016) requisitos são o início de tudo, dando origem aos projetos de software. O autor ainda complementa que um requisito pode ser definido como uma necessidade, exigência, desejo ou solicitação. Assim notamos que ele é parte fundamental de um sistema é partir dele que um software é projetado, desenvolvido e evoluído.

Essa pesquisa visa mostrar hipoteticamente que a utilização do *framework* Scrum pode funcionar conjuntamente com a engenharia de requisitos para efetivar um dos produtos mais importantes da engenharia de software, o requisito. Permitindo que sua criação seja documentada utilizando a agilidade e flexibilidade do *framework* com a técnica de documentação da engenharia de requisitos. Visto que a engenharia de software trabalha com recursos e inovações para a construção de sistemas, assim vemos que evolução é uma constante nesse meio e trabalhar com aprimoramentos no desenvolvimento é necessário, buscando melhorias e aperfeiçoamentos internos na própria engenharia de software.

## 2. ENGENHARIA DE SOFTWARE

A engenharia de software surgiu como uma necessidade ao crescente meio do desenvolvimento de software, pois não havia um padrão para criação de sistemas e isso sempre causava problemas com a mantenabilidade e o próprio desenvolvimento do sistema, dessa forma padronizar era algo necessário e em meados dos anos 70 surgiu um movimento que deu início a engenharia de software formalmente dita e foi adquirindo força nos anos conseguintes, embora esse conceito tenha sido abordado muitos anos antes do seu surgimento, mas foi retomado quando existia de fato uma necessidade vigente de padrões, regras, processos e normas para o desenvolvimento de software. Segundo (IEEE. *Standard Glossary of Software Engineering Terminology, New York*. 1990) a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção podem ser definidos como engenharia de software. Onde através das formalizações cresceu e assimilou muitas extensões dando origem a composições de requisitos que visam normatizar padrões e métodos para elicitação e validação de requisitos.

A criação de sistemas torna-se cada vez mais necessária para o mundo moderno, visto que computadores e dispositivos necessitam de um software que o controle e execute todas as funcionalidades para qual foi projetado. A tecnologia se expande e torna-se presente no cotidiano social do homem moderno, sendo hoje indispensável para diversas atividades em diferentes áreas do conhecimento como medicina, engenharia, administração, dentre outras. Softwares controlam máquinas que por sua vez auxiliam nessas áreas e são extremamente importantes em pesquisas e diagnósticos, como combate a doenças através de mapeamento genético e realização de exames, softwares também auxiliam na comunicação seja ela interna a um empresa ou instituição de ensino de um país ou mesmo de um país a outro como investimento em bolsas de valores que é algo indispensável economicamente, assim como o controle de transportes; como exemplos desses meios de transportes temos os aviões, metrôs e até mesmo navios. Software em sua essência está vinculado ao mundo moderno e dessa forma é preciso buscar sempre maneiras de aprimorar o desenvolvimento de sistemas através de regras que permitam agilizar o processo de desenvolvimento, porém, sem perder a qualidade e a segurança que são fatores indispensáveis em um software, afinal um software sem essas características não seria rentável ou mesmo confiável para serem utilizados.

A engenharia de software é a responsável por arquiteturas e regras que auxiliam na criação de software enumerando técnicas e métodos que não só possibilitam a criação de um sistema com qualidade, como também na economia de recursos, pois como toda indústria minimizar os gastos sem perder qualidade é de fundamental importância para que um sistema seja adaptável aos recursos disponíveis. Para (COSTA, 2014) uma arquitetura deficiente pode refletir organizações mal definidas com projetos ineficiências, o que consequentemente causa má comunicações e decisões precipitadas. Assim sendo a engenharia de software possui diversas maneiras para criar software desde algo que permita criar software de forma rápida até algo mais formalizado que permita um software ter um ciclo de vida consideravelmente grande aplicando as técnicas da engenharia de requisitos para fase inicial do desenvolvimento, entretanto devido a inúmeros sistemas terem uma certa semelhança em sua estrutura de dados ainda é possível utilizar reuso para se refatorar um sistema de acordo com o desejado pelo cliente, onde pode ou não usar partes de um outro sistema para se refazer um novo, caso o software em questão torne-se obsoleto, ou seja, ultrapassado para a tecnologia atual vigente.

Na sociedade moderna sistemas no geral tornam-se de total importância, pois provem o funcionamento de equipamentos necessário para que seu propósito de funcionamento ocorra da maneira admissível. Segundo (SOMMERVILLE, 2011) o mundo moderno não poderia existir sem software. Afinal todo e qualquer aparelho eletrônico que desempenhe funções

automatizadas irá necessitar de um programa que controle o seu funcionamento. Na sociedade contemporânea sinais de trânsito, dispositivos móveis, controle de tráfego aéreo ou terrestre, GPS, dentre outros são controlados por um software que orienta todas as funções e os mantém ativos, afim de permitir que cotidiano social siga de maneira fluída. Outro autor que menciona a importância de um software é (PRESSMAN, 2011) onde ele menciona que "software distribui o produto mais importante da nossa era, a informação", seguindo a linha desse pensamento podemos concluir que software está presente em uma das áreas da modernidade mais indispensável para o funcionamento do planeta, os aparelhos eletrônicos voltados a informação, visto que nesse estágio do desenvolvimento humano a comunicação é vista como o meio mais importante seja de comércio como o mercado de ações que negocia bilhões em dinheiro todos os dias ou da difusão da informação como telejornais, transmissões de rádio, até mesmo pela Internet, dentre outros, todos eles utilizam softwares computacionais que prezam a segurança e a qualidade de suas ações, seja ela para meio externo ou interno, o que demanda desenvolvimento para a computação, ou seja, prazos serão estipulados e deverão ser cumpridos.

A gerencia de como deve ser distribuído o tempo de desenvolvimento pode ser destacada no *framework* Scrum enquanto a elicitação de requisitos e sua validação pode ser expressa pela engenharia de requisitos. Todavia a composição da engenharia de requisitos demanda uma etapa inteira apenas para o processamento dos requisitos, afim de compor a documentação do sistema o que pode demandar bastante tempo do prazo de entrega. Contudo essa etapa é uma das mais cruciais para o desenvolvimento de sistemas, pois será a partir dela que todas as informações necessárias estarão presentes para a implementação de software. Entretanto analisando a necessidade do desenvolvimento de maneira mais flexível um grupo de desenvolvedores renomados da engenharia de software, criou técnicas e métodos que economizariam tempo para a grande demanda de construção de software que estava cada vez mais crescente, essas técnicas ficaram conhecidas como metodologias ágeis baseadas no manifesto ágil[2]. Dessa maneira o campo das metodologias ágeis exige da construção de um software menos tempo do que o prazo que seria estipulado pela engenharia de requisitos, o que é uma grande vantagem para o desenvolvimento de sistemas.

Nesse meio surgiu o Scrum ainda visto por muitos como uma metodologia ágil, porém, o Scrum não é baseado em processos ou técnicas o que não o caracteriza como metodologia, mas o Scrum permite que sejam acoplados em seu escopo diversos processos e

[2] Disponível em: <http:// www.manifestoagil.com.br> Acesso em: 12 fev, 2018.

técnicas para permitir que seja adaptado de uma forma mais personalizada a maneira de desenvolver o que o torna flexível o definindo como um *framework* de desenvolvimento.

Muitos dos processos e métodos que a engenharia de software possui são divididos entre a engenharia de requisitos (ER) e nas metodologias ágeis que permitem a quem as use o desenvolvimento de sistemas de qualidade e seguros, além do mais dependendo da forma que está sendo realizada a construção de um software pode ser realizada mais rapidamente, o que necessariamente não é uma desvantagem para a engenharia de software, porém, pode prejudicar a rastreabilidade de requisitos assim como a mantenabilidade futura para sistemas com ciclo de vida grande, já que não necessariamente será a equipe que desenvolveu o software que irá prestar manutenção.

Desenvolvimentos de sistemas em engenharia de software através de seus processos e métodos resultam em produtos denominados artefatos, que podem ser definidos como todo e quaisquer produtos de trabalho que forneçam informações com descrições e definições tangíveis. Artefatos fornecem meios para os quais sejam possíveis identificar e compreender funções pertinentes ao sistema, porém, nem todo artefato gerado pode ser considerado um documento.

Além do que a área correspondente ao desenvolvimento baseado em flexibilidade trabalha com equipes distribuídas o que pode prejudicar a inserção de novos membros durante o desenvolvimento de um sistema, pois iriam necessitar de um conhecimento prévio sobre as técnicas e métodos de aplicação para aquele sistema antes de começar a contribuir para a construção do software, no Scrum são utilizados *users stories*, *epics* e *themes* para gerar os requisitos de um sistema através de *stories*. Contudo são gerados durante o desenvolvimento sendo considerados artefatos temporais, visto isso são descartados a cada etapa finalizada.

Segundo (BECK, 2004) uma *story* é uma unidade funcional de planejamento visível para usuário/cliente. Dessa forma permite que o cliente possa interagir ativamente no projeto podendo apontar quais pontos prioritários devem ser implementados inicialmente, neutralizando possíveis ambiguidades que venham a surgir nos requisitos. Todavia minimizando uma das maiores causas de falha em projetos de softwares, a falta da participação dos usuários. Os *epics* são considerados grandes *users stories*, ou seja, demanda uma carga maior de trabalho e a partir dele podem surgir outras *stories*, um *epic* geralmente é determinada pela falta de informações suficientes que permitam serem desenvolvidas individualmente. Enquanto os *themes* são coleções de *stories* com a mesma importância, ou seja, um conjunto de *stories* que realizem tarefas similares.

Assim sendo as equipes que desenvolvem utilizando a engenharia de requisitos tendem a possuírem equipes fixas e com uma taxa muita baixa de flexibilidade em todo projeto, pois a utilização desse processo deve indicar que o projeto está bem definido e não está suscetível a mudança, afinal no gerenciamento que utiliza a ER existe um prazo que deve ser cumprido e é basicamente inalterado, todavia se houver alguma eventualidade que acarrete em alguma alteração durante o projeto é possível que exista um atraso em todo o desenvolvimento. Além do que pode ou não haver a negociação de um novo prazo, fazendo dessa consequência uma desvantagem que pode prejudicar o projeto inteiro, assim como os integrantes e os interessados. Por outro lado, a flexibilidade dos métodos ágeis indica que mudanças podem ocorrer em qualquer momento nos requisitos, possibilitando que melhorias possam ser feitas, entretanto, a falta de especificações normativas para os requisitos pode trazer compleições em alguma fase do projeto. Logo encontrar dentro da engenharia de software a técnica de desenvolvimento ideal é o primeiro passo para estabelecer regras de negócio e gerencia que permitam o desenvolvimento sem perdas para o software e todos os envolvidos no projeto.

## 2.1. Métodos ágeis

As metologias ágeis surgiram na década de 1990 pela necessidade de construir software de forma mais rápida eliminando a etapa da documentação do sistema, pois sua base de construção é voltada interações com pessoas, ou seja, preza-se pela interação entre a equipe de desenvolvimento e o *stakeholder*. As metodologias ágeis são motivadas pelo manifesto ágil que define parâmetros de como devem ser as práticas ágeis fundamentadas pelos engenheiros de softwares mais conceituados da época. Para (FERNANDES, 2013) ela objetiva propiciar a comunicação face a face ao invés de inúmeros documentos, ele ainda cita duas qualidades para aplicação dos métodos ágeis, o trabalho com pequenas equipes e projetos que necessitam de várias mudanças nos requisitos. As metodologias ágeis possuem características distintas que as tornam bastantes popularizadas , porém, todas tem o mesmo propósito criar sistemas de maneira rápida e flexível, as metodologias ágeis mais utilizadas entre as equipes de desenvolvimento são *Extreme Programming* (XP) e Scrum que embora tenha sua base originalmente fundamentada em métodos ágeis não é mais considerada uma metodologia e sim um *framework* de desenvolvimento, pois sua estrutura permite assimilar

técnicas e processos de outras metodologias que não comprometam a flexibilidade do desenvolvimento.

As equipes que utilizam metodologias ágeis por consequência de seus métodos e técnicas são mais maleáveis quanto a mudanças e a prazos, pois é trabalhado de forma que haja sempre interações no desenvolvimento com todos os envolvidos no projeto, em outras palavras "orientação a pessoa" termo utilizado por (DARWISH; MEGAHED, 2016), dessa forma as interações entre os membros da equipe de desenvolvimento e o *stakeholder* é constante, ou seja, sempre há uma pequena reunião com o cliente para definir, alterar ou testar requisitos no sistema. Nesse tipo de metodologia por prezar pela interação entre os envolvidos a documentação do sistema é um item dispensável para o projeto e dessa maneira o *stakeholder* explora as opções diretamente com o responsável por estabelecer os requisitos para o sistema chamado de *Product Owner* e fixa o que almeja ser desenvolvido o que pode ocorre não em apenas uma etapa do desenvolvimento, como também durante toda construção do software. Para (CARNEVALI, 2014) uma preocupação de gerar uma documentação é o excesso de papéis que são gerados com documento que muitas vezes nunca serão lidos.

Dialogar de forma regular com o cliente é uma das grandes vantagens dessa metodologia ao desenvolvimento, pois se comparada a engenharia de requisitos que geralmente tem a fase inicial com seu *stakeholder* sendo a mais ativa, os métodos ágeis por outro lado podem suprir essa carência. Visto que a presença do *stakeholder* ocorre de forma constante do início até o fim do desenvolvimento. Além disso nem todo *stakeholder* possui conhecimento em programação e dessa maneira muitas vezes o que ele quer pode não ser algo viável para ser concluído no prazo estipulado, contudo cogitar a possibilidade de deixar para versões posteriores a desenvolvida pode não ocorrer exatamente como ele gostaria no momento que pensou na sua ideia ou ainda nem ocorrer, afinal não há a criação de um artefato que irá ser utilizado a longo prazo, mas sim um artefato temporal que será descartado ao fim de uma etapa. Onde durante esse período de construção a ideia proposta pode mudar ou ser esquecida, pois o sistema pode solucionar a necessidade principal do *stakeholder*, porém, talvez não solucionar as demais consecutivas, perdendo o conhecimento de uma proposta que antes poderia ser utilizada para resolver as necessidades com incrementos conseguintes a versão desenvolvida.

Visto que a memória humana é algo considerada volátil e informações caso não sejam bem fixadas podem ser perdidas, o que pode ser um problema na área da programação que trabalha com informação para desenvolver softwares, afinal tudo se origina de uma ideia que está armazenada na memória de curto prazo, para (DIVIDINO, 2013) a Memória de Curto

Prazo possui duração de armazenamento limitado, onde ele ainda completa que a cada instante que passa, um item nela armazenado se torna mais fraco até por fim desaparecer. Assim podemos concluir que uma memória sendo uma ideia é frágil, pois está armazenada em local volátil e se não contextualizada pode mudar ou ser esquecida.

Todavia essa forma de construção de software tão direta pode minimizar inclusive o tempo de entrega do software por fazer o *stakeholder* explanar opções de funcionalidades em período de desenvolvimento, podendo solicitar a incorporação de outras funções ou mesmo a exclusão de outras, dessa forma através de prototipagem o *stakeholder* pode testar as funcionalidades do programa sem necessariamente esperar o prazo de entrega chegar ao fim. Com essa abordagem é possível minimizar a má compreensão da descrição de cada requisito eliminando ambiguidades que poderiam ser implementadas o que evita o desperdício do prazo entrega. Entretanto por ser uma forma mais flexível de desenvolvimento é plausível perceber as evoluções que o sistema possui durante todo desenvolvimento, sem necessariamente seguir algo linear quando se compara com a ER, o que pode continuar a ser uma vantagem como também pode ser uma desvantagem, visto que a parte documentacional desse dessa forma de desenvolvimento é simplesmente o código o que pode ser confuso para prestação da manutenção ou consultar eventuais informações sobre o sistema que foram mencionadas em etapas anteriores ao desenvolvimento, porém, nada impede de que haja a criação de artefatos do sistema.

O Scrum baseia seus princípios para gerencia das atividades de desenvolvimento em nos métodos ágeis o que não o torna necessariamente uma metodologia, entretanto foi fundamentado com mesmos princípios do manifesto ágil, porém, é considerado um *framework* por agregar em seu escopo técnicas e processos de outras metodologias. Segundo (PRESSMAN, 2011, P.95) o Scrum incorpora atividades estruturais como requisitos, análise, projeto, evolução e entrega. (BASSI, 2008, p. 21) afirma que o Scrum pode ser expresso da seguinte forma, “caracteriza-se como um processo empírico e adaptativo”. Podemos perceber que o autor cita processos existentes que assimilam características de um outro modelo, pois mostra a flexibilidade no desenvolvimento, afim de realizar mudanças. Suas iterações baseiam-se fortemente no ciclo de Deming também conhecido como PDCA que pode ser visto na Figura 1 e as definições das etapas podem ser vistas na Tabela 1.

O ciclo de Deming/PDCA é visto como ciclo de melhoria continua, pois em qualquer uma das etapas é possível integrar melhorias em todo processo que esteja sendo aplicado. O autor ainda completa que devido a forma similar de funcionamento desse modelo com os métodos ágeis o Scrum por ter princípios ágeis integra algumas etapas em seu escopo onde

elas podem ser definidas como; Planejamento, *Sprint* e avaliação o que corresponderiam as etapas iniciais do processo, além do mais a etapa de ação também acontece de modo implícito no Scrum com as alterações no próximo planejamento baseada nas terminações da etapa de avaliação. A fase do planejamento ocorre durante todo desenvolvimento, contudo o primeiro planejamento tem como objetivo conhecer as necessidades do produto de forma satisfatória e assim evoluir de acordo com as necessidades do *stakeholder*. As etapas no Scrum são denominadas *Sprints*, e cada *Sprint* utiliza padrões de processos definidos inicialmente para executá-las de modo eficiente e rápido e a fase de avaliação também acontece durante todo desenvolvimento é a partir das conclusões que se cria um novo planejamento e evolui o produto, dessa forma não é preciso seguir o desenvolvimento de maneira sequencial, pois permite retornar a qualquer uma das fases sempre que necessário. Segundo (CARNEVALI, 2014) uma *sprint* pode possuir um prazo de trinta dias, mas devido a flexibilidade dos processos ter esse prazo alterado para mais ou para menos e ao se estabelecer tarefas as mesmas não podem exceder o período de uma semana, porém, não é uma regra, podendo assim ser alterado isso garante qualidade no desenvolvimento, pois permite quebrar grandes tarefas em outras menores tornando o trabalho flexível para aquela faixa de tempo.

Tabela 1 - Etapas do ciclo de Deming/ PDCA

| Etapa | Objetivos |
|---|---|
| **Planejamento** | Definir objetivos; estabelecer um caminho para atingir os objetivos; definir padrões; escolher métodos e ferramentas. |
| **Execução** | Treinar e aprender os métodos escolhidos; executar o planejado; coletar dados para avaliação. |
| **Verificação** | Verificar se os padrões estão seguidos; analisar os dados coletados; avaliar se os objetivos foram atingidos. |
| **Ação** | Investigar as razões dos problemas; se necessário, tomar ações para atender os padrões estabelecidos; corrigir os problemas e tomar ações para evitar que eles reapareçam; identificar maneiras de |

| | |
|---|---|
| | melhorar o processo de trabalho. |

Fonte: BASSI (2008)

Figura 1 - Ciclo de Deming / PDCA


Fonte: BASSI (2008)

O Scrum assimila formas de desenvolvimento que o permite ser personalizado de modo que se adeque as necessidades de todos os envolvidos no projeto, tornando-o um *framework* popular e bastante utilizado atualmente. Para (BISSI, 2007) utiliza-lo pode gerar benefícios; como diminuição dos riscos, maior integração entre os membros das equipes, rápida solução de problemas, progresso medido continuamente, os clientes se tornam parte da equipe de desenvolvimento, entregas frequentes de funcionalidades funcionando, discussões diárias de status com a equipe, os profissionais de negócios e tecnologias trabalham juntos.

## 2.2. A engenharia de requisitos

A engenharia de requisitos veio como forma de facilitar a compreensão dos requisitos em um sistema diante do desenvolvimento de artefatos gerados durante na primeira fase do desenvolvimento de um software, ou seja, no planejamento afim de estabelecer quais serão as funcionalidades que estarão presentes no sistema. Segundo a norma (IEEE. *Recommended Practice for Software Requirements Specifications. New York*, 1998) engenharia de requisitos

é definida como o processo de aquisição, refinamento e verificação das necessidades do cliente para um sistema de software, objetivando-se ter uma especificação completa e correta dos requisitos de software. (FRANCESCHI; DUARTE, 2011) completa que a engenharia de requisitos por ser um subprocesso da engenharia de software tem o objetiva identificar, analisar, documentar e validar os requisitos de um sistema. Dessa forma podemos notar que um sistema deve ter seu escopo bem detalhado com informações necessárias para que todas as funções desejadas sobre um produto sejam compreendidas de forma correta.

Segundo (ALVES, 2008) a elicitação de requisitos é um processo engloba o adequado entendimento da organização e de seu processo de negócios, assim sendo é uma das fases mais importantes do desenvolvimento de software, pois será através dela que serão conhecidas as necessidades do cliente a serem moldadas, embora trabalhosa e consuma um tempo considerável do prazo de desenvolvimento, é constituído pela construção dos artefatos necessários para documentação de requisitos de um sistema. Para (VALASKI, 2013) a que a elicitação dos requisitos necessita de habilidade para discutir problemas e por meio deles projetar as soluções computacionais. Um processo que exigem interpretações exatas, almejando a eliminação de qualquer ambiguidade que esteja presente nos artefatos.

Dentre os artefatos gerados na fase de concepção de um sistema usando as bases da ER podem estar inclusos o documento de requisitos, os diagramas, os casos de uso e ainda podendo ser apoiado por gráficos e tabelas para demonstrar desempenho ou delimitar prazos. Contudo nessa pesquisa daremos foco apenas ao documento de requisito, mostrando a importância sobre os requisitos no sistema.

O documento de requisitos delimita os recursos que o sistema irá utilizar baseado no ambiente no qual será inserido, isso permite que a equipe possa desenvolver apenas o que é desejado pelo cliente sem demandar mais tempo e recursos. (IEEE Std 830-1998; VOLERE, 2012) citam modelos estruturais de como uma documentação de requisitos deve ser seguida, afinal para se criar este artefato deve-se ao menos definir um glossário para tal finalidade, podemos ver esse modelo nas partes destacadas em azul no Quadro 1.

Quadro 1 – Modelos para documento de requisitos

| **Volere** | **IEEE-830** |
|---|---|
| DIRETIVAS DO PROJETO<br>1. O Propósito do Projeto | 1. INTRODUÇÃO<br>1.1 Propósito |

| | |
|---|---|
| 2. Os *Stakeholder*s | 1.2 Escopo |
| RESTRIÇÕES DO PROJETO | 1.3 Definições, acrônimos e abreviações |
| 3. Restrições Obrigatórias | 1.4 Referências |
| 4. Convenções de Nomenclatura e Terminologia | 1.5 Visão geral |
| 5. Fatos Relevantes e Premissas | 2. DESCRIÇÃO GLOBAL |
| REQUISITOS FUNCIONAIS | 2.1 Perspectivas do produto |
| 6. O Escopo do Trabalho | 2.2 Funções do produto |
| 7. Modelo de Dados Corporativos e Dicionário de Dados | 2.3 Características dos usuários |
| 8. O Escopo do Produto | 2.4 Restrições |
| 9. Requisitos Funcionais | 2.5 Suposições e dependências |
| REQUISITOS NÃO FUNCIONAIS | 3. REQUISITOS ESPECÍFICOS |
| 10. Exigências de aparência e sensibilidade | APÊNDICES |
| 11. Requisitos de Usabilidade e Humanidade | |
| 12. Requisitos de Desempenho | |
| 13. Requisitos Operacionais e Ambientais | |
| 14. Requisitos de manutenção e suporte | |
| 15. Requisitos de segurança | |
| 16. Requisitos Culturais | |
| 17. Requisitos Legais | |
| QUESTÕES DO PROJETO | |
| 18. Questões Abertas | |
| 19. Soluções Off-the-Shelf | |
| 20. Novos problemas | |
| 21. Tarefas | |
| 22. Migração para o novo produto | |
| 23. Riscos | |
| 24. Custos | |
| 25. Documentação e Treinamento do Usuário | |
| 26. Sala de Espera | |
| 27. Ideias para Soluções | |

Fonte: VOLERE (2012); IEEE Std 830-1998

Equipes que utilizam técnicas da ER para desenvolver tendem a ter um prazo de entrega maior do que equipes que trabalham com metodologias ágeis, pois nessa parte da engenharia de software o foco inicial está voltado para rastreabilidade dos requisitos necessários para o sistema, o que demanda um pouco mais de tempo para desenvolvimento, afim de focar na criação de artefatos validos que ajudem a compor a documentação do sistema. Dessa forma a equipe de desenvolvimento terá material de consultar e reconhecimento caso algum membro da equipe seja substituído durante o processo de construção de software, embora seja pouco provável por se tratar de equipes fixas.

Além do mais a documentação poderá ajudar em possíveis incrementos que o sistema possa ter ao longo do seu ciclo de vida, caso não tenha havido prazo suficiente para implementação de todas as metas, sendo assim alocadas para versões posteriores. Levando em consideração que as funcionalidades deixadas não iriam comprometer o fluxo principal do sistema. O documento de requisitos ajuda a identificar em um requisito as definições em que se baseia sua implementação, isso ajuda aos desenvolvedores a minimizar o tempo de conhecimento sobre o sistema, contudo devemos considerar que a equipe de desenvolvimento seja diferente do original.

A engenharia de requisitos define formas para construir um documento de requisitos claro sobre um sistema, o que é utilizado durante todas as fases de desenvolvimento inclusive após, mostrando a importância desse artefato em software, pois plataformas de sistemas podem evoluir e necessitam que sistemas a acompanhem para fins de compatibilidade, e o conhecimento de um requisito auxilia nessa temática, tornando esse artefato um dos mais importante da engenharia de software.

### 2.2.1. Requisitos de um sistema

Os requisitos são o alicerce de qualquer projeto de software, pois serão a partir deles que um software irá obter forma. Segundo apresenta (SOMMERVILLE, 2011) os requisitos são considerados descrições do que o sistema deve fazer, assim como os serviços que oferecem e as restrições a seu funcionamento. Todavia seguindo essa linha de raciocínio podemos observar que requisitos não podem ser tratados de formas iguais para todos os casos, visto que desempenham papeis diferentes em diversas partes de um sistema, dessa forma vemos que requisitos podem ser divididos em duas classes: requisitos funcionais (RF) e os requisitos não funcionais (RNF).

Requisitos funcionais descrevem as funcionalidades presentes no sistema, ou seja, as funções que normalmente ficam visíveis como interações entre usuário e sistema, entretanto não sendo restrito a apenas esse tipo de ação. Segundo (SOMMERVILLE, 2011, p.59) define que um RF são como declarações de serviços que o sistema deve fornecer, além de como o sistema deve reagir com entradas especificas e de como o sistema deve se comportar em determinadas situações, ou seja, no fluxo principal que é para o qual o sistema foi projetado sem esperar nenhuma anormalidade e os fluxos alternativos que hipoteticamente simulam um resultado que o sistema não deveria realizar. Dessa forma fica claro que RF é toda e qualquer funcionalidade que o sistema possuirá, desempenhado a partir de uma entrada que foi proposta pelo cliente durante a abordagem dos requisitos, afim de uma saída que eventualmente é o esperado que o sistema faça.

Requisitos não funcionais são os que delimitam por onde o sistema deve iniciar e finalizar, segundo (SOMMERVILLE, 2011) os RNF são as restrições aos serviços ou funções oferecidas pelo sistema. RNF podem se dividir em categorias, afim de melhor compreender se comportam e como cada tipo pode fornecer um serviço apropriado ao sistema, restringindo mais claramente o que se deseja ao software para que sejam eliminadas as ambiguidades dos requisitos na documentação. Podem ser divididos em; requisitos de usabilidade, requisitos de confiabilidade, requisitos de desempenho, requisitos de reusabilidade, requisitos de segurança, requisitos de padrão, requisitos legais e requisitos de interoperabilidade o que fornece para cada tipo uma peculiaridade diferente sobre o sistema.

Segundo (SILVA, 2008, p.6-11) as definições que representam cada tipo de requisito RNF estão descritas como requisitos de usabilidade, “especificam tanto o nível de desempenho quanto a satisfação do usuário no uso do sistema”, ou seja, a interações entre humano e máquina. Requisitos de confiabilidade, “compreendem restrições sobre o comportamento do sistema de software em tempo de execução” esse tipo de requisito mostra o comportamento do sistema dada uma entrada específica em um determinado período de tempo. Requisito de desempenho, “restringe a velocidade de operação de um sistema de software” o que interfere diretamente na usabilidade, pois fatores como tempo de resposta pode prejudicar a produtividade que se espera do sistema. Requisito de reusabilidade “determinará quão fácil será conseguir componentes reutilizáveis e a interdependência ou acoplamento entre os componentes” o que auxilia na redução de retrabalho para novos projetos, podendo utilizar componentes de um sistema antigo para implementação de um novo. Requisito de segurança “assegurada a integridade do sistema quanto a ataques intencionais ou acidentes” dessa forma é possível garantir o bloqueio de acessos não

autorizados ao sistema, além do que esse tipo de requisito é fundamental para qualquer sistema e essencial para sistemas considerados críticos como sistema de controle de voo ou mesmo sistema de controle para dispositivos médicos, visto que é impossível confiar em um sistema dessa natureza se ele não for seguro.

Um outro autor ainda acrescenta outros tipos de RNF, (VENTURA, 2016, p.26) define os seguintes tipos de requisitos: requisitos de padrão, "padrões aplicáveis ao software e ao projeto: [...], metodologia para desenvolvimento do sistema, padrões de projeto (*design patterns*) a serem aplicados, padrões arquiteturais" define assim se um sistema deve ser fraco ou fortemente acoplado, prezando pela separação de cada atividade do sistema, onde necessariamente uma função não dependa muito de outra para executar seu serviço. Requisitos legais, "exigências de conformidade do software com alguma legislação pertinente ao projeto" aborda o atendimento a alguma norma nacional quanto a criação de produtos específicos para alguma área, por exemplo, temos no Brasil legislações que devem ser seguidas para softwares que serão utilizados dentro de hospitais, pois é uma área que trabalha com vidas e dessa forma deve ter um rigoroso controle quanto à qualidade do produto no nosso caso software. Requisitos de interoperabilidade "integração do sistema com outros sistemas, integração com APIs[3], componentes, banco de dados externos" descrevendo assim como o sistema deve se comportar ao ser acoplado com outros sistemas e quais condições serão necessárias para isso acontecer, ou seja, a compatibilidade entre eles. Esses são os principais tipos de requisitos encontrados no mercado durante o desenvolvimento de um software, todos possuem seu significado de importância no processo de construção de sistemas, porém alguns tipos de requisitos podem ganhar maior importância do que os demais dependendo da natureza do software a ser desenvolvido.

Baseando-se nessas definições podemos notar que um sistema é estruturado com diferentes tipos de requisitos, onde cada um deles é tratado de forma apropriada a sua natureza, dessa forma organizar e agrupar requisitos pode levar a uma ampliação da eficácia em fases de desenvolvimento. Uma forma apropriada para organizar os requisitos a longo prazo seria utilizando documentação, que define e estrutura de modo formal um requisito e suas dependências.

A documentação de requisitos já se provou auxiliadora no desenvolvimento de software, pois permite visualizar as dependências no sistema entre os requisitos, além de facilitar a rastreabilidade e gerir possíveis mudanças que venham a surgir, porém, temos que

[3] Interface de Programação de Aplicativos (*Application Programming Interface*)

ter em mente que se o gerenciamento de projeto for baseado em engenharia de requisitos após sua validação um requisito normalmente ele não seria alterado a menos que o cliente queira muda-lo, entretanto é algo que iria comprometer o prazo estipulado no escopo do desenvolvimento. A flexibilidade a mudanças é uma característica imposta apenas as metodologias ágeis, onde o Scrum foi incialmente criado. A falta de uma documentação adequada pode afetar todos os pontos de um sistema podendo dificultar o desenvolvimento do software mesmo com a interação constante com *stakeholder*. Para (ESPINOZA; GARBAJOSA, 2011) a gerência de forma imprópria, pode ocasionar problemas sérios para outros processos como o gerenciamento de mudanças, análise de impacto e estimativa, que se baseiam em um modelo de ciclo de vida originado de um requisito habitual a um processo de análise, podemos perceber através da citação desse autor que a ausência de algum componente no desenvolvimento pode comprometer não apenas aquela função como também o sistema inteiro o que pode o acarretar em uma situação inesperada ao usuário do sistema mesmo tendo explorados cenários alternativos.

A existência de cenários pode variar de acordo com o que o usuário utilize no sistema, dessa forma é considerado não apenas o cenário principal como também os alternativos, contudo nem todos os cenários alternativos são mapeados o que pode comprometer o software e eventualmente tornar ainda mais difícil a sua mantenabilidade, levando em consideração que a técnica aplicada tenha sua vertente baseada em metodologia ágil.

Mesmo o desenvolvimento ágil gera documentação, porém nem sempre é suficiente para suprir as necessidades que podem surgir sobre a rastreabilidade dos requisitos. Segundo (TRINDADE; LUCENA, 2016) priorizar interações entre pessoas ao invés de processos e ferramentas leva a uma falta de documentação de requisitos formais, dificultando a engenharia de requisitos, esse processo mencionado pelos autores orbita em torno das premissas de elicitação seguido da validação para um requisito que é primordial para que um requisito se torne efetivo. Podemos perceber que que alguns autores citam que a falta de documentação dos requisitos formal pode muitas vezes prejudicar o projeto como um todo, pois existem fatores de risco que não seriam são abordados. Todavia no âmbito da programação a mudança pode trazer a perda de informações e devem ser documentadas de maneira adequada, afinal um software pode assimilar ao longo de sua operabilidade um ciclo de vida grande necessitando de informações sobre sua estrutura e granularidade.

A documentação de requisitos é um artefato necessário e pode ser necessária inclusive após o desenvolvimento, pois durante a construção de um software pode haver a substituição de um membro da equipe e dessa forma a integração de um novo membro na equipe irá

demandar trabalho extra, já que o novato terá de possuir conhecimento do sistema que irá ser assimilado no decorrer de sua estadia no meio alocando trabalho para outros integrantes da equipe fazendo com que algumas fases sejam menos exploradas que outras. Para isso na ausência da documentação de requisitos pode significar prejuízo sobre o prazo.

Visto que no ramo do desenvolvimento de software existe uma rotatividade consideravelmente grande de pessoal a migração pode ocorrer pode diversos fatores, porém, esse fato torna-se um problema, afinal a substituição de um integrante por outro sem conhecimento prévio do sistema irá acarretar consequências. Segundo (ANG; SLAUGHTER, 2004) as políticas de recursos humanos das empresas e o desequilíbrio entre a baixa disponibilidade e a crescente demanda de profissionais são fatores que influenciam os profissionais a migrarem para outras organizações, causando o *turnover*, (AMBRÓSIO, 2008) ainda complementa que o *turnover* contribui para aumentar a quantidade de erros de requisitos, visto que o esforço de trabalho seja maior para algumas atividades e menores para outras podendo prejudicar inclusive a qualidade do software, o autor afirma que devido a prática a taxa de probabilidade de se cometer erros na especificação de um requisito fica entre 5% e 20% e a de alteração entre 10% e 30%. Esses valores podem ser determinantes para que se ocorra a falha em um projeto, afinal requisitos estruturam um sistema e se sua mudança é afeta o sistema consequentemente afetará o projeto como um todo.

A fundamentação de um requisito permite que a composição de um sistema seja possível e a sua ausência pode tornar a implementação inexistente ou mesmo a falta de formalização com artefatos geradores de requisitos com uma estruturação malformada pode acarretar em falha crítica no projeto de software. Todavia vemos que um requisito deve possuir informações suficientes para criação e validação, afim de que seja implementado corretamente. Assim sendo, a importância de um requisito pode durar inclusive após o desenvolvimento na fase de manutenção que pode incluir a inseminação de novas funcionalidades e a evolução das existentes, afetando o modo de como o usuário interage com o sistema.

# 3. FRAMEWORK SCRUM

O Scrum é um *framework* bem popular e bastante utilizando, pois permite agregar em seu escopo muitas possibilidades metodológicas para se dá início ao desenvolvimento de sistemas, afinal possui funções bem definidas quanto a equipe que o utiliza, sendo descritos

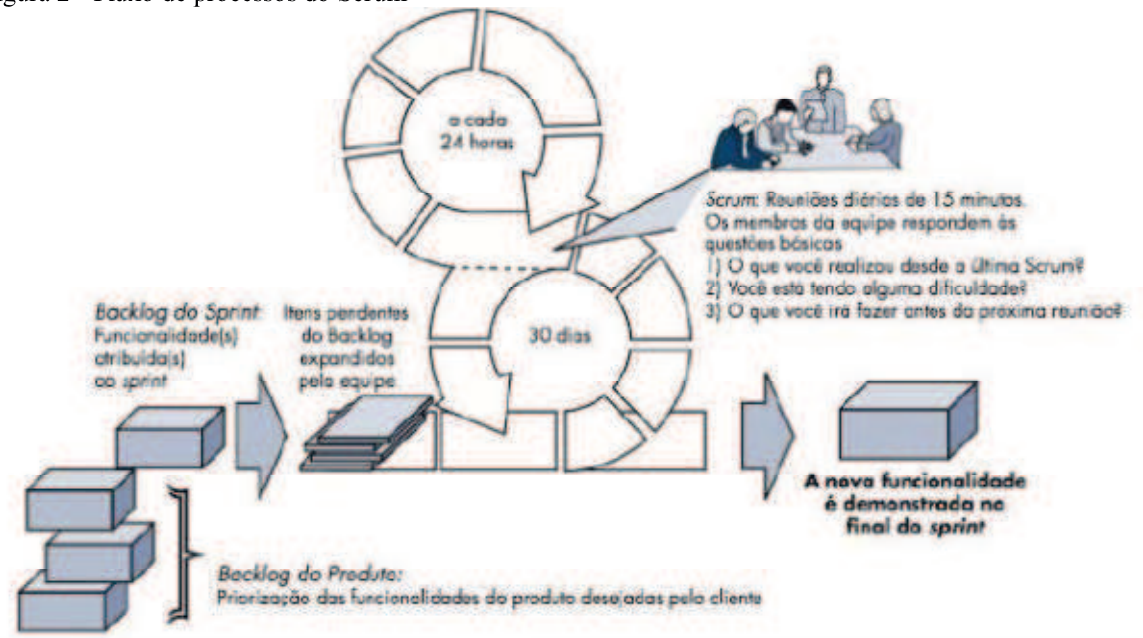
como: Time Scrum, *Scrum Master*, *Product Owner*, Time de Desenvolvimento, *Product Backlog*, *Sprint Backlog*, Revisão da *Sprint*, *Sprint retrospective*.

A cada um dos papeis tem funções especificas que tornam o Scrum uma metodologia completa para o desenvolvimento de software, segundo (SCHWARZ; SUTHERLAND, 2013) o Time Scrum é composto pelo Product Owner, o Time de Desenvolvimento e o Scrum Master, [...] entregam produtos de modo iterativo e incremental, maximizando as oportunidades de realimentação do projeto, fazendo com que o projeto tenha sempre melhorias que o tornem eficiente e com qualidade. O Scrum *Master* é o responsável por garantir que o Scrum seja compreendido e aplicado, ou seja, ele é o gerente que comanda todos os processos do desenvolvimento no Scrum. *Product Owner* o dono do produto, é o responsável por maximizar o valor do produto e do trabalho do Time de Desenvolvimento, no Scrum esse papel é atribuído ao nosso *stakeholder*, em outras palavras são os interessados no projeto. Time de desenvolvimento, consiste de profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto ao final de cada *Sprint*, ou seja, são apenas os integrantes do time que criam incrementos que podem ser aplicados ao projeto. *Product Backlog* é uma lista ordenada de tudo que deve ser necessário no produto, e é uma origem única dos requisitos para qualquer mudança a ser feita no produto, em resumo seria uma lista de todos os requisitos, funções, melhorias e correções, ou seja, todas as características do sistema, pois o produto no Scrum é algo dinâmico e está em constante evolução. *Sprint Backlog* é um conjunto de itens do Backlog do Produto selecionados para a *Sprint*, juntamente com o plano para entregar o incremento do produto e atingir o objetivo da *Sprint*, ou seja, um plano para mudanças durante as reuniões diárias permitindo acréscimos e mudanças na *Sprint* fornecidos pelos Time de Desenvolvimento. Revisão da *Sprint* é executada no final da *Sprint* com finalidade de inspecionar o incremento e adaptar o *Product Backlog* se necessário, ao contrário das reuniões diárias que duram alguns minutos essa revisão possui um tempo maior de duração sendo definidos em horas e tem fins de prover colaborações e apresentar incrementos na *sprint. Sprint retrospective* é uma oportunidade para o Time Scrum inspecionar a si próprio e criar meios para se alcançar melhorias a serem aplicadas na próxima *Sprint*.

Uma *sprint* em sua forma mais direta é definida como tarefas, ou seja, são ações que levam aos incrementos e mudanças durante um projeto. Segundo (SCHWABER, 2009) *sprints* são eventos com duração fixa, onde o *Scrum Master* garante que não será feita nenhuma mudança que possa afetar a meta da *sprint*. A base do Scrum gira em torno de *sprints*, assim sendo durante cada fase de desenvolvimento uma *sprint* é desenvolvida com

um prazo fixo de forma que a equipe possa realiza-la da maneira que foi definida na sua criação, porém, durante uma *sprint* pode-se eventualmente haver alguma complicação no desenvolvimento que permita ter a sua duração aumentada ou mesmo reduzida e em um caso extremo, visto que a tarefa torne-se algo fora do contexto proposto ela pode ser cancelada caso a equipe considere essa possibilidade viável para o projeto, contudo quando uma *sprint* possui suas definições alteradas a sua complexidade aumenta, assim como os riscos a um projeto. A seguir na Figura 2 podemos ver o funcionamento do Scrum através de suas *Sprints*.

Figura 2 - Fluxo de processos do Scrum




Fonte: DOS SANTOS (2014)

As *sprints* possuem duas formas de interação com a equipe de desenvolvimento atribuídas a uma reunião diária também conhecida como *Sprint Backlog* e uma reunião ao final de cada *sprint* denominada de revisão da *Sprint*, como forma de revisar o que foi feito e dar sugestões de como se pode melhorar os processos utilizados, dessa maneira é possível melhorar não apenas o sistema mas também as formas de tratamento de como a equipe constrói software, assim como os próprios processos utilizados. Afinal Scrum usa o princípio ágil como referência e uma das vertentes dele é a evolução.

Através da definição de cada papel presente no Scrum vemos que sua utilização possui uma base que torna esse *framework* completo para se utilizar no desenvolvimento. (ANDRADE, 2009) afirma que ao se utilizá-lo é possível produzir software em um menor tempo, sem perder a qualidade do produto. Entretanto pode não ser suficiente. Determinados

autores defensores da engenharia de requisitos citam que devido ao Scrum trabalhar princípios ágeis a ausência de documentação formal pode prejudicar o desenvolvimento do sistema pela falta de informação para se incrementar novas funcionalidades em versões conseguintes do sistema assim como a sua mantenabilidade.

Segundo (ESPINOZA, 2011) a falta de documentação adequada pode dificultar a rastreabilidade e consequentemente afetar a gestão de mudança, estimativa e a análise de impacto durante uma alteração. (TRINDADE, 2016) ainda completa que, muitas vezes as informações sobre onde uma mudança irá afetar são buscadas no conhecimento da própria equipe e com a documentação mínima exigida pelas MA, o que não garante que uma informação esteja sendo lembrada. Todavia a fase que sucede a construção do sistema necessita de uma documentação de requisitos adequada para garantir a informação e a qualidade do software. (CATUNDA, 2011) em uma aplicação prática do Scrum notou que o desenvolvimento pode ser prejudicado ao se implementar modelos de melhorias de processos aderentes a modelos de qualidades. Podemos notar que o Scrum, embora possua suas vantagens ainda possui características que podem acarretar alguns problemas e em um caso extremo a falha de projeto.

O levantamento de requisitos no Scrum difere da engenharia de requisitos, pois devido as interações constantes com o cliente o interesse dele sobre o sistema é avaliado e fundamentado. Segundo (ZANATTA, 2005) os requisitos no Scrum são levantados diretamente com o *stakeholder* isso permite saber através dele o que anseia. Entretanto a efetivação do requisito apenas será dada quando o mesmo for testado pelo próprio usuário. Para (DOS PRAZERES LOPES, 2017) destaca que não se espera que o produto seja totalmente entregue, mas sim que ele seja incrementado a cada fase até que atinja sua completude, isso permite que sejam criadas interações múltiplas que possibilitam a diminuição dos riscos sobre o projeto, afim de que o produto final tenha todas os apontamentos do cliente.

# 4. ENGENHARIA DE REQUISITOS COM APOIO DA METODOLOGIA ÁGIL

Muito se fala que ambas as áreas são controversas por uma ser mais flexível que a outra quanto a mudanças inclusive para prazos. Contudo a ER por possuir um prazo rígido e geralmente mais longo não oferece muita abertura para mudanças, por ter em seu escopo de desenvolvimento algo bem definido na fase inicial quanto à os processos e técnicas, dessa

forma é analisado por diversos ângulos o que deve ser feito no desenvolvimento baseando-se nas formas de abordagens que houve com o *stakeholder* para elicitação dos requisitos que compõe o documento de requisitos, que é uma das partes da documentação do sistema que é útil não apenas na fase de desenvolvimento como também nas fases conseguintes. Enquanto na metodologia ágil não existe etapa de documentação o que pode ser um problema, pois dificulta a rastreabilidade dos requisitos mesmo estando em contato constante com o *stakeholder*, visto que não existe algo consolidado que possa ser fonte de consulta ao sistema, porém, mesmo tendo essa característica diferente se associada a ER essa pratica pode resolver a questão da rastreabilidade em requisitos, assim como a manutenção do próprio sistema, inclusive com a evolução do próprio sistema com incremento de novas funcionalidades, assim sendo temos duas áreas que criam de forma eficiente, mas com vantagens distintas. (ALVES, 2008) afirma em sua pesquisa que 58,3% das empresas analisadas não possuem processos de requisitos bem definidos e formalizados, isso pode prejudicar o projeto em todos os pontos, a autora ainda completa que essas mesmas empresas buscam adaptar processos da ER a sua realidade local, em outras palavras personalizar o desenvolvimento de acordo com as necessidades. (LOPES, 2005) afirma que não existe um processo único para ser utilizado por organizações, cada uma deve adapta-los ao seu contexto ambiental.

Todavia seria possível criar um sistema com as vantagens de ambas as áreas sem perder recursos e qualidade? A resposta mais plausível seria sim, segundo (DARWISH; MEGAHED, 2016) exemplos de técnicas de ER que podem ser aplicadas em métodos ágeis seriam: Entrevista, JAD, Modelagem, Prototipação e documentação, todas essas técnicas poderiam ser aplicadas não apenas em uma fase do desenvolvimento, mas durante todo o desenvolvimento. Afinal permitem serem mais flexíveis na mudança de sua estrutura podendo ser alterada durante quase todo processo, minimizando em tempo de desenvolvimento possíveis problemas ou mesmo interpretações errôneas dos requisitos.

A necessidade principal é criar uma forma de transição entre o desenvolvimento e a manutenção que é algo comum utilizado na ER que pode ser diferenciado dos processos ágeis, pois a forma de documentação é geralmente o código, o que não impede de se criar artefatos que possam servir de apoio na construção do software ou mesmo na consulta durante, como também depois do desenvolvimento. Porém, de modo geral não existe uma documentação formal do sistema, levando em consideração que a engenharia de requisitos também deixa lacunas que podem levar a complicações no projeto, podemos citar como exemplo a estruturação errônea de requisitos por possuírem ambiguidades ou mesmo pela má qualidade na qual foram estruturados. Para (CARVALHO, 2011) a baixa qualidade da documentação

leva os codificadores a implementar funcionalidades diferentes das descritas, ele completa que a existência de um requisito exige uma especificação e intepretação para que possa ser traduzido e convertido em código, assim podemos notar que ambos os processos da engenharia de software podem ter brechas que permitem falhas, entretanto, vemos que baseada na definição de ambas podemos hipoteticamente completa-las com suas qualidades distintas.

Além do que podemos citar o *framework* Scrum que permite a inserção de outros métodos em seu escopo onde seria possível obter processos flexíveis em quaisquer uma das fases do desenvolvimento, porém, ao se fazer isso o Scrum perderia uma de suas vantagens mais visíveis a agilidade sobre o desenvolvimento de uma sistema, afinal se fosse aplicada a técnica de documentação seria exigido do desenvolvimento mais uma fase o que exigiria um prazo maior tornando inviável a aplicação dessa metodologia, pois confrontava com os princípios ágeis que buscar redução de prazo e flexibilidade. Entretanto podemos perceber que a técnica de prototipação não é uma característica exclusiva dos processos ágeis, mas também da ER o que reforça características em comum de ambas.

Nos processos ágeis podem ocorrer a criação de artefatos que auxiliem no desenvolvimento como a utilização de modelagens para representar o sistema seja para o dono do produto ou mesmo para equipe que o desenvolve, mas para fins de visualização das interações do sistema, sem apoiar-se apenas na entrega de protótipos. Porém, de forma não documentacional, ou seja, ao termino da construção do sistema esses artefatos não seriam utilizados para criação de uma documentação formal, apenas seriam abandonados. Entretanto podemos concluir que algumas características da ER também foram assimiladas pela abordagem ágil, mostrando que a possibilidade de unir ambas as áreas de desenvolvimento não apenas é uma hipótese, mas sim uma possibilidade de melhorar a engenharia de software no quesito desenvolvimento de sistemas.

As técnicas da engenharia de requisitos mais comuns que poderão ser utilizadas em conjunto são descritas a seguir por (RODRIGUES, 2013), sendo elas; Entrevista que consiste series de reuniões entre desenvolvedor e usuário, dessa forma é possível entender como o usuário deseja modelar seu sistema de maneira clara para minimizar ao máximo possíveis equívocos nos requisitos a serem elicitados. Brainstorming é uma técnica utilizada basicamente para gerar ideias, consiste na realização de reuniões envolvendo desenvolvedores e usuários que estão ligados diretamente ao produto que está sendo desenvolvido, onde se permite que as pessoas sugiram e explorem ideias sem que sejam criticadas ou julgadas, onde as ideias geradas podem ou não se tornarem requisitos para o sistema. Prototipagem é outra

técnica utilizada na extração de requisitos, onde se utiliza um software já existente como referência, voltada para aqueles clientes que tem maior dificuldade de expressar suas necessidades, nessa técnica pode ser utilizada tanto uma prototipagem própria do sistema em desenvolvimento quanto um software similar já existente como forma de reuso e redução de trabalho e consequentemente o prazo. JAD - *Joint Application Design,* proporciona maior integração dos usuários e assim estrutura o debate, focando os requisitos necessários para a produção do software, ou seja, expande a visão de compreendimento da equipe de desenvolvimento afim de envolver todos com o sucesso do produto. Durante a aplicação das técnicas de levantamento de requisitos serão gerados artefatos que serão utilizados como forma de consulta durante o desenvolvimento ou mesmo para manutenção posterior ao desenvolvimento do software.

O questionamento maior que determinados autores possuem é sobre utilizar métodos ágeis juntamente com técnicas da engenharia de requisitos, pois alguns acreditam que ao se fazer isso os métodos ágeis podem perder sua essência, ou seja, suas vantagens baseadas no manifesto ágil como a flexibilidade sobre mudanças durante o desenvolvimento. Contudo, podemos perceber baseadas no conhecimento descrito que se aplicadas de maneira correta os métodos ágeis não perdem sua qualidade, pelo contrário adquirem outras, aprimorando assim as formas de criação de sistemas na engenharia de software.

# 5. UTILIZAÇÃO DA ENGENHARIA DE REQUISITOS NO FRAMEWORK SCRUM

A construção de um sistema remete a um conjunto de funcionalidades originadas dos requisitos, pois é a partir da criação, análise e validação de um requisito que funções são estabelecidas assim como suas restrições. Requisitos são abstrações que tornam a criação de sistemas possível, ele pode ser estabelecido de diferentes maneiras em diferentes metodologias de desenvolvimento. Através dos conhecimentos obtidos na pesquisa vemos que os requisitos podem ser validados como artefatos formais o que dão origem a uma parte da documentação de um sistema que é um método baseado na engenharia de requisitos ou mesmo podem ser descritos como artefatos temporais que é geralmente utilizado pelo Scrum. Entretanto requisitos são a essência de um sistema já que os torna possível a estruturação de qualquer sistema, porém, como podemos ver nem sempre na engenharia de software ele é formalizado, o que ao término de um desenvolvimento pode ser um problema, afinal para se

permitir a manutenção seja ela para fins de manter o sistema em funcionamento ou mesmo para incremento de novas funcionalidades será necessário conhecer os requisitos e sem uma formalização deles a dificuldade quanto a prazo poderia ser aumentada.

O Scrum que utiliza artefatos temporais pode ser prejudicado quanto as suas vantagens, pois ao término da construção do sistema não haveria um documento do sistema e outrora o que podia ser uma vantagem quanto ao desenvolvimento o Scrum perde a vantagem da agilidade sobre o sistema, já que não será necessariamente a equipe de desenvolvimento que irá prestar a manutenção do sistema, visto que para realizar tal procedimento seria necessário o conhecimento prévio do sistema o que seria apoiado pela documentação de requisitos, mas esse artefato não existe junto ao Scrum como artefato de criação para sistemas. Porém, a possibilidade de se utilizar a técnica de documentação no *framework* é possível já que ele permite assimilar processos e técnicas de outras metodologias. Contudo a fase da documentação é uma das mais longas do desenvolvimento na engenharia de requisitos e ao se utilizar esse método no Scrum pode-se comprometer a sua essência que se baseia no manifesto ágil sobre a flexibilidade e agilidade. Entretanto podemos notar que o Scrum é flexível quanto a evolução e integração de outros processos, aceitando durante todo seu desenvolvimento conceitos de outras metodologias que possam melhorar esse propósito, permitindo a personalização do escopo de um projeto.

A fase de desenvolvimento é algo importante, pois será a partir dela que o produto desejado pelo *stakeholder* será construído, mas em alguns casos o sistema não é algo que seja dito como temporário quanto a seu ciclo de vida, pois algumas funcionalidades poderiam ter sido desejadas após o desenvolvimento como forma de incremento para o sistema, sendo dificultado pela falta de informação. Afinal o Scrum trabalha com artefatos temporais e o único artefato que resta é o próprio código, mas o código não necessariamente pode fornecer as informações necessárias para evoluir o sistema após sua construção.

Os artefatos que o Scrum utiliza que geram requisitos são denominados de *users stories*, *epics* e *themes*, eles fornecem informações sobre o sistema que permitem a criação de funções ou mesmo restrições. Contudo esses artefatos são descartados durante o desenvolvimento, mas poderiam ser reutilizados ao termino para a construção de um documento de requisitos que pudesse ser útil para etapas conseguintes. Embora as informações presentes nos artefatos para requisitos serem não tão completas é possível melhorar essas informações e torna-las completas para que possam serem utilizadas em outros propósitos não apenas a criação de funções pertinentes ao sistema durante o desenvolvimento. O melhoramento desses artefatos pode prover a evolução de software a longo prazo utilizando

o método da documentação de requisitos proveniente da engenharia de requisitos, ainda assim preservando a essência do manifesto ágil que preza pela flexibilidade e agilidade.

A inserção do método de documentação da engenharia de requisitos no Scrum não é com intuito de se criar um documento técnico sobre o sistema, afinal para se criar algo desse porte seria necessário uma etapa inteira para ser possível ter uma documentação formal o que poderia confrontar os princípios ágeis pelo qual o Scrum é baseado, em vez disso adaptar esse método para o Scrum e utilizar seus artefatos gerados que sejam ligados a requisitos e documenta-los de maneira formal. Todavia sabemos que os artefatos gerados no Scrum são considerados artefatos temporais e são descartados, entretanto se fosse reutilizado ao final do desenvolvimento e estruturado de maneira formal para um documento de requisitos que não fosse tão técnico e burocrático a ponto de ser igual ao método da engenharia de requisitos, mas como um documento baseado nos artefatos voltados para requisitos gerados no Scrum que permita a quem utiliza-lo vê a estrutura que o sistema possui e qual a correspondência desse requisito no processo ágil.

## 5.1. Analise dos artefatos gerados no framework Scrum

As *user story* são os artefatos do Scrum que definem através de descrições denominadas histórias de usuário uma funcionalidade a ser implementada, inclusive restrições, logo visto a natureza dessas descrições podemos considera-las como uma forma de requisito informal para o sistema. Porém, esses artefatos são descartados ao se implementar as histórias, todavia se em vez de descarta-las fossem melhoradas e arquivada para no final da construção do software ser reutilizada para estrutura uma fonte de pesquisa para o sistema como um manual de como o sistema foi feito permitindo a equipes futuras prestar manutenção do sistema.

A possibilidade de criação de um documento de requisitos pautado na engenharia de requisitos inserida no Scrum é algo totalmente possível, já que o *framework* permite a assimilação da técnica, assim como criar uma engenharia reversa da documentação sobre o sistema já desenvolvido. Entretanto ao se fazer isso poderia demandar uma fase extra no desenvolvimento, onde será composta pelas etapas de identificação, validação e implementação, consequentemente tendo que ter o prazo aumentado. Essa prática iria inviabilizar parcialmente a essência do *framework* que por utilizar princípios ágeis se tornaria mais rígida, sendo equiparada em partes ao mesmo estilo de desenvolvimento sequencial da

engenharia de requisitos, sem a possibilidade da flexibilização sobre mudanças. Além mais iria tornar mais restrito a participação do cliente que possivelmente só teria sua participação ativa na etapa de elicitação e validação de requisitos.

Todavia a utilização de *stories* provém uma linguagem de alto nível que possibilita a participação ativa dos *stakeholder*s, afinal são eles que juntamente com o *Product Owner* criam esses artefatos. Contudo a forma de validação se distingue da engenharia de requisitos, pois o próprio usuário quem validaria a função proveniente da story e caso fosse de fato o que ele esperava o requisito seria efetivado como funcionalidade, isso torna o Scrum bem difundido e popular, já que minimiza a má compreensão dos requisitos. No entanto, quando um *stakeholder* busca desenvolver algo, em geral nem ele sabe exatamente o que deseja até que possa ter um produto funcional palpável. Onde se eventualmente não for o esperado simplesmente a story seria descartada, assim como o requisito originado, isso torna mais fácil descartar uma funcionalidade do que ter que reestruturar um sistema completo.

O intuito principal dessa pesquisa é demonstrar a importância que requisitos possuem no desenvolvimento de sistemas, pois são a partir deles que todo o software será estruturado, mas não é apenas para isso que os requisitos servem já que existe a fase de evolução do sistema onde se presta a manutenção para mantê-lo em funcionamento podendo incrementar novas funções ao sistema. Dessa forma ter uma fonte de consulta sobre o sistema seria um auxilio que é defendido pelo manifesto ágil a agilidade para se executar alguma ação sobre o software.

Mas como se fazer um documento de requisitos baseado em *user story*, *epic* ou *theme*? Podemos responde-la ao se analisar a técnica de documentar presente na engenharia de requisitos, contudo como já foi citado o Scrum por ser um *framework* flexível permite assimilar em seu escopo processos e técnicas de outras metodologias para tornar o desenvolvimento mais flexível e ágil quanto as necessidades do produto, logo podemos inserir no Scrum essa técnica e adaptá-la de acordo com os processos ágeis no qual o Scrum foi baseado. Os artefatos gerados no Scrum podem ser utilizados para se criar um documento que diferencie requisitos funcionais de não funcionais, afinal esses artefatos são criados para referenciar um requisito, ou seja, será a partir dele que um requisito será implementado. Dessa forma podemos utiliza-los para arquivar formas e métodos de implementação dos requisitos que geraram artefatos em cada revisão da *sprint*. Pois ao final da construção do sistema reunir todos os artefatos e criar um documento de requisitos baseados nos princípios do Scrum. Mas um dos questionamentos que podemos fazer seria se ao utilizar essa abordagem o prazo seria afetado, vemos que se baseando no tempo de duração dessa fase a reunião das review dura em

torno de quatro horas e como as *stories* estariam prontas, seriam preenchidos os parâmetros adaptados nelas que duraria em média no máximo dez minutos para cada uma. O tempo máximo que seria acrescentado não afetaria o prazo nem prejudicaria a construção, visto no Gráfico.

Uma *user story* descreve a história de um usuário que necessita de algo a ser resolvido e dessa forma se constrói um requisito que resolve algo no sistema. Assim sendo uma *user story* é dividida em "QUEM", "O QUE" e o "POR QUÊ", ou seja, para "QUEM" define a quem é direcionada a necessidade, "O QUE" descreve qual é a necessidade que precisa ser resolvida, em geral os requisitos são criados a partir dessa definição e o "POR QUÊ" delineia o motivo pelo qual essa necessidade existe. Assim podemos perceber que uma *user story* é bem similar a um requisito levantado na documentação de requisitos, pois descreve para quem, a necessidade e com qual motivo ele é construído. Entretanto existe uma diferença só que podemos destacar é que uma definição seria mais técnica restrita apenas a pessoas que possuem conhecimento para interpretação, enquanto a outra seria mais alto nível podendo ser interpretada por quem a ler-se, mas todas com o mesmo propósito, gerar um requisito.

## 5.2. Aplicação das modificações propostas em artefatos geradores de requisitos do framework Scrum

Através da engenharia de requisitos é possível gerar uma documentação técnica dos requisitos, consequentemente o Scrum por ser um *framework* que abriga processos e técnicas de outras metodologias é plausível gerar uma documentação de requisitos em alto nível que sirva não apenas à profissionais da computação como também para usuários entenderem a motivação da criação de um requisito.

A melhora das histórias de usurário poderiam ser feitas adicionando o "ACOPLAMENTO" baseando-o em duas características sendo elas, alto acoplamento e baixo acoplamento para definir o nível no qual as ligações entre requisitos, não como a definição motivacional, mas como parâmetros de desenvolvimento que foram utilizados para se criar aquelas funcionalidades específicas do sistema, ou seja, um requisito, isso permite que a atenção seja voltada para riscos maiores no qual poderia desestruturar o sistema. Entretanto, o acoplamento ainda deve vir acompanhado pelo identificador da *user story* que requer a sua necessidade e o "por quê" de está acoplado a ele, podendo ser visto no Quadro 2. Como exemplo podemos utilizar para contextualizar o cenário o jogo Jenga, que consiste em uma torre com blocos, onde o objetivo principal é retirar os blocos sem permitir que a torre seja

desestruturada, visto a peça que for retirada. O mesmo ocorre com um sistema, já que caso um sistema esteja em sua fase de manutenção existe a possibilidade da adição de novas funcionalidades ou a remoção de outras, suponhamos que a torre do Jenga seja o sistema e as funcionalidades correspondam as peças, visto que a necessidade seria retirar uma funcionalidade (peça), mas deve-se notar quais peças mantém a sustentação da torre e quais peças podem ser retidas sem abalar permanentemente sua estrutura, pois caso seja retirada uma peça importante a torre inteira pode ruir.

O alto acoplamento de um sistema nos diz quais funções mantem o sistema funcionando de modo satisfatório. Assim caso um requisito com alta acoplamento seja alterado ou excluído o sistema inteiro pode entrar em falha, consequentemente dificultando sua mantenabilidade. Ainda podemos notar que o mesmo pode ocorrer com a migração de um membro da equipe durante a construção de um sistema e ser substituído por um novato que não possui informações suficientes sobre o sistema e tenha que alterar algo, no qual pode ser possível ter consequências críticas sobre a estrutura e funcionamento do software. Além do mais com a inclusão de um novo membro existirá uma exploração maior de alguns processos, enquanto outros importantes não terão tanta atenção como deveriam, como exemplo temos a garantia de qualidade. Mas a falta de informações pode ser suprida com a existência de um artefato que contenham esses elementos.

A aplicação dessa metodologia hipoteticamente pode ser assimilada na revisão da *sprint*, assim em cada reunião de final de *sprint* preencher esse novo parâmetro para *story*, dessa forma utilizar o conhecimento da própria equipe sobre as imposições utilizadas sobre as ligações dos requisitos, citando qual modulo do sistema requer uma atenção mais elevada. Podemos ver na Figura 4 o modelo adaptado para uma *user story* mostrando que mesmo com a adição desse parâmetro ainda é dotado de simplicidade, contudo vale salientar que essa parte não terá a participação do *stakeholder*, apenas dá equipe de desenvolvimento.

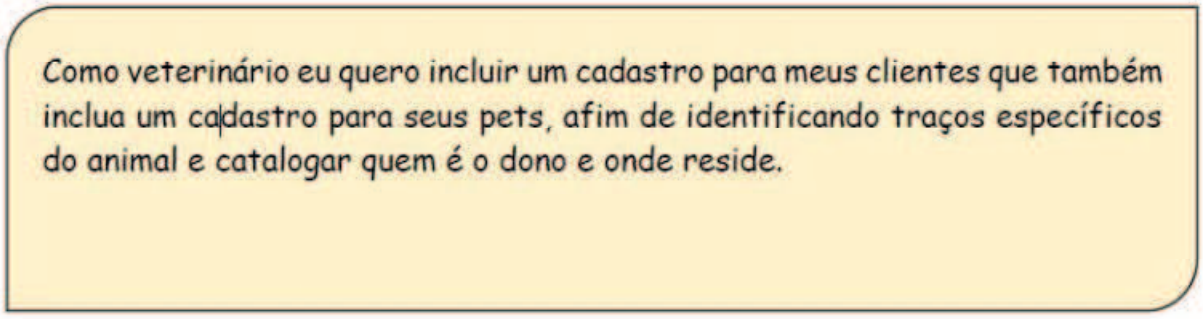



Figura 3 – Modelo de user story

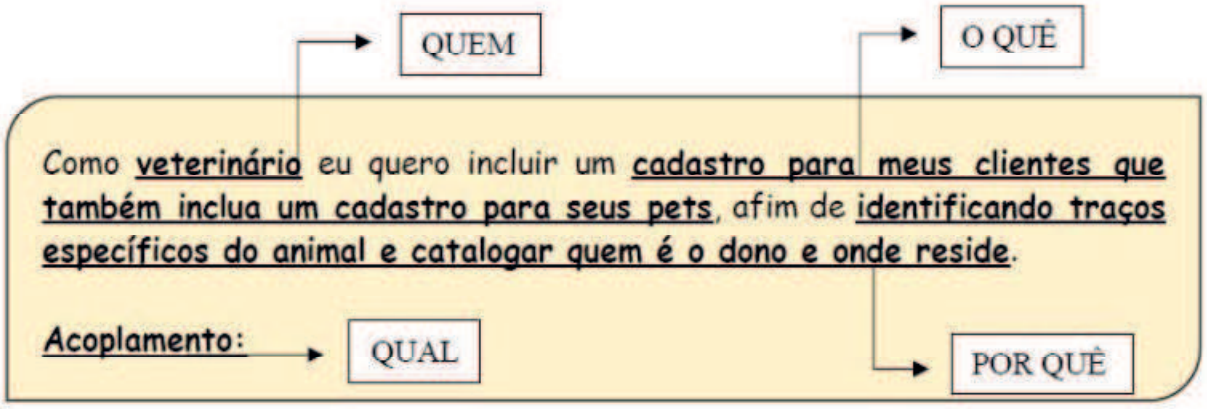



Figura 4 – Modelo de user story adaptado

As mudanças das *stories* adaptam informações que podem ser usadas como referencial de identificação de um requisito, visto que um requisito deve conter informações sobre especificações de como o sistema executará uma função, sem se preocupar de como o sistema será implementado. Todavia essa adaptação permite que o requisito seja mais visível ao sistema em escala global, ou seja, o nível de acoplamento de um requisito terá com outro. Por esse motivo essa seria a melhoria mais interessante para um requisito pelo simples fato de que se um requisito esteja altamente acoplado no sistema significaria dizer que para evolui-lo teria de existir um cuidado maior do que um requisito com baixo acoplamento que pela ideia central iria representar a inexistência do mesmo. Dessa forma seria possível minimizar complicações que um software teria caso o requisito tivesse uma especificação com essa significância.

Entretanto essa melhoria poderia ser vista como dispensável, já que seria utilizada uma linguagem em alto nível para se criar uma documentação de requisito. Contudo essa informação não seria para o usuário e sim para o desenvolvedor, pensando não apenas durante o desenvolvimento que é importante, mas também após ele. Vemos que com a utilização do Scrum assimilando processos ágeis e originado do mesmo, a flexibilidade de utiliza-lo é altíssima, dessa forma mudança é algo intrínseco a ele. Porém também existe a possibilidade de um requisito já implementado mudar por vontade do cliente para se adequar a sua necessidade ou mesmo como forma de sugestão para implementação mais simplificada para reduzir prazo no software. Assim uma documentação que especifique esse requisito pode funcionar como recordação do seu processo de criação.

Ainda podemos falar sobre os demais artefatos gerados no Scrum que se referenciam a requisitos como os *Epics* e os *Themes*. Os *Epics* são vistos com uma grande *user story*, dessa forma é possível dividi-los em tarefas menores dando origens a outras *users stories* durante

todo desenvolvimento no Scrum, visto que fizemos uma adaptação desse artefato, um *Epic* pode ser encaixado nesse contexto. Além do mais temos os *themes* que podem ser vistos como grupos específicos de *stories*, ou seja, se comparado com a engenharia de requisitos que trabalha em sua documentação de requisitos com dois tipos de requisitos os funcionais e os não funcionais, podemos utilizar os *themes* para agrupar *stories* no Scrum que contenham regras de negócio similares adaptando com um novo parâmetro que categorize cada tipo de *story*, já que irão derivar requisitos. Utilizando essas adaptações podemos construir um documento de requisitos em alto nível para sistemas com princípios ágeis que podem ser compreendidos tanto pela equipe de desenvolvimento quanto por usuários, por derivar de uma linguagem de alto nível. Na Figura 5 podemos visualizar a estrutura de um *theme* adaptado.

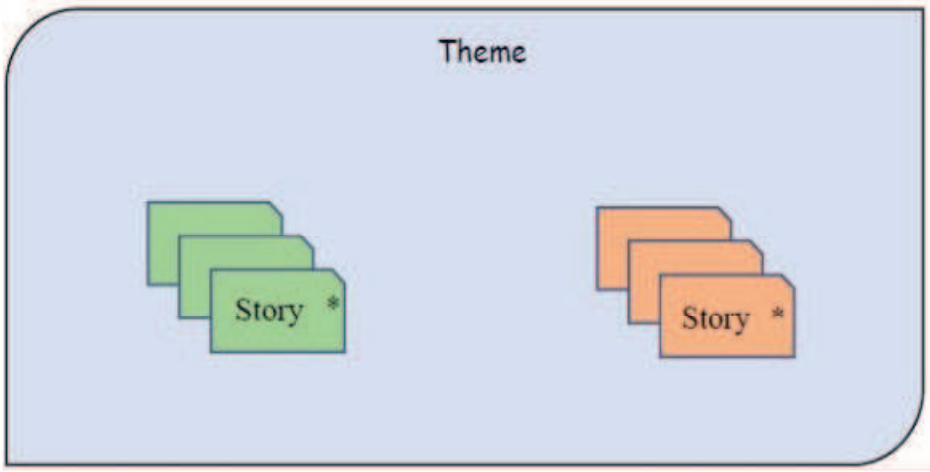



Figura 5 – Modelo de theme adaptado

A estrutura adaptada de um *theme* é mais simples que de uma *user story*, afinal o único parâmetro que será colocado é se a *story* possui dependência ou não, agrupando cada *story* com finalidade de organizar o padrão do documento de requisitos. Esse novo parâmetro é simbolizado pelo asterisco "*", assim caso uma *story* possua essa simbologia significará que ela tem uma ou mais dependências e qual seria ela, como pode ser visto no quadro 2, permitindo essa característica facilite na formalização da mesma, afinal quando uma *user story* possui regras de negócio, intrinsicamente uma ou mais restrições estarão sendo estabelecidas, pois a partir disso serão criados os critérios de aceitação, esses critérios restringem como deve prosseguir a implementação, para atender no escopo da *user story*. Sendo as funções representadas pelos requisitos funcionais e as restrições representadas pelos requisitos não funcionais em comparação a ER, afinal os requisitos a depender do tipo recebem tratamento diferenciado. Vale salientar que esse modelo é proposto a priori apenas para facilitar o arquivamento das *users stories,* sendo que após a finalização do projeto e a

formalização do artefato normativo, seria deixado de ser utilizado, visto que todas as informações já estariam presentes na documentação.

Todavia *users stories* trazem consigo testes de aceitação que permite impor parâmetros que devem ser seguidos pela equipe, podendo serem transcritos através de cartões ou mesmo de programas criados para esse propósito, afim de se chegar no resultado que a *story* descreve. Dessa forma podemos utiliza-la como premissa para o desenvolvimento e consequentemente uma documentação baseada nelas, assim podemos torna-las mais completas acrescentando os possíveis acoplamentos das *stories*, além de citar caso exista parâmetros de restrições embutidos na *story*, assim podemos criar um Modelo normativo que sirva para compor a documentação. Podemos ver os modelos propostos nos Quadros de 2 a 4.

Quadro 2 - Modelo documentacional de uma *user story* adaptada

| **Identificador: [US001]** |
|---|
| **Descrição:** Sou um professor e necessito realizar a chamada dos meus alunos pelo smartphone, para reduzir o tempo gasto em caderneta. |
| **Critérios de aceitação:** é necessário que o sistema seja pelo menos 5.0 ou mais |
| **Parâmetros de restrições:** O sistema utilizado nos aparelhos deriva do C# |
| **Acoplamentos:** alto; necessita da US002 para analisar a versão do sistema |

Quadro 3 - Modelo documentacional da *user story* [US002]

| **Identificador: [US002]** |
|---|
| **Descrição:** Sou um analista e necessito verificar a versão do sistema, para adequação de todos os aparelhos. |
| **Critérios de aceitação:** Atualizar todos os aparelhos para versão 5.0 do sistema |
| **Parâmetros de Restrições:** Aparelhos celulares devem ter quatro núcleos de processamento |
| **Acoplamento:** baixo, não necessita de outra *user story* |

Quadro 4 - Definições dos campos da *user story* adaptada

| |
|---|
| • **Identificador:** o identificador das *user story* que pode deverá ser iniciado por uma numeração. |
| • **Descrição:** detalhes da *user story*, seguindo o padrão da sua construção, QUEM, O QUÊ e POR QUÊ. |

| |
|---|
| • **Critérios de aceitação:** Testes para validação das *users stories*, afim de verificar se todas estão de acordo com as solicitações. |
| • **Parâmetros de restrições:** As restrições que podem existir nas *users stories* (opcional). |
| • **Acoplamentos:** Define o nível de acoplamento no sistema e descreve quais seriam (opcional, caso o seja considerado baixo, ou seja, não necessite de outra *user story*). |

Algumas formas de desenvolvimento não possuem registros normativos sobre o que os requisitos representam e suas categorias, dessa forma existe uma dificuldade latente que pode dificultar fases posteriores a construção do sistema ou mesmo durante. Entretanto uma documentação poderia agilizar processos como o de manutenção provendo incrementos de novas funcionalidades de maneira mais eficaz. Visto que para se realizar esses procedimentos é necessário o conhecimento prévio do sistema. A engenharia de requisitos por ter métodos para documentar a construção de um sistema torna o pós-desenvolvimento algo mais ágil na compreensão como também durante, já que as *users stories* seriam arquivada na medida que fossem validadas, pois facilita a compreensão do sistema para quaisquer equipes a prestar serviços ao software, visto que a linguagem utilizada nas *users stories* torna mais facilitada a compreensão da proposta para um requisito.

Por outro lado, temos o Scrum que é um *framework* bastante usado para desenvolvimento de sistemas atualmente, pela sua flexibilidade e agilidade nos processos de desenvolvimento. Contudo uma de suas maiores falhas é a falta de informações para etapas conseguintes do sistema, tornando a agilidade uma vantagem quase inexistente, pois demandará uma análise do sistema. Todavia o Scrum não trabalhar com o método de documentar formalmente e a única forma que pode ser equiparada a um possível documento do sistema seria o próprio código. Além do mais o código do sistema pode se tornar confuso para quem o estuda caso não se conheça para qual propósito ele foi criado, mesmo que esse código tenha comentários que situem para qual finalidade a linha estruturada tenha sido criada, inclusive para o próprio desenvolvedor quem a escreveu. Requisitos em sua essência estão presentes não apenas durante o processo de desenvolvimento como também após, tornando o artefato de documentação indispensável, visto a analise de todas as fases de um sistema ativo.

Requisitos são a base para construção de um sistema são eles que provem todas as suas funções e restrições de um sistema, além do mais pode fornecer informações sobre a

compreensão de como o sistema funciona sendo útil em quaisquer fases, assim vemos que todo artefato que se refere a um requisito deveria ser preservando enquanto o software se manter ativo. A existência de um requisito é tão essencial que se um requisito for alterado o sistema inteiro pode sofre alteração dependendo dos acoplamentos desse requisito com os demais, requisito no geral torna um sistema possível.

A documentação é algo que permite que os requisitos sejam formalizados e utilizados a longo prazo. Assim vimos que ao se utilizar a engenharia de requisitos em especifico o método da documentação, é possível adaptar os artefatos do Scrum para que se tornem normalizados, afim serem compreendidos por quem os lerem utilizando as próprias *stories*. Todavia sem prejudicar as fases dos Scrum quanto a sua flexibilidade e agilidade, apenas complementando o que já existe com intuito de documentar e maximizar o potencial dos requisitos.

Essa pesquisa mostra a importância que um requisito possui ao longo das etapas de desenvolvimento, assim mapear *users stories* validadas para em teoria aplicar a técnica de documentação da engenharia de requisitos no escopo do *framework* Scrum, mostrando uma possibilidade que pode ser seguida para maximizar as qualidades de software e preservar o produto mais importante do desenvolvimento de sistemas, o requisito. Adaptando *users stories* para que tenha informações suficientes e sucintas que possibilitem sua compreensão, dando origem a um produto no Scrum que podemos chamar de documentação ágil. Valorizando um requisito em qualquer etapa de um software.

## 5.3. Impacto sobre as adaptações das users stories em desenvolvimento de software

A falta de comunicação durante o desenvolvimento é uma das grandes causas de falha ou complicações no projeto seja ela interna, entre a equipe de desenvolvimento ou mesmo externa com os *stakeholder*s. Contudo diversos fatores podem implicitamente tornar esse evento possível. Segundo (STANDISHGROUP, 2014) menciona que a falha de projeto é dada por fatores específicos onde 13,1% é a faixa atribuída para requisitos incompletos, esses requisitos não possuem informações suficientes ou mesmo informações ambíguas indefinindo o que é requerido no mesmo, além do que 8,7% correspondem a mudanças nas especificações dos requisitos, o que torna esse processo falho é a falta de informações sobre o próprio requisito, pois sem um escopo inicial do requisito a definição pós implementação se tornariam

flutuantes, impulsionando a uma definição um pouco distante da que originou o requisito. Ainda são citados outros fatores que impactam sobre o sucesso de um projeto vistos a seguir.

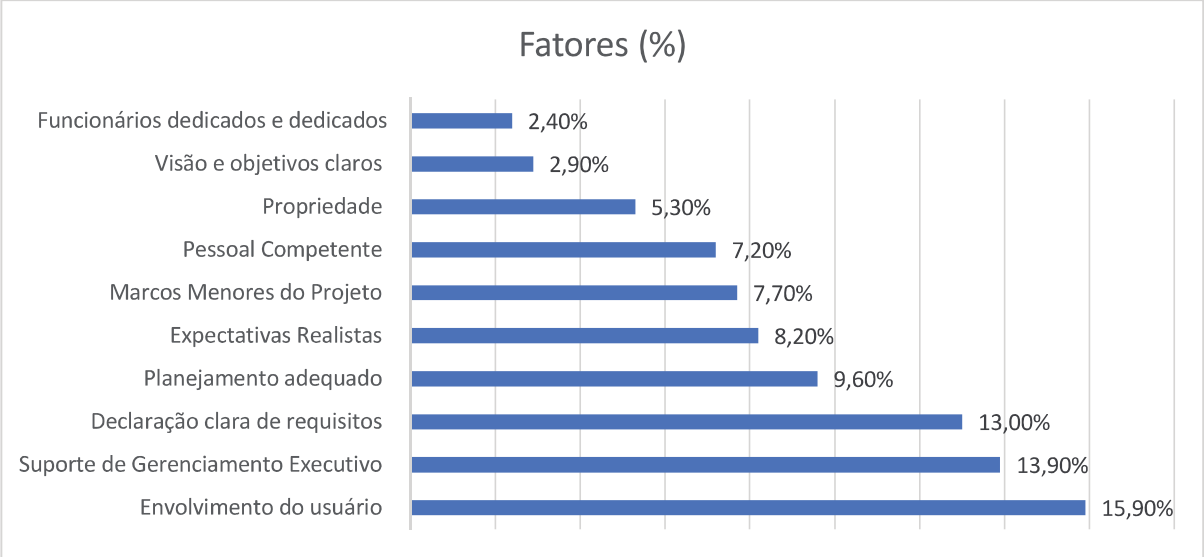
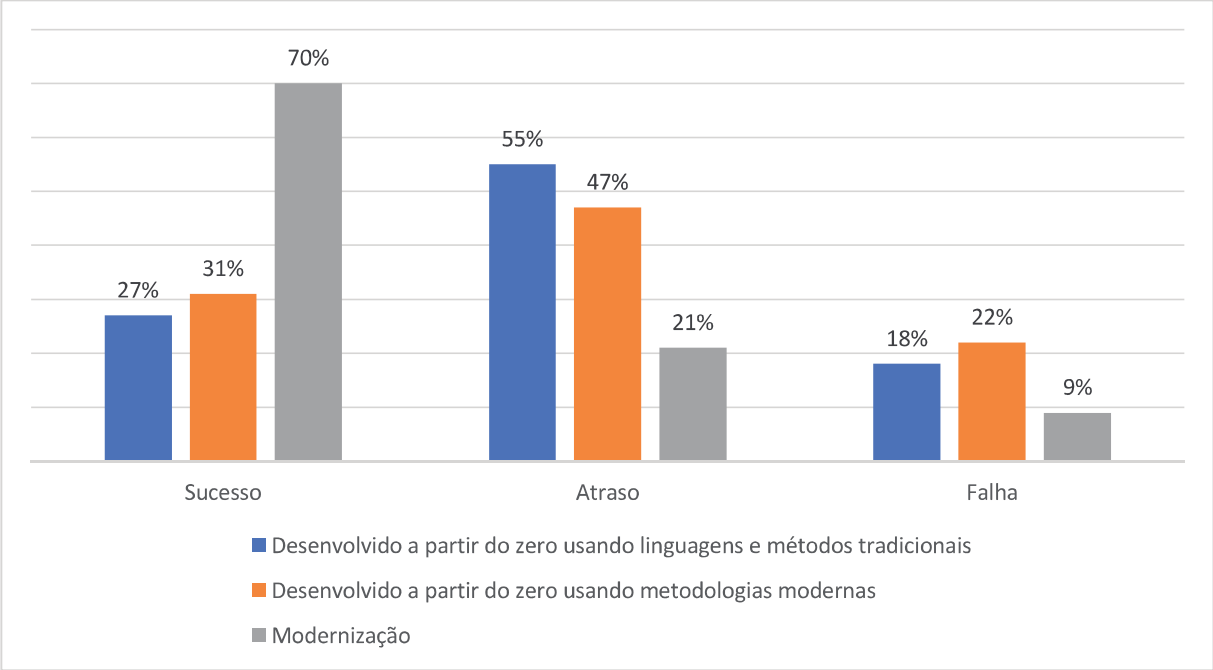



Gráfico 1 - Fatores de impacto sobre um projeto

Fonte: *The Standish Group* (*Chaos Report*, 2014)

Esses fatores são cruciais e representam uma porção considerável do sucesso de um projeto e devem ser minuciosamente considerados e estudados para torná-los possíveis. A pesquisa ainda faz uma comparação entre o sucesso, o atraso e a falha de projetos que utilizam técnicas tradicionais, metodologias modernas e a modernização. A busca de meios que melhores as metodologias da engenharia de software é algo incessante que necessita está sempre sendo aprimorados, visto que resultados sobre o sucesso de um projeto gira em torno de 70% para modernização das técnicas de desenvolvimento o que ultrapassa muito as outras duas, as comparações podem ser consultadas no Gráfico 2.

Gráfico 2 - Comparação de parâmetros sobre o gerenciamento de projetos




Fonte: *The Standish Group* (*CHAOS REPORT*, 2014)

Os artefatos têm um papel fundamental na compreensão de um sistema seja ele temporal ou não isso fornece informações que auxiliam em qualquer fase de um sistema, além do que vimos na Gráfico 2 que a modernização ocupa uma grande faixa sobre o sucesso de projetos, ou seja, buscas por melhorias nas técnicas de engenharia de software tornaram mais eficientes os processos de desenvolvimento de sistemas, consequentemente aumentando a taxa de sucesso. Todavia conceitos como melhor comunicação e conhecimento dos requisitos são fatores essenciais nesse processo. Assim podemos notar que uma documentação abrange alguns desses conceitos que podem auxiliar no projeto e minimizar falhas.

Entretanto a documentação possui um processo muito burocrático para sua validação em engenharia de requisitos, fazendo com que consuma uma parte considerável, podendo causar atrasos nas entregas e exceder o prazo. Mas vemos que adição de um conceito ágil nessa técnica podem moderniza-la maximizando sua eficiência e a tornando viável para ser utilizada em desenvolvimento ágil integralmente baseando-se em *user stories* utilizadas no *framework* Scrum.

Utilizando as *users stories* como forma documentacional podemos suprir a falta de informação e aprimorar o processo de comunicação no desenvolvimento. Dessa forma teríamos uma documentação para *stories* que utiliza conceitos da engenharia de requisitos com informações sobre dependências juntamente com agilidade dos processos ágeis. Visto que a documentação seria constituída à medida que funções fossem validadas pelo usuário,

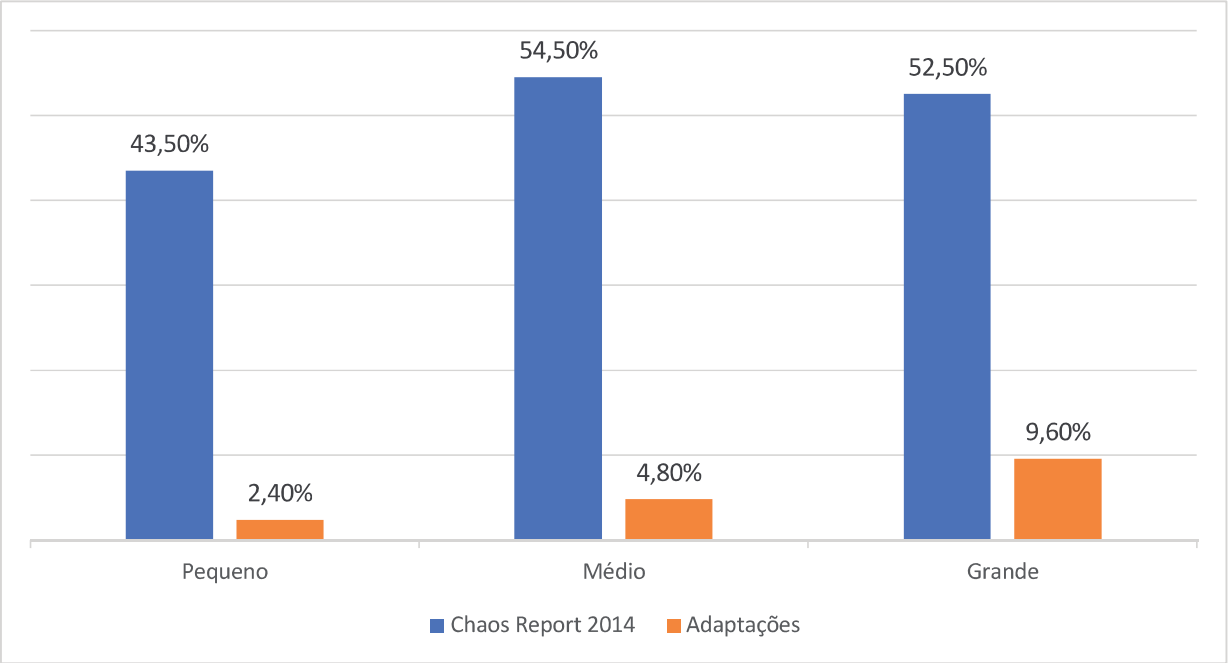
sendo finalizadas durante a revisão da *sprint*, onde a documentação completa apenas seria montada após todas as *users stories* fossem completamente testadas e aprovadas. Podemos consultar um modelo para essa documentação de *users stories* no apêndice A.

Podemos ressaltar que mesmo com a adaptação desse artefato gerado pelo Scrum, ainda seguiria o acrômio do modelo "INVEST" proposto por (WAKE, 2003) que descreve que uma *user story* desejável deve ser independente (*Independent*), negociável (*Negotiable*), valiosa (*Valuable*), estimável (*Estimable*) pequena (*Small*) e testável (*Testable*), para que se torne ideal, onde caso algum desses parâmetros não sejam assimilados pela *story* ela deve ser refeita.

Todavia mesmo aplicando essas mudanças nas *users stories* o conceito referente a independência seria o único que poderia ser afetado, mas ainda assim prevalece, visto que a implementação da *story* se daria de forma independente das outras sem comprometer a sua construção, contudo ao se compor um conjunto de funcionalidades do sistema, as dependências seriam estabelecidas atualizando o parâmetro proposto. O conceito referente a negociação continuaria imutável com a adaptação, assim como o conceito do valor que ela representa, o conceito da estimação, o conceito de permanência da sua simplicidade o que a torna pequena e o conceito de testabilidade, presentes no acrômio INVEST. Assim vemos que a mudanças na estrutura da *user story* não prejudicam sua essência pelo contrario apenas acrescenta e a torna mais completa, levando em consideração que o proposito principal dessa alteração seria a sua formalização em um documento de *user stories*.

Como podemos ver o impacto sobre o desenvolvimento a ponto de prejudica-lo é baixo como podemos consultar no Gráfico 3, visto que as modificações foram feitas com intuito de aperfeiçoar o artefato e mantê-lo útil a longo prazo, afim de preservar o requisito ao qual ele deu origem. O sucesso de um projeto não vem apenas da escolha das técnicas para seu desenvolvimento, mas dos aprimoramentos que são feitos sobre essas metodologias, visto que atualmente projetos são modificados constantemente e buscar meios para adapta-los é mais trabalhoso que aprimorá-los, fazendo assim as técnicas se adaptarem aos projetos e não os projetos as técnicas.

Gráfico 3 - Comparação do impacto sobre desenvolvimento




Fonte: *The Standish Group* (*CHAOS REPORT*, 2014)

Assim utilizando dados estatísticos do (STANDISHGROUP, 2016) que fez a comparação do atraso dos métodos tradicionais e ágeis, obtivemos uma média que nos fornece informações sobre o atraso que as técnicas possuem sobre projetos de pequeno, médio e grande porte. Dessa forma podemos comparar com as adaptações em *user stories* realizadas nessa pesquisa, visto que os procedimentos para preencher os parâmetros de uma *user story* adaptada citados no tópico 5.2 seriam realizados na revisão da *sprint*. Estipulando um tempo de dez minutos para cada *user story*, hipotetizando um cenário em que para cada mês composto por quatro semanas, quatro *users stories* seriam implementadas adicionando assim um tempo extra de quarenta minutos. Todavia o impacto sobre o atraso em projetos de diferentes portes foi menor que 10% nas alterações propostas em *users stories* o que em comparação com as informações obtidas sobre as diferentes metodologias é bastante minimizado. Concluímos que esse processo de modernização pode não apenas auxiliar com a redução dos prejuízos no desenvolvimento de software como também causa um impacto reduzido sobre o prazo tornando-as viáveis para aplicação.

# 6. METODOLOGIA

Os requisitos embora possuam diferentes abordagens nos dois processos tem o mesmo objetivo primário a sua identificação e validação, pois serão a partir deles que um sistema tomará forma. Segundo (EASTERBROOK, 2000) a engenharia de requisitos é o processo de identificar o propósito através da identificação de *stakeholder*s e suas necessidades, documentar esses requisitos de forma adequada para análise, comunicação e posterior implementação da solução.

A utilização das técnicas de levantamentos de requisitos pode muitas vezes serem cruciais para que um projeto de software seja desenvolvido de maneira adequada, afim de diminuir os possíveis erros que venham a surgir durante o incremento do software. Segundo (SOMMERVILLE, 2011), a engenharia de requisitos é definida pelo processo de descobrir, analisar, documentar e verificar esses serviços e restrições, o que nos mostra a importância dessa fase no desenvolvimento de software. Porém, uma abordagem mais direta sem a preocupação da documentação pode muitas vezes antecipar etapas de elicitação de requisitos como é o caso das metodologias ágeis, onde segundo (SOMMERVILLE, 2011), o método ágil é caracterizado por desenvolvimento rápido de softwares, ou seja, o software não é desenvolvido como única unidade, mas como uma serie de incrementos.

Para chegar no resultado desejado são utilizados processos e técnicas da engenharia de requisitos inseridas no escopo do *framework* Scrum que se baseia nos princípios ágeis mostrando que é possível trabalharem juntas, sem que demande mais tempo para o cliente e a equipe, inibindo a possibilidade da perda de prazo.

A partir de uma revisão bibliográfica de artigos, teses e livros necessários para o conhecimento adequado e hipotetizar a inserção da engenharia de requisitos no escopo de desenvolvimento do Scrum, mostrando a importância dos requisitos no sistema e como são tratados em ambas as áreas. Todavia adaptando a técnica da documentação presente na engenharia de requisitos para Scrum com a finalidade de preservar e maximizar o potencial de um requisito.

A pesquisa exploratória foi utilizada para fundamentar a hipótese de assimilação e adaptação de métodos para preservar os requisitos em um documento com linguagem de alto nível, afim de utiliza-lo como apoio para fases conseguintes do desenvolvimento.

# 7. CONCLUSÃO

Os requisitos de softwares como podemos notar é uma parte comum na engenharia de softwares, pois está presente em todos os métodos e processos que nela existe, já que são através deles que as funcionalidades de um sistema serão estabelecidas com finalidades distintas, assim como as restrições do sistema sobre plataformas e outros sistemas, visando definir o que foi proposto pelo *stakeholder*. A utilização de técnicas para levantamento de requisitos se distingue de cada metodologia, afim de proporcionar uma melhor compreensão quando as equipes que as utilizam em um projeto.

A distinção dos processos torna o levantamento e validades dos requisitos complexo para cada técnica. Onde podem serem mais maleáveis que outras, dentre os processos da engenharia de software que tornam esse cenário possível estão inclusas as metologias ágeis que atualmente está bastante difundida e popular devido a sua flexibilidade e agilidade e a engenharia de requisitos que embora tenha um processo rígido de construção permite visualizar com mais clareza dependências e ferramentas úteis a serem utilizadas.

Todavia vemos que ambas os processos possuem premissas únicas capazes de efetivar a informação para o sistema desenvolvendo vantagens para cada uma que possibilitam o sucesso de um projeto. Porém, se durante alguma fase ocorrer um imprevisto afetaria todo desenvolvimento, podendo ser de modo mais grave afetando as funcionalidades inclusive o prazo ou de modo mais leve afetando apenas um modulo do sistema. Geralmente o que falta em um processo que o prejudica é explorado por outro e por esse motivo foi possível obter uma problemática que possibilite uma hipótese de melhoria, podendo compensar essas lacunas no desenvolvimento.

Em primeira instância resolvemos falar sobre como a engenharia de software afeta a sociedade moderna, afinal software é o produto de maior circulação da atualidade, gerenciando outra representação da modernidade a informação, que atualmente está sendo considerada a moeda mais valiosa desse século e por esse fato fez-se necessário a padronização da criação de um sistema, estabelecendo processos e técnicas para auxiliar nesse fim.

Em segunda instância mostramos dois dos processos bem conhecidos a engenharia de requisitos e as metodologias ágeis que possuem características especificas que as distinguem uma da outra, permitindo que sejam aplicadas para um mesmo fim com didáticas diferentes. No entanto mesmo tendo suas vantagens elas também possuem brechas que por sua vez pode permitir o fracasso de um projeto. Visto isso impúnhamos formas de minimizar essa

consequência, onde vemos que ao utilizar um *framework* originário dos métodos ágeis denominado Scrum, juntamente com a engenharia de requisitos seria possível estabelecer um elo entre esses processos. Podemos perceber que uma característica comum a ambos os processos são os requisitos que embora sejam elicitados diferentemente tem a mesma finalidade estabelecer a estrutura de um sistema, elucidando funcionalidade e restrições para que o sistema funcione de modo satisfatório baseado nas necessidades do cliente.

O Scrum foi a escolha mais indicada por ser um *framework* que permite a assimilação de técnicas em seu escopo pertencente a outras metodologias, assim podendo personalizar o processo de construção de um software para adequar as necessidades do produto e todos com envolvimento no projeto. Dessa forma buscamos meios para estruturar o escopo do *framework* sem prejudicar a sua vantagem principal proveniente da sua origem, o manifesto ágil sendo elas a agilidade e flexibilidade. Assim formamos pontes entre os dois processos destacando seus artefatos que representam meios para o estabelecimento de requisitos. Dentre os artefatos gerados pelo Scrum temos as user *stories*, *epics* e t*hemes* que fornecem descrições em alto nível para construir requisitos e/ou restrições, por outro lado, a engenharia de requisitos trabalha com métodos documentacionais que processam o requisito de maneira mais técnica e linear.

Em terceira instância buscamos trabalhar com métodos documentacionais sobre os artefatos do Scrum associando a engenharia de requisitos em seu escopo para possibilitar que a criação de uma documentação que sustentem informações sobre requisitos, afim de permitir assistência ao sistema em longo prazo para torna-lo sustentável com possibilidades de incremento e evolução. Entretanto é possível questionar se a inserção dessa metodologia não comprometeria a vantagem sobre mudanças no *framework*, porém, vemos que os artefatos do Scrum podem ser vistos como descrições que são utilizadas como parâmetros para implementação de funções e restrições, onde o mesmo processo ocorre em engenharia de requisitos através das descrições da documentação de requisitos.

Dessa forma podemos aplicar adaptações nos artefatos recorrentes do Scrum parar formas seções de um documento de requisitos seguindo a mesma estrutura desse método na engenharia de requisitos sendo estruturado como requisito funcionais e requisitos não funcionais, enquanto nos artefatos do Scrum seguindo essa estrutura poderiam ser divididos como funções e restrições.

A aplicação seria feita seguindo o fluxo de desenvolvimento Scrum e ao final de uma *sprint* com a revisão da *sprint*, onde durante a reunião a seriam preenchidos os parâmetros adaptados nas *users stories* e arquivando-a em um grupo que possuam *stories* similares para

dividir em seções um documento de *stories*, assim mostrando tornando o sistema ativo a longo prazo e preenchendo as lacunas que causam desvantagens associadas e agilizando o processo de mantenabilidade de um sistema, o que por consequência é uma característica do manifesto ágil.

Em conclusão vemos que o aprendizado nessa pesquisa foi dado de forma gradativa, pois diversos conceitos e meios a priori desconhecidos foram sendo conhecidos e caracterizados para melhor compreensão, além do que através desse estudo foi possível complementar o conhecimento obtido no meio acadêmicos. Ainda uma área embora subestimada é mais complexa e indispensável, já que é fundamental para o sucesso social, visto que a depender da natureza de um software uma falha poderia causar um abalo econômico considerável no meio utilizado, além do que em sistemas críticos pode significar uma ou várias fatalidades. O conhecimento obtido abriu um leque de opções para as mais variadas áreas de atuação da engenharia de software e foram através desses estudos que foi possível direcionar um caminho para atuação profissional. Entretanto essa pesquisa ainda está na sua fase hipotética, podendo ser lapidada e aplicada como técnica no desenvolvimento de software, visto que pode possibilitar inúmeras vantagens que complementam a agilidade e eficiências dos processos.

# REFERÊNCIAS

ALVES, Carina. Uma Experiência de Engenharia de Requisitos em Empresas de Software. In: **EIG**. 2008. p. 13-24.

AMBRÓSIO, Bernardo Giori; BRAGA, José Luis; OLIVEIRA, Alcione de Paiva. Um modelo dinâmico para análise dos impactos da rotatividade de pessoal durante a fase de requisitos. 2008.

ANDRADE, Antonio José F. et al. Gestão de projeto com Scrum: Um estudo de caso. **Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE). Aracat**i, p. 12, 2009.

ANG, Soon; SLAUGHTER, Sandra. Turnover of information technology professionals: the effects of internal labor market strategies. **ACM SIGMIS Database: the DATABASE for Advances in Information Systems**, v. 35, n. 3, p. 11-27, 2004.

BASSI FILHO, D. L. Experiências com desenvolvimento ágil. 2008. **Instituto de Matemática e Estatística, Universidade de São Paulo**, 2008.

BECK, Kent et al. **Manifesto for Agile Software Development**. Disponível em: <http://agilemanifesto.org/ >. Acesso em: 20 de fev. 2017.

BISSI, Wilson. SCRUM-Metodologia de desenvolvimento ágil. **Campo Digital**, v. 2, n. 1, p. 03-06, 2007.

CARNEVALI, Felipe Luiz; LUCRÉDIO, Daniel. Aplicando a Metodologia Ágil Scrum para apoio ao Gerenciamento de Requisitos. **Revista TIS**, v. 3, n. 2, 2014.

CARVALHO, Carlos Eduardo Costa de; ABRANTES, Carolina Thomé de; CAMEIRA, Renato Flórido. **Métodos ágeis de desenvolvimento de software: Um caso prático de aplicação do Scrum**. 2011.

CATUNDA, Edmar et al. Implementação do Nível F do MR-MPS com Práticas Ágeis do Scrum em uma Fábrica de Software. **X Simpósio Brasileiro de Qualidade de Software (SBQS'11), Curitiba–Brasil**, 2011.

COSTA, Nuno et al. Delivering user stories for implementing logical software architectures by multiple scrum teams. In: **International Conference on Computational Science and Its Applications**. Springer, Cham, 2014. p. 747-762.

DARWISH, Nagy Ramadan; MEGAHED, Salwa. Requirements Engineering in Scrum Framework. **Requirements Engineering**, v. 149, n. 8, 2016.

DA SILVA, Antonio Carlos; GARCIA, Ricardo Alexandre Martins. TEORIA DOS *STAKEHOLDER*S E RESPONSABILIDADE SOCIAL: algumas considerações para as organizações contemporâneas. **Trabalho de conclusão de curso para obtenção de nota parcial no curso de pós-graduação lato sensu à distância em MBA-Executivo em Gestão Empresarial pelo convênio UCDB/Portal da Educação**, 2011.

DE SOUZA RODRIGUES, Débora Carolina**. A Importância do Uso das Técnicas de Extração de Requisitos para o Desenvolvimento de Softwares**. 2013. p. 1-8.

DE REZENDE ALVES, Sérgio; ALVES, André Luiz; ALVES–ESPECIALISTA, André Luiz. Engenharia De Requisitos Em Metodologias Ágeis. 2010.

DIVIDINO, Renata Queiroz; FAIGLE, Ariadne. Distinções entre memória de curto prazo e memória de longo prazo. **Consultado a**, v. 8, 2004.

DOS SANTOS, Vinicius Salustiano Alves; CANEDO, Edna Dias. Agile methodology in the software development: Case study: Electoral justice of Brazil. In: **Information Systems and Technologies (CISTI), 2014 9th Iberian Conference on**. IEEE, 2014. p. 1-6.

DOS SANTOS SOARES, Michel. Metodologias ágeis extreme programming e scrum para o desenvolvimento de software. **Revista Eletrônica de Sistemas de Informação ISSN 1677-3071 doi: 10.21529/RESI**, v. 3, n. 1, 2004.

DOS PRAZERES LOPES, Luísa. **APLICAÇÃO DA METODOLOGIA SCRUM EM UMA ÁREA DE ENGENHARIA DE PROCESSOS DE UMA EMPRESA DO VAREJO**. 2017. Tese de Doutorado. Universidade Federal do Rio de Janeiro.

NUSEIBEH, Bashar; EASTERBROOK, Steve. Requirements engineering: a roadmap. In: **Proceedings of the Conference on the Future of Software Engineering**. ACM, 2000. p. 35-46.

ESPINOZA, Angelina; GARBAJOSA, Juan. A study to support agile methods more effectively through traceability. **Innovations in Systems and Software Engineering**, v. 7, n. 1, p. 53-69, 2011.

FERNANDES, Fabio et al. APLICABILIDADE DA ENGENHARIA DE REQUISITOS DE SOFTWARE COMO ATIVIDADE DA METODOLOGIA ÁGIL SCRUM. In: **Proceedings of International Conference on Engineering and Computer Education**. 2013. p. 258-261.

FRANCESCHI, Rudiney Altair; DUARTE, Ana Marcia Debiasi. Uma abordagem para gerência de requisitos integrada com práticas ágeis de gerência de projetos. 2011.

GIL, Antonio Carlos. Métodos e Técnicas de Pesquisa Social**. São Paulo. Atlas**.1994. p. 64-74.

IEEE Std 830-1998, **IEEE Recommended Practice for Software Requirements Specifications.**

IEEE Std 610.12-1990, **IEEE Standard Glossary of Software Engineering Terminology**, p. 67.

LOPES, Leandro Teixeira et al. **Um modelo de processo de engenharia de requisitos para ambientes de desenvolvimento distribuído de software**. 2005

NUSEIBEH, Bashar; EASTERBROOK, Steve. Requirements engineering: a roadmap. In: **Proceedings of the Conference on the Future of Software Engineering**. ACM, 2000. p. 35-46.

PRESSMAN, Roger S. Engenharia de software: Uma abordagem profissional. 7. ed. **Porto Alegre: AMGH**, 2011. 771 p. Tradução Ariovaldo Griesi.

PRODANOV, Cleber Cristiano; DE FREITAS, Ernani Cesar. **Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico-2ª Edição**. Editora Feevale, 2013.

ROBERTSON, James & Suzanne. Volere: Requirements Resources.16. ed. **Association Atlantic Systems**, 2012. 80 p.

SCHWABER, Ken; SUTHERLAND, Jeff. Um guia definitivo para o Scrum: As regras do jogo. **Julho de**, 2013.

SOMMERVILLE, Ian. Engenharia de Software. 9. ed. **São Paulo: Pearson Prentice Hall**, 2011. 529 p. Tradução Ivan Bosnic e Kalinga G. de O. Gonçalves.

SILVA, Antônio Mendes Filho. Requisitos Não Funcionais: Critérios para analise arquitetural. **Engenharia de Software Magazine**, ano 1, 3ª edição. 2008. p. 4-12.

SOARES, Rafael; CABRAL, Thiago; ALENCAR, Fernanda MR. Gerenciamento de Requisitos em Scrum baseado em Test Driven Development. In: **ER@ BR**. 2013.

STANDISH GROUP et al. **O relatório CHAOS**. 2014. [Consult. 17 Mai. 2018]. Disponível na internet <URL: http://www. standishgroup.com>.

TRINDADE, Gabriela Oliveira; LUCENA, Márcia. **Rastreabilidade de Requisitos em Metodologias Ágeis: um Estudo Exploratório**. 2016.

VALASKI, Joselaine et al. Apoio Semântico à Engenharia de Requisitos. In: **ER@ BR**. 2013.

VENTURA, Plínio. **Requisitos de software: Uma visão detalhada sobre requisitos funcionais, requisitos não-funcionais e regras de negócio**. 2016. [Consult. 5 Jan. 2018]. Disponível na internet <URL: https://pt.scribd.com/ document/321987497/eBook-Requisitos-Software-Plinio-Ventura>.

WAKE, Bill. **INVEST in Good Stories, and SMART Tasks**. 2003. [Consult. 17 Mai. 2018]. Disponível na internet <URL: https://xp123.com/articles/invest-in-good-stories-and-smart-tasks>.

ZANATTA, Alexandre Lazaretti; VILAIN, Patrícia. Uma análise do método ágil Scrum conforme abordagem nas áreas de processo Gerenciamento e Desenvolvimento de Requisitos do CMMI. In: **WER**. 2005. p. 209-220.

## APÊNDICE A – MODELO PARA DOCUMENTO DE USERS STORIES

# Documento de Users Stories

## Nome do sistema

Versão 0.0

# Histórico de Alterações

| Versão | Descrição | Data | Autor(es) |
|---|---|---|---|
| 0.0 | Descrição das alterações | dia/mês/ano | Nome |

# 1. Introdução

## 1.1. Objetivo

Descrição dos objetivos que deseja alcançar com o sistema.

## 1.2. Definições Siglas e Abreviações

Este tópico deve apresentar e definir todas as siglas ou abreviações que o documento possui.

## 1.3. Identificação das users stories

Este tópico deve definir todos os identificadores presente no documento, visto que cada um deve ser único. Exemplo: Identificador da *user story*[US001].

# 2. Descrição Geral do Sistema

Descrição porque, definindo para quem, onde e o público geral que irá utiliza-lo.

# 3. Users Stories

Esta sessão deve apresentar as *users stories* com todos os parâmetros inclusos.

---

**[US001]**

---

**Descrição**:

**Critérios de aceitação:**

**Parâmetros de restrição:**

**Acoplamento:**

---

**[US002]**

---

**Descrição**:

**Critérios de aceitação:**

**Parâmetros de restrição:**

**Acoplamento:**