

INSTITUTO FEDERAL DE EDUCAÇÃO,  
CIÊNCIA E TECNOLOGIA DA PARAÍBA

COORDENAÇÃO DO CURSO SUPERIOR DE  
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO



Jardel Brandon de Araujo Regis

## **GESTOR DE PROJETOS E PROCESSOS: GPP+**

**Campina Grande  
2022**

**Jardel Brandon de Araujo Regis**

## **GESTOR DE PROJETOS E PROCESSOS: GPP+**

Monografia apresentada à Coordenação do Curso de Engenharia de Computação do Instituto Federal de Educação Ciência e Tecnologia - Campus Campina Grande, como parte dos requisitos para a obtenção do grau de Engenheiro da Computação.

**Orientador:** Prof. Dr. Danyllo Wagner Albuquerque

**Campina Grande  
2022**

**Jardel Brandon de Araujo Regis**

## **GESTOR DE PROJETOS E PROCESSOS: GPP+**

Monografia apresentada à Coordenação do Curso de Engenharia de Computação do Instituto Federal de Educação Ciência e Tecnologia - Campus Campina Grande, como parte dos requisitos para a obtenção do grau de Engenheiro da Computação.

### **BANCA EXAMINADORA**

---

Nome do Orientador - **IFPB**

---

Nome do Primeiro Membro da Banca - **IFPB**

---

Nome do Segundo Membro da Banca - **IFPB**

**Instituto Federal De Educação, Ciência e Tecnologia  
Da Paraíba - Campus Campina Grande**

*Dedico este trabalho primeiramente a Deus, por ser essencial em minha vida e aos meus pais. Eles são o motivo pelo qual me esforço e me desafio todos os dias para ser uma pessoa melhor.*

## AGRADECIMENTOS

Ao Instituto Federal de Educação Ciência e Tecnologia da Paraíba (IFPB) - Campus Campina Grande pela experiência maravilhosa vivenciada durante a graduação. Agradeço a todos os docentes, técnicos e terceirizados que passaram em minha trajetória estudantil.

Obrigado a toda equipe de Coordenação de Apoio ao Estudante do IFPB Campina Grande (CAEST), por toda excelência nos trabalhos prestados no apoio e auxílio aos estudantes.

Gratidão eterna aos meus pais, à minha mãe Poliana Bezerra de Araujo Regis por sempre me incentivar nos estudos e por todo amor, cuidado e paciência dedicados a mim e ao meu pai João Bosco Regis por toda sua lealdade para comigo e por ser esse exemplo de ótima referência de ser humano, ao qual posso me espelhar sempre com muito orgulho.

Gratidão imensa a todos os meus familiares e amigos pelas orações e boas vibrações enviadas a mim durante a graduação, pela confiança e pela compreensão nas ausências necessárias para a conclusão desta jornada. Também agradeço ao meu irmão, Jamerson Brennon, porque sei que tenho e sempre terei um amigo e companheiro, sempre poderemos contar um com o outro.

Aproveito e agradeço também a todos os meus tios, em especial a Maria Tereza - mais conhecida por Dinha - por todo o apoio e orações dispensadas a mim de formas poderosas, grato também para com Suede Bezerra pelo apoio emocional e companheirismo que cresce a cada dia e presto o mesmo a, Lúcio Bezerra, que além de tio é um grande amigo e nunca mediou esforços para ajudar sempre que pudesse, estando presente desde o primeiro dia e a Josefa Teresa Regis Costa "*in memoriam*", que não está presente fisicamente neste momento, mas que queria agradecê-la por tudo, esteja onde estiver. Por fim, obrigado a todos os familiares que estiveram no auxílio de alguma forma fazendo possível essa aspiração acontecer.

Agradeço ao meu orientador, professor Dr. Danylo Wagner Albuquerque, que mesmo com um convite inesperado me aceitou com muito carinho como orientando para desenvolvermos esta pesquisa. Obrigado pelo tempo e esforços dedicados em fazer parte deste trabalho.

*“Todas as grandes conquistas da  
humanidade — da construção das  
grandes pirâmides à cura da poliomielite,  
bem com a visita do homem à Lua —  
começaram como um projeto”*

LARSON GRAY

## **RESUMO**

Considerando a crescente relevância do papel da gestão de projetos para a engenharia de *software* e boas práticas aos desenvolvedores independentes ou em organizações, que aumentam a demanda sobre produtos para este fim. Assim, esta pesquisa apresenta como objetivo geral. O desenvolvimento de uma ferramenta guia para fluxos de trabalho auxiliares a utilização de metodologias ágeis em projetos, propondo com o gestor de projetos, uma sugestão de alinhamento gradual de acordo com as práticas recomendadas. Trata-se de uma pesquisa qualitativa em que se foi realizada um levantamento bibliográfico de natureza básica estratégica, com o objetivo exploratório, de método hipotético-dedutivo, visando utilizar procedimentos de pesquisas documental, bibliográfica e experimental, para embasar e guiar o desenvolvimento da melhor forma possível do *software* proposto. O resultado da pesquisa visa trazer de forma prática em alinhamento com as disciplinas práticas do curso o desenvolvimento de um *software web* que realiza a comunicação via *interface* de aplicação com a ferramenta *Notion*, para que de acordo com os levantamentos das referências teóricas, utilizar como embasamento a principal literatura sobre o tema e todas as experiências em tais matérias relacionadas.

**Palavras-Chave:** Gestão de projetos; Engenharia de *software*; *Framework web* com API para *database*.

## **ABSTRACT**

Considering the growing importance of the role of project management for software engineering and good practices for independent developers or in organizations, which increase the demand on products for this purpose. Thus, this research presents as general objective. The development of a guide tool for workflows helps in the use of agile methodologies in projects, proposing with the project manager, a suggestion of gradual alignment according to recommended practices. It is a qualitative research in which a bibliographical survey of a basic strategic nature was carried out, with the exploratory objective, of a hypothetical-deductive method, through the use of documental, bibliographical and experimental research procedures, to support and guide the development of the best possible form of the proposed software. The result of the research aims to bring, in a practical way, in alignment with the practical disciplines of the course, the development of a web software that performs communication via the application interface with the Notion tool, so that, according to the surveys of theoretical references, use as a basis the main literature on the topic and all experiences in such related matters.

**Keywords:** Project management; Software Engineering; Web framework with API database.

## LISTA DE FIGURAS

Figura 1 - Ciclos de vida do gerenciamento de projetos	29
Figura 2 - Áreas de conhecimento	31
Figura 3 - Camadas da engenharia de software	45
Figura 4 - Custos de alterações como uma função do tempo em desenvolvimento	51
Figura 5 — Fluxo do processo Scrum	55
Figura 6 — Exemplo de quadro Kanban	58
Figura 7 — Scrum + Kanban = Scrumban	61
Figura 8 — Página inicial Scrumban+	71
Figura 9 — Time em cartões	73
Figura 10 — Quadro do time por papéis	74
Figura 11 — Quadro do time por papéis	76
Figura 12 — Quadro do time por papéis	77
Figura 13 — Quadro do time por papéis	77
Figura 14 — Backlog do produto: Projetos	79
Figura 15 — Backlog do produto: Tarefas	79
Figura 16 — Planejamento da sprint: Épicos	83
Figura 17 — Planejamento da sprint: Atividades	83
Figura 18 — Planejamento da sprint: Histórias do usuário	84
Figura 19 — Kanban board	84
Figura 20 — Registro das reuniões diária	85
Figura 21 — Entregáveis	89
Figura 22 — Registros das revisões de Sprint	89
Figura 23 — Registro das retrospectivas de Sprint	90
Figura 24 — Definição do estado de finalização de tarefas	90

## **LISTA DE ABREVIATURA E SIGLAS**

ABNT	Associação Brasileira de Normas Técnicas
ANA	Agência Nacional de Águas e Saneamento Básico
ANSI	<i>American National Standards Institute</i>
CBAP	Comissão Brasileira de Agricultura de Precisão
EMBRAPA	Empresa Brasileira de Pesquisa Agropecuária
ISO	<i>International Organization for Standardization</i>
MAPA	Ministério da Agricultura, Pecuária e Abastecimento
NBR	Norma Brasileira
OMS	Organização Mundial da Saúde
PMBOK	<i>Project Management Body Of Knowledge</i>
PMI	<i>Project Management Institute</i>

## SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>13</b>
1.1 CONSIDERAÇÕES GERAIS.....	14
1.2 APRESENTAÇÃO DO PROBLEMA.....	15
1.3 MOTIVAÇÃO DA SOLUÇÃO PROPOSTA.....	17
1.4 OBJETIVOS E CONTRIBUIÇÕES.....	18
1.4.1 Objetivo geral.....	18
1.4.2 Objetivos específicos.....	18
<b>2 REFERENCIAL TEÓRICO.....</b>	<b>19</b>
<b>2.1 GERENCIAMENTO DO PROJETO.....</b>	<b>19</b>
2.1.1 Project Management Institute (PMI).....	21
2.1.2 O guia PMBOK.....	22
2.1.3 Definição de projetos.....	22
2.1.3.1 <i>Diferenças entre Projetos e Processos</i> .....	24
2.1.3.2 <i>Fases de desenvolvimento de um projeto</i> .....	25
2.2 ENGENHARIA DE SOFTWARE.....	29
<b>2.2.1 Objetivo.....</b>	<b>30</b>
<b>2.2.2 Histórico.....</b>	<b>31</b>
<b>2.2.3 Requisitos de Software.....</b>	<b>32</b>
2.2.3.1 Design de Software.....	33
2.2.3.2 Construção de Software.....	33
2.2.3.3 Teste de Software.....	34
2.2.3.4 Manutenção de Software.....	34
2.2.3.5 Gerência de Configuração de Software.....	35
2.2.3.6 Gerência da Engenharia de Software.....	36
2.2.3.7 Processo de Engenharia de Software.....	36
2.2.3.8 Ferramentas e Métodos da Engenharia de Software.....	37
2.2.3.9 Qualidade de Software.....	37
<b>2.2.4 Metodologias.....</b>	<b>38</b>
<b>2.2.5 Desenvolvimento ágil.....</b>	<b>40</b>
2.2.5.1 Agilidade.....	41
2.2.5.2 Mudanças.....	43
2.2.5.3 Abordagens abrangentes.....	44
2.2.5.3.1 <i>Scrum</i> .....	45
2.2.5.3.2 <i>Kanban</i> .....	49
2.2.5.3.3 <i>Scrumban</i> .....	51
<b>3 METODOLOGIA.....</b>	<b>53</b>
3.1 ABORDAGEM.....	54
3.2 NATUREZA.....	55

3.3 OBJETIVOS.....	55
3.4 PROCEDIMENTOS.....	56
<b>4 DESENVOLVIMENTO.....</b>	<b>57</b>
4.1 O TIME SCRUM.....	58
<b>4.1.1 Product Owner.....</b>	<b>58</b>
<b>4.1.2 Scrum Master.....</b>	<b>58</b>
<b>4.1.3 Desenvolvedores.....</b>	<b>58</b>
4.1.3.1 Tamanho do time de desenvolvimento.....	59
<b>4.1.4 Notion - Atribuições do time.....</b>	<b>59</b>
4.1.4.1 Visualização em cartões.....	59
4.1.4.2 Visualização em quadro agrupado por função.....	59
4.2 GERENCIAMENTO DO ESCOPO.....	60
<b>4.2.1 Notion - Escopo.....</b>	<b>61</b>
4.2.1.1 Notion - Requerimentos.....	61
4.2.1.2 Notion - Produtos.....	62
4.2.1.3 Notion - Protótipos.....	63
4.3 CICLOS DO PROCESSO.....	63
<b>4.3.1 Pré-planejamento (Pre-game phase).....</b>	<b>64</b>
4.3.1.1 Artefatos.....	64
4.3.1.1.1 Backlog do produto.....	64
4.3.1.2 Notion - Artefatos.....	64
4.3.1.2.1 Notion - Backlog dos projetos.....	64
4.3.1.2.2 Notion - Backlog das atividades.....	65
<b>4.3.2 Desenvolvimento (Game phase).....</b>	<b>66</b>
4.3.2.1 Eventos.....	66
4.3.2.1.1 Planejamento da Sprint.....	66
4.3.2.1.2 Sprint.....	67
4.3.2.1.3 Reunião Diária.....	67
4.3.2.2 Notion - Eventos.....	68
4.3.2.2.1 Notion - Planejamento da Sprint.....	68
4.3.2.2.2 Notion - Sprint.....	70
4.3.2.2.3 Notion - Reunião diária.....	71
<b>4.3.3 Pós-planejamento (Post-game phase).....</b>	<b>71</b>
4.3.3.1 Encerramento.....	71
4.3.3.1.1 Incrementos.....	72
4.3.3.1.2 Revisão de Sprint.....	72
4.3.3.1.3 Retrospectiva da Sprint.....	73
4.3.3.1.4 Definição de “Pronto”.....	74
4.3.3.2 Notion - Encerramento.....	74
4.3.3.2.1 Notion - Incrementos.....	74

4.3.3.2.2 <i>Notion - Revisão da Sprint</i> .....	75
4.3.3.2.3 <i>Notion - Retrospectiva da Sprint</i> .....	75
4.3.3.2.4 <i>Notion - Definição de “Pronto”</i> .....	76
<b>5 CONSIDERAÇÕES FINAIS.....</b>	<b>77</b>
<b>REFERÊNCIAS.....</b>	<b>79</b>

## **1 INTRODUÇÃO**

Um dos grandes desafios da humanidade sempre foi produzir mais com menos recursos, com o desenvolvimento de *hardware* e *software* não se é diferente; obter maiores resultados, empregando menores esforços, vêm sendo constantemente um dos objetivos almejados no decorrer da evolução das tecnologias.

Considerando a grande ânsia da humanidade na busca da solução dos seus desafios, têm-se o desenvolvimento tecnológico como ferramentas aplicadas desde os primórdios da civilização humana, em prol do bem-estar populacional, de tal forma que ao se olhar para o passado vemos a importância de cada recurso ferramental tecnológico desenvolvido. A história da humanidade foi construída sempre na busca de conseguir dominar os recursos à sua disposição e transformá-los em produtos que pudessem proporcionar conforto, satisfação e, finalmente, crescimento socioeconômico.

Entretanto, com o passar do tempo a competição mercadológica e de soberania entre nações, levou-os a buscarem soluções provenientes de soberanias competitivas. (CANDIDO *et al.*, 2012).

O crescimento do gerenciamento de projetos também pode ser visto em sala de aula. Há 10 anos as universidades mais importantes ofereciam uma ou duas aulas em gerenciamento de projetos, principalmente para engenheiros. Atualmente, muitas delas oferecem múltiplas sessões de aulas sobre gerenciamento de projetos, em especial para grupos de engenheiros, bem como para estudantes de negócios com ênfase em *marketing*, sistemas de informações gerenciais, finanças e também de outras disciplinas como oceanografia, ciências da saúde, computação e comunicação.

Esses estudantes têm aprendido que sua exposição ao gerenciamento de projetos oferece vantagens distintas na hora de buscar emprego. Mais e mais empregadores procuram graduados com habilidades em gerenciamento de projetos. O ponto inicial lógico para o desenvolvimento dessas habilidades é o entendimento do caráter único de um projeto e de seus gerentes. (GRAY; LARSON, 2010).

Nesse sentido, o pressuposto de conclusão desta pesquisa é o desenvolvimento de um protótipo de um fluxo de trabalho na plataforma *Notion*,

utilizando a junção das ferramentas *Scrum* e *Kanban*, para que possa servir como base venha a ser utilizados por todos aqueles que considere adequada sua utilização em seu processo de gerenciamento do projeto e que possa ser utilizado por quaisquer pessoas que tenha acesso e conhecimento básicos a sistemas digitais, sejam para grandes e pequenas empresas ou como mais uma alternativa para pequenas equipes e desenvolvedores solos.

Com isso, esta pesquisa apresenta como objetivo geral utilizar a história do desenvolvimento da engenharia de *software* quanto ao gerenciamento de projetos como fio condutor para a apresentação de um fluxo de trabalho e metodologias que sugerem introduzir um novo estilo de gestão.

Sendo dividida, portanto, em cinco capítulos a saber: no primeiro capítulo que se estiver corrente, se é apresentado uma introdução geral e específica acerca da proposta de trabalho do projeto desenvolvido. No segundo capítulo vai ser apresentado o referencial teórico, contendo o levantamento do estado da arte de cada tópico da engenharia de *software* que embasam teoricamente esta pesquisa, possibilitando assim o completo entendimento das realizações propostas no capítulo um. No terceiro capítulo será apresentada a metodologia utilizada no desenvolvimento desta pesquisa, a fim de que seja possível a realização da conferência de corretude dos passos tomados, bem como, seja possível a realização da replicabilidade do estudo em diferentes condições, de tal forma conforme os princípios que regem a ciência pregam. O quarto capítulo conterá as etapas de desenvolvimento do projeto, que terá presente as etapas adotadas para a construção da pesquisa e também o seu desenrolar enquanto em busca dos objetivos projetados. No quinto capítulo vão ser apresentadas as conclusões a respeito do que foi descoberto durante a construção deste estudo.

## 1.1 CONSIDERAÇÕES GERAIS

Os avanços da ciência e da tecnologia permitiram ao homem ter uma melhor qualidade de vida. Mas, por outro lado, com o desenvolvimento humano, estamos assistindo a profundos aumentos de demandas das energias vitais do homem, nunca na história se produziu tanto em tão pouco tempo quanto agora, na era da informação.

Com isso, é necessário fazer uma gestão sustentável na criação e gestão de novos projetos, otimizar este gerenciamento deixou de ser opcional e tornou-se uma obrigação, tendo em vista a importância de garantir que os planejamentos sejam bem sucedidos, evitando assim, os prejuízos do fracasso. Para alcançar esse objetivo com primazia, é preciso entender e defender antes da gestão em si, aquilo que existe para garantir o direito a uma vida saudável e produtiva em harmonia com o meio, buscando sempre o desenvolvimento sustentável.

Pode-se considerar o Gerenciamento de Projetos como uma área autônoma nas organizações, assim como é tratada pelo Instituto Nacional de Ensino (INE). A Gerência de Projetos já existe há mais de 50 anos, e vem sendo praticada em diversos ramos de negócios. É o principal meio para lidar com a mudança de produtos, de serviços e de processos nas organizações contemporâneas. As pessoas envolvidas em projetos têm desenvolvido processos e técnicas especializadas para lidar com desafios de planejamento, organização e motivação dos membros das equipes, liderança de equipes de projetos, acompanhamento, avaliação e controle do emprego dos recursos. (INE, [s. d.])

Por conseguinte, o desenvolvimento de um sistema prático que possibilite a realização do gerenciamento de projetos do início ao fim, utilizando duas das maiores e mais utilizadas metodologias de trabalhos ágeis, tende a se tornar um recurso valioso, para aqueles que prezam pela qualidade do projeto e não desejam investir tempo na criação de um ferramental do “zero”.

Destarte, para realização deste projeto, foi-se realizado um embasamento teórico da área de gestão de projetos e engenharia de software, mais especificamente a utilização de metodologias ágeis para o desenvolvimento do mesmo, focando nos métodos *Scrum*, *Kaban* e um híbrido entre ambos denominado *Scrumban* como princípios norteadores do fluxo de gestão, que teve como ferramenta de criação o *Notion*, permitindo desta uma sugestão de fluxo de trabalho para gestores e time de desenvolvedores, que possam experienciar, caso estejam enfrentando alguma das dificuldades citadas ou apreciem as vantagens apresentadas pelo trabalho desenvolvido.

## 1.2 APRESENTAÇÃO DO PROBLEMA

Há décadas a crise do *software* instiga e inspira pesquisadores e profissionais a criarem propostas de soluções visando combatê-la e tornar projetos de criação de *software* mais produtivos e previsíveis; e com resultados de melhor qualidade.

Em verdade, uma consequência dessas propostas firmadas é a engenharia de *software*. Segundo o *Institute of Electrical and Electronic Engineering* (IEEE), a engenharia de *software* é a aplicação sistemática, disciplinada e com abordagem quantitativa para desenvolvimento, operação e manutenção de software, conforme definição descrita em IEEE STD 610.12-1990 (1990, p. 67).

Como sabido, os problemas acontecidos durante o período conhecido como a crise do *software* nunca deixaram de ocorrer, podendo se levantar a hipótese sobre se é correto realmente afirmar a denominação “crise do *software*” ou “calamidade crônica”.

De toda forma, também se é sabido que a criação da engenharia de *software*, pós período da crise do *software* em meados da década de 1979, embora providencial para solucionar a recorrência dos mais graves problemas ocorridos pela falte de controle e gerenciamento, mesmo assim, não foi uma solução definitiva para estes problemas, que ainda ocorrem atualmente, contudo, bem como sabemos por lógica, temos um ambiente melhor gerenciável e propício a criação de softwares, como podemos observar que em meio a complexidade e diversos desafios intrínsecos ao ramo, contudo, existe a evolução constante da tecnologia e dos atuais programas criados, que superam esta e várias outras crises já enfrentadas.

Destarte, em paralelo ao fazer a apresentação do problema, trago a situação obstáculo e faço a sugestão de fluxo de trabalho para lidar com ela, assim tais quais em semelhança houveram a crise do *software* e a engenharia de *software* respectivamente, onde a dificuldade mencionada foi encontrada durante desenvolvimento deste próprio trabalho, em um primeiro momento onde eu pretendia construir um protótipo de sistema de irrigação automática, sendo que para isto eu busquei, conforme disciplinas recomendadas durante meu curso de graduação, um software para realizar o gerenciamento do projeto, sendo este meu principal após a dificuldade de aquisição dos equipamentos, não por falta de opções existentes, mas sim, por falta de adaptabilidade ao contexto ou pela rigidez provocada pela falta de personalização do fluxo de trabalho. Por conseguinte, ao perceber tal deficiência e por estar utilizando muito tempo com esta parte em específico, optei então pela

migração do tema para o vigente neste trabalho desenvolvendo, procurando como aprimorar um fluxo de trabalho seguindo as recomendações propostas pelas práticas da engenharia de software e principalmente pensando na relevância que o mesmo pode tomar em ajudar pessoas em situação semelhante de forma prática?

### 1.3 MOTIVAÇÃO DA SOLUÇÃO PROPOSTA

Ao propor em um primeiro momento o desenvolvimento de um Sistema modular automatizado para monitoramento e controle de irrigação residencial/rural como projeto de conclusão de curso e após realizar a compra dos materiais necessários para início da montagem, durante etapas iniciais, onde buscava um método prático e eficaz para gerência do projeto que propus anteriormente, acabei desamparado por não ter nenhum opção satisfatória.

De tal forma, que o não cumprimento de tais características que estava em busca, levaram-me a direcionar as motivações desta pesquisa para um novo sentido, no qual me vi obrigado a investir um tempo considerável e necessário para resolução deste empecilho, optando por pivotar de temática e me dedicar primeiramente a algo que possa me ajudar e ajudar a todos no quesito prático da gestão de projetos e engenharia de software, que por conseguinte este seja um projeto embasador para não somente o desenvolvimento do projeto anteriormente iniciado mas, como também de auxílio a trabalhos vindouros, fazendo valer desta maneira o tempo investido nesta resolução

Como realizar a gestão de projetos de uma maneira mais eficiente, prática e com menor esforço? Ribeiro; Cunha e Ribeiro (2015, p. 10), em seu trabalho sobre métodos ágeis discorrem o seguinte acerca da necessidade da utilização das mesmas:

Pela necessidade de um aumento contínuo de competitividade, com o dinamismo e a velocidade com que a informação e o conhecimento circulam, o ambiente corporativo de várias empresas necessita utilizar a gerência através de projetos para se tornar mais eficaz, ágil e competitivo.

Não é exagero dizer que o tempo é o bem mais valioso da humanidade, esta necessidade do ser humano em utilizar o tempo da melhor forma possível, considerando que é impossível recuperá-lo, tem-se assim cenários como o citado

acima, onde a busca pela utilização deste recurso da melhor forma possível eleva as soluções já empregadas a fim de se economizar o tempo.

Desta maneira, a presente pesquisa é motivada pela busca de criação de uma ferramenta que proporcione o processo de planejamento e controle do tempo, de tal maneira que possa ser verificado métricas de utilização do tempo gasto de forma bem ou mal sucedidas, além dos registros geradas pela mesma, que possibilita sempre a melhoria contínua da gestão recursivamente.

#### 1.4 OBJETIVOS E CONTRIBUIÇÕES

A fim de buscar elementos que ajudem a discutir as questões norteadoras deste trabalho, relevantes a construção da plataforma gestora de projetos, foram desenvolvidos os seguintes objetivos:

##### 1.4.1 Objetivo geral

Elaborar, de acordo com os métodos ágeis, um fluxo de trabalho que permita o gerenciamento de projetos seguindo os melhores padrões do desenvolvimento de *software*, baseando-se nas plataformas de trabalho *Scrum*, *Kanban* e o híbrido de ambas. Utilizando para isto, a ferramenta *Notion*, que permite aplicar de forma prática todos os ricos conceitos teóricos abordados.

##### 1.4.2 Objetivos específicos

- Compreender a teoria de Cynefin.
- Compreender os principais da área de gestão de projetos relacionados ao desenvolvimento de *hardware* e *software*.
- Compreender o manifesto ágil e sobre as principais metodologias abrangidas pelo conceito, enfatizando o *Scrum* e *Kanban* que norteiam este trabalho.
- Criar o fluxo de trabalho com auxílio da plataforma *notion*, que permita a gestão e controle do desenvolvimento de projetos de *hardware* e *software*, baseado no *Scrumban*.

## **2 REFERENCIAL TEÓRICO**

Podemos imaginar de acordo com a definição do conceito de projeto dada logo mais nesta seção, que a realização de projetos se é algo intrínseco a todos os seres vivos e que permite a evolução da vida.

Com a evolução humana, projetos mais complexos e de maiores portes foram surgindo, de tal forma que, aos que possuíssem técnicas eficientes de execução dos projetos e fossem detentores de sucessos em seus planos, obtivessem maiores eficácia diante das abordagens na vida, assim naturalmente se deu a evolução da gestão de projetos, pelos benefícios trazidos dessa área do conhecimento científico.

Dessa forma se faz importante discorrer nesta seção os principais quesitos acerca da gestão de projetos em geral e a respeito também da engenharia de software, partindo-se do pressuposto de que há a utilização dessa ciência e ferramentas dela no desenvolvimento deste projeto teórico e prático abrangido pela engenharia de computação.

### **2.1 GERENCIAMENTO DO PROJETO**

A gestão de projetos não é uma ciência nova. Desde o início da existência humana verifica-se indícios de planejamento e organização de recursos para o alcance de algum objetivo, como citado anteriormente.

Voltando à antiguidade, relatos demonstram vestígios do gerenciamento de projetos arcaicos e princípios de engenharia básica. Exemplos clássicos estão nas pirâmides do Egito, Muralha da China e o Coliseu de Roma, destacados entre vários outros exemplos. Diversas técnicas sofisticadas de engenharia e gestão eram aplicadas como, por exemplo, para a construção de sistemas de esgoto e irrigação, embarcações, canais, palácios e templos. (VALLE et al., 2010).

Gerenciar Projeto é a aplicação dos conhecimentos, habilidades, ferramentas e técnicas nas atividades do projeto com o objetivo de atender os seus requisitos, e ainda, atender ou exceder as necessidades e expectativas dos

*stakeholders*<sup>1</sup>, envolvendo as variáveis, tais como: Escopo, prazo, custo e qualidade. O *Project Management Body of Knowledge* (PMBOK) serve como um documento que norteia um gerente de projeto a estruturar a execução de um projeto que esteja sendo desenvolvido. (VALLE *et al.*, 2010).

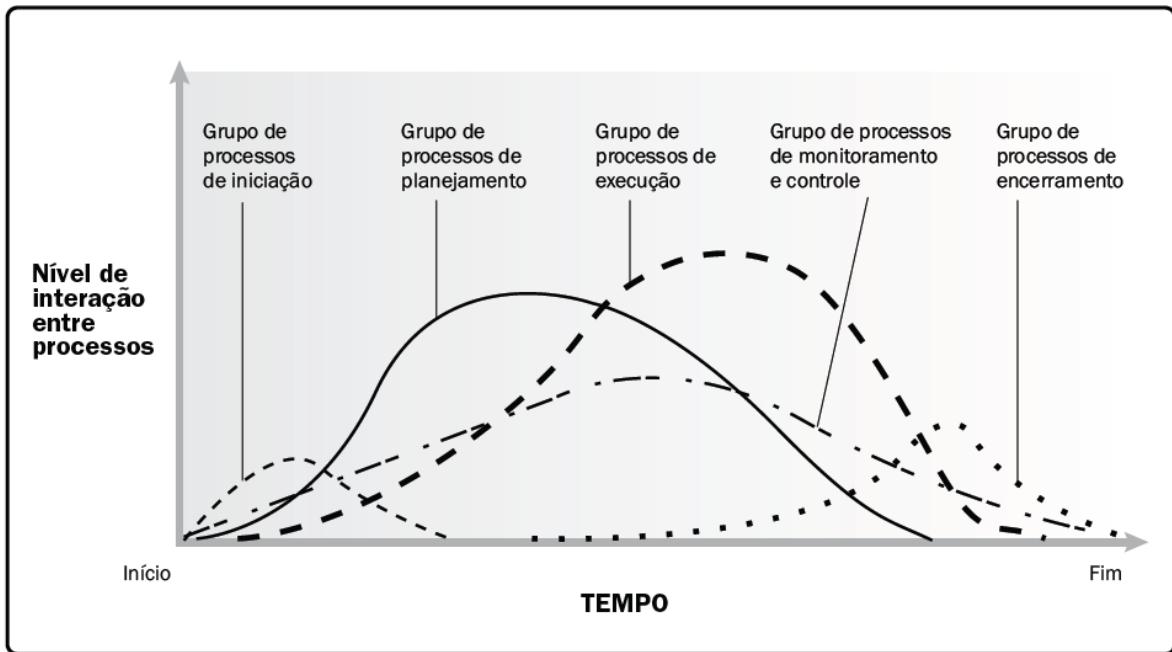
O gerenciamento de projetos é realizado por meio dos processos. Envolve as etapas selecionadas para desenvolver um projeto estesjam alinhados com uma visão sistêmica. Convém que cada fase do ciclo de vida do projeto tenha entregas específicas. Acorda-se que estas entregas possuam ligação com cada fase do ciclo de vida do projeto tenha entregas específicas. Combinando entregas que sejam regularmente analisadas e criticamente analisadas durante o projeto para atender os requisitos do patrocinador, clientes e outras partes interessadas. (ABNT, 2012).

Um projeto, por ser um empreendimento temporário, necessita que seja bem delimitado. Entretanto, isso não é uma tarefa simples de planejar sua evolução. Reconhecer onde termina um projeto e começa o ciclo de vida de seus produtos, ou compreender a relação entre as etapas do gerenciamento e as etapas de desenvolvimento do projeto (fases), que estão representadas na Figura 1 em um gráfico em que relaciona o tempo das fases com o nível de interações entre os processos.

---

<sup>1</sup> De acordo com Sommerville e Sawyer (1997) a definição dos “envolvidos” é dada como “qualquer pessoa que se beneficie de forma direta ou indireta do sistema que está sendo desenvolvido” (PRESSMAN; MAXIM, 2021, p. 244)

**Figura 1 - Ciclos de vida do gerenciamento de projetos**



Fonte: PMI, 2013, p. 51

Na prática, o ciclo de vida é utilizado por alguns grupos para descrever a distribuição de tempo para tarefas mais importantes durante um projeto. Por exemplo, a equipe de design deve planejar um maior comprometimento de recursos durante o estágio de definição, enquanto a equipe de qualidade deverá esperar que seus esforços aumentem durante os estágios finais do ciclo de vida do projeto. Já que muitas organizações possuem um portfólio de projetos que ocorrem ao mesmo tempo, cada um deles em um estágio diferente de seu ciclo de vida, o planejamento e a administração cuidadosa nos níveis da organização e do projeto são imperativos. (GRAY; LARSON, 2010).

### **2.1.1 Project Management Institute (PMI)**

O Project Management Institute (PMI), é uma organização internacional que tem como objetivo definir, detalhar e padronizar práticas e processos do gerenciamento de projetos. Em 1969, no auge dos projetos espaciais da NASA, um grupo de cinco profissionais de vanguarda que entendiam o valor do networking, do compartilhamento das informações dos processos e da discussão dos problemas comuns de projetos, da Filadélfia, Pensilvânia, nos EUA, reuniu-se para discutir as melhores práticas, e Jim Snyder fundou então o Project Management Institute - PMI (EUA). O PMI atualmente é a maior instituição internacional dedicada à

disseminação do conhecimento e ao aprimoramento das atividades de gestão profissional de projetos, sendo responsável pelo guia PMBOK (ENAP, 2014).

### **2.1.2 O guia PMBOK**

O PMBOK é um guia para gerenciamento de projetos que se tornou um marco na história da ciência do gerenciamento de projetos, sendo reconhecido, em 1999, como um padrão de gerenciamento de projetos pelo *American National Standards Institute* (ANSI) (PAES, 2014).

O objetivo do Guia PMBOK é o de identificar o subconjunto do conjunto de conhecimentos em gerenciamento de projetos que é amplamente reconhecido como boa prática. Uma boa prática não significa que o conhecimento descrito deverá ser sempre aplicado uniformemente em todos os projetos; a organização e/ou equipe de gerenciamento de projetos é responsável por determinar o que é adequado para um projeto específico.

O objetivo principal deste guia é identificar e condensar os conhecimentos, visões e práticas aplicáveis de forma que a aplicação correta dessas habilidades, ferramentas e técnicas contribuam para o aumento das chances de sucesso de uma série de projetos diferentes. (PMI, 2013).

O guia é baseado em processos e identifica um subconjunto do conjunto de conhecimentos em gerenciamento de projetos. Não se tratando apenas de uma metodologia e, sim, de uma padronização que identifica processos, áreas de conhecimento, técnicas, regras e métodos. Hoje ele se consolida como a mais popular referência em gerenciamento de projetos colocando mais de 4 milhões de cópias no mercado. (PAES, 2014).

### **2.1.3 Definição de projetos**

Em uma definição dada do que seria um projeto, por Gray e Larson (2010, p. 5), destacam que:

Um projeto está sempre submetido a alcançar objetivos com uma limitação de escopo e tempo. Um projeto é um esforço único, complexo e não rotineiro, limitado por orçamento, tempo e/ou recursos e com especificações de desempenho criadas de acordo com as necessidades de um cliente.

Na realidade, o conceito de projeto já é há décadas explorado de uma forma mais ampla, mesmo que indefinido informalmente. Há muito tempo a abordagem por projetos é usada desde a antiguidade com atividades de sobrevivência, até a atualidade seja para fazer negócios, no setor de construção, nas relações com *Hollywood* e com grandes empresas de consultoria. Dessa maneira, atualmente o gerenciamento de projetos atinge todos os tipos de trabalho, tendo assim de forma garantida sua importância.

Já a definição de projeto apontada pelo *Project Management Institute* (PMI), proposta pelo seguinte trabalho (PMI, 2013, p. 3) no seu guia de conhecimento sobre gerenciamento de projetos (PMBOK), que descreve a definição de projeto da seguinte forma:

Projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo. A natureza temporária dos projetos indica que eles têm um início e um término definidos. O término é alcançado quando os objetivos do projeto são atingidos ou quando o projeto é encerrado porque os seus objetivos não serão ou não podem ser alcançados, ou quando a necessidade do projeto deixar de existir.

Corroborando com os conceitos das definições aqui dadas, temos também segundo a Associação Brasileira de Normas Técnicas (ABNT) regulamentadora da Norma Brasileira (NBR) que foi certificada em conformidade com a International Organization for Standardization (ISO) de Nº 21500, decorrente de um grande esforço multinacional de 37 países, com participação majoritária do Brasil, a seguinte definição. (ABNT, 2012, p. 5):

Um projeto é um conjunto único de processos que consiste em atividades coordenadas e controladas com datas de início e fim, empreendidas para atingir os objetivos do projeto. O alcance dos objetivos do projeto é definido com datas de início e fim, empreendidas para atingir os objetivos do projeto. O alcance dos objetivos do projeto requer provisão de entregas, conforme requisitos específicos.

Assim temos vários conceitos convergentes sobre o que é um projeto, que no geral corresponde às expectativas dos conceitos informalmente formados, contudo, para se aprofundar ainda mais neste tema, se faz necessário saber que o projeto pode estar sujeito a múltiplas restrições, que o descharacterizam.

### 2.1.3.1 Diferenças entre Projetos e Processos

Levando em consideração que existem ambivalência e uma forte ligação entre processos e projetos, cabe abrir uma ressalva quanto às suas diferenças e peculiaridades, trazendo assim uma maior compreensão quanto ao tema, de forma que a seguir serão realizadas breves notações com este intuito.

A gestão de projetos e processos é fundamental para o sucesso de qualquer planejamento organizacional em rumo do objetivo específico, sabendo-se que o dinamismo e o conhecimento, são os pontos mais importantes para a gestão das organizações (CARDOSO; CARDOSO, 2018). Dar-se-ão:

- Projetos

Trabalho temporário que produz um resultado único, possui um começo, meio e fim. São atividades temporárias realizadas nas organizações que procuram gerar resultados melhores e possuem um começo, um meio e um fim.

- Exemplos

- Construção de um prédio;
    - Escrita de um livro;
    - Criação de uma nova linha de automóveis

- Características

- Temporário;
    - Gera resultado único;
    - Elaborado progressivamente.

- Processos

Trabalho contínuo que produz resultados padronizados, possuindo métodos bem definidos que asseguram o resultado. São atividades contínuas realizadas nas organizações pelos seus atores organizacionais, que possuem entradas, ferramentas, técnicas e saídas.

- Exemplos

- Manutenção de equipamentos;
    - Revisão de livros;
    - Fabricação de automóveis.

- Características

- Contínuo;
    - Gera resultados padronizados;
    - Fortemente definido.

- Resumo

Um processo é reconhecido pelo aprendizado através da repetição das atividades, onde quanto maior for a frequência e experiência da prática, teoricamente melhor será sua execução, já um projeto estará sempre ligado à inovação, ou seja, mesmo que dois projetos tenham objetivos e metodologias similares, sempre haverá características que lhes diferem. (CARDOSO; CARDOSO, 2018)

### 2.1.3.2 Fases de desenvolvimento de um projeto

Para orientar como deve ser elaborado um gerenciamento efetivo, o PMI apresenta algumas áreas que devem ser levadas em consideração durante a execução de um projeto. A Figura 2 apresenta as áreas da Gestão de Projetos de acordo com o Guia PMBOK.

Um projeto pode ser dividido em qualquer número de fases. A fase de um projeto é um conjunto de atividades relacionadas de maneira lógica que culmina na conclusão de uma ou mais entregas. As fases do projeto são usadas quando a natureza do trabalho a ser executado é única para uma parte do projeto, e são normalmente ligadas visando o desenvolvimento de uma entrega principal específica do pmi, conforme descrito (PMI, 2017, p. 547).

Ciclo de vida do projeto é a série de fases pelas quais um projeto passa, do início à conclusão. A fase de um projeto é um conjunto de atividades relacionadas de maneira lógica que culmina na conclusão de uma ou mais entregas. As fases podem ser sequenciais, iterativas ou sobrepostas. Os nomes, a quantidade e a duração das fases do projeto são determinados pelas necessidades de gerenciamento e controle das organizações envolvidas no projeto, pela natureza do projeto em si e sua área de aplicação.

Uma fase pode enfatizar os processos de um grupo específico de processos de gerenciamento do projeto, mas é provável que a maioria ou todos os processos sejam executados de alguma forma em cada fase (PMI, 2021).

Geralmente as fases são terminadas sequencialmente, mas podem se sobrepor em algumas situações do projeto. Fases distintas normalmente têm durações ou esforços diferentes. A natureza de alto nível das fases de um projeto as torna um elemento do ciclo de vida do projeto. (PMI, 2013).

**Figura 2 - Áreas de conhecimento**



Fonte: OLIVEIRA; CHIARI, 2015, p. 30

Os cinco grupos de processos descritos anteriormente são formados por processos que estão distribuídos por dez áreas de conhecimento em gerenciamento de projetos. São elas em termos gerais, as áreas vistas acima que podem ser interpretadas, segundo o PMI em seu guia **PMBOK (2013)**, como:

- Integração do Projeto:
  - área relacionada ao controle e acompanhamento das atividades do projeto para que a execução do mesmo se dê efetivamente;
  - Inclui os processos e as atividades necessárias para identificar, definir, combinar, unificar e coordenar os vários processos e atividades de gerenciamento do projeto dentro dos grupos de processos de gerenciamento do projeto.
- Gerenciamento do Escopo do Projeto:
  - área que lista detalhadamente todas as atividades necessárias que deverão ser realizadas para a conclusão do projeto;

- Inclui o processo de enunciar, de forma sucinta, os produtos ou serviços a serem fornecidos ao cliente.
- Gerenciamento do Tempo ou Prazo do Projeto:
  - área que visa alinhar o cumprimento das atividades com prazos pré-determinados a fim de garantir que o cronograma do projeto possa ser cumprido sem se extrapolar o prazo de execução do mesmo;
  - Inclui os processos necessários para gerenciar o término pontual do projeto.
- Gerenciamento de Custos:
  - área voltada por acompanhar e controlar os custos do projeto para garantir que o orçamento do projeto não seja extrapolado; Projeto Integração Escopo Tempo Custo Qualidade Recursos Humanos Comunicação Risco Aquisições Partes Interessadas;
  - Inclui os processos envolvidos em estimativas, orçamentos e controle dos custos, de modo que o projeto possa ser terminado dentro do orçamento aprovado.
- Gerenciamento da Qualidade do Projeto:
  - área destinada a acompanhar o desenvolvimento das atividades do projeto de modo a garantir que as necessidades dos clientes sejam satisfeitas ao final da execução do projeto;
  - Inclui os processos e as atividades da organização executora que determinam as políticas de qualidade, os objetivos e as responsabilidades, de modo que o projeto satisfaça as necessidades para as quais foi empreendido.
- Gerenciamento dos Recursos Humanos:
  - área que coordena a distribuição das atividades entre a equipe executora do projeto de modo a maximizar a utilização dos recursos humanos para a execução do projeto;
  - Inclui os processos que organizam e gerenciam a equipe do projeto. Abrange todos os aspectos do gerenciamento e interação das pessoas, incluindo liderança, contratação,

treinamento, resolução de conflitos, avaliação de desempenho etc.

- Gerenciamento da Comunicação:
  - área voltada para assegurar que a geração, distribuição e armazenamento da comunicação durante o projeto seja feita de forma eficiente de modo que todas as pessoas recebam as informações que necessitam para o desenvolvimento do projeto;
  - Inclui os processos necessários para assegurar que as informações do projeto sejam geradas, coletadas, distribuídas, armazenadas, recuperadas e organizadas de maneira oportuna e adequada.
- Gerenciamento de Riscos do Projeto:
  - área que realiza o prévio levantamento de todos os riscos associados à execução de um projeto antes mesmo de seu início. Este gerenciamento além de levantar os riscos, propõe um plano de ação para tratar adequadamente esses riscos de acordo com o objetivo da empresa e do projeto.
  - Inclui os processos relacionados com o planejamento, identificação, análise, elaboração de respostas, monitoramento e controle dos riscos em um projeto. Compreende a maximização dos resultados de eventos positivos e a minimização das consequências de eventos negativos.
- Gerenciamento das Aquisições:
  - área voltada para coordenar como serão realizadas as compras e aquisições do projeto em conformidade com o orçamento do mesmo e dos prazos estabelecidos;
  - Inclui os processos de compra ou aquisição de produtos, serviços ou resultados externos à equipe do projeto necessários para realizar o trabalho.
- Gerenciamento das Partes Interessadas:
  - área que acompanha e organiza os *stakeholders* de cada projeto e acompanha as expectativas e necessidades dos mesmos com relação à execução de um projeto.

- Inclui os processos necessários para identificar todas as pessoas ou organizações impactadas pelo projeto, analisando as expectativas dessas partes interessadas e seu impacto no projeto, bem como adotando estratégias apropriadas para o efetivo engajamento delas nas decisões e na execução do projeto.

## 2.2 ENGENHARIA DE SOFTWARE

No mundo moderno, tudo é *software*. Hoje em dia, por exemplo, empresas de qualquer tamanho dependem dos mais diversos sistemas de informação para automatizar seus processos. Governos também interagem com os cidadãos por meio de sistemas computacionais, por exemplo, para coletar impostos ou realizar eleições.

Empresas vendem, por meio de sistemas de comércio eletrônico, uma gama imensa de produtos, diretamente para os consumidores. O Software está também embarcado em diferentes dispositivos e produtos de engenharia, incluindo automóveis, aviões, satélites, robôs, etc. Por fim, o software está contribuindo para renovar indústrias e serviços tradicionais, como telecomunicações, transporte em grandes centros urbanos, hospedagem, lazer e publicidade. (VALENTE, 2020).

Portanto, devido a sua relevância no nosso mundo, não é surpresa que exista uma área da Computação destinada a investigar os desafios e propor soluções que permitam desenvolver sistemas de *software* — principalmente aqueles mais complexos e de maior tamanho — de forma produtiva e com qualidade. Essa área é chamada de Engenharia de *Software*. (VALENTE, 2020).

O mundo moderno não poderia existir sem o *software*. Infraestruturas e serviços nacionais são controlados por sistemas computacionais, e a maioria dos produtos elétricos inclui um computador e um software que o controla. A manufatura e a distribuição industriais são totalmente informatizadas, assim como o sistema financeiro. A área de entretenimento, incluindo a indústria da música, jogos de computador, cinema e televisão, faz uso intensivo de *software*. Portanto, o rumo que estuda os métodos de desenvolvimento de *software* é essencial para o funcionamento de sociedades nacionais e internacionais. (SOMMERVILLE, 2011b).

Os sistemas de *software* são abstratos e intangíveis. Eles não são restringidos pelas propriedades dos materiais, nem governados pelas leis da física ou pelos processos de manufatura. Isso simplifica a engenharia de *software*, porque não há limites naturais para o potencial do *software*. No entanto, devido a essa falta de restrições físicas, os sistemas de software podem se tornar extremamente complexos de modo muito rápido, difíceis de entender e caros para alterar. (SOMMERVILLE, 2011b).

Existem vários tipos de sistemas de *software*, desde os simples sistemas embutidos até os sistemas de informações complexos, de alcance mundial. Não faz sentido procurar notações, métodos ou técnicas universais para a engenharia de *software*, porque diferentes tipos de *software* exigem abordagens diferentes. Desenvolver um sistema de informações corporativo é totalmente diferente de desenvolver um controlador para um instrumento científico. Nenhum desses sistemas tem muito em comum com um jogo computacional com gráficos intensos. Todas essas aplicações precisam de normas, embora não necessitem das mesmas técnicas. (SOMMERVILLE, 2011b).

### 2.2.1 Objetivo

Tratar da aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para desenvolver, operar, manter e evoluir *software*. Ou seja, Engenharia de *Software* é a área da Computação que se preocupa em propor e aplicar princípios de engenharia na construção de *software*.(IEEE STD 610.12-1990, 1990).

Inúmeras pessoas escrevem programas. Pessoas envolvidas com negócios escrevem códigos em planilhas para simplificar seu trabalho; cientistas e engenheiros escrevem códigos para processar seus dados experimentais; e há aqueles que escrevem códigos como *hobby*, para seu próprio interesse e diversão.

No entanto, a maior parte do desenvolvimento de *software* é uma atividade profissional, em que o *software* é desenvolvido para um propósito específico de negócio, para inclusão em outros dispositivos ou como produtos de *software* como sistemas de informação, sistemas CAD etc. O *software* profissional, que é usado por alguém além do seu desenvolvedor, é normalmente criado por equipes, em vez de indivíduos. Ele é mantido e alterado durante sua vida. (SOMMERVILLE, 2011b).

Por outro lado, esta engenharia, tem por objetivo apoiar o desenvolvimento profissional de *software*, mais do que a programação individual. Ela inclui técnicas que apoiam especificação, projeto e evolução de algoritmos, que normalmente não são relevantes para o desenvolvimento de *software* pessoal. Para ajudá-lo a ter uma visão geral sobre o que trata o objeto de estudo.(SOMMERVILLE, 2011b).

Muitas pessoas pensam que *software* é simplesmente outra palavra para arquivos executáveis de computador. No entanto, quando falamos de engenharia de *software*, não se trata apenas do algoritmo em si, mas de toda a documentação associada e dados de configurações necessários para fazer esse algoritmo funcionar corretamente. Um sistema de *software* desenvolvido profissionalmente é, com frequência, mais do que apenas uma interface de comandos de linha (CUI = *Command-Line Interface*) ou uma interface gráfica de usuário (GUI = *Graphical User Interface*); ele normalmente consiste em uma série de linhas de instruções separados por contextos e arquivos de configuração que são usados para configurar esses códigos. Isso pode incluir a documentação do sistema, que descreve a sua estrutura; documentação do usuário, que explica como usar o sistema; e sites, para usuários baixarem a informação recente do produto. (SOMMERVILLE, 2011b).

Essa é uma diferença importante entre desenvolvimento de *software* profissional e amador. Se você está escrevendo um programa para si mesmo, que ninguém mais usará, você em tese não precisa se preocupar em escrever o manual do trabalho desenvolvido, documentar sua arquitetura etc. No entanto, se você está escrevendo um *software* que outras pessoas usarão e no qual outros engenheiros farão alterações, então você provavelmente deve fornecer informação adicional, assim como a sequência de instruções escritas para serem interpretadas por um computador para executar as tarefas específicas do produto. (SOMMERVILLE, 2011b).

### 2.2.2 Histórico

Historicamente, a área surgiu no final da década de 60 do século passado. Nas duas décadas anteriores, os primeiros computadores modernos foram projetados e começaram a ser usados principalmente para resolução de problemas científicos. Ou seja, nessa época o *software* não era uma preocupação central, mas sim construir máquinas que pudessem executar alguns poucos programas. Em resumo, computadores eram usados por poucos e para resolver apenas problemas científicos.

No entanto, progressos contínuos nas tecnologias de construção de *hardware* mudaram de forma rápida esse cenário. No final da década de 60, computadores já eram mais populares, já estavam presentes em várias universidades norte-americanas e europeias e já chegavam também em algumas grandes empresas. Os cientistas da computação dessa época se viram diante de um novo desafio: como os computadores estavam se tornando mais populares, novas aplicações não apenas se tornavam possíveis, mas começavam a ser demandadas pelos usuários dos grandes computadores da época. Na verdade, os computadores eram grandes no sentido físico e não em poder de processamento, se comparado com os computadores atuais. Dentre essas novas aplicações, as principais eram sistemas comerciais, como folha de pagamento, controle de clientes, controle de estoques, etc. (VALENTE, 2020).

Em 1958, John Tukey, o estatístico de renome mundial, cunhou o termo *software*. O termo "Engenharia de Software" foi utilizado no título de uma conferência da Otan, realizada na Alemanha em 1968. Em 1972, a IEEE Computer Society publica pela primeira vez seu relatório sobre Engenharia de *Software*. Em 1976 foi fundada uma comissão dentro da IEEE Computer Society com a incumbência de desenvolver padrões desta "ciência".

A primeira visão geral sobre que saiu da IEEE Computer Society resultou de um esforço liderado por Fletcher Buckley, no padrão IEEE 730 que versava sobre garantia de qualidade de *software* e que foi concluído em 1979. O propósito do padrão IEEE 730 era fornecer requisitos mínimos de uniformidade para a preparação e conteúdo do planejamento de *software*.

Esta norma influenciou o desenvolvimento e conclusão de outros temas: gerenciamento de configuração, teste de *software*, requisitos de *software*, design de *software*, verificação e validação de *software*. No período de 1981-1985, a IEEE Computer Society realizou uma série de oficinas sobre a aplicação de padrões de engenharia de

*software*. Nessas oficinas, profissionais envolvidos compartilham suas experiências com as atuais normas.

Também se realizavam nestas oficinas sessões onde eram discutidos os rumos para as normas, incluindo as medidas e métricas para a engenharia de software, produtos e processos. O planejamento resultou também no padrão IEEE 1002, padrão de taxonomia (1986), que proporcionou uma nova visão da nova engenharia abordada. Ele explica vários tipos de padrões, seus relacionamentos funcionais e externos bem como o papel das várias funções participantes no ciclo de vida do *software*. (SANTIAGO, 2011).

### 2.2.3 Requisitos de Software

Um requisito é definido como uma propriedade que deve ser exposta para resolver algum problema do mundo real. A primeira subárea de conhecimento é chamada de Fundamentos de Requisitos de *Software*. Ela inclui definições dos próprios requisitos de *software*, mas também os principais tipos de requisitos como: diferença entre produto e processo, propriedades funcionais e não funcionais e propriedades emergentes. A subárea também descreve a importância de requisitos quantificáveis e também apresenta a diferença entre requisitos de *software* e requisitos de sistemas.

A segunda subárea de conhecimento é o Processo de Requisitos, que introduz o próprio processo, orientando o resto das cinco subáreas e exposição como a engenharia de requisitos se relaciona com os demais processos de engenharia de *software*. Ela descreve modelos de processo, atores de processo, processos de suporte e gerenciamento, e o processo de qualidade e melhoria. (SANTIAGO, 2011).

A terceira subárea é a Elicitação de Requisitos, que descreve de onde os requisitos de *software* vêm e como o engenheiro de *software* pode obtê-los. Ele inclui as fontes de requisitos e técnicas elicitação.

A quarta subárea, Análise de Requisitos, se preocupa com o processo de analisar os requisitos para:

- Detectar e resolver conflitos entre requisitos
- Descobrir os limites do *software* e como ele deve interagir com o seu ambiente
- Elaborar requisitos de sistema de forma a se transformarem requisitos de *software*

A análise de requisitos inclui a classificação dos requisitos, a modelagem conceitual, o desenho da arquitetura e alocação de requisitos, e a negociação de requisitos.

A quinta subárea é a Especificação de Requisitos. A especificação de requisitos tipicamente refere-se à produção de um documento, ou o seu equivalente eletrônico, que pode ser sistematicamente revisto, avaliado e aprovado. Para sistemas complexos, em particular os que implicam o uso de componentes que não necessariamente são *software*. No mínimo, três tipos diferentes de documentos são produzidos: definição de sistema, especificação de requisitos do sistema, e especificação de requisitos de *software*. A subárea descreve os três documentos e as atividades subjacentes.

A sexta subárea é a Validação de Requisitos, cujo objetivo é buscar qualquer problema antes que os recursos sejam implantados no envio dos requisitos. A validação de requisitos preocupa-se com o processo de examinar os documentos de

requisitos para assegurar que eles estão definindo o sistema correto (isto é, o sistema que o usuário espera). É subdividido em descrições de procedimento de revisão de requisitos, prototipação, validação de modelo e testes de aceitação.

A sétima subárea, que é a última subárea, é chamada de Considerações Práticas. Ela descreve os tópicos que têm de ser compreendidos na prática. O primeiro tópico é a natureza iterativa do processo de requisitos. Os próximos três tópicos são fundamentalmente sobre a gerência de mudanças e a manutenção de requisitos em um estado que exatamente reflete o *software* a ser construído, ou que já tenha sido construído. Ele inclui gerência de mudanças, atributos de requisitos, e investigação de requisitos. O tópico final é a medição de requisitos. (SANTIAGO, 2011).

#### 2.2.3.1 Design de Software

De acordo com a definição do IEEE [IEEE 610.12-90], design é "o processo de definir a arquitetura, componentes, interfaces, e outras características de um sistema ou componente" e também "o resultado de processo". A *Knowledge Area* (KA) Design de Software é dividida em seis subáreas.

A primeira subárea apresenta os Fundamentos de Design de Software, que formam uma base subjacente à compreensão do papel e do escopo do design de *software*. Existem conceitos de *software* genéricos, o contexto do design de *software*, o processo de design de *software*, e as técnicas de autorização de design de *software*.

A segunda subárea agrupa o conjunto de Questões-chave em Design de Software. Essas questões chaves incluem colaboração, controle e tratamento de eventos, a distribuição de componentes, tratamento de exceções e erros e tolerância à falha, interação e apresentação, e persistência de dados.

A terceira subárea é Estrutura e Arquitetura de Software, os tópicos da qual são estruturas de arquiteturas e pontos de vista, estilos de arquitetura, padrões de design, e, finalmente, as famílias dos programas e *frameworks*.

A quarta subárea descreve Análise de Qualidade do Design e Avaliação do *software*. Enquanto existe uma KA inteira dedicada à qualidade de *software*, esta subárea apresenta os tópicos especificamente relacionados ao design de *software*. Esses aspectos são atributos de qualidade, análise de qualidade, e técnicas de avaliação e medidas.

A quinta subárea é Notações de Design de Software, que é dividida em descrições estruturais e comportamentais.

A última subárea descreve Estratégias de Design de Software e Métodos. Primeiro, as estratégias gerais são descritas, seguidas por métodos de design orientados por função, métodos de design orientados a objeto, design centrando na estrutura de dados, design baseado em componentes e outros. (SANTIAGO, 2011).

#### 2.2.3.2 Construção de Software

Construção de *software* se refere à criação detalhada de trabalho, significando *software* através da combinação de codificação, verificação, testes de unidades, testes de integração, e depuração. A KA possui três subáreas.

A primeira subárea é chamada de Fundamentos de Construção de Software.

Os três primeiros tópicos são princípios básicos da construção: minimização da complexidade, antecipação de mudanças e construção com foco na verificação. O tópico último discute padrões da construção.

A segunda subárea é descreve o Gerenciamento da Construção. Os tópicos tratados são modelos de construção, planejamento da construção e mensuração da construção.

A terceira subárea cobre as Considerações Práticas. Os tópicos são design de construção, linguagens de programação, codificação, teste de construção, reutilização, qualidade de construção, e integração. (SANTIAGO, 2011).

#### 2.2.3.3 Teste de *Software*

Teste de *Software* compõe-se da verificação dinâmica de uma seleção de domínios de execuções normalmente infinito, contra o comportamento esperado. Ela inclui cinco subáreas. A KA inicia com a descrição de Fundamentos de Testes de *Software*. Primeiro a terminologia de testes é apresentada, então são descritos assuntos relacionados com testes de *software*, e finalmente, os relacionamentos entre testes e as outras atividades são descritos.

A segunda subárea é chamada de Níveis de Teste. É dividida entre os alvos dos testes e os objetivos dos testes.

A terceira subárea é chamada de Técnicas de Teste. A primeira categoria inclui os testes baseados na intuição e experiência do testador. Um segundo grupo compreende técnicas baseadas na especificação, seguidas por baseadas no código, em falhas, no uso e baseadas na natureza da aplicação. Uma discussão de como selecionar e combinar as técnicas apropriadas também é apresentada.

A quarta subárea cobre Mensurações Relacionadas aos Testes. As medidas são agrupadas conforme o relacionamento de forma a avaliar o programa que está sofrendo testes e avaliar os testes executados.

A última subárea descreve o Processo de Testes e inclui considerações práticas e as atividades de teste. (SANTIAGO, 2011).

#### 2.2.3.4 Manutenção de *Software*

Uma vez na operação, as anomalias são descobertas, modificação no ambientes operacional, e novos requisitos dos usuários são expostos. A fase de manutenção do ciclo de vida começa a partir da entrega, porém, as atividades de manutenção ocorrem muito antes. A KA de Manutenção de *Software* é dividida em quatro subáreas.

Primeiramente é apresentado os Fundamentos de Manutenção de *Software*: definições e terminologia, a natureza de manutenção, a necessidade de manutenção, os principais custos de manutenção, evolução de *software* e as categorias de manutenção.

A segunda subárea agrupa as Questões-chave em Manutenção de *Software*. São as questões técnicas, de gerência, estimativas de custos de manutenção, e a mensuração da manutenção de *software*.

A terceira subárea descreve o Processo de Manutenção. Os tópicos aqui são os processos de manutenção e atividades de manutenção.

Técnicas para Manutenção constituem a quarta subárea. Essa subárea inclui a compreensão de programa, a reengenharia, e a engenharia reversa. (SANTIAGO, 2011).

#### 2.2.3.5 Gerência de Configuração de *Software*

A Gerência de Configuração de *Software* (GCS) é a disciplina de identificar a configuração do *software* em pontos distintos no tempo com o propósito de sistematicamente controlar modificações à configuração e de manter a integridade e a auditoria da configuração em todas as partes do ciclo de vida de sistema. Este KA inclui seis subáreas.

A primeira subárea é a Gerência do Processo de GCS. Ela cobre os tópicos do contexto organizacional de GCS, limites e orientação para o GCS, planejamento para o GCS, o próprio plano de GCS, e a monitoração da GCS.

A segunda subárea é a Identificação de Configuração de *Software*, que identifica itens a serem controlados, estabelece esquemas de identificação dos itens e as suas versões, também estabelece os instrumentos e técnicas a serem utilizadas na aquisição e no gerenciamento dos itens controlados. Os primeiros tópicos nesta subárea são a identificação dos itens a serem controlados e a biblioteca de *software*.

A terceira subárea é o Controle de Configuração de *Software*, que é a gerência de mudanças durante o ciclo de vida de *software*. Os tópicos são: primeiro, solicitando, avaliando, e aprovando alterações no *software*; em segundo lugar, implementação de modificações no *software*; e terceiro, desvios e não aprovações (renúncia de alterações).

A quarta subárea é a Registros de Estado de Configuração de *Software*. Os seus tópicos são a informação sobre posição de configuração do *software* e a reportagem do estado de configuração do *software*. A quinta subárea é a Auditoria de Configuração de *Software*. Ele compõe-se da auditoria de configuração funcional do *software*, auditoria de configuração física do *software*, e auditorias no processo a partir de uma base de referência do *software*.

A última subárea é a Gerência de Liberação e Entrega de *Software*, cobrindo elaboração de versões de *software* e gerenciamento de liberação de *software*. (SANTIAGO, 2011).

#### 2.2.3.6 Gerência da Engenharia de *Software*

O KA Gerenciamento da Engenharia de *Software*, aponta sobre o gerenciamento e mensuração. Enquanto que a mensuração é um aspecto importante em todas as KAs, está aqui que o tópico de programas de mensuração é apresentado. Há seis subáreas na KA de gerenciamento de engenharia de *software*. As cinco primeiras cobrem assuntos relacionados com gerenciamento de projetos de *software* e a sexta descrevem executáveis de mensuração de *software*.

A primeira subárea é a Iniciação e Definição de Escopo, que compreende a determinação e a negociação de requisitos, análise de praticabilidade, e processo da reavaliação e a revisão de requisitos.

A segunda subárea é o Planejamento de Projeto de *Software* e inclui o planejamento de processo, determinando pacotes de trabalhos (*deliverables*), o esforço, agendamento e a estimativa de custos, a alocação de recursos, a gerência dos riscos, a gerência de qualidade, e o plano de gerenciamento.

A terceira subárea é a Aprovação do Projeto de *Software*. Os tópicos tratados nesta KA são a implementação de planos, gerência de contrato de fornecedores, a implementação do processo de mensuração, monitoração de processo, controle de processo, e a divulgação de informações do projeto.

A quarta subárea é Revisão e Avaliação, que inclui os tópicos de determinar a satisfação dos requisitos e revisão e avaliação da performance.

A quinta subárea descreve o Fechamento: determinação de fechamento e atividades de fechamento.

Finalmente, a sexta subárea descreve a Mensuração da Engenharia de *Software*, mais especificamente, programas de mensuração. As mensurações de produto e processos são descritos na KA Processo de Engenharia de *Software*. Muitas outras KA também descrevem medidas específicas para a sua respectiva KA. Os tópicos desta subárea incluem o estabelecimento e comprometimento da mensuração, planejamento do processo de mensuração, execução do processo de mensuração, e avaliação da mensuração. (SANTIAGO, 2011).

#### 2.2.3.7 Processo de Engenharia de Software

A KA Processo de Engenharia de Software trata da definição, implementação, avaliação, mensuração, gerenciamento, alterações e melhora do próprio processo. É dividida em quatro subáreas.

A primeira subárea apresenta o Processo de Implementação e Mudanças. Os tópicos nesta KA são a infraestrutura de processo, o ciclo do processo de gerenciamento do *software*, modelos de implementação do processo e mudanças e considerações práticas.

A segunda subárea trata da Definição do Processo. Ela inclui os tópicos de modelos de ciclo de vida do *software*, processos de ciclo de vida de *software*, notações das definições do processo, adaptação do processo e automação.

A terceira subárea é a Avaliação de Processo. Os tópicos aqui incluem modelos de avaliação de processo e métodos de avaliação do processo.

A quarta subárea descreve Mensuração de Produto e Processo. O processo de engenharia de *software* cobre a medição do produto de processo em geral. As mensurações específicas de cada KA são abordadas nas respectivas KA. Os tópicos são mensuração do processo, mensuração do produto de *software*, a qualidade da mensuração dos resultados, modelos de informação de *software* e técnicas de mensuração do processo. (SANTIAGO, 2011).

#### 2.2.3.8 Ferramentas e Métodos da Engenharia de Software

A KA Ferramentas e Métodos para Engenharia de *Software* inclui, como a própria descrição apresenta, ferramentas para serem aplicadas à engenharia de *software* e também métodos para serem aplicados no objeto de estudo.

A subárea de Ferramentas para a regulamentação para criação de programação de computadores usa a mesma estrutura que Guia, com um tópico de cada uma das nove KAs. Em um tópico adicional é fornecido: assuntos sobre ferramentas diversas, como ferramentas para técnicas de integração, que são potencialmente aplicáveis a todas as classes de ferramentas.

A subárea Métodos é dividida em quatro subseções: métodos heurísticos que tratam com aproximações informais, métodos formais que se baseiam em aproximações matematicamente, e métodos de prototipação se baseiam em aproximações de desenvolvimento de *software* em torno de múltiplas formas de prototipação. (SANTIAGO, 2011).

#### 2.2.3.9 Qualidade de *Software*

A KA Qualidade de *Software* aborda considerações relativas à qualidade de software que vão além dos processos de ciclo de vida de *software*. Uma vez que a qualidade de *software* é um assunto presente em todas as partes na engenharia de *software*, também é considerado em muitas outras KAs, e o leitor notará pontos desta KA nas outras KAs do Guia. Esta KA possui três subáreas.

A primeira subárea descreve Fundamentos da Qualidade de *Software* como uma cultura e ética na engenharia de *software*, os valores e custos da qualidade, características de modelos e qualidade, e melhoria da qualidade.

A segunda subárea trata dos Processos de Gerenciamento da Qualidade do *Software*. Os tópicos tratados aqui são garantia da qualidade de *software*, verificação e validação, e por fim, revisões e auditorias.

A terceira e última subárea descreve Considerações Práticas relacionadas à qualidade de *software*. Os tópicos são requisitos de qualidade de *software*, caracterização de defeito, técnicas de gerenciamento da qualidade do *software*, e

mensuração da qualidade do *software*. (SANTIAGO, 2011).

#### 2.2.4 Metodologias

Um processo de *software* é um conjunto de atividades relacionadas que levam à produção de um produto de *software*. Essas atividades podem envolver o desenvolvimento de *software* a partir do zero em uma linguagem padrão de programação como Java ou C. No entanto, aplicações de negócios não são necessariamente desenvolvidas dessa forma. Atualmente, novos *softwares* de negócios são desenvolvidos por meio da extensão e modificação de sistemas existentes ou componentes do sistema ou por meio da configuração e integração de um produto de software prateleira, que de acordo com (SOMMERVILLE, 2011, p. 307) “é um sistema de software que pode ser adaptado às necessidades de diferentes clientes sem alterar o código-fonte do sistema. Praticamente todos os *softwares* de *desktop* e uma grande variedade de produtos de servidor são *softwares* prateleira”.

Os processos de *software* são complexos e, como todos os processos intelectuais e criativos, dependem de pessoas para tomar decisões e fazer julgamentos. Não existe um processo ideal, a maioria das organizações desenvolve os próprios processos de desenvolvimento de *software*. Os processos têm evoluído de maneira a tirarem melhor proveito das capacidades das pessoas em uma organização, bem como das características específicas do sistema em desenvolvimento. Para alguns sistemas, como sistemas críticos, é necessário um processo de desenvolvimento muito bem estruturado; para sistemas de negócios, com requisitos que se alteram rapidamente, provavelmente será mais eficaz um processo menos formal e mais flexível. (SOMMERVILLE, 2011b).

Hoje, o *software* assume um duplo papel. Ele é um produto e, ao mesmo tempo, o veículo para distribuir um produto. Como produto, fornece o potencial computacional representado pelo *hardware* ou, de forma mais abrangente, por uma rede de computadores que podem ser acessados por *hardware* local. Independentemente de residir em um celular ou operar dentro de um *mainframe*, *software* é um transformador de informações — produzindo, gerenciando, adquirindo, modificando, exibindo ou transmitindo informações que podem ser tão simples quanto um único bit ou tão complexas quanto uma apresentação multimídia derivada de dados obtidos de dezenas de fontes independentes. Como veículo de distribuição do produto, o *software* atua como a base para o controle do computador (sistemas operacionais), a comunicação de informações (redes) e a criação e o controle de outros programas (ferramentas de *software* e ambientes). Pressman (2021, p. 46) define o que é *software* da seguinte forma: “*Software* de computador é o produto que profissionais de *software* desenvolvem e ao qual dão suporte por

muitos anos. Esses artefatos incluem programas executáveis em computador de qualquer porte ou arquitetura”.

O *software* distribui o produto mais importante de nossa era — a informação. Ele transforma dados pessoais (por exemplo, transações financeiras de um indivíduo) de modo que possam ser mais úteis num determinado contexto; gerencia informações comerciais para aumentar a competitividade; fornece um portal para redes mundiais de informação (Internet) e os meios para obter informações sob todas as suas formas.

O papel desempenhado pelo *software* tem passado por grandes mudanças ao longo dos últimos cinquenta anos. Aperfeiçoamentos significativos no desempenho do *hardware*, mudanças profundas nas arquiteturas computacionais, vasto aumento na capacidade de memória e armazenamento, e uma ampla variedade de exóticas opções de entrada e saída, tudo isso resultou em sistemas computacionais mais sofisticados e complexos. Sofisticação e complexidade podem produzir resultados impressionantes quando um sistema é bem-sucedido, porém, também podem trazer enormes problemas para aqueles que precisam desenvolver sistemas robustos. (PRESSMAN, 2011).

**Figura 3 - Camadas da engenharia de software**



Fonte: PRESSMAN, 2011, p. 39

A engenharia de *software* é uma tecnologia em camadas. Referindo-se à Figura 3, qualquer abordagem de engenharia (inclusive a de *software*) deve estar fundamentada em um comprometimento organizacional com a qualidade. A gestão da qualidade total Seis Sigma e filosofias similares promovem uma cultura de aperfeiçoamento contínuo de processos, e é esta cultura que, no final das contas, leva ao desenvolvimento de abordagens cada vez mais efetivas. Sendo que a pedra fundamental que a sustenta, é o foco na qualidade.

A base para a engenharia de *software* é a camada de processos. O processo realizado é a liga que mantém as camadas de tecnologia coesas e possibilita o desenvolvimento de *software* de forma racional e dentro do prazo. O processo define uma metodologia que deve ser estabelecida para a entrega efetiva de tecnologia de na entrega com qualidade das rotinas executáveis via computação.

O processo de *software* constitui a base para o controle do gerenciamento de projetos de *software* e estabelece o contexto no qual são aplicados métodos técnicos, são produzidos produtos derivados (modelos, documentos, dados, relatórios, formulários etc.), são estabelecidos marcos, a qualidade é garantida e mudanças são geridas de forma apropriada. Os métodos padronizados pela engenharia e definições para a criação de *softwares* estabelecidas, fornecem as informações técnicas necessárias para se desenvolver um bom projeto.

Os métodos envolvem uma ampla gama de tarefas, que incluem: comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte. Os métodos utilizados pelos engenheiros e cientistas da computação, baseiam-se em um conjunto de princípios básicos que governam cada área da tecnologia e inclui atividades de modelagem e outras técnicas descriptivas.

As ferramentas da área abordada neste tópico, fornecem suporte automatizado ou semi automatizado para o processo e para os métodos. Quando as ferramentas são integradas, de modo que as informações criadas por uma ferramenta possam ser usadas por outra, é estabelecido um sistema para o suporte ao desenvolvimento de *software*, denominado engenharia de *software* com o auxílio do computador. (PRESSMAN, 2011).

### **2.2.5 Desenvolvimento ágil**

Nos dias de hoje, as empresas operam em um ambiente global, com mudanças rápidas. Assim, precisam responder a novas oportunidades e novos mercados, às mudanças nas condições econômicas e ao surgimento de produtos e serviços concorrentes. *Softwares* fazem parte de quase todas as operações de negócios, assim, novos *softwares* são desenvolvidos rapidamente para obterem proveito de novas oportunidades e responder às pressões competitivas.

“*Software* é um lugar onde sonhos são plantados e pesadelos são colhidos, um pântano abstrato e místico onde demônios terríveis competem com mágicas panaceias, um mundo de lobisomens e balas de prata.” (BRAD J. COX apud PRESSMAN, 2011, p.31)

O desenvolvimento e entrega rápidos são, portanto, o requisito mais crítico para o desenvolvimento de sistemas de *software*. Na verdade, muitas empresas estão dispostas a trocar a qualidade e o compromisso com requisitos do *software* por uma implantação mais rápida do *software* de que necessitam.

Essas empresas operam em um ambiente de mudanças rápidas, e por isso, muitas vezes, é praticamente impossível obter um conjunto completo de requisitos de *software* estável. Os requisitos iniciais inevitavelmente serão alterados, pois os clientes acham impossível prever como um sistema afetará as práticas de trabalho, como irá interagir com outros sistemas e quais operações do usuário devem ser automatizadas. Pode ser que os requisitos se tornem claros apenas após a entrega do sistema e à medida que os usuários ganhem experiência. Mesmo assim, devido a

fatores externos, os requisitos são suscetíveis a mudanças rápidas e imprevisíveis. Por exemplo, quando for entregue, o *software* poderá estar desatualizado.

Processos de desenvolvimento de *software* que planejam especificar completamente os requisitos e, em seguida, projetar, construir e testar o sistema não estão adaptados ao desenvolvimento rápido de *software*. Com as mudanças nos requisitos ou a descoberta de problemas de requisitos, o projeto do sistema ou sua implementação precisa ser refeito ou retestado. Como consequência, um processo convencional em cascata ou baseado em especificações costuma ser demorado, e o *software* final é entregue ao cliente bem depois do prazo acordado.

Para alguns tipos de *software*, como sistemas críticos de controle de segurança, em que uma análise completa do sistema é essencial, uma abordagem dirigida a planos é a melhor opção. No entanto, em um ambiente de negócios que se caracteriza por mudanças rápidas, isso pode causar problemas reais. Quando o *software* estiver disponível para uso, a razão original para sua aquisição pode ter mudado tão radicalmente que o *software* será efetivamente inútil. Portanto, para os sistemas de negócios, particularmente, os processos de desenvolvimento que se caracterizem por desenvolvimento e entrega rápidos de *software* são essenciais. (SOMMERVILLE, 2011a).

#### 2.2.5.1 Agilidade

Na década de 80 e início da de 90, havia uma visão generalizada de que a melhor maneira para conseguir o melhor *software* era por meio de um planejamento cuidadoso do projeto, qualidade da segurança formalizada, do uso de métodos de análise e projeto apoiado por ferramentas CASE (*Computer-aided software engineering*) e do processo de desenvolvimento de *software* rigoroso e controlado. Essa percepção veio da comunidade de engenharia de *software*, responsável pelo desenvolvimento de sistemas de *software* grandes e duradouros, como sistemas aeroespaciais e de governo.

Esse *software* foi desenvolvido por grandes equipes que trabalham para diferentes empresas. Geralmente, as equipes eram dispersas geograficamente e trabalhavam com o *software* por longos períodos. Um exemplo desse tipo de *software* é o sistema de controle de uma aeronave moderna, que pode demorar até dez anos desde a especificação inicial até a implantação. Tais abordagens dirigidas a planos envolvem um *overhead* significativo no planejamento, projeto e documentação do sistema. Esse *overhead* se justifica quando o trabalho de várias equipes de desenvolvimento tem de ser coordenado, quando o sistema é um sistema crítico e quando muitas pessoas diferentes estão envolvidas na manutenção do *software* durante sua vida.

No entanto, quando essa abordagem pesada de desenvolvimento dirigido a planos é aplicada aos sistemas corporativos de pequeno e médio porte, o *overhead* envolvido é tão grande que domina o processo de desenvolvimento de *software*.

Gasta-se mais tempo em análises de como o sistema deve ser desenvolvido do que no desenvolvimento de executáveis e testes. Como os requisitos do sistema se alteram, o retrabalho é essencial, e, pelo menos em princípio, a especificação e o projeto devem mudar com o programa.

A insatisfação com essas abordagens pesadas da engenharia de *software* levou um grande número de desenvolvedores de *software* a proporem, na década de 1990, novos métodos ágeis<sup>1</sup>. Estes permitiram que a equipe de desenvolvimento focassem no *software* em si, e não em sua concepção e documentação. Métodos ágeis, universalmente, baseiam-se em uma abordagem incremental para a especificação, o desenvolvimento e a entrega do *software*. Eles são mais adequados ao desenvolvimento de aplicativos nos quais os requisitos de sistema mudam rapidamente durante o processo de desenvolvimento. Destinam-se a entregar o software rapidamente aos clientes, em funcionamento, e estes podem, em seguida, propor alterações e novos requisitos a serem incluídos nas iterações posteriores do sistema. Têm como objetivo reduzir a burocracia do processo, evitando qualquer trabalho de valor duvidoso de longo prazo e qualquer documentação que provavelmente nunca será usada. (SOMMERVILLE, 2011a).

Em essência, métodos ágeis se desenvolveram em um esforço para sanar fraquezas reais e perceptíveis da produção de *software* convencional. O desenvolvimento ágil oferece benefícios importantes, no entanto, não é indicado para todos os projetos, produtos, pessoas e situações. Também não é a antítese da prática de engenharia de *software* consistente e pode ser aplicado como uma filosofia geral para todos os trabalhos de *software*.

Na economia moderna é frequentemente difícil ou impossível prever como um sistema computacional (por exemplo, uma aplicação baseada na Web) irá evoluir com o tempo. As condições de mercado mudam rapidamente, as necessidades dos usuários finais se alteram e novas ameaças competitivas emergem sem aviso. Em muitas situações, não se conseguirá definir completamente requisitos antes que se inicie o projeto. É preciso ser ágil o suficiente para dar uma resposta ao ambiente de negócios fluido.

Fluidez implica mudanças, e mudanças são caras. Particularmente, se forem sem controle e mal gerenciadas. Uma das características mais convincentes da abordagem ágil é sua habilidade de reduzir os custos da mudança ao longo de todo o processo de *software*.

Isso significa que o reconhecimento dos desafios apresentados pela moderna realidade faz com que se descartem valiosos princípios da ciência de desenvolvimento de *software*, conceitos, métodos e ferramentas? Absolutamente não! Como todas as disciplinas de engenharia, a de *software* continua a evoluir, podendo ser adaptada facilmente aos desafios apresentados pela demanda por agilidade.

Em textos que nos levam à reflexão sobre desenvolvimento de *software* ágil, existe o argumento que o modelo de processo prescritivo, tem uma falha essencial: esquece das fragilidades das pessoas que desenvolvem o *software*. Os engenheiros

de *software* não são robôs. Eles apresentam grande variação nos estilos de trabalho; diferenças significativas no nível de habilidade, criatividade, organização, consistência e espontaneidade. Alguns se comunicam bem na forma escrita, outros não.

Para que funcionem, os modelos de processos devem fornecer um mecanismo realista que estimule a disciplina necessária ou, então, devem ter características que apresentem “tolerância” com as pessoas que realizam trabalhos de engenharia de software. Invariavelmente, práticas tolerantes são mais facilmente adotadas e sustentadas pelas pessoas envolvidas, porém podem ser menos produtivas. Como a maioria das coisas na vida, deve-se considerar os prós e os contras.

Entretanto, agilidade consiste em algo mais que uma resposta à mudança, abrangendo a filosofia proposta no manifesto citado no início deste capítulo. Ela incentiva a estruturação e as atitudes em equipe que tornam a comunicação mais fácil (entre membros da equipe, entre o pessoal ligado à tecnologia e o pessoal da área comercial, entre os engenheiros de *software* e seus gerentes). Enfatiza a entrega rápida do *software* operacional e diminui a importância dos artefatos intermediários (nem sempre um bom negócio); assume o cliente como parte da equipe de desenvolvimento e trabalha para eliminar a atitude de “nós e eles”, que continua a invadir muitos projetos de *software*; reconhece que o planejamento em um mundo incerto tem seus limites e que o plano (roteiro) de projeto deve ser flexível.

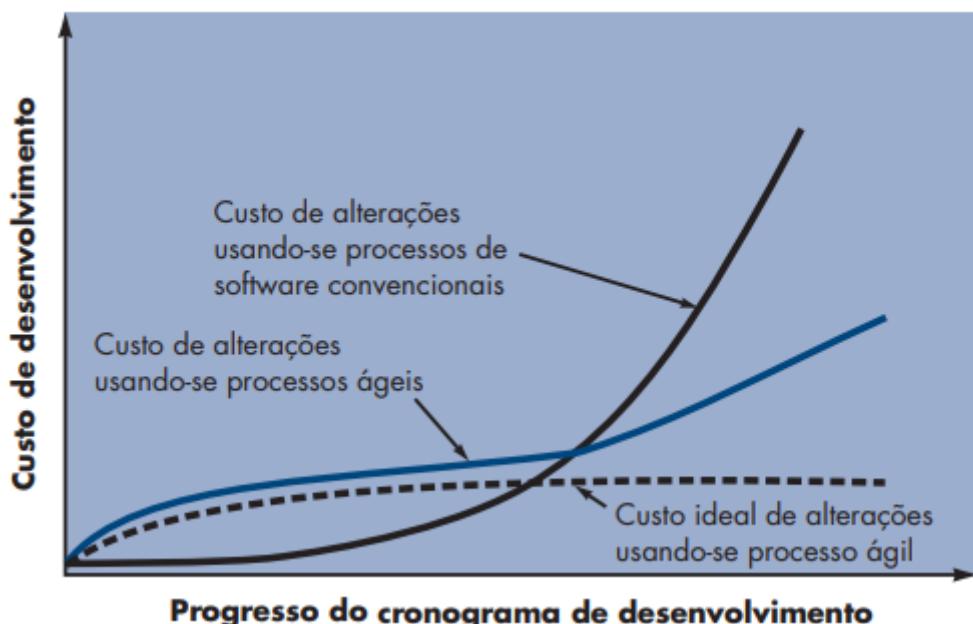
A agilidade pode ser aplicada a qualquer processo de *software*. Entretanto, para obtê-la, é essencial que seja projetado para que a equipe possa adaptar e alinhar (racionalizar) tarefas; possa conduzir o planejamento compreendendo a fluidez de uma abordagem do desenvolvimento ágil; possa eliminar tudo, exceto os artefatos essenciais, conservando-os enxutos; e enfatize a estratégia de entrega incremental, conseguindo entregar ao cliente, o mais rapidamente possível, o *software* operacional para o tipo de produto e ambiente operacional. (PRESSMAN, 2011).

#### 2.2.5.2 Mudanças

A sabedoria convencional em desenvolvimento de *software* (baseada em décadas de experiência) afirma que os custos de mudanças aumentam de forma não linear conforme o projeto avança (Figura 4, curva em preto contínuo). É relativamente fácil assimilar uma mudança quando uma equipe de software está juntando requisitos (no início de um projeto). Pode-se ter de alterar um detalhamento do uso, ampliar uma lista de funções ou editar uma especificação por escrito. Os custos de tal trabalho são mínimos e o tempo demandado não afetará negativamente o resultado do projeto. Mas se adiantarmos alguns meses, o que aconteceria? A equipe está em meio aos testes de validação (que ocorre relativamente no final do projeto) é um importante interessado está requisitando uma

mudança funcional de vulto. A mudança requer uma alteração no projeto da arquitetura do *software*, o projeto e desenvolvimento de três novos componentes, modificações em outros cinco componentes, projeto de novos testes, e assim por diante. Os custos crescem rapidamente e não serão triviais o tempo e custos necessários para assegurar que a mudança seja feita sem efeitos colaterais inesperados. (PRESSMAN, 2011).

**Figura 4** - Custos de alterações como uma função do tempo em desenvolvimento



Fonte: PRESSMAN, 2011, p. 83

Os defensores da agilidade argumentam que um processo ágil bem elaborado “achata” o custo da curva de mudança (Figura 4, curva em linha azul), permitindo que uma equipe de *software* assimile as alterações, realizadas posteriormente em um projeto de *software*, sem um impacto significativo nos custos ou no tempo. Já foi mencionado que o processo ágil envolve entregas incrementais. O custo das mudanças é atenuado quando a entrega incremental é associada a outras práticas ágeis, como testes contínuos de unidades e programação por pares. Há evidências que sugerem que se pode alcançar redução significativa nos custos de alterações, embora haja um debate contínuo sobre qual o nível em que a curva de custos torna-se “achatada”. (PRESSMAN, 2011).

#### 2.2.5.3 Abordagens abrangentes

Na história da engenharia de *software* há dezenas de metodologias e descrições de processos, métodos e notações de modelagem, ferramentas e tecnologias obsoletas. Cada um atingiu grande notoriedade e foi então ofuscado por algo novo e (supostamente) melhor. Com a introdução de uma ampla gama de

modelos de processos ágeis; todos disputando por aceitação pela comunidade de desenvolvimento de *software*; o movimento ágil está seguindo o mesmo caminho histórico.

Isso não é algo ruim. Antes que um ou mais modelos ou métodos sejam aceitos como um padrão de fato, todos devem competir para conquistar os corações e mentes dos engenheiros de *software*. Os “vencedores” evoluem e se transformam nas melhores práticas, enquanto os “perdedores” desaparecem ou se fundem aos modelos vencedores. (PRESSMAN, 2011).

Embora esses métodos ágeis sejam todos baseados na noção de desenvolvimento e entrega incremental, eles propõem diferentes processos para alcançar tal objetivo. No entanto, compartilham um conjunto de princípios, com base no manifesto ágil, e por isso têm muito em comum. Diferentes métodos ágeis instituem esses princípios de maneiras diferentes, e eu não tenho espaço para discutir todos os métodos ágeis. Em vez disso, foco em dois dos mais usados: Kanban e Scrum que são utilizados neste trabalho.

A principal responsabilidade dos gerentes de projeto de *software* é gerenciar o projeto para que o *software* seja entregue no prazo e dentro do orçamento previsto. Eles supervisionam o trabalho dos engenheiros de *software* e acompanham quanto bem o desenvolvimento de *software* está progredindo.

A abordagem-padrão para gerenciamento de projetos é a dirigida a planos. Os gerentes devem elaborar um plano para o projeto mostrando o que deve ser entregue, quando deve ser entregue e quem vai trabalhar no desenvolvimento das entregas do projeto. Uma abordagem baseada em planos necessita de um gerente que tenha uma visão estável de tudo o que tem de ser desenvolvido e os processos de desenvolvimento. Contudo, essa abordagem não funciona bem com os métodos ágeis, nos quais os requisitos são desenvolvidos de forma incremental, o *software* é entregue em incrementos curtos e rápidos, e as mudanças nos requisitos e no *software* são a norma.

Como todos os outros processos profissionais de desenvolvimento de *software*, o desenvolvimento ágil tem de ser gerenciado de modo que se faça o melhor uso com o tempo e os recursos disponíveis para a equipe. Isso requer do gerenciamento de projeto uma abordagem diferente, adaptada para o desenvolvimento incremental e para os pontos fortes dos métodos ágeis. (SOMMERVILLE, 2011a).

#### **2.2.5.3.1 Scrum**

A abordagem Scrum é um método ágil geral, mas seu foco está no gerenciamento do desenvolvimento iterativo, ao invés das abordagens técnicas específicas da engenharia de *software* ágil. Scrum não prescreve o uso de práticas de programação, como programação em pares e desenvolvimento *test-first*. Portanto, pode ser usado com abordagens ágeis mais técnicas, como XP, para fornecer um *framework* de gerenciamento do projeto.

No Scrum, existem três fases. A primeira é uma fase de planejamento geral, em que se estabelecem os objetivos gerais do projeto e da arquitetura do *software*. Em seguida, ocorre uma série de ciclos de *sprint*, sendo que cada ciclo desenvolve um incremento do sistema. Finalmente, a última fase do projeto encerra o projeto, completa a documentação exigida, como quadros de ajuda do sistema e manuais do usuário, e avalia as lições aprendidas com o projeto. (SOMMERVILLE, 2011a).

A característica inovadora do Scrum é sua fase central, chamada ciclos de *sprint*. Um *sprint* do Scrum é uma unidade de planejamento na qual o trabalho a ser feito é avaliado, os recursos para o desenvolvimento são selecionados e o *software* é implementado. No fim de um *sprint*, a funcionalidade completa é entregue aos *stakeholders*.

As principais características desse processo são:

- *Sprints* são de comprimento fixo, normalmente de duas a quatro semanas. Eles correspondem ao desenvolvimento de um release do sistema em XP
- O ponto de partida para o planejamento é o *backlog* do produto, que é a lista do trabalho a ser feito no projeto. Durante a fase de avaliação do *sprint*, este é revisto, e as prioridades e os riscos são identificados. O cliente está intimamente envolvido nesse processo e, no início de cada *sprint*, pode introduzir novos requisitos ou tarefas.
- A fase de seleção envolve todos da equipe do projeto que trabalham com o cliente para selecionar os recursos e a funcionalidade a ser desenvolvida durante o *sprint*.
- Uma vez que todos estejam de acordo, a equipe se organiza para desenvolver o *software*. Reuniões diárias rápidas, envolvendo todos os membros da equipe, são realizadas para analisar os progressos e, se necessário, repriorizar o trabalho. Nessa etapa, a equipe está isolada do cliente e da organização, com todas as comunicações canalizadas por meio do chamado '*Scrum Master*'. O papel do *Scrum Master* é proteger a equipe de desenvolvimento de distrações externas. A maneira como o trabalho é desenvolvido depende do problema e da equipe. Diferentemente do XP, a abordagem Scrum não faz sugestões específicas sobre como escrever os requisitos ou sobre o desenvolvimento *test-first* etc. No entanto, essas práticas de XP podem ser usadas se a equipe achar que são adequadas.
- No fim do *sprint*, o trabalho é revisto e apresentado aos *stakeholders*. O próximo ciclo *sprint* começa em seguida

A ideia por trás do Scrum é que toda a equipe deve ter poderes para tomar decisões, de modo que o termo gerente de projeto tem sido deliberadamente evitado. Pelo contrário, o *Scrum Master* é um facilitador, que organiza reuniões diárias, controla o *backlog* de trabalho, registra decisões, mede o progresso comparado ao *backlog* e se comunica com os clientes e a gerência externa à equipe.

Toda a equipe participa das reuniões diárias; às vezes, estas são feitas com os participantes em pé *stand-up*, muito rápidas, para a manutenção do foco da equipe. Durante a reunião, todos os membros da equipe compartilham informações,

descrevem seu progresso desde a última reunião, os problemas que têm surgido e o que está planejado para o dia seguinte. Isso garante que todos na equipe saibam o que está acontecendo e, se surgirem problemas, poderão replanejar o trabalho de curto prazo para lidar com eles. Todos participam desse planejamento de curto prazo; não existe uma hierarquia *top-down* a partir do *Scrum Master*. (SOMMERVILLE, 2011a).

*Scrum* (o nome provém de uma atividade que ocorre durante a partida de *rugby*) é um método de desenvolvimento ágil de *software* concebido por Jeff Sutherland e sua equipe de desenvolvimento no início dos anos 1990. Mais recentemente, foram realizados desenvolvimentos adicionais nos métodos gráficos *Scrum*.

Os princípios do *Scrum* são consistentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades estruturais: requisitos, análise, projeto, evolução e entrega. Em cada atividade metodológica, ocorrem tarefas a realizar dentro de um padrão de processo (discutido no parágrafo a seguir) chamado *sprint*. O trabalho realizado dentro de um *sprint* (o número de *sprints* necessários para cada atividade metodológica varia dependendo do tamanho e da complexidade do produto) é adaptado ao problema em questão e definido, e muitas vezes modificado em tempo real, pela equipe *Scrum*. O fluxo geral do processo *Scrum* é ilustrado na Figura 5.

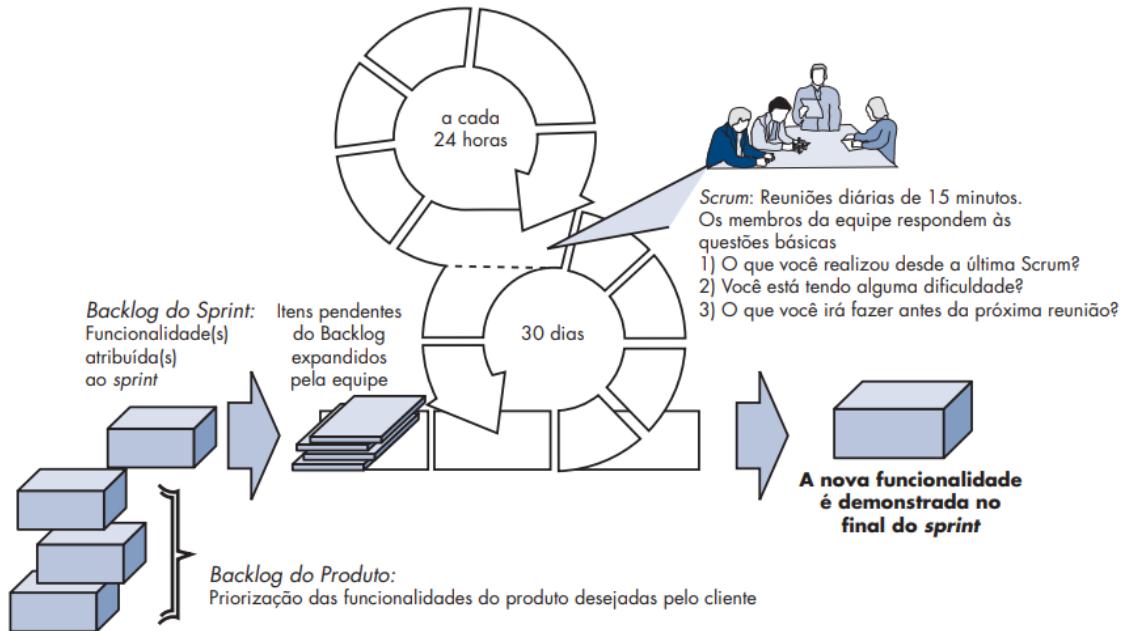
O *Scrum* enfatiza o uso de um conjunto de padrões de processos de *software* que provaram ser eficazes para projetos com prazos de entrega apertados, requisitos mutáveis e críticos de negócio. Cada um desses padrões de processos define um conjunto de ações de desenvolvimento:

Registro pendente de trabalhos (*Backlog*) — uma lista com prioridades dos requisitos ou funcionalidades do projeto que fornecem valor comercial ao cliente. Os itens podem ser adicionados a esse registro em qualquer momento (é assim que as alterações são introduzidas). O gerente de produto avalia o registro e atualiza as prioridades conforme requisitado.

Urgências (corridas de curta distância) *sprints* — consistem de unidades de trabalho solicitadas para atingir um requisito estabelecido no registro de trabalho (*backlog*) e que precisa ser ajustado dentro de um prazo já fechado (janela de tempo)<sup>14</sup> (tipicamente 30 dias).

Alterações (por exemplo, itens do registro de trabalho — *backlog work items*) não são introduzidas durante execução de urgências (*sprint*). Portanto, o *sprint* permite que os membros de uma equipe trabalhem em um ambiente de curto prazo, porém estável. (PRESSMAN, 2011).

**Figura 5 — Fluxo do processo Scrum**



Fonte: PRESSMAN, 2011, p.96

Reuniões *Scrum* — são reuniões curtas (tipicamente 15 minutos), realizadas diariamente pela equipe *Scrum*. São feitas três perguntas-chave e respondidas por todos os membros da equipe:

- O que você realizou desde a última reunião de equipe?
- Quais obstáculos está encontrando?
- O que planeja realizar até a próxima reunião da equipe?

Um líder de equipe, chamado *Scrum master*, conduz a reunião e avalia as respostas de cada integrante. A reunião *Scrum*, realizada diariamente, ajuda a equipe a revelar problemas potenciais o mais cedo possível. Ela também leva à socialização do conhecimento e, portanto, promove uma estrutura de equipe auto-organizada.

Demos — entrega do incremento de *software* ao cliente para que a funcionalidade implementada possa ser demonstrada e avaliada pelo cliente. É importante notar que a demo pode não ter toda a funcionalidade planejada, mas sim funções que possam ser entregues no prazo estipulado.

Sobre esses padrões: “O *Scrum* pressupõe a existência do caos...”. Os padrões de processos do *Scrum* capacitam uma equipe de *software* a trabalhar com sucesso em um mundo onde a eliminação da incerteza é impossível. (PRESSMAN, 2011).

### 2.2.5.3.2 *Kanban*

*Kanban* é um método para definir, gerenciar e melhorar serviços que entregam trabalho de conhecimento, tais como serviços profissionais, atividades criativas e o design de produtos físicos e de *software*. Pode ser caracterizado como um método “começar pelo que você já faz” — um catalisador para uma mudança rápida e focada dentro das organizações — que reduz a resistência a mudanças benéficas alinhando com os objetivos da organização. (ANDERSON; CARMICHAEL, 2016).

O método *Kanban* é baseado em tornar visível o que é de outra forma, trabalho intangível de conhecimento, para garantir que o serviço funcione com a quantidade adequada de trabalho — o trabalho que é necessário e solicitado pelo cliente e que a equipe tem a capacidade de entregar. Para isso, usamos um sistema *kanban* — um sistema de fluxo de entrega que limita a quantidade de trabalho em progresso (WiP) usando sinais visuais.

Os mecanismos de sinalização, às vezes referidos como *kanbans*, são exibidos em quadros *kanban* e representam limites de WiP, que impedem que muito ou muito pouco trabalho entre no sistema, melhorando assim o fluxo de valor para os clientes. As Políticas de limite do WiP criam um sistema puxado: O trabalho é “puxado” para o sistema quando outro trabalho é concluído e a capacidade torna-se disponível, ao invés de “empurrado” para ele quando novo trabalho é exigido. (ANDERSON; CARMICHAEL, 2016).

*Kanban* se concentra na entrega de serviços por uma organização — uma ou mais pessoas colaborando para produzir produtos de trabalho (geralmente intangíveis). Um serviço tem um cliente, que solicita o trabalho ou cujas necessidades são identificadas, e que aceita ou considera a entrega do trabalho concluída. Mesmo quando há um produto físico dos Serviços, o valor reside menos no produto em si e mais em seu conteúdo informacional (o *software*, no sentido mais geral). (ANDERSON; CARMICHAEL, 2016).

No desenvolvimento de *software*, estamos usando um sistema *kanban* virtual para limitar o trabalho-em-progresso. Embora “*kanban*” signifique “cartão sinalizador” e não existem cartões utilizados na maioria das implementações de *Kanban* no desenvolvimento de *software*, estes cartões não funcionam realmente como sinais para puxar mais trabalho. Em vez disso, eles representam itens de trabalho. Daí o termo “virtual”, porque não há nenhum cartão sinalizador físico. O sinal para puxar o novo trabalho é inferido da quantidade visual dos trabalhos-em-progresso subtraído de algum indicador do limite (ou capacidade). Alguns profissionais aplicaram *kanban* físico usando técnicas como clipe autocolante ou *slots* físicos em uma placa. Mais frequentemente o sinal é gerado a partir de um sistema de acompanhamento de trabalho em *software*.

Paredes de cartões se tornaram um mecanismo de controle visual popular no desenvolvimento de *software* ágil. Usando um quadro de aviso de cortiça com cartões de índice fixados em uma placa ou um quadro de comunicações com notas

auto-adesivas para acompanhar os trabalhos em progresso (WIP) tornou-se comum. Vale a pena observar, nesta fase inicial, que apesar de alguns comentários da Comunidade em contrário, estes muros de cartão não são inherentemente sistemas kanban. Eles são sistemas de controle visual apenas. Eles permitem às equipes observarem visualmente os trabalhos em progresso e a se auto-organizarem, atribuindo suas próprias tarefas e movendo o trabalho de um *backlog* para conclusão sem orientação de um gerente de projeto ou linha. No entanto, se não houver nenhum limite explícito para trabalho-em-progresso e uma sinalização para puxar o novo trabalho através do sistema, ele não é um sistema *kanban*.

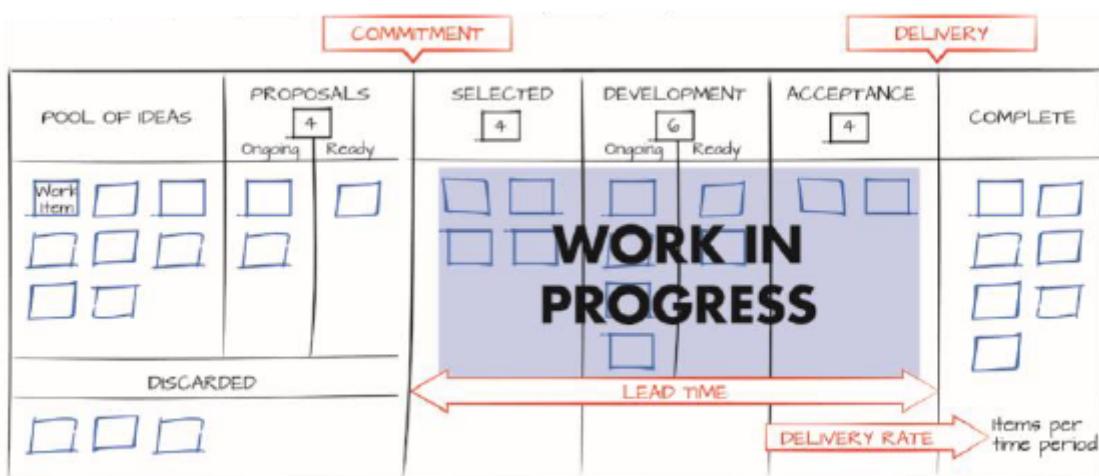
(REINERTSEN; ANDERSON; PINTO, 2011).

*Kanban* é usado para definir, gerenciar e melhorar sistemas que fornecem serviços de valor para clientes. Como *Kanban* é aplicado ao trabalho do conhecimento, onde as entregas consistem em informações de diversas formas, em vez de itens físicos, os processos podem ser definidos como uma série de conhecimentos, descoberta de etapas e suas políticas associadas, tornando-se visível em um quadro de *kanban*, exemplificado na figura 6.

O quadro retrata um sistema de fluxo no qual os itens de trabalho fluem através de várias etapas de um processo, ordenados da esquerda para a direita. Várias condições devem existir para que este sistema de fluxo seja um sistema *kanban*. Em primeiro lugar, devem existir sinais (normalmente sinais visuais) para limitar o trabalho em progresso (WiP). Neste caso, os sinais derivam da combinação dos cartões, do Limite do Trabalho em Progresso visível (nos retângulos no topo das colunas), e a coluna que representa a atividade. Além disso, sistemas *kanban* devem ter identificados pontos de compromisso e entrega.

(ANDERSON; CARMICHAEL, 2016).

**Figura 6 — Exemplo de quadro Kanban**



Fonte: ANDERSON; CARMICHAEL, 2016, p. 8

Como deve se tornar evidente nos próximos capítulos, nós usamos um sistema *kanban* para limitar o trabalho-em-progresso de uma equipe para definir a capacidade e equilibrar a demanda sobre a equipe em relação ao rendimento do trabalho entregue. Fazendo isso, podemos conseguir um ritmo sustentável de

desenvolvimento para que todos os indivíduos possam alcançar um equilíbrio entre trabalho e vida pessoal.

Como será visto, *Kanban* também elimina rapidamente os problemas que prejudicam o desempenho e desafia uma equipe a se concentrar em resolver esses problemas para manter um fluxo constante de trabalho. Ao fornecer visibilidade para problemas de qualidade e de processos, torna óbvio o impacto de defeitos, gargalos, variabilidade e custos econômicos no fluxo e na vazão. O simples ato de limitar o trabalho-em-progresso com o *kanban* incentiva maior qualidade e maior desempenho. A combinação de fluxo aperfeiçoado e melhor qualidade ajuda a reduzir os prazos de entrega e a melhorar o desempenho da data de entrega e a previsibilidade. Ao estabelecer uma cadência regular de liberação e entregando consistentemente, *Kanban* ajuda a construir a confiança com os clientes e confiança ao longo da cadeia de valor com outros departamentos, fornecedores e parceiros.

Ao fazer tudo isso, *Kanban* contribui para a evolução cultural das organizações. Ao expor problemas, fazendo com que uma organização dê enfoque em resolvê-los e eliminando seus efeitos no futuro, *Kanban* facilita o surgimento de uma organização altamente colaborativa, de alta confiança, altamente habilitada, e em constante melhoria.

*Kanban* tem mostrado melhorar a satisfação do cliente através de entregas regulares, confiáveis e de alta qualidade de *software* de alto valor. Ele também tem mostrado aumentar a produtividade, qualidade e prazos de entrega. Além disso, há evidências de que *kanban* é um catalisador fundamental para o surgimento de uma organização mais ágil através de mudança cultural evolutiva.  
(REINERTSEN; ANDERSON; PINTO, 2011).

#### 2.2.5.3.3 *Scrumban*

Conhecemos *Scrum* e *Kanban* como opções dos métodos ágeis. *Scrum* é mais adequado para projetos de produtos e de desenvolvimento. Cada uma dessas estruturas tem seus próprios benefícios e desvantagens, e combinar dois métodos pode ser a melhor maneira de evitar quaisquer armadilhas de desenvolvimento. Esta combinação é conhecida como metodologia *Scrumban* que é um híbrido de *Scrum* e *Kanban*.

- *Scrumban*
  - Base do *Scrumban*
    - É fundamentado em sistema puxado, onde a equipe já não planeja o trabalho durante a sprint planning, em vez disso, o refinamento é feito continuamente.
      - Usa a natureza prescritiva do *Scrum* para ser Ágil.
      - Usa a melhoria de processo do *Kanban* para permitir que a equipe melhore continuamente seu processo.

- Fluxo de trabalho no Scrumban
  - Usando o sistema puxado o fluxo se torna mais suave à medida que a capacidade de processo melhora.

Kanban é compatível com a mecânica do Scrum, o método de gerenciamento de projetos. A adição do WIP (trabalho em processo) e visualização ao Scrum (ou seja, Scrumban) ajuda a melhorar a eficácia do Sprint.

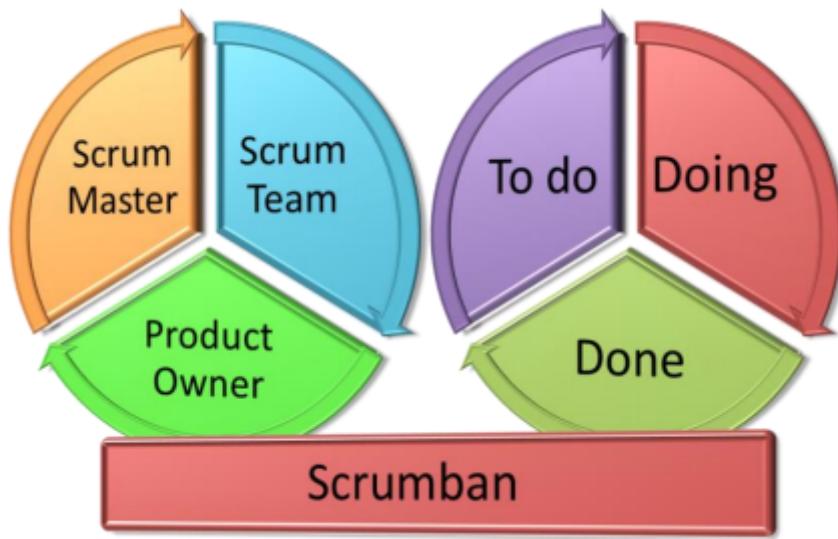
Além disso, também está introduzindo o limite do WIP como um mecanismo para catalisar mudanças incrementais. O limite de WIP elimina a necessidade de compromisso para impulsionar a mudança, reduz a dependência disfuncional do esforço heroico e melhora o sistema ao considerar possíveis melhorias. Na prática se parece um pouco como o Scrum, mas no nível cultural ele se parecerá com *Kanban* – evolução suave ao invés de tratamento de choque e revolução.

Com a citação de Reis (2021, p.58) pode-se analisar a brevidade da metodologia, assim como também apresenta sua potencialidade ao dizer que “O termo *Scrumban* foi primeiramente mencionado por Corey Ladas, um praticante de *Lean-Kanban* que por meio de diversos estudos criou um método de migrar do *Scrum* para o *Kanban*. Apesar de novo, os estudos de Ladas levaram a crer que a adoção da mescla seria muito benéfica para os já praticantes do framework *Scrum*, pois preenchem lacunas deixadas em aberto”.

Assim, com a visibilidade do método híbrido aumentado e à medida que mais pessoas se interessam pelas ideias *Lean* e sua aplicação ao trabalho do conhecimento e gerenciamento de projetos, é útil encontrar maneiras que tornem mais fácil começar ou aprender alguns conceitos básicos que podem levar a *insights* mais profundos posteriormente. De uma forma ou de outra, os usuários Scrum são um importante constituinte do público *Kanban*. Visto que o Scrum pode ser descrito como uma declaração na linguagem que usamos para descrever os sistemas *Kanban*, também é bastante fácil elaborar esse caso para descrever os híbridos *Scrum / Kanban*. Isso pode ser útil para times *Scrum* existentes que procuram melhorar sua escala ou capacidade. Também pode ser útil para novos usuários mais cautelosos que encontram conforto em um método “estabelecido”. (Apesar do fato de que a ideia *Kanban* é pelo menos 40 anos mais velha.) (REIS, 2021).

O *Scrumban*, como o híbrido de dois métodos ágeis, ilustrado na figura 7 como uma abstração dessa junção, fornece às equipes de desenvolvimento de software a flexibilidade para se adaptar e alterar as partes interessadas e os requisitos de produção sem sobrecarga. Esta é uma abordagem versátil para gerenciamento de fluxo de trabalho, pois fornece a estrutura Scrum completa com a visualização e flexibilidade de Kanban. Você pode facilmente aplicar o *Scrumban Agile* para passar do *Scrum* para o *Kanban* sem dor. (REIS, 2021).

**Figura 7 — Scrum + Kanban = Scrumban**



Fonte: STOICA *et al.*, 2016, p. 11

Neste trabalho pôde-se mostrar como a evolução da metodologia ágil impacta diretamente nas áreas de desenvolvimento de *software*. Nota-se que as metodologias *Kanban* e *Scrum* conseguem trabalhar de forma satisfatória separadamente, porém um novo patamar é alcançado ao ser utilizadas juntas.

O *Scrum*, uma das metodologias mais utilizadas no desenvolvimento de *software*, mostra pequenas fraquezas quando olhamos o fluxo de trabalho, enquanto o *Kanban* peca principalmente na parte de documentação de processos, como o DOD.

Quando as empresas percebem que precisam mudar para uma forma mais enxuta e ágil de trabalhar, o *Scrumban* é uma maneira fácil de eliminar qualquer empecilho que acaba surgindo eventualmente, além de permitir que as empresas continuem a trabalhar de maneira mais eficiente, mas ao mesmo tempo implementem as melhores práticas enxutas e ágeis específicas para o ambiente de *streaming*.

O *Scrumban* pode fornecer o nível apropriado de rigor de que a empresa precisa, bem como a flexibilidade, eficiência e visibilidade de *Kanban* e *Lean*. (REIS, 2021).

### 3 METODOLOGIA

Diante disto, para a escrita desta pesquisa contou com um levantamento bibliográfico, sendo que o atual capítulo visa descrever as práticas que foram seguidas na elaboração da presente pesquisa, apresentando o objetivo, abordagem, procedimento, finalidade e método que foram utilizados, assim como suas devidas

explicações dos porquês foram escolhidas, assim como também sobre como se enquadram as etapas seguidas e as ferramentas que foram adotados na construção do sistema.

Para a busca de trabalhos como fonte de pesquisa bibliográfica, foram utilizados os seguintes descritores: Gestão de projetos; metodologias ágeis; API com desenvolvimento web; scrum e kanban. Nas bases de dados: Google acadêmico<sup>2</sup>, Scientific Electronic Library Online (Scielo)<sup>3</sup> e Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES)<sup>4</sup>, no período de novembro de 2022 a novembro de 2023. Dando-se a devida prioridade para a escolha dos documentos escritos na língua nacional<sup>5</sup>, contudo, considerando a representatividade, abundância e importância de materiais escritos em língua inglesa, também o foram utilizados em quantidade proporcional bem menor comparado ao idioma predominantemente adotado aqui no Brasil. Sendo a análise das informações realizadas por meio de leitura exploratória do material encontrado, em uma abordagem qualitativa.

### 3.1 ABORDAGEM

No que se refere à classificação da pesquisa quanto à questão da “abordagem”, tem-se por definição que o caso aqui tratado é do tipo qualitativo, pois realiza a abordagem das análises bibliográficas realizadas de maneira subjetiva, filtrando e apresentando o que de maior valor é agregado ao tema central.

Quando o autor do trabalho analisa criticamente os dados coletados sobre o tema. Na pesquisa qualitativa, estamos falando sobre a visão do autor do trabalho sobre uma questão escolhida para abordar. Aqui, os dados são subjetivos, porque abordam motivações, comportamentos ou emoções que não podem ser quantificadas numericamente, assim como apresentado por Freitas e Prodanov (2013, p.70).

Considera que há uma relação dinâmica entre o mundo real e o sujeito, isto é, um vínculo indissociável entre o mundo objetivo e a subjetividade do sujeito que não pode ser traduzido em números. A interpretação dos fenômenos e a atribuição de significados são básicas no processo de pesquisa qualitativa. Esta não requer o uso de métodos e

---

<sup>2</sup> <https://scholar.google.com/>

<sup>3</sup> <https://www.scielo.br/>

<sup>4</sup> <https://www-periodicos-capes-gov-br.ezl.periodicos.capes.gov.br/>

<sup>5</sup> Português Brasileiro

técnicas estatísticas. O ambiente natural é a fonte direta para coleta de dados e o pesquisador é o instrumento-chave. Tal pesquisa é descritiva. Os pesquisadores tendem a analisar seus dados indutivamente. O processo e seu significado são os focos principais de abordagem.

Portanto, conforme definição acima realizada e de acordo com os métodos e objetivos da pesquisa anteriormente explicados, pode-se inferir que a metodologia da abordagem é considerada como uma pesquisa qualitativa.

### 3.2 NATUREZA

A finalidade do trabalho, natureza como também é conhecida, ou ainda aplicação, visa esclarecer para o próprio pesquisador e para o leitor do trabalho a problematização da pesquisa, ou seja, quais questões e pontos do tema serão discutidos e avaliados. Assim, a natureza da pesquisa relaciona-se à contribuição de suas conclusões à ciência.

O tipo de pesquisa básico, diferentemente do tipo aplicado que objetiva gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos, apresenta um escopo mais amplo, que pode ser geralmente aplicados em diversas situações, sendo comumente utilizado um cunho de enfoque mais teórico para a solução apresentada, assim como se segue em citação “Objetiva gerar conhecimentos novos úteis para o avanço da ciência sem aplicação prática prevista. Envolve verdades e interesses universais.” (**FREITAS; PRODANOV, 2013, p.51**).

Esse subtipo de natureza corresponde à metodologia do caso aqui utilizado de acordo com a proposta e do conceito caracterizado, tendo em vista que a ferramenta de gestão de projetos propõe uma sugestão de fluxo de trabalho baseando-se em modelos já existentes, aproveitando definições acadêmicas e tendo referência em outras ferramentas já difundidas, mas traz especificidades de cunho inovador e que também poderá ser modelado para as utilizações específicas em cada ocasião, assim classificando-se como básica estratégica.

### 3.3 OBJETIVOS

Com a utilização da pesquisa chega-se a um conhecimento totalmente ou parcialmente novo, contribuindo assim para a formação da consciência crítica do pesquisador aprendendo algo que antes ignorava e para que isso ocorra com êxito..

A pesquisa exploratória permite uma maior interação entre você e o tema que será desenvolvido, já que é um assunto pouco conhecido e explorado. Por ser bastante específica, assume a forma de um estudo de caso. Segundo Cervo, Bervian, e Silva, (2007, p.63 e 64):

Pesquisa exploratória realiza descrições precisas da situação e quer descobrir as relações existentes entre seus elementos componentes. Esse tipo de pesquisa requer um planejamento bastante flexível para possibilitar a consideração dos mais diversos aspectos de um problema ou de uma situação. Recomenda-se a pesquisa exploratória quando há pouco conhecimento sobre o problema a ser estudado.

De acordo com as definições levantadas, torna-se perceptível que um dos objetivos da pesquisa é justamente o exploratório, pela realização das atividades de inovação necessárias ao protótipo de software e hardware do mesmo, além dos assuntos que durante o momento da pesquisa apresentam pouco repertório bibliográfico, realizando-se pesquisas em diversas fontes.

### 3.4 PROCEDIMENTOS

Quanto aos procedimentos técnicos, ou seja, a maneira pela qual obtemos os dados necessários para a elaboração da pesquisa, torna-se necessário traçar um modelo conceitual e operativo dessa, denominado de design, que pode ser traduzido como delineamento, uma vez que expressa as ideias de modelo, sinopse e plano (FREITAS; PRODANOV, 2013).

Um dos principais fatores de apresentação de um trabalho de pesquisa, é tornar clara a forma como o estudo foi aplicado, ou seja, os métodos utilizados para atingir os resultados desejados. Torna-se o momento propício de apresentar as técnicas utilizadas para fazer a coleta dos dados. Relacionando os procedimentos que serão aplicados durante a pesquisa com o objetivo de alcançar os resultados esperados, os que foram aqui utilizados seguem eles aqui listados:

- Revisão bibliográfica
- Pesquisa documental
- Pesquisa experimental

## 4 DESENVOLVIMENTO

De acordo com o estado da arte levantado no referencial teórico sobre o *framework* da seção [2.2.5.3.3 Scrumban](#), demonstrado na figura 8, tratando justamente de um modelo híbrido dos métodos ágeis *Scrum* e *Kanban*, que embasam e guiam a criação do fluxo de trabalho sugerido neste trabalho, têm-se nos próximos tópicos desta seção a apresentação do exemplo de *workflow* e seu desenvolvimento alicerçado pelo conceitos teóricos presentes na literatura referenciada para esta metodologia.

**Figura 8 —** Página inicial Scrumban+



The figure shows a screenshot of the Scrumban+ application. At the top, there is a navigation bar with three horizontal lines, a user icon, and the text "Scrumban+". Below the navigation bar, there is a photograph of two people working at a desk. One person is visible from the side, wearing a dark shirt, and another person is facing the camera, wearing a white polo shirt. They are both looking at laptops. In the background, there is a wall with several colorful sticky notes attached to it, which are typical for Kanban boards. Below the photograph, the word "Scrumban+" is displayed in a large, bold, black font. Underneath this, there is a vertical list of menu items, each preceded by a small colored arrow icon:

- + :: ▶ Team
- ▶ Requirements
- ▶ Sprint
- ▶ Deployment
- ▶ Management
- ▶ Attachments

Fonte: Elaborado pelo autor (2022)

## 4.1 O TIME SCRUM

O Time Scrum é composto pelo *Product Owner*, o Time de Desenvolvimento e o Scrum Master. Times Scrum são auto-organizáveis e multifuncionais. Times auto-organizáveis escolhem qual a melhor forma para completarem seu trabalho, em vez de serem dirigidos por outros de fora do Time. Times multifuncionais possuem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe. O modelo de time no Scrum é projetado para aperfeiçoar a flexibilidade, criatividade e produtividade.

Times Scrum entregam produtos de forma iterativa e incremental, maximizando as oportunidades de realimentação. Entregas incrementais de produto “Pronto” garantem que uma versão potencialmente funcional do produto do trabalho esteja sempre disponível. (**SCHWABER; SUTHERLAND, 2017**).

### 4.1.1 Product Owner

O Product Owner, ou dono do produto, é o responsável por maximizar o valor do produto resultado do trabalho do Time de Desenvolvimento. Como isso é feito pode variar amplamente através das organizações, Times Scrum e indivíduos. (**SCHWABER; SUTHERLAND, 2017**).

### 4.1.2 Scrum Master

O Scrum Master é responsável por promover e suportar o Scrum como definido no Guia Scrum. O Scrum Master faz isso ajudando todos a entenderem a teoria, as práticas, as regras e os valores do Scrum.

O Scrum Master é um servo-líder para o Time Scrum. O Scrum Master ajuda aqueles que estão fora do Time Scrum a entender quais as suas interações com o Time Scrum são úteis e quais não são. O Scrum Master ajuda todos a mudarem estas interações para maximizar o valor criado pelo Time Scrum.

### 4.1.3 Desenvolvedores

O Time de Desenvolvimento consiste de profissionais que realizam o trabalho de entregar um incremento potencialmente liberável do produto “Pronto” ao final de cada Sprint. Um incremento “Pronto” é requerido na Revisão da Sprint. Somente integrantes do Time de Desenvolvimento criam incrementos.

Os Times de Desenvolvimento são estruturados e autorizados pela organização para organizar e gerenciar seu próprio trabalho. A sinergia resultante aperfeiçoa a eficiência e a eficácia do Time de Desenvolvimento como um todo. (**SCHWABER; SUTHERLAND, 2017**).

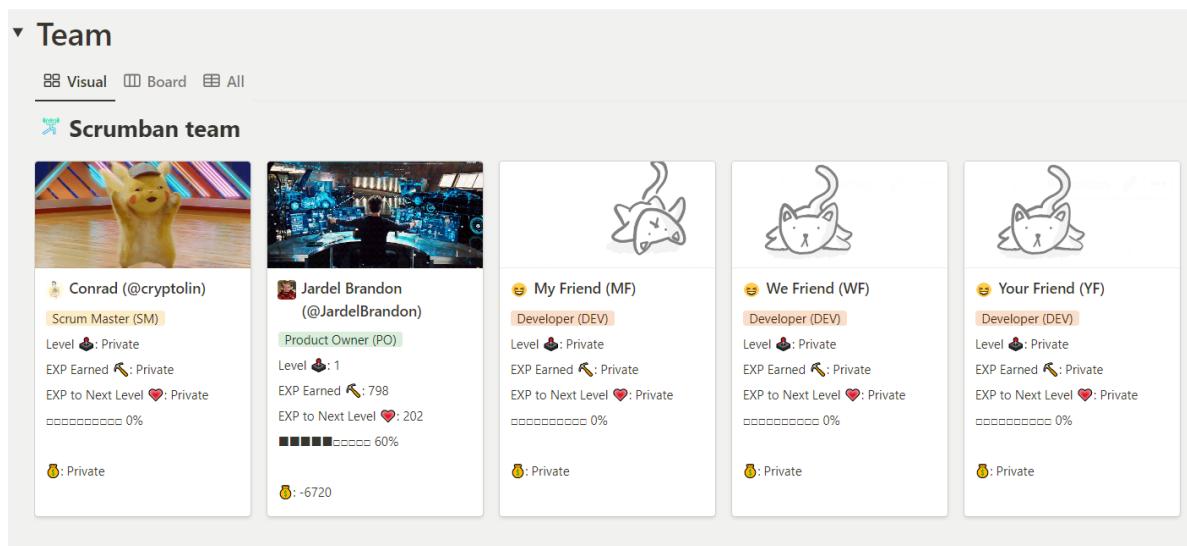
#### 4.1.3.1 Tamanho do time de desenvolvimento

O tamanho ideal do Time de Desenvolvimento é pequeno o suficiente para se manter ágil e grande o suficiente para completar um trabalho significativo dentro da Sprint. Menos de três integrantes no Time de Desenvolvimento diminuem a interação e resultam em um menor ganho de produtividade. Times de desenvolvimento menores podem encontrar restrições de habilidades durante a Sprint, gerando um Time de Desenvolvimento incapaz de entregar um incremento potencialmente liberável. Havendo mais de nove integrantes é exigida muita coordenação. Times de Desenvolvimento grandes geram muita complexidade para que um processo empírico seja útil. Os papéis de Product Owner e de Scrum Master não são incluídos nesta contagem, a menos que eles também executem o trabalho do Backlog da Sprint. (SCHWABER; SUTHERLAND, 2017).

#### 4.1.4 Notion - Atribuições do time

##### 4.1.4.1 Visualização em cartões

**Figura 9 — Time em cartões**



Fonte: Elaborada pelo autor (2022)

##### 4.1.4.2 Visualização em quadro agrupado por função

**Figura 10 — Quadro do time por papéis**

The screenshot shows a digital team board titled "Scrumban team". At the top, there are three tabs: "Visual", "Board" (which is selected), and "All". Below the title, there are three main sections representing team roles:

- Product Owner 1**: Shows a person working at a control panel with many screens. Below the image is the name "Jardel Brandon (@JardelBrandon)".
- Scrum Master (1)**: Shows a Pikachu wearing a cap. Below the image is the name "Conrad (@cryptolin)".
- Developer (DEV 3)**: Shows three cartoon cat icons. Below the first icon is the name "We Friend (WF)". Below the second icon is the name "Your Friend (YF)". Below the third icon is the name "My Friend (MF)".

At the bottom left, there is a "+ New" button.

Fonte: Elaborada pelo autor (2022)

#### 4.2 GERENCIAMENTO DO ESCOPO

De acordo com o PMBoK® (2013), o escopo do projeto deve incluir todas atividades necessárias para alcançar o produto final, de acordo com as necessidades e restrições apresentadas pelos envolvidos e de maneira que só seja executado o trabalho necessário para sua realização, nunca trabalho a mais ou a menos.

Com os requisitos bem definidos é possível elaborar o escopo, onde serão descritos detalhadamente os passos para alcançar o produto final do projeto por meio das características que os resultados devem apresentar (PMBoK®, 2013).

Segundo o **PMI (2017)** O gerenciamento do escopo do projeto inclui os processos necessários para assegurar que o projeto inclua todo o trabalho, e apenas o necessário, para que termine com sucesso. O gerenciamento do escopo do projeto está relacionado principalmente com definir e controlar o que está e o que não está incluído no projeto.

Os processos de gerenciamento do escopo do projeto são:

- 5.1 Planejar o gerenciamento do escopo—O processo de criar um plano de gerenciamento do escopo que documenta como os escopos do projeto e do produto serão definidos, validados e controlados.
- 5.2 Coletar os requisitos—O processo de determinar, documentar e gerenciar as necessidades e requisitos das partes interessadas a fim de atender aos objetivos do projeto..
- 5.3 Definir o escopo—O processo de desenvolver uma descrição detalhada do projeto e do produto.
- 5.4 Criar a EAP—O processo de subdividir as entregas e o trabalho do projeto em componentes menores e mais facilmente gerenciáveis.
- 5.5 Validar o escopo— O processo de formalizar a aceitação das entregas concluídas do projeto.
- 5.6 Controlar o escopo—O processo de monitorar o status do escopo do projeto e do produto e gerenciar as mudanças feitas na linha de base do escopo.

Para ajudar o gerenciamento de escopo do projeto, é possível definir no notion:

#### **4.2.1 Notion - Escopo**

##### **4.2.1.1 Notion - Requerimentos**

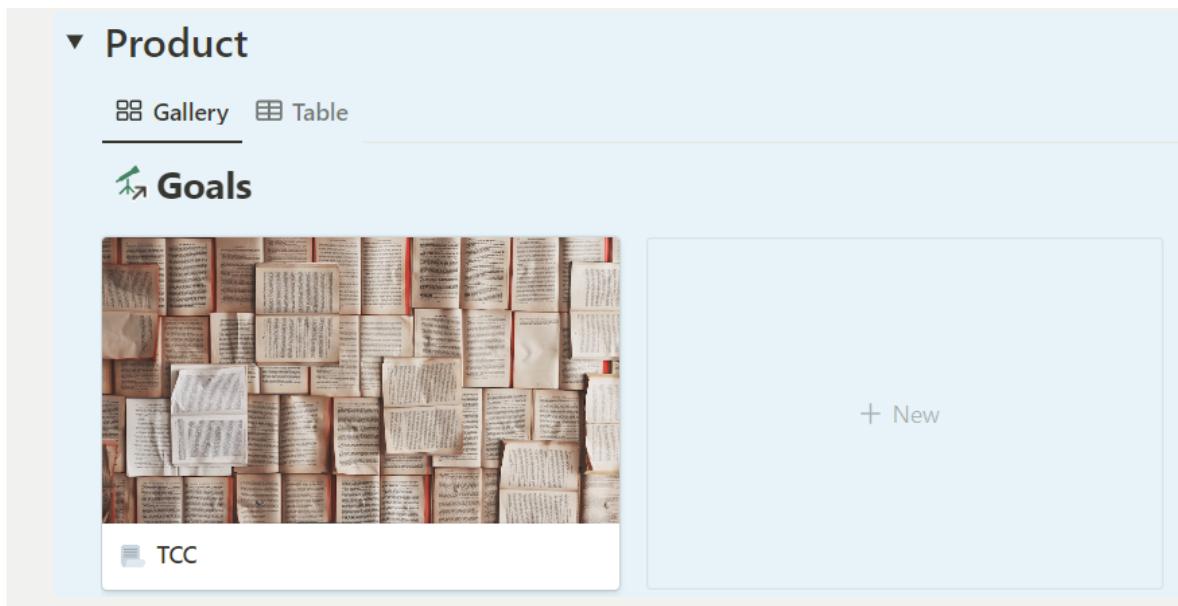
**Figura 11 — Quadro do time por papéis**

The screenshot shows a Notion workspace with a light gray header bar. Below it, a vertical sidebar on the left contains a green header 'Requeriments' with a downward arrow, followed by a green header 'Scope' with a downward arrow. Under 'Scope', there are two buttons: 'Gallery' (with a grid icon) and 'Table' (with a table icon). A horizontal line separates this from the main content area. In the main content area, there is a card titled 'Visions' with a green eye icon. The card features a thumbnail image of a person in a red graduation gown holding a diploma and a red tassel. Below the thumbnail, the title 'Computer Engineer' is displayed next to a person icon. To the right of the card is a large, empty rectangular box with a plus sign and the word 'New'. At the bottom of the card, there are three status indicators: 'Doing' (with a person icon), '18 day(s) to go!' (with a clock icon), and 'TCC' (with a document icon).

Fonte: Elaborada pelo autor (2022)

#### 4.2.1.2 Notion - Produtos

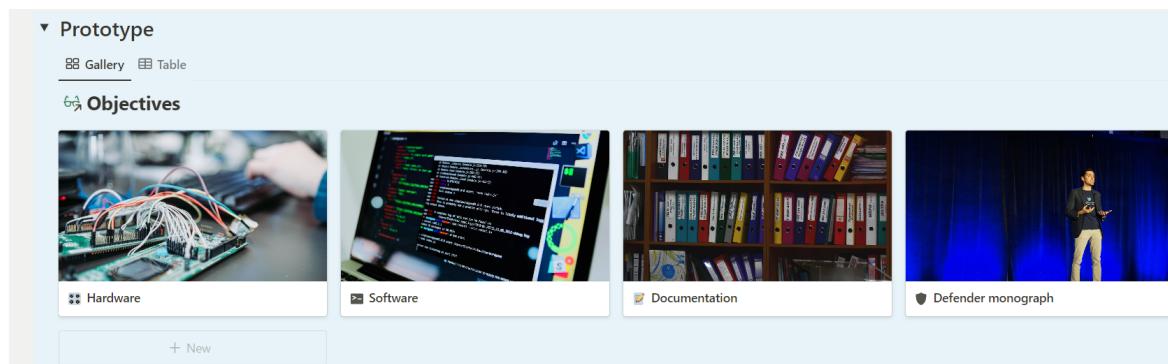
**Figura 12 — Quadro do time por papéis**



Fonte: Elaborada pelo autor (2022)

#### 4.2.1.3 Notion - Protótipos

**Figura 13 — Quadro do time por papéis**



Fonte: Elaborada pelo autor (2022)

### 4.3 CICLOS DO PROCESSO

O ciclo de vida da Scrumban é baseado em três fases principais, divididas em sub-fases:

### **4.3.1 Pré-planejamento (*Pre-game phase*)**

Além dos desenvolvimentos dos *backlogs*. O planejamento inclui também, entre outras atividades, a definição da equipe de desenvolvimento, as ferramentas a serem usadas, os possíveis riscos do projeto e as necessidades de treinamento. Finalmente é proposta uma arquitetura de desenvolvimento. Eventuais alterações nos requisitos descritos no backlog são identificadas, assim como seus possíveis riscos.

#### **4.3.1.1 Artefatos**

Os artefatos do Scrum representam o trabalho ou o valor para o fornecimento de transparência e oportunidades para inspeção e adaptação. Os artefatos definidos para o Scrum são especificamente projetados para maximizar a transparência das informações chave de modo que todos tenham o mesmo entendimento dos artefatos. (SCHWABER; SUTHERLAND, 2017)

##### **4.3.1.1.1 Backlog do produto**

O Backlog do Produto é uma lista ordenada de tudo que é conhecido ser necessário no produto. É a única origem dos requisitos para qualquer mudança a ser feita no produto. O Product Owner é responsável pelo Backlog do Produto, incluindo seu conteúdo, disponibilidade e ordenação. Um Backlog do Produto nunca está completo. Os primeiros desenvolvimentos estabelecem os requisitos inicialmente conhecidos e melhor entendidos.

O Backlog do Produto evolui tanto quanto o produto e o ambiente no qual ele será utilizado evoluem. O Backlog do Produto é dinâmico; mudando constantemente para identificar o que o produto necessita para ser mais apropriado, competitivo e útil. Se um produto existe, seu Backlog do Produto também existe.

(SCHWABER; SUTHERLAND, 2017)

#### **4.3.1.2 Notion - Artefatos**

##### **4.3.1.2.1 *Notion - Backlog dos projetos***

**Figura 14 —** Backlog do produto: Projetos

Fonte: Elaborada pelo autor (2022)

#### **4.3.1.2.2 Notion - Backlog das atividades**

**Figura 15 —** Backlog do produto: Tarefas

Product Backlog								
Tasks			Objectives			Projects		
Name	Status	Difficulty	Priority	Objectives	Projects	Next Tasks	Previous Tasks	
Materials catalog	Doing	Medium	High	Hardware	BOM (Bill Of Materials)	Materials test		
Materials test	To-Do	Medium	High	Hardware	BOM (Bill Of Materials)	Materials set-up	Materials catalog	
Materials set-up	To-Do	Medium	High	Hardware	BOM (Bill Of Materials)		Materials test	
Define hardware configurations	To-Do	Medium	High	Hardware	Hardware set-up	Base structure of the hardware		
Base structure of the hardware	To-Do	Medium	High	Hardware	Hardware set-up	Configuration of DAQ	Define hardware configurations	
Configuration of DAQ	To-Do	Medium	High	Hardware	Hardware set-up		Base structure of the hardware	
Design of the hardware	OPEN	To-Do	Medium	Semi-High	Hardware	Hardware MVP	Implementation of design	
Implementation of design	To-Do	Medium	Medium	Semi-High	Hardware	Hardware MVP	Finish the MVP project	
Finish the MVP project	To-Do	Medium	Medium	Semi-High	Hardware	Hardware MVP	Design of the hardware	

Fonte: Elaborada pelo autor (2022)

### **4.3.2 Desenvolvimento (*Game phase*)**

As muitas variáveis técnicas e do ambiente identificadas previamente são observadas e controladas durante o desenvolvimento. Ao invés de considerar essas variáveis apenas no início do projeto, como no caso das metodologias tradicionais, na SCRUM o controle é feito continuamente, o que aumenta a flexibilidade para acompanhar as mudanças. Nesta fase o software é desenvolvido em ciclos (sprints) em que novas funcionalidades são adicionadas.

#### **4.3.2.1 Eventos**

Eventos prescritos são usados no Scrumban para criar uma regularidade e minimizar a necessidade de reuniões não definidas no Scrumban. Todos os eventos são eventos time-boxed, de tal modo que todo evento tem uma duração máxima. Uma vez que a Sprint começa, sua duração é fixada e não pode ser reduzida ou aumentada. Os eventos restantes podem terminar sempre que o propósito do evento é alcançado, garantindo que uma quantidade adequada de tempo seja gasta não permitindo desperdícios no processo.

Além da Sprint, que é um container para outros eventos, cada evento no Scrumban é uma oportunidade de inspecionar e adaptar alguma coisa. Estes eventos são especificamente projetados para permitir uma transparência e inspeção criteriosa. Falhas na inclusão de qualquer um destes eventos resultará na redução da transparência e na perda de oportunidades para inspecionar e adaptar.  
**(SCHWABER; SUTHERLAND, 2017).**

##### **4.3.2.1.1 Planejamento da Sprint**

O trabalho a ser realizado na Sprint é planejado durante o planejamento da Sprint. Este plano é criado com o trabalho colaborativo de todo o Time Scrum.

O Planejamento da Sprint é um time-boxed com no máximo oito horas para uma Sprint de um mês de duração. Para Sprints menores, este evento é usualmente menor. O Scrum Master garante que o evento ocorra e que os participantes entendam seu propósito. O Scrum Master ensina o Time Scrum a manter-se dentro dos limites do time-box.

O planejamento da Sprint responde às seguintes questões:

- O que pode ser entregue como resultado do incremento da próxima Sprint?
- Como o trabalho necessário para entregar o incremento será realizado?

**(SCHWABER; SUTHERLAND, 2017).**

#### **4.3.2.1.2 Sprint**

O coração do Scrum é a Sprint, um time-boxed de um mês ou menos, durante o qual um “Pronto”, incremento de produto potencialmente liberável é criado. Sprints têm durações consistentes ao longo de todo o esforço de desenvolvimento. Uma nova Sprint inicia imediatamente após a conclusão da Sprint anterior.

As Sprints contém e consistem de um planejamento da Sprint, reuniões diárias, o trabalho de desenvolvimento, uma revisão da Sprint e uma retrospectiva da Sprint.

Durante a Sprint:

- Não são feitas mudanças que possam por em perigo o objetivo da Sprint;
- As metas de qualidade não diminuem; e,
- O escopo pode ser clarificado e renegociado entre o Product Owner e o Time de Desenvolvimento quanto mais for aprendido.

Cada Sprint pode ser considerada um projeto com horizonte não maior que um mês. Como os projetos, as Sprints são utilizadas para realizar algo. Cada Sprint tem uma meta do que é para ser construído, um plano previsto e flexível que irá guiar a construção, o trabalho e o produto resultante do incremento.

Sprints são limitadas a um mês corrido. Quando o horizonte da Sprint é muito longo, a definição do que será construído pode mudar, a complexidade pode aumentar e o risco pode crescer. Sprints permitem previsibilidade que garante a inspeção e adaptação do progresso em direção à meta pelo menos a cada mês corrido. Sprints também limitam o risco ao custo de um mês corrido.

(SCHWABER; SUTHERLAND, 2017).

#### **4.3.2.1.3 Reunião Diária**

A Reunião Diária do Scrum é um evento time-boxed de 15 minutos para o Time de Desenvolvimento. A Reunião Diária é realizada todos os dias da Sprint. Nela o Time de Desenvolvimento planeja o trabalho para as próximas 24 horas. Isso otimiza a colaboração e a performance do time através da inspeção do trabalho desde a última Reunião Diária, e da previsão do próximo trabalho da Sprint. A Reunião Diária é mantida no mesmo horário e local todo dia para reduzir a complexidade.

O Time de Desenvolvimento usa a Reunião Diária para inspecionar o progresso em direção ao objetivo da Sprint e para inspecionar se o progresso tende na direção de completar o trabalho do Backlog da Sprint. A Reunião Diária aumenta

a probabilidade do Time de Desenvolvimento atingir o objetivo da Sprint. Todos os dias, o Time de Desenvolvimento deve entender como o mesmo pretende trabalhar em conjunto, como um time auto-organizado, para completar o objetivo da Sprint e criar o incremento previsto até o final da Sprint.

A estrutura da reunião é definida pelo Time de Desenvolvimento e pode ser conduzida de diferentes formas desde que estas foquem no progresso em direção à Meta da Sprint. Alguns Times de Desenvolvimento utilizarão perguntas, outros se basearão em discussões. Aqui segue um exemplo do que pode ser utilizado:

- O que eu fiz ontem que ajudou o Time de Desenvolvimento a atingir a meta da Sprint?
- O que eu farei hoje para ajudar o Time de Desenvolvimento atingir a meta da Sprint?
- Eu vejo algum obstáculo que impeça a mim ou o Time de Desenvolvimento no atingimento da meta da Sprint?

O Time de Desenvolvimento ou membros da equipe frequentemente se encontram imediatamente após a Reunião Diária para discussões detalhadas, ou para adaptar, ou replanejar, o restante do trabalho da Sprint.

O Scrum Master assegura que o Time de Desenvolvimento tenha a reunião, mas o Time de Desenvolvimento é responsável por conduzir a Reunião Diária. O Scrum Master ensina o Time de Desenvolvimento a manter a Reunião Diária dentro do time-box de 15 minutos.

A Reunião Diária é uma reunião interna do Time de Desenvolvimento. Se outros estiverem presentes, o Scrum Master deve garantir que eles não perturbem a reunião.

Reuniões Diárias melhoram as comunicações, eliminam outras reuniões, identificam e removem impedimentos para o desenvolvimento, destacam e promovem rápidas tomadas de decisão, e melhoram o nível de conhecimento do Time de Desenvolvimento. Esta é uma reunião chave para inspeção e adaptação. ([SCHWABER; SUTHERLAND, 2017](#)).

#### 4.3.2.2 Notion - Eventos

##### 4.3.2.2.1 Notion - Planejamento da Sprint

- Épicos

**Figura 16 — Planejamento da sprint: Épicos**

Aa Name	Date	Objectives
W3	November 13, 2022 → November 19, 2022	Hardware Documentation
W4	November 20, 2022 → November 26, 2022	Software
W5	November 27, 2022 → November 30, 2022	Software
W1	December 1, 2022 → December 3, 2022	Software
W2	December 4, 2022 → December 10, 2022	< Now >
W3	December 11, 2022 → December 17, 2022	Defender monograph

Fonte: Elaborada pelo autor (2022)

- Sprint

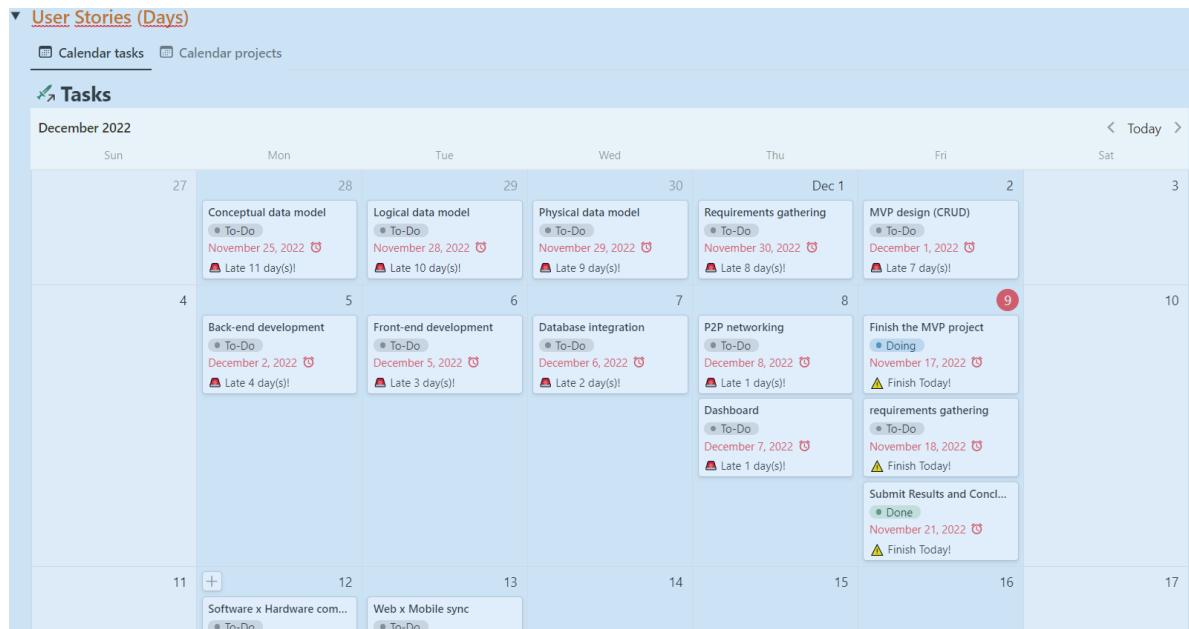
**Figura 17 — Planejamento da sprint: Atividades**

Aa Title	Projects
04/12/2022 @	
05/12/2022 @	Analysis of the results
06/12/2022 @	Analysis of the results
07/12/2022 @	Hardware MVP
07/12/2022 @	Conclusion
08/12/2022 @	Conclusion
09/12/2022 @	Conclusion
10/12/2022 @	
11/12/2022 @	
12/12/2022 @	Presentation
13/12/2022 @	Presentation

Fonte: Elaborada pelo autor (2022)

- User Stories

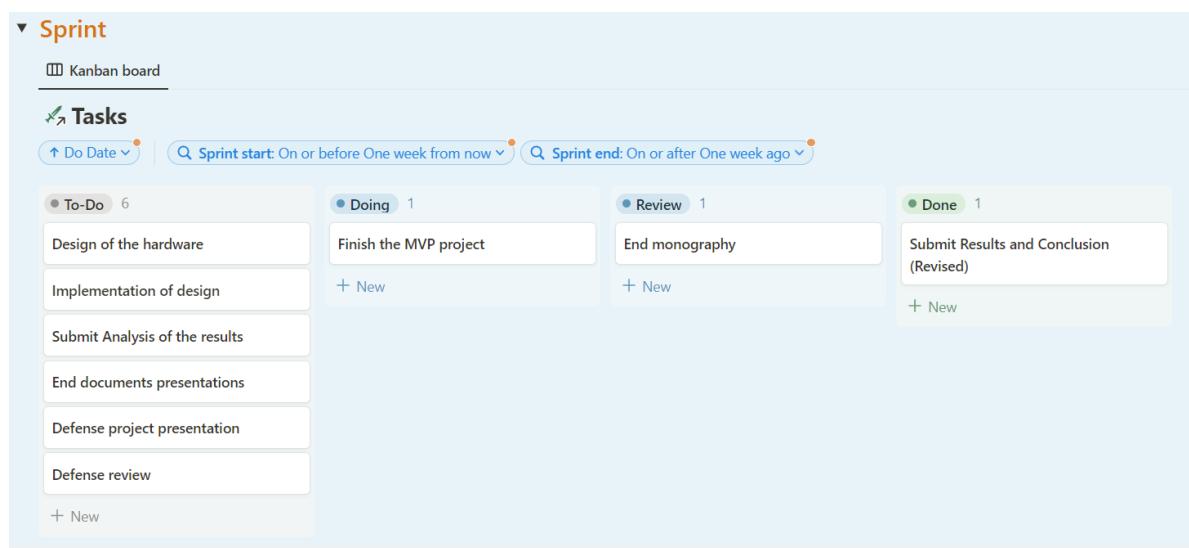
**Figura 18 — Planejamento da sprint: Histórias do usuário**



Fonte: Elaborada pelo autor (2022)

#### 4.3.2.2.2 Notion - Sprint

**Figura 19 — Kanban board**



Fonte: Elaborada pelo autor (2022)

#### 4.3.2.2.3 Notion - Reunião diária

Figura 20 — Registro das reuniões diárias

Aa Name	Date	Days	Did yest...	Do today	Impediments	Records	Video	Tags	...
Daily @Today	December 9, 2022	📅 09/12/2022 @Today	TCC	TCC	Birthday		<a href="https://www.youtube.com/watch?v=SWDhgSZNF9M">https://www.youtube.com/watch?v=SWDhgSZNF9M</a>		
Daily @Yesterday	December 8, 2022	📅 08/12/2022 @Yesterday	TCC	TCC	World cup game (Brazil)				
Daily @Wednesday	December 7, 2022	📅 07/12/2022 @Wednesday	TCC	TCC					

Fonte: Elaborada pelo autor (2022)

#### 4.3.3 Pós-planejamento (*Post-game phase*)

Após a fase de desenvolvimento são feitas reuniões para analisar o progresso do projeto e demonstrar o software atual para os clientes. Nesta fase são feitas as etapas de integração, testes finais e documentação.

##### 4.3.3.1 Encerramento

A meta da Sprint é um objetivo definido para a Sprint que pode ser satisfeita através da implementação do Backlog do Produto. Este fornece uma direção para o Time de Desenvolvimento sobre o porquê de estar construindo o incremento. Este é criado durante a reunião de planejamento da Sprint. O objetivo da Sprint dá ao Time de Desenvolvimento alguma flexibilidade a respeito da funcionalidade que será completada dentro da Sprint. Os itens do Backlog do Produto selecionados entregam uma função coerente, que pode ser o objetivo da Sprint. O objetivo da Sprint pode ser qualquer outro coerente que faça o Time de Desenvolvimento trabalhar em conjunto em vez de em iniciativas separadas.

Conforme o Time de Desenvolvimento trabalha, eles mantêm o objetivo da Sprint em mente. A fim de satisfazer o objetivo da Sprint, implementando funcionalidade e tecnologia. Caso o trabalho acabe por ser diferente do esperado pelo Time de Desenvolvimento, então eles colaboram com o Product Owner para negociar o escopo do Backlog da Sprint dentro da Sprint.

Tendo definido o objetivo da Sprint e selecionado os itens de Backlog do Produto da Sprint, o Time de Desenvolvimento irá construir essas funcionalidades durante a Sprint e transformá-las em um incremento de produto “Pronto”. Os itens de Backlog do Produto selecionados para a Sprint, junto com o plano de entrega destes itens é chamado de Backlog da Sprint. (SCHWABER; SUTHERLAND, 2017).

Após a finalização do ciclo definido de desenvolvimento da sprint, o time Scrum deve apresentar os entregáveis que foram escolhidos para o Backlog da Sprint, podendo acontecer de diversas formas conforme for combinado, as entregas dos incrementos pode também ser conjuntamente atrelada com a reunião de Revisão de Sprint.

#### **4.3.3.1.1 *Incrementos***

O incremento é a soma de todos os itens do Backlog do Produto completados durante a Sprint e o valor dos incrementos de todas as Sprints anteriores. Ao final da Sprint um novo incremento deve estar “Pronto”, o que significa que deve estar na condição de ser utilizado e atender a definição de “Pronto” do Time Scrum. Um incremento é uma parte principal inspecionável de trabalho pronto que suporta empirismo no final da sprint. O incremento é um passo na direção de uma visão ou de um objetivo. O incremento deve estar na condição de ser utilizado independente do Product Owner decidir por liberá-lo ou não. (SCHWABER; SUTHERLAND, 2017).

#### **4.3.3.1.2 *Revisão de Sprint***

A Revisão da Sprint é realizada no final da Sprint para inspecionar o incremento e adaptar o Backlog do Produto se necessário. Durante a Revisão da Sprint o Time Scrum e as partes interessadas colaboram sobre o que foi feito na Sprint. Com base nisso e em qualquer mudança no Backlog do Produto durante a Sprint, os participantes colaboram nas próximas coisas que podem ser feitas para otimizar valor. Esta é uma reunião informal, não uma reunião de status, e a apresentação do incremento destina-se a motivar e obter feedback e promover a colaboração.

Esta é uma reunião de no máximo 4 horas de duração para uma Sprint de um mês. Para Sprints menores, este evento é usualmente menor. O Scrum Master garante que o evento ocorra e que os participantes entendam o seu propósito. O Scrum Master ensina todos os envolvidos a manter a reunião dentro do Time-box.

A Revisão da Sprint inclui os seguintes elementos:

- Os participantes incluem o Time Scrum e os *Stakeholders* chaves convidados pelo Product Owner;

- O Product Owner esclarece quais itens do Backlog do Produto foram “Prontos” e quais não foram “Prontos”;
- O Time de Desenvolvimento discute o que foi bem durante a Sprint, quais problemas ocorreram dentro da Sprint, e como estes problemas foram resolvidos;
- O Time de Desenvolvimento demonstra o trabalho que está “Pronto” e responde às questões sobre o incremento;
- O Product Owner discute o Backlog do Produto tal como está. Ele (ou ela) projeta os prováveis alvos e datas de entrega baseado no progresso até a data (se necessário);
- O grupo todo colabora sobre o que fazer a seguir, e é assim que a Revisão da Sprint fornece valiosas entradas para o Planejamento da Sprint subsequente;
- Revisão de como o mercado ou o uso potencial do produto pode ter mudado e o que é a coisa mais importante a se fazer a seguir; e,
- Revisão da linha do tempo, orçamento, potenciais capacidades, e mercado para a próxima versão esperada de funcionalidade ou de capacidade do produto.

O resultado da Revisão da Sprint é um Backlog do Produto revisado que define os prováveis Itens de Backlog do Produto para a próxima Sprint. O Backlog do Produto pode também ser ajustado completamente para atender novas oportunidades. (**SCHWABER; SUTHERLAND, 2017**).

#### **4.3.3.1.3 *Retrospectiva da Sprint***

A Retrospectiva da Sprint é uma oportunidade para o Time Scrum inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima Sprint.

A Retrospectiva da Sprint ocorre depois da Revisão da Sprint e antes do planejamento da próxima Sprint. Esta é uma reunião de no máximo três horas para uma Sprint de um mês. Para Sprint menores, este evento é usualmente menor. O Scrum Master garante que o evento ocorra e que os participantes entendam seu propósito.

O propósito da Retrospectiva da Sprint é:

- Inspecionar como a última Sprint foi em relação às pessoas, aos relacionamentos, aos processos e às ferramentas;
- Identificar e ordenar os principais itens que foram bem e as potenciais melhorias; e,

- Criar um plano para implementar melhorias no modo que o Time Scrum faz seu trabalho;

O Scrum Master encoraja o Time Scrum a melhorar, dentro do processo do framework do Scrum, seu processo de desenvolvimento e suas práticas para torná-lo mais efetivo e agradável para a próxima Sprint. Durante cada Retrospectiva da Sprint, o Time Scrum planeja formas de aumentar a qualidade do produto melhorando o processo de trabalho ou adaptando a definição de “Pronto”, se apropriado e sem entrar em conflito com os padrões do produto ou organização.

Ao final da Retrospectiva da Sprint, o Time Scrum deverá ter identificado melhorias que serão implementadas na próxima Sprint. A implementação destas melhorias na próxima Sprint é a forma de adaptação à inspeção que o Time Scrum faz a si próprio. Apesar de que melhorias podem ser implementadas a qualquer momento, a Retrospectiva da Sprint fornece uma oportunidade formal focada em inspeção e adaptação. (SCHWABER; SUTHERLAND, 2017).

#### **4.3.3.1.4 Definição de “Pronto”**

Quando um item do Backlog do Produto ou um incremento é descrito como “Pronto”, todos devem entender o que o “Pronto” significa. Embora, isso possa variar por Time Scrum, os integrantes devem ter um entendimento compartilhado do que significa o trabalho estar completo, assegurando a transparência. Esta é a “Definição de Pronto” para o Time Scrum e é usado para assegurar quando o trabalho está completado no incremento do produto.

A mesma definição orienta o Time de Desenvolvimento no conhecimento de quantos itens do Backlog do Produto podem ser selecionados durante o Planejamento da Sprint. O propósito de cada Sprint é entregar incrementos de funcionalidades potencialmente liberáveis que aderem à definição atual de “Pronto” do Time Scrum. (SCHWABER; SUTHERLAND, 2017).

#### **4.3.3.2 Notion - Encerramento**

##### **4.3.3.2.1 Notion - Incrementos**

**Figura 21 — Entregáveis**

Aa Name	⌚ Implementation	📎 Attachments	☰ Tags	↗ Releases	+ ...
Integration	<a href="https://github.com/JardelBrandon/TCC">https://github.com/JardelBrandon/TCC</a>	Scrum-Guide-P...		<a href="#">V1.0</a>	
Hardware	<a href="https://github.com/JardelBrandon/TCC">https://github.com/JardelBrandon/TCC</a>	Scrum-Guide-P...		<a href="#">V1.0</a>	
Software	<a href="https://github.com/JardelBrandon/TCC">https://github.com/JardelBrandon/TCC</a>	Scrum-Guide-P...		<a href="#">V1.0</a>	

+ New

Fonte: Elaborada pelo autor (2022)

#### **4.3.3.2.2 Notion - Revisão da Sprint**

**Figura 22 — Registros das revisões de Sprint**

Aa Name	📅 Date	↗ Weeks	☰ Goals	☰ Feedback & Notes	☰ Next steps	⌚ Video
Review Sprint: W2-December	December 4, 2022 → December 10, 2022	<a href="#">W2</a>	TCC	Presentation Monograph	Defender Monograph	<a href="https://www.youtube.com/watch?v=OSXmWiZVT0g">https://www.youtube.com/watch?v=OSXmWiZVT0g</a>
Review Sprint: W1-December	November 27, 2022 → December 3, 2022	<a href="#">W1</a> <a href="#">W5</a>	TCC	Analisis of results		
Review Sprint: W4-November	November 20, 2022 → November 26, 2022	<a href="#">W4</a>	TCC			

+ New

Fonte: Elaborada pelo autor (2022)

#### **4.3.3.2.3 Notion - Retrospectiva da Sprint**

**Figura 23 — Registro das retrospectivas de Sprint**

Sprint Retrospective Reports						
Aa Name	Date	Sprint Review Reports	What worked well	What went well	What could be improved	Improvements in next s...
Retrospective Sprint: W2-December	December 4, 2022 → December 10, 2022	<a href="#">Review Sprint: W2-December</a>	TCC	TCC	Time manager	Quality
Retrospective Sprint: W1-December	November 27, 2022 → December 3, 2022	<a href="#">Review Sprint: W1-December</a>	TCC	TCC	Time manager	Quality
Retrospective Sprint W4-November	November 20, 2022 → November 26, 2022	<a href="#">Review Sprint: W4-November</a>	TCC	TCC	Time manager	Quality

Fonte: Elaborada pelo autor (2022)

#### 4.3.3.2.4 Notion - Definição de “Pronto”

**Figura 24 — Definição do estado de finalização de tarefas**

Definition of Done						
Aa Name	Tasks	Initial	Criterias	Final	Tags	...
DoD: W2-December	<ul style="list-style-type: none"> <li><a href="#">Finish the MVP project</a></li> <li><a href="#">P2P networking</a></li> <li><a href="#">Submit Metodology (Revised)</a></li> </ul>	<input checked="" type="checkbox"/>	<ul style="list-style-type: none"> <li>- Example 01</li> <li>- Example 02</li> <li>- Example 03</li> </ul>	<input checked="" type="checkbox"/>		
DoD: W1-December	<ul style="list-style-type: none"> <li><a href="#">Materials catalog</a></li> <li><a href="#">Defense review</a></li> </ul>	<input checked="" type="checkbox"/>	<ul style="list-style-type: none"> <li>- Example 01</li> <li>- Example 02</li> <li>- Example 03</li> </ul>	<input checked="" type="checkbox"/>		
DoD: W4-November	<ul style="list-style-type: none"> <li><a href="#">Implementation of design</a></li> <li><a href="#">Add improvements for the project</a></li> </ul>	<input checked="" type="checkbox"/>	<ul style="list-style-type: none"> <li>- Example 01</li> <li>- Example 02</li> <li>- Example 03</li> </ul>	<input type="checkbox"/>		

Fonte: Elaborada pelo autor (2022)

## **5      CONSIDERAÇÕES FINAIS**

Ao adentrar no universo da engenharia de software para pesquisar quais são as melhores maneiras de se desenvolver um fluxo de trabalho para criação e gestão de projetos, foi constatado que se trata de uma ciência bem abrangente e que no geral propõem-se muitos conceitos teóricos. De tal forma, que a abordagem deste trabalho foi de se realizar uma síntese destes principais conceitos e aplicá-los em um fluxo de trabalho de forma prática, utilizando para isso as ferramentas disponibilizadas na plataforma Notion.

O objetivo geral desta monografia foi compreender as metodologias de gerenciamento de projetos e realizar o levantamento do estado da arte acerca das metodologias ágeis. Para atingir este propósito, foram estudadas as características da criação de programas aplicados aos modelos e processos da gerência de projetos e análise e projeto de sistemas. A partir do diagnóstico, de que estas são áreas do conhecimento imprescindíveis para todo desenvolvimento de software, tendo em vista a grande crise ocorrida pela não utilização destas, que vieram justamente a levar a criação da Engenharia de Software. Essas medidas contribuem para estabilidade, eficiência e qualidade dos projetos de hardwares e software entregues após implantação destes novos métodos, apresentando alto nível de confiabilidade, possibilitando a evolução tecnológica.

No desenvolver da pesquisa, pode-se perceber a relevância do tema tratado, tanto pela quantidade e qualidade de literaturas abrangentes, com suas produções realizadas por autores renomados no meio acadêmico, quanto pelas instituições e organizações que tratam deste assunto, geralmente consumando relações internacionais para criação de padronizações globais. Onde são de forma colaborativa definidas normas e regulações para todos aqueles que hão de utilizar dos conhecimentos padronizados como boas práticas para a codificação de algoritmos, para o gerenciamento de projetos, tendo em vista que a não utilização destas melhores práticas tendem a levar os projetos no caminho de crises, de acordo com o já vivenciado na história da tecnologia e conforme todas as criações das soluções supracitadas, até o presente momento, se apresentam efetivas e eficazes se utilizadas de maneira correta.

As empresas ou equipes de desenvolvimento precisam adotar processos que a engenharia de software sugere como um padrão para a criação de um produto de qualidade. É importante destacar que esses processos precisam passar por modificações ao longo do tempo para atender as necessidades de cada empresa ou equipe. Provavelmente, esse seja um caminho para solução dos problemas ou, pelo menos, diminuição do índice de software de baixa qualidade.

Toda a ferramenta está disponível para duplicação no Notion, através do seguinte link:

<https://furtive-appeal-514.notion.site/Jardel-Brandon-a3ae59d77dd942a18b75aecde636e82f>

Limitações ...

Perspectivas de trabalhos futuros ...

## REFERÊNCIAS

- ABNT. Projeto 93:000.00-001 (ISO/FDIS 21500): Orientações sobre gerenciamento de projeto. *Em: ABNT/CEE-93, 2012. Anais [...]. [S. I.: s. n.], 2012.* Disponível em: <https://edoc.pub/abnt-nbr-21500-2012-gerenciamento-de-projetos-rascunho-pdf-free.html>. Acesso em: 22 nov. 2022.
- ANDERSON, D. J.; CARMICHAEL, A. **Kanban Essencial Condensado**. Illustrated First Editioned. Seattle, Washington: Lean-Kanban University, 2016. (Essential Kanban). *E-book*. Disponível em: <https://a.co/d/7nNcr9c>. Acesso em: 18 abr. 2023.
- CANDIDO, R. et al. **Gerenciamento de projetos**. [S. I.]: Aymará Educação, 2012. *E-book*. Disponível em: <http://repositorio.utfpr.edu.br:8080/jspui/handle/1/2061>. Acesso em: 14 dez. 2022.
- CARTONI, D. M. **Apostila Metodologia da Pesquisa Científica**. Campinas: [s. n.], 2011.
- CERVO, A. L.; BERVIAN, P. A.; SILVA, R. da. **Metodologia Científica - 6ª edição**. 6. ed. São Paulo: Pearson Prentice Hall, 2007. *E-book*. Disponível em: <https://www.bvirtual.com.br/NossoAcervo/Publicacao/341>. Acesso em: 16 nov. 2022.
- DIEHL, A. A.; TATIM, D. C. **Pesquisa em ciências sociais aplicadas: Métodos e Técnicas**. 1. ed. São Paulo: Pearson Prentice Hall, 2004. *E-book*. Disponível em: <https://doceru.com/doc/x1c1111>. Acesso em: 17 nov. 2022.
- FREITAS, E. C. de; PRODANOV, C. C. **Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico - 2ª Edição**. [S. I.]: Editora Feevale, 2013.
- GIL, A. C. **Métodos e técnicas de pesquisa social**. 6. ed. São Paulo: Atlas, 2008. *E-book*. Disponível em: <https://ayanrafael.files.wordpress.com/2011/08/gil-a-c-mc3a9todos-e-tc3a9cnicas-de-pesquis-a-social.pdf>. Acesso em: 17 nov. 2022.
- GRAY, C. F.; LARSON, E. W. **Gerenciamento de Projetos: O processo gerencial**. 4ºed. Porto Alegre: AMGH, 2010. *E-book*. Disponível em: [https://edisciplinas.usp.br/pluginfile.php/5687071/mod\\_resource/content/1/Gerenciamento%20de%20Projetos%20%28Gray%20-%20Larson%29%20-%20McGrawHill.pdf](https://edisciplinas.usp.br/pluginfile.php/5687071/mod_resource/content/1/Gerenciamento%20de%20Projetos%20%28Gray%20-%20Larson%29%20-%20McGrawHill.pdf). Acesso em: 21 nov. 2022.
- HARARI, Y. N. **Sapiens - Uma Breve História da Humanidade**. 1ª ediçãoed. [S. I.]: L&PM, 2015.
- IEEE STD 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. **IEEE Standard Glossary of Software Engineering Terminology, identifies terms currently in use in the field of Software Engineering. Standard definitions for those terms are established.**, [s. I.], p. 1–84, 1990.
- KINCHESKI, G. F.; ALVES, R.; FERNANDES, T. R. T. TIPOS DE METODOLOGIAS ADOTADAS NAS DISSERTAÇÕES DO PROGRAMA DE PÓS-GRADUAÇÃO EM ADMINISTRAÇÃO UNIVERSITÁRIA DA UNIVERSIDADE FEDERAL DE SANTA CATARINA, NO PERÍODO DE 2012 A 2014. [s. I.], 2015.
- OLIVEIRA, A. B. de; CHIARI, R. **Fundamentos em Gerenciamento de Projetos baseado no PMBOK 5ª edição**. 2ªed. São Paulo: Communit, 2015. *E-book*. Disponível em: <https://www.docsity.com/pt/cms-files-2206-1428511246-ebook-fundamentos-em-gerenciamento-de-projetos-pmbok5aed/4878320/>. Acesso em: 24 nov. 2022.
- PAES, L. A. B. A UTILIZAÇÃO DA METODOLOGIA PMBOK NO GERENCIAMENTO DE PROJETOS: UMA ANÁLISE DOS DAS NOVAS PRÁTICAS PROPOSTAS NA 5ª EDIÇÃO. **REGRAD - Revista Eletrônica de Graduação do UNIVEM - ISSN 1984-7866**, [s. I.], v. 7, n. 1, 2014. Disponível em: <https://revista.univem.edu.br/REGRAD/article/view/764>. Acesso em: 24 nov. 2022.
- PMI, P. M. I. **PMBOK: Um guia do Conhecimento em Gerenciamento de Projetos**. 6ª

Ediçãoed. Newtown Square, Pensilvânia: Project Management Institute, 2017. *E-book*. Disponível em:  
<https://analisederequisitos.com.br/wp-content/uploads/2020/10/pmbok-6-portugues.pdf>.

PMI. **Um guia do conhecimento em gerenciamento de projetos (Guia PMBOK)**. 5<sup>a</sup> ediçãoed. Newtown Square, Pennsylvania: Project Management Institute, 2013. *E-book*. Disponível em: [https://wiki.tce.go.gov.br/lib/exe/fetch.php/acervo\\_digital:pmbok5.pdf](https://wiki.tce.go.gov.br/lib/exe/fetch.php/acervo_digital:pmbok5.pdf).

PRESSMAN, R. S. **Engenharia De Software: Uma Abordagem Profissional**. 7<sup>a</sup> ediçãoed. Porto Alegre: AMGH, 2011.

REINERTSEN, D. G.; ANDERSON, D. J.; PINTO, A. **Kanban: Mudanca Evolucionaria de Sucesso Para Seu Negocio de Tecnologia: Mudança Evolucionária de Sucesso para seu Negócio de Tecnologia**. 1<sup>a</sup> ediçãoed. [S. I.]: Blue Hole Press, 2011.

REIS, A. A. **Scrumban-metodologia híbrida com scrum e kanban para desenvolvimento de software**. 2021. 68 f. - FACULDADE DE TECNOLOGIA DE SÃO PAULO, São Paulo, 2021. Disponível em:  
[http://ric.cps.sp.gov.br/bitstream/123456789/8502/1/ads\\_2021\\_2\\_augustoalbuquerquereis\\_scrumbanmetodologiah%c3%adbida.pdf](http://ric.cps.sp.gov.br/bitstream/123456789/8502/1/ads_2021_2_augustoalbuquerquereis_scrumbanmetodologiah%c3%adbida.pdf).

RIBEIRO, R. D.; CUNHA, H. da; RIBEIRO, S. **Métodos Ágeis: em Gerenciamento de Projetos**. 1<sup>a</sup>ed. Rio de Janeiro: SPIN Educação Profissional, 2015. *E-book*. Disponível em:  
<https://www.faeterj-rio.edu.br/downloads/bbv/0059.pdf>. Acesso em: 8 dez. 2022.

SANTIAGO, M. R. **Ensaio do SWEBOK – Software Engineering Body Of Knowledge**. 2011. 243 f. Tese de Mestrado - UNIVERSIDADE GAMA FILHO, Goiânia, 2011. Disponível em: <https://dokumen.tips/documents/swebok-traduzido.html>. Acesso em: 2 dez. 2022.

SCHWABER, K.; SUTHERLAND, J. Um guia definitivo para o Scrum: As regras do jogo. *Em: GUIA DO SCRUM*. [S. I.: s. n.], 2017. p. 20. *E-book*. Disponível em:  
<https://scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Portuguese-Brazilian.pdf>. Acesso em: 8 dez. 2022.

SOMMERVILLE, I. **Engenharia de Software**. 9<sup>a</sup> ediçãoed. São Paulo: Pearson Prentice Hall, 2011a.

SOMMERVILLE, I. Engenharia de software, 9a. **São Palo, SP, Brasil**, [s. I.], p. 63, 2011b.

STOICA, M. *et al.* Analyzing agile development-from waterfall style to scrumban. **Informatica Economica**, Romania, v. 20, n. 4, p. 5, 2016.

VALENTE, M. T. **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade**. 1<sup>a</sup> ediçãoed. [S. I.]: Independente, 2020.

VALLE, A. B. do *et al.* **Fundamentos do Gerenciamento de Projetos**. 2<sup>a</sup> ediçãoed. Rio de Janeiro: FGV, 2010. *E-book*. Disponível em:  
[https://kupdf.net/download/fundamentos-do-gerenciamento-de-projetos-fgv-andre-bittencourt-do-valle\\_5afd056ae2b6f5e1140a7f9c\\_pdf](https://kupdf.net/download/fundamentos-do-gerenciamento-de-projetos-fgv-andre-bittencourt-do-valle_5afd056ae2b6f5e1140a7f9c_pdf).