Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Plan And Go: Desenvolvimento de uma Proposta de Modelo de Processo Funcional para Controle de Atividades de Desenvolvimento de Software

Maurício Moura Dos Santos Júnior

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Alexandre Magno de Sousa

Fevereiro, 2024
João Monlevade–MG

Maurício Moura Dos Santos Júnior

Plan And Go: Desenvolvimento de uma Proposta de Modelo de Processo Funcional para Controle de Atividades de Desenvolvimento de Software

Orientador: Alexandre Magno de Sousa

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação

Universidade Federal de Ouro Preto

João Monlevade

Fevereiro de 2024

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Maurício Moura Dos Santos Júnior

***Plan And Go*: Desenvolvimento de uma Proposta de Modelo de Processo Funcional para Controle de Atividades de Desenvolvimento de Software**

Monografia apresentada ao Curso de Engenharia de Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Aprovada em 26 de fevereiro de 2024.

Membros da banca

Mestre - Professor Alexandre Magno de Sousa - Orientador - Universidade Federal de Ouro Preto
Doutora - Professora Gilda Aparecida de Assis - Universidade Federal de Ouro Preto
Mestre - Professor Igor Muzetti Pereira - Universidade Federal de Ouro Preto

Professor Alexandre Magno de Sousa, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 20/03/2024.

Documento assinado eletronicamente por **Alexandre Magno de Sousa**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 20/03/2024, às 23:03, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0688410** e o código CRC **0678736A**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Este trabalho é dedicado à mim, minha namorada que sempre me apoiou em todos os meus projetos e a minha família que sempre me deu suporte quando necessário.

Agradecimentos

Em primeiro lugar agradeço a minha família que me permitiu realizar minha graduação sem preocupações externas e provendo um lar agradável e tranquilo.

Agradeço à minha namorada que me deu ajuda e suporte em todos os momentos de dificuldade, sendo minha luz guia em momentos de escuridão e incertezas.

Agradeço ao corpo discente que me municiou com os conhecimentos necessários para estruturar o trabalho aqui apresentado.

Ademais, agradeço ao meu orientador pelo comprometimento com o trabalho e por oferecer o suporte técnico e moral para a conclusão do mesmo.

"To keep moving forward, you have to leave something behind."

— Joseph A. Cooper

Resumo

O acompanhamento de projetos de *software* tem se tornado um desafio eminente nos dias atuais. Seja pela complexidade dos projetos desenvolvidos ou pelo tamanho das equipes envolvidas, os projetos dessa natureza têm se tornado alvos de estudo. Analisar projetos de software tem como principal objetivo levantar métricas e processos de software compatíveis com a natureza peculiar que cada projeto pode apresentar. Com características que diferem de projetos em outras áreas, empregar metodologias de desenvolvimentos tradicionais podem gerar resultados negativos como, por exemplo, a evolução lenta de um trabalho. Este trabalho explora a idealização e o uso de um modelo de processo para acompanhamento e melhoria do desenvolvimento de software de forma eficiente. Por meio de esquemáticos e de uma proposta de protótipo, o projeto busca levantar as possibilidades e limitações de um modelo que seja adaptável a diversos contextos visando o tratamento e avaliação de requisitos de software. Além disso, também se propõe definir papéis a serem assumidos pela equipe de desenvolvimento no processo de criação e melhoria de projetos de software. No desenvolvimento do trabalho foram utilizadas ferramentas como o Microsoft Visio que fornece uma interface de criação de esquemas e fluxogramas que apresentam uma visão sistêmica do modelo idealizado. Também foi utilizada a ferramenta Figma na elaboração de um protótipo de acordo com o modelo de processo proposto, assim, ele mostra como requisitos podem ser rascunhados a partir de informações básicas de uma possível ferramenta funcional. Esse protótipo foi utilizado em dois momentos, no primeiro para demonstrar como protótipos de alto nível se encaixam na cadeia de requisitos levantados e sua avaliação e, posteriormente, no uso do modelo de processo idealizado. No desenvolvimento do projeto, estratégias para melhoria de fluxos de decisão foram consideradas, especialmente em contextos de desenvolvimento de software recentes do mercado os quais utilizam metodologias ágeis. Assim, como produto final, foi desenvolvido um modelo de processo que concatena as etapas desde o levantamento de requisitos, passando pela entrega de resultados gerados por um software até a entrega de soluções ao cliente. Por fim, foi possível identificar fatores catalisadores em um processo de software, idealizar uma distribuição de atividades para uma equipe e indicar os artefatos e documentos que precisam ser gerados a partir de cada etapa do processo de desenvolvimento.

Palavras-chaves: Desenvolvimento de software. Métodos ágeis. Elicitação de requisitos. *Sprint* de desenvolvimento. Modelo de processo. Protótipo. Estudo de caso. Plan and Go.

Abstract

Nowadays, monitoring software projects has become an eminent challenge. This is due to the complexity of the projects developed or the size of the teams involved, so that projects of this type have become the focus of research. The main goal of analyzing and studying software projects is to identify metrics, processes and software operations that are compatible with the particular characteristics of each project. The characteristics of software projects are different from those of other fields, so the application of standard development methods can lead to negative results such as slow development of the work. This work examines the idealization and use of a software process model for the efficient monitoring and improvement of software development. Using schemas and a prototype, the possibilities and limitations of an adaptable model for different contexts are shown, with a focus on handling and evaluating software requirements. In addition, we propose to define the roles of the development team in the creation process and in the improvement of software projects. We also use tools such as Microsoft Visio, which provides an interface to create schemas and flowcharts that represent a systemic view of the idealized model. Furthermore, we used Figma in the construction of the prototype according to our proposed process model to show how the requirements can be derived from the basic information about a possible functional tool. The prototype is used in two moments, firstly to show how high-level prototypes fit into the chain of requirements elicitation and their evaluation, and secondly in the use of the idealized process model. During the project development, strategies to improve the decision-making processes were considered, especially in the context of modern software development with agile methods. As a final product, we then developed a process model that links the steps from requirements elicitation to the delivery of software results and solutions to customers. Finally, we identified catalytic factors in a software process, idealized the distribution of activities in the team, and identified the artifacts and documents required for each step of the software development process.

Key-words: Software development. Agile methodologies. Elicitation of requirements. Development sprint. Process model. Prototyping. Case study. Plan and Go.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SSCs *Small Software Companies*

XP Extreme Programming

Sumário

1 Introdução

As plataformas de apoio à desenvolvimento de software têm se tornado ferramentas de grande valia para equipes de desenvolvimento, seja em grupos com ciclos iniciais de aprendizado, para empresas que pretendem adotar métricas de produção com o objetivo de aumentar sua produtividade geral, ou por alunos e professores de graduação que adotam essas ferramentas de software para controlar e/ou melhorar o processo de criação de seus projetos de pesquisa (PAASIVARA, 2014). Em geral, tais ferramentas são voltadas para a implantação de um processo coeso, linear e sem grande abertura para modificações no seu ciclo de tarefas, visto que os projetos acompanhados por essas plataformas têm como uma de suas principais características a determinação de datas para a realização de entregas de artefatos finais ou parciais.

Apesar da vasta gama de abordagens existentes, observa-se que, na prática, algumas das técnicas existentes acabam por ser demasiadamente vagas, longevas e desconectadas da realidade da produção de software em contextos dinâmicos. Por exemplo, o trabalho de Rasheed (2021) aponta como principais dificuldades a falta de clareza para desenvolvedores com relação ao seu papel no contexto do projeto e de seus fluxos de trabalho, bem como a necessidade de que os mesmos sejam "mestres de todas as habilidades". Isso também ressalta a falta de clareza no processo, além disso, observa-se a introdução de metodologias que não estão fundamentadas em conhecimentos presentes na equipe. Esse último caso pode ser analisado com clareza quando metodologias ágeis são instauradas sem que o treinamento adequado seja realizado para a compreensão de tal metodologia e seus impactos.

Outras dificuldades, como as citadas por Cho (2019), podem surgir da falta de lideranças experientes em contextos dinâmicos de projetos ágeis e também da falta de incentivo e motivação para implantar sistemas de controle que se adaptem aos projetos que já estão em curso, algo que demanda uma curva de aprendizado acentuada. De forma geral, nota-se a necessidade de um olhar mais específico para cada tipo de projeto desenvolvido e suas características, assim como para a sua equipe de desenvolvimento. Como consequência, surge um grande espaço para a criação de novas abordagens que podem variar na forma de aplicação dos métodos ágeis dentro do contexto de cada empresa. Tais abordagens variam em tamanho, escala e complexidade de projetos, bem como Saeeda, Ahmad e Gustavsson (2023) e Hajjdiab (2011) descrevem em suas buscas por um melhor entendimento dos problemas de integração entre times, metodologias e ambiente de trabalho.

1.1 Motivação e Justificativa

Em primeiro lugar, este trabalho é motivado pela necessidade de complementar e explorar de uma maneira mais geral, temas levantados pelo autor em um trabalho prévio e que foi base para o trabalho atual. O trabalho anterior foram levantadas questões sobre a versatilidade e adaptabilidade de processos de software ditos abrangentes quando aplicados a um contexto de desenvolvimento de nicho, no caso, equipamentos embarcados que requerem uma visão bastante particular principalmente em seu processo de levantamento de requisitos. É possível observar na literatura uma lacuna entre a existência de projetos diversos que requerem a adaptação de processos de software e a real existência de abordagens que supram a necessidade de tais projetos como visto no trabalho de Acuna (2019). A partir de tal estudo, o presente trabalho se justifica pela necessidade de explorar os processos de software existentes e levantar possibilidades de criação de modelos de processos assistidos que sejam adaptáveis para diversos tipos de projetos.

1.2 Definição do Problema

No trabalho de Moura (2019) foi possível levantar características e falhas de processos de desenvolvimento de software para dispositivos embarcados em um estudo de caso. As ocorrências de problemas se davam pela falta de documentação dos projetos, aliadas à falta de integração com outros projetos de suporte. Nesse trabalho, os seguintes pontos críticos foram identificados na:

- **Descrição de tarefas:** as atividades são descritas de maneira superficial, poucas informações são retidas e a sua continuidade por outro membro da equipe é dificultada pela curva de aprendizado que se faz necessária a partir do fato de que a atividade não era bem descrita. Atualmente, a descrição de tarefas pode ser feita de modo individual para cada tarefa ou tratada como um sistema que integra tarefas e constrói suas descrições de maneira conjunta (BRANISLAV RUZICKY, 2018). A principal vantagem da abordagem individual para cada tarefa é a liberdade que todos os interessados possuem para avaliar e criar uma linha do tempo para desenvolver suas atividades. Em contrapartida, a abordagem conjunta ajuda o time de desenvolvimento a ter uma visão mais ampla de como o entregável será formatado, além de fornecer datas de entrega mais realistas para um entregável de tamanho considerável;
- **Temporização de atividades:** atividades com diferentes complexidades e pré-requisitos são alocadas em períodos de tempo os quais não são realistas. Dessa forma, o desenvolvedor acaba por optar por uma abordagem em que ele desenvolve a tarefa no período de tempo indicado. Porém, como esse período não se mostra compatível com a realidade da tarefa, a qualidade do entregável desenvolvido é comprometida;

- **Comunicação entre desenvolvedores:** as mudanças dentro dos projetos, tais como a finalização de tarefas, alteração de datas e a criação de novas atividades, necessitam de uma reunião formal com toda a equipe para serem informadas.

Diante desse cenário, percebe-se que cada projeto possui uma configuração específica, por isso, os seguintes problemas são identificados: (*i*) alta complexidade; (*ii*) pouca integração de dados; (*iii*) documentação escarça; e, por fim, (*iv*) artefatos técnicos sem possibilidade imediata de serem legados. Além disso, também foram analisadas outras configurações de projetos presentes na literatura, tal como o de Paasivara (2014), que descreve os desafios que surgem quando softwares legados ou não recebem a implementação de metodologias ágeis. Outra configuração observada foi dada pela construção de projetos de software embarcado, como visto por Patil (2010), no qual pode ser analisada a falta de robustez e confiabilidade dos processos que são geralmente utilizados e para levantar requisitos para uma variação de software (embarcados) que necessitam de grande precisão e abrangência. Por fim, o trabalho de Mori Fabio Paterno (2015) mostra como aplicações Web de página única podem ser dificultadas em seu processo de levantamento de requisitos e desenvolvimento quando elementos como usabilidade e fluxo de interações não são abordadas e refinadas de maneira aceitável.

Em suma, observou-se que os problemas identificados no trabalho *Plan And Go*, destacado aqui como trabalho anterior, também são comuns na maioria dos casos e se agravam em situações em que as metodologias de desenvolvimento e equipes estavam em constante mudança. Assim, este projeto pretende expandir o horizonte de avaliação do trabalho anterior *PlanAndGo* e apresentar uma proposta de solução para a problemática da criação de um fluxo de software que contemple o levantamento de requisitos, sua avaliação e documentação.

1.3 Objetivos Gerais e Específicos

O objetivo geral deste trabalho é criar uma proposta de modelo de processo funcional para controle de atividades de desenvolvimento de software. Assim, por meio deste modelo, espera-se que os desenvolvedores possam aprimorar o andamento de suas atividades para que possam gerenciá-las de uma maneira mais eficiente. Além disso, pretende-se obter uma documentação e comunicação mais eficaz dentro do processo de desenvolvimento software. Para isso, será proposto um modelo de processo funcional baseado nos pilares da metodologia ágil, mais especificamente, do padrão Scrum (SACHDEVA, 2016).

No modelo de processo proposto por este trabalho, os processos de identificação e levantamento de requisitos, bem como a passagem desses requisitos pela cadeia de desenvolvimento de software até sua entrega como funcionalidades em estado de utilização pelos usuários poderão ser analisados em cada etapa de desenvolvimento. Como consequência,

as atividades do modelo de processo podem então ser aplicadas à um protótipo de software de gestão de processos e tarefas que pode apresentar informações essenciais do projeto de software que está sendo desenvolvido pela equipe. Esse protótipo poderá permitir a gestão das atividades, a comunicação entre participantes e a alocação temporal das atividades de acordo com sua complexidade.

Para alcançar o objetivo geral, os objetivos específicos são definidos a seguir:

- Identificar as atividades base de um processo de software para elicitação e refinamento de requisitos;
- Formular um modelo de desenvolvimento com características cíclicas e baseado em metodologias ágeis para criação de projetos de desenvolvimento de software;
- Selecionar um modelo Figma[1] mais adequado para criação do protótipo funcional desejado;
- Criar um *layout* de telas para o protótipo a partir do modelo Figma por meio da identificação de possibilidades e limitações do modelo de processo que será idealizado, desde etapas de levantamento, refinamento, avaliação técnica e desenvolvimento de requisitos;
- Aplicar o modelo criado em um estudo de caso simulado para projetar os possíveis resultados esperados dentro de um contexto real.

1.4 Resultados e Contribuições

Os resultados apresentados apontam para uma melhora na identificação e integração das tarefas dentro de uma equipe de desenvolvimento. Com a elaboração do modelo de processo de software é possível realizar melhor rastreio dos requisitos levantados e explorá-los de forma mais sistêmica. Após a comparação dos resultados obtidos dentro do estudo de caso é possível identificar que o Plan And Go fornece uma abordagem que favorece a integração entre membros de uma equipe e que a aplicação de seu fluxo pode contribuir para uma análise mais profunda, técnica e bem-sucedida dos requisitos e funcionalidades levantadas. Puderam também ser identificadas limitações relativas as mapeamento de informações a longo prazo como a manutenção de um *backlog* longevo. Além de propostas para trabalhos futuros como a melhoria do mapeamento de tarefas relacionadas a outros fatores como a qualidade de software, indo assim além de apenas rastrear requisitos imediatos de uma sprint corrente.

[1] Figma é um editor gráfico vetorial e prototipagem de projetos de design baseado principalmente em navegador Web, também conta com ferramentas *offline* adicionais para aplicações desktop, tais como para GNU/Linux, macOS e Windows. Pode ser acessado em: <http://figma.com>.

1.5 Estrutura da Monografia

Este trabalho está estruturado conforme descrito a seguir. No próximo capítulo, Capítulo 2, é apresentada a fundamentação teórica na qual são apresentados os principais conceitos abordados e utilizados neste projeto. No Capítulo 3 são descritos os trabalhos relacionados que abordam o tema do trabalho com diferentes tipos de visões. Por sua vez, o Capítulo 4 aborda os processos utilizados para desenvolvimento do modelo de processo e do *layout* do protótipo. O capítulo 5 demonstra a aplicação do modelo de processo criado integrado ao protótipo proposto dentro do contexto de um estudo de caso simulado. Já no Capítulo 6, são apresentadas as análises dos resultados deste trabalho. Por fim, no Capítulo 6 são discutidos os resultados finais do trabalho, considerando suas limitações, contribuições e possíveis trabalhos futuros.

2 Fundamentação Teórica

Este capítulo apresenta uma síntese das práticas, métodos e conceitos utilizados como fundamentos para a elaboração deste trabalho. Os assuntos aqui tratados tem influência direta nos tópicos restantes do texto e formam uma base sólida de conhecimento sobre o ferramental clássico e moderno, isto é, mais recentes, e também traz visões futuras do desenvolvimento de software realizado com suporte de ferramentas e métodos ágeis.

Este capítulo está estruturado conforme descrito a seguir. A Seção 2.1 descreve um trabalho previamente realizado pelo autor onde foram abordadas questões sobre a engenharia de requisitos e processos de software adaptados. A Seção 2.2 apresenta a definição e principais objetivos dos métodos ágeis dentro dos contextos em que são aplicados. Na Seção 2.3 são tratadas as abordagens de desenvolvimento de software com metodologias clássicas, apresenta suas características limitantes e benéficas a um processo de software que tem foco no levantamento de requisitos e desenvolvimento de funcionalidades. Por sua vez, a Seção 2.3.3 descreve as técnicas modernas para elicitação de requisitos em projetos que empregam metodologias ágeis e levanta seus impactos nesses contextos. Por fim, a Seção 2.4 apresenta as considerações finais sobre metodologias ágeis, processos de levantamento de requisitos e como esses conceitos podem ser utilizados para se alcançar uma abordagem que seja satisfatória em ambientes de desenvolvimento que necessitam de abordagens ágeis.

2.1 Diretrizes da Engenharia de Requisitos

Em um trabalho anterior, como parte de um estudo voltado para a aplicação de engenharia de requisitos e metodologias ágeis para desenvolvimento de softwares embarcados, foi desenvolvido pelo autor deste projeto um trabalho em que foram abordadas questões e práticas interessantes sobre como métodos padronizados de engenharia de requisitos poderiam ser adaptados e implantados no desenvolvimento de aplicações com características peculiares – como os softwares embarcados (MOURA, 2016).

O trabalho foi desenvolvido em ambiente acadêmico, mais precisamente no laboratório iMobilis , localizado dentro do campus avançado da Universidade Federal de Ouro Preto na cidade de João Monlevade, Minas Gerais. Esse laboratório de pesquisas é voltado para a criação de projetos de softwares embarcados e possuía diversos alunos e professores colaboradores que desenvolviam trabalhos nessa área de pesquisa.

Os principais objetivos do trabalho são: (*ii*) compreender como a engenharia de requisitos para sistemas embarcados é realizada em empresas do mercado de tecnologia da

informação; (*ii*) entender na literatura quais são as abordagens propostas para a indústria e para a academia; (*iii*) observar como é realizada esta disciplina pelos alunos do iMobilis em seus projetos; (*iv*) propor uma perspectiva que contempla às características do BOPE [1]e do iMobilis; (*v*) melhorar continuamente a perspectiva proposta através de *feedbacks* constantes dos envolvidos; e, por fim, (*vi*) documentar a nova perspectiva, implantar no laboratório e reportar os resultados obtidos.

Para alcançar tais objetivos, foi realizada um revisão geral da literatura a cerca das características dos softwares embarcados e em seguida uma revisão da literatura sobre o estado da arte da engenharia de requisitos aplicada nesses softwares. Posteriormente, foram realizadas com a equipe pertencente ao laboratório entrevistas baseadas em questionários, buscando explicitar o sistema desenvolvido e o porquê das decisões tomadas durante o período de criação do sistema. Ao fim dos questionamentos, foi elaborado um documento que envolve como tratar as principais dificuldades dos desenvolvedores. Os dados colhidos foram documentados e armazenados para posterior consulta.

Com a coleta dos dados, foi possível alcançar resultados que indicavam os seguintes aspectos:

- Existe dentro da fase inicial de levantamento de requisitos para softwares embarcados uma grande necessidade de identificar requisitos já obtidos em projetos correlatos, tal prática facilita a integração de novas funcionalidades e reaproveitamento de código;
- A utilização de *frameworks* de desenvolvimento consolidados é bem vinda já que softwares embarcados, por natureza, necessitam de precisão alta em sua execução;
- Definir requisitos baseados em testes é de suma importância, já que os testes necessários para os softwares embarcados estão diretamente ligados ao consumo de recursos;
- Foi identificado que uma solução embarcada que passa pelos testes identificados nem sempre é a solução final. Assim, otimizar a solução final para atender critérios de tempo de resposta e consumo energético, por exemplo, é ideal. Isso se tornou ainda mais importante quando foi levado em consideração o contexto acadêmico, onde recursos são limitados.

O estudo de diferentes tipos de software apresenta na prática desafios que devem ser considerados desde sua concepção e não apenas levados em conta no momento de aplicar testes que são usados como padrão em outros modelos de software. Nesse ponto, foi observada a necessidade de considerar limitações além das técnicas, como a limitação

[1] BOPE é um processo mistura práticas de engenharia de software oriundas de processos tradicionais e ágeis para se adaptarem ao contexto acadêmico de um laboratório de pesquisa e desenvolvimento de software. Pode ser acessado em: <www.monografias.ufop.br/handle/35400000/621>/

de recursos financeiros, de tempo e de curva de aprendizado. Por fim, observar que ambientes acadêmicos contribuem para tornar o desenvolvimento de software de nicho ainda mais desafiador é importante para levantar proposições sobre a necessidade da adaptação de métricas padrões para tipos de softwares diferentes e ambientes diversos de desenvolvimento.

2.2 Métodos Ágeis

Os métodos ágeis podem ser definidos como um conjunto de princípios que facilitam a abordagem de tarefas, formulação de objetivos a curto e médio prazo, e o alcance de metas dentro de um período de tempo variável que depende da complexidade do projeto em desenvolvimento (ANAND, 2016). Tais princípios tem como principais características:

- **Entrega constante de valor**: a busca constante de entregas vem para principalmente demonstrar avanço constante do projeto. A evolução dos entregáveis também demonstra a maturidade do software que evolui com base nos *feedbacks* e que orientam a priorização de tarefas e as melhorias entregues;
- **Iteratividade**: flexibilidade de prazos, rotação de tarefas e recondicionamento de entregáveis de forma planejada fazem com que a qualidade do software melhore de forma constante e sem perda de mapeamento temporal;
- **Colaboração constante entre pessoas interessadas no software**: tais métodos também incentivam tanto a colaboração quanto a comunicação entre todas as pessoas da equipe, bem como com os *stakeholders* do projeto. Uma das características das equipes ágeis é trabalhar de forma interdisciplinar, o que promove a troca constante de ideias, conhecimentos e experiências;
- **Autonomia de equipes e indivíduos**: em paralelo com a colaboração constante, a troca de informação entre as partes acaba por promover também maior autonomia dos indivíduos e equipes que conseguem e são incentivados a se auto-organizarem, a tomar decisões e a definir o melhor caminho para atingir os objetivos do projeto em desenvolvimento;
- **Resposta imediata à mudanças**: ao observar o crescimento do projeto e a maturidade dos entregáveis passa a ser necessário avaliar as mudanças necessárias na estrutura do projeto. Os métodos ágeis têm como característica permitir mudanças estruturais, tanto nas pessoas quanto nas abordagens de desenvolvimento escolhidas, em detrimento de seguir um plano não-flexível.

Seguindo tais princípios, as metodologias de desenvolvimento ágil de software buscam ampliar um horizonte de possibilidades que era limitado pelas abordagens em

cascata e guiadas por planos, metodologias essas que, por apresentarem rigidez elevada, não se adaptam bem à um mercado de criação de software que exige adaptabilidade e criatividade para enfrentar novos desafios.

2.3 Elicitação de Requisitos

Com a criação do manifesto ágil no início do século XXI e a formalização das ideias do que compõem uma metodologia que segue tal manifesto, surgiram as primeiras abordagens para a engenharia de software e a engenharia de requisitos que compõem as metologias ágeis juntamente com a mudança de paradigma que tal marco representa (BRANISLAV RUZICKY, 2018).

Apesar dos diversos anos de evolução e refinamento dos métodos e abordagens ágeis, as primeiras abordagens são conhecidas e utilizadas até os dias atuais e recebem constantes atualizações. Por exemplo, o Extreme Programming (XP) é uma metodologia que gira em velocidade, rotatividade, micro ciclos de desenvolvimento e documentação reduzida. Essa metodologia é baseada em *feedback* constante, abordagem incremental e comunicação proativa, sendo assim, é voltada para pequenas e médias empresas onde as regras de negócio são conhecidas a fundo por todos os envolvidos (SOARES, 2004). O uso do XP foi e ainda é polêmico, pois as noções de documentação e transmissão de conhecimento legado são deixadas de lado em diversas ocasiões, principalmente em troca do favorecimento das entregas de código constantes, mesmo que tais possam ser prejudicadas no longo prazo.

Por sua vez, o Scrum, metodologia mais conhecida e dissipada nas empresas que empregam os métodos ágeis como forma de abordagem ao desenvolvimento, traz consigo a necessidade constante das práticas de ritos de integração, documentação e passagem de conhecimento legado (SACHDEVA, 2016). Com nível de robustez elevado, o Scrum consegue ser aplicado em empresas de diversos portes e fornece flexibilidade na tomada de decisões aliada à segurança que vem por meio da comunicação e integração da equipe envolvida.

2.3.1 Modelos Clássicos de Elicitação de Requisitos

Antes da definição e propagação dos métodos ágeis, algumas práticas que são empregadas nesses métodos já existiam e eram utilizadas com algum nível de sucesso. Dentro dessas práticas, são incluídos os modelos de elicitação de requisitos que podem ser utilizados. Assim, em primeiro lugar, é necessário definir os seguintes conceitos:

- **Requisitos Funcionais**: segundo Yousuf (2015), os requisitos funcionais são definidos como um conjunto base e não negociável de funcionalidades que serão desenvol-

vidas em sua totalidade, depois de serem finalizados, outros requisitos funcionais podem ser levantados para complementar o software a ser desenvolvido;

- **Requisitos não Funcionais**: são requisitos totalmente necessários para a construção de um software (YOUSUF, 2015), são definidos como um conjunto de especificações que descrevem o sistema desejado, suas capacidades e restrições, além de definir métricas para avaliação, tal como, por exemplo, segurança e velocidade.

A elicitação de requisitos funcionais e não funcionais de um projeto é parte crucial na estruturação das tarefas, prazos e abordagens de codificação, pois as mesmas pautam desde a estrutura de determinados módulos do sistema até a complexidade das tarefas distribuídas para a equipe. Os métodos de elicitação primordiais em desenvolvimento de software partem de um pressuposto muito perigoso, eles consideravam que os *stakeholders* possuíam todas as respostas definidas em relação ao projeto, algo que é claramente visto como falso quando se observa um desenvolvimento em médio ou longo prazo. Esse pressuposto direciona a um levantamento de requisitos limitado e que acarreta lacunas de dados e funcionalidades cruciais no momento de transcrever requisitos em código (YOUSUF, 2015). A seguir, são descritas as principais abordagens clássicas de elicitação de requisitos:

- **Entrevistas**: focadas primeiramente em uma abordagem não guiada, as entrevistas podem ser realizadas com o intuito de extrair informações de maneira informal dos *stakeholders*, como usuários e gerentes de área (GOGUEN, 2015). As entrevistas tendem a obter uma visão voltada para o indivíduo e podem ser proveitosas quando se desenvolve um software de utilização pessoal, mas, podem levar a conclusões equivocadas quando o software criado será utilizado em conjunto;

- **Questionários**: partindo na contramão das entrevistas, Goguen (2015) mostra que, geralmente, os questionários têm grande foco em guiar o usuário para uma coleção de artefatos e *features* que os donos do produto sabem que serão úteis no projeto. Assim, por meio da eliminação e refinamento, é possível alcançar um conjunto de informações úteis para a criação de entregáveis;

- **Análise de Documentação**: aqui é vista uma abordagem que se afasta do *stackholder* de forma direta e busca analisar o contexto em que ele está inserido por meio da documentação gerada por ou para ele. Goguen (2015) também mostra que tal abordagem é mais utilizada em casos em que o projeto deve ter funcionalidades de digitalização e interpretação. Assim, a análise de documentação sempre requer um trabalho paralelo com o usuário para que os dados e documentos não sejam interpretados de maneira errônea.

2.3.2 Modelos de Elicitação de Requisitos e Métodos Ágeis

Levando em consideração que os métodos previamente citados também podem ser aplicados em contextos ágeis, segundo Saeeda, Ahmad e Gustavsson (2023), é necessário analisar novos contextos e buscar novos meios de retirar informações dos usuários. Em ambientes distribuídos, em que cliente e equipe de desenvolvimento necessitam praticar a agilidade de forma completa, é interessante utilizar a reunião de pessoas interessadas para gerar discussões sobre o produto criado como é visto no trabalho de Branislav Ruzicky (2018). Assim, os modelos de elicitação de requisitos atualmente mais ligados aos métodos ágeis procuram tirar vantagem da reunião de pessoas. A seguir podem ser citados alguns exemplos dessas reuniões:

- ***Workshops* de requisitos**: buscam explorar ideias já existentes, esses *workshops* tentam trazer a tona comportamento e atividades já existentes, trabalhar seu refinamento e entregar um conjunto de documentos que discute e apresenta possíveis soluções e *features* que atendam as necessidades do cliente;
- ***Workshops* de desenvolvimento**: assim como os *workshops* anteriores, os de desenvolvimento buscam explorar ideias já existentes. Porém, nesse caso, essas ideias são exploradas com o apoio dos desenvolvedores. Dessa forma, são criadas soluções técnicas ou protótipos de baixo nível que dão ao usuário uma apresentação de contexto visual e de usabilidade para as *features* e possíveis entregáveis criados.

Diante desse cenário, a abordagem utilizada neste trabalho trata-se de uma visão mista que busca manter padrões clássicos proveitosos, mas que ajudam as equipes a se organizar e a documentar requisitos de maneira eficiente dentro de um projeto de desenvolvimento de software. Sendo assim, as equipes podem produzir uma documentação robusta sobre o software e que ajude na estruturação de tarefas definidas a partir dos requisitos estabelecidos.

2.3.3 Modelos de Elicitação de Requisitos Modernos e Métodos Ágeis

A elicitação de requisitos quando aplicada em contextos ágeis depende diretamente da inserção do profissional de produto no contexto em que o software criado será aplicado. Assim, busca tomar proveito da flexibilidade que os métodos ágeis como o Scrum fornecem, dessa forma, torna-se interessante analisar os dados que irão alimentar o sistema, as pessoas envolvidas em seu uso e como elas irão compartilhar as informações geradas pelo sistema. Por fim, deve-se traçar uma relação direta entre esses pontos anteriormente citados e uma interface que será a forma de interação entre usuário e sistema como descrito no trabalho de Goguen (2015). Com isso, os itens a seguir são utilizados em diversas abordagens ágeis que requerem elicitação e análise de requisitos:

- **Observação**: segundo Goguen (2015), inserir um profissional de produto diretamente no local de utilização do software a ser criado gera possibilidades de observação que a elicitação de requisitos por meios padrões não seria capaz de fornecer. A Interação direta com os *stackholders* pode ser crucial nesse ponto;

- **Prototipação Direta**: essa abordagem é tratada no trabalho de Sousa (2019), o profissional de requisitos trabalha juntamente na maioria das vezes com o profissional de *design*. A prototipação direta gera artefatos de baixa e média complexidade, principalmente quando os requisitos levantados têm foco em interface e interação entre dados em telas do sistema. Essa técnica tem como vantagem a possibilidade de ser combinada com diversas outras. Dessa forma, a prototipação é largamente aplicada pois gera resultados visuais que são bastante explicativos e em grande parte levam usuários a darem *feedbacks* muito construtivos.

2.4 Considerações Finais

O gerenciamento de atividades, sejam elas voltadas para elicitação de requisitos ou desenvolvimento, compreende grande parte do processo de criação de um software. Mesmo quando o contexto apresentado tem características simples, a organização de tarefas tende no curto, médio e longo prazo facilitar a rastreabilidade e melhoria de artefatos e funcionalidades desenvolvidas.

Dentre as possíveis abordagens metodológicas existentes, os métodos ágeis se apresentam como a proposta, em sua maior parte, mais adaptável e propícia a mudanças contextuais. Assim, sua evolução tem se mostrado constante e sua aplicação tem apresentado resultados satisfatórios quando aplicados corretamente desde seu surgimento e na evolução de suas características iniciais até as suas facetas mais recentes. Algumas dessas características como, por exemplo, a integração de profissionais voltados para a manutenção de uma cultura ágil ou a integração de softwares de suporte ao desenvolvimento com funcionalidades que auxiliem a propagação de conceitos, as quais são o foco das metodologias tratadas neste trabalho, podem ser observadas no trabalho de Branislav Ruzicky (2018).

Os métodos ágeis e suas práticas intrínsecas abrangem as atividades de todo o fluxo de criação de um software. Essas atividades tendem, quando não abordadas de forma correta, a se tornarem dispersas e sem uma pessoa totalmente responsável dentro de tantas tarefas existentes no desenvolvimento de soluções de software. Assim, desenvolver um modelo de processo de software que explicite um ciclo de atividades claro, com papéis determinados para os integrantes de um time que mitigue a dispersão de informação e que possua escopo claro para um membro da equipe se torna importante, ainda mais quando contextos ágeis e/ou projetos com requisitos voláteis estão em jogo.

3 Trabalhos Relacionados

No mercado, existem diversas plataformas que buscam aprimorar o desenvolvimento de tarefas de projetos de software através de controle de atividades. Neste capítulo, são abordados trabalhos que contextualizam e abordam temas que contribuem para diversos aspectos utilizados no projeto aqui desenvolvido. Estes temas podem variar desde a escolha de uma abordagem satisfatória para o desenvolvimento de uma plaicação com características únicas até a visão de profissionais sobre a adoção de metodologias de controle de software guiadas por ideias ágeis. Suas respectivas características, vantagens e desvantagens em comparação com a proposta deste trabalho.

Este capítulo está estruturado conforme descrito a seguir. As Seções 3.1 até 3.5 descrevem os trabalhos que contribuíram para criar base de comparação e de estudo para este projeto. Esses trabalhos abordam os temas centrais abordados por este estudo e foram selecionados por terem alguns pontos em comum com esta proposta. Na Seção 3.6 é apresentada uma visão comparativa entre os trabalhos correlatos e este projeto.

3.1 Estudo Empírico sobre Desenvolvimento Multiplataformas para Dispositivos Movéis

O trabalho produzido por Biørn-Hansen é um estudo contextual que demonstra o estado da arte do desenvolvimento multiplataforma de produtos de software no âmbito do desenvolvimento móveis (BIøRN-HANSEN et al., 2019). Os principais objetivos do trabalho são amostrar a popularidade, a taxa de adoção e os problemas que surgem ao adotar metodologias de desenvolvimento em contextos ágeis, multiplataforma e, nesse caso, com foco em dispositivos móveis. Como um dos objetivos, está a criação de um método e/ou padrão de questionário que ajude o profissional que trabalha com o gerenciamento de produtos de software a elucidar dúvidas na transcrição de pedidos que vem da necessidade do cliente para histórias do usuário e tarefas dentro de um projeto.

A principal motivação do trabalho é a busca da sugestão de que existe uma lacuna entre abordagens que prezam por métricas qualitativas e quantitativas no desenvolvimento de software que se aplicam às diversas plataformas em uso. Assim, a atenção desse trabalho é voltada para o desenvolvimento de aplicativos móveis com ênfase no que chama de desenvolvimento multiplataforma. Além disso, um questionário é proposto para mostrar como os desenvolvedores veem, utilizam e gerenciam os projetos em que trabalham dentro das perspectivas nativa e multiplataforma e como diferentes abordagens impactam nos

entregáveis[1], manutenção de documentação, reuso de codificação e mapeamento de tarefas.

De acordo com Biørn-Hansen et al. (2019), existem como pontos positivos para a escolha da abordagem, como:

- Integração de reuso de código quando as plataformas compartilham arquitetura de linguagens;
- Relativa facilidade para integrar testes;
- Facilidade de encontrar ajuda em comunidades externas como fóruns destinados ao levantamento de dúvidas e exposição de ideias ou erros encontrados no uso das plataformas.

Por outro lado os pontos negativos apresentados podem servir como indicador das lacunas tanto na criação de uma abordagem correta para a integração de plataformas, bem como para a construção do questionário utilizado como método de pesquisa. Como pode ser visto nos itens que serão descritos a seguir, os pontos negativos apresentados são de grande impacto no desenvolvimento do projeto:

- Perda de desempenho técnico que é observado como o aumento de tempos de respostas e de *loadings*. Estes podem ser atribuídos a uma integração de plataformas não otimizada;
- Integração de aspectos visuais longe de um estado ótimo. Nesse caso, comportamentos de componentes em tela podem variar e a definição clara de como as plataformas vão interagir com esses componentes é necessária para não gerar conflitos;
- *Frameworks* em geral são imaturos para questões de integração da documentação gerando dificuldade na transmissão de conhecimento, ganho de aprendizado e mapeamento de tarefas.

Em suma, o trabalho apresenta os ganhos e perdas vistas quando se trabalha em sistemas multiplataforma e que necessitam de interação constante entre pessoas desenvolvedoras e pessoas que analisam e realizam entregas. Além disso, nota-se a falta de proposta para uma solução dos problemas encontrados no projeto como, por exemplo, a dificuldade em analisar pedidos que não passam diretamente por dados visíveis na interface de acesso do usuário. Assim, o trabalho se resume ao levantamento de ideias, opiniões e conclusões sobre métodos e avaliações disponíveis no mercado juntamente com uma proposta de questionário para ajudar no levantamento de requisitos. A proposta deste

[1] Entregáveis de um processo de desenvolvimento de software são artefatos, funcionalidades ou até resultados teóricos obtidos ao final de um período de tempo determinado para a conclusão de uma atividade (HEAGNEY, 2016).

trabalho é complementar ao trabalho descrito nesta seção, pois pretende apontar melhores práticas de distribuição de tarefas ao longo do desenvolvimento de software e atribuindo atividades para diversos membros do time para que estes participem em momentos chave do processo.

3.2 Desafios no Desenvolvimento Ágil em Larga Escala

O trabalho de Saeeda, Ahmad e Gustavsson (2023) aborda a maneira como os métodos ágeis obtiveram sucesso dentro de indústrias de pequeno e médio porte, abrindo portas e desburocratizando processos que em tempos passados eram tidos como rígidos e sem abertura para mudanças. Visando ampliar a visão para a adoção de metodologias ágeis e os tipos de suporte oferecidos a essas metodologias para as indústrias de grande porte, os autores apontam como objetivos a necessidade de avaliar questões como: (a) complexidade de tarefas; (b) tempo de entendimento das ferramentas; e (c) a aplicação de ferramental teórico para estruturar um projeto, as quais representam grandes desafios na indústria de software, tanto para gestores como para desenvolvedores. O trabalho demonstra por meio de estudos de caso em duas empresas e diversas entrevistas como a aderência dos métodos ágeis tem crescido na medida em que as equipes se tornam cada vez mais diversas. Além disso, mostra como as equipes se tornam cada vez mais eficientes quando trabalham com a ajuda de ferramentas de suporte contínuo.

Diante desse contexto, o trabalho consegue apontar como pontos positivos os seguintes tópicos:

- Melhoria na rotação de membros de uma equipe;
- Maior número de iterações dentro de um *backlog* estruturado;
- Maior alinhamento entre equipes seguindo ritos ágeis[2].

Por sua vez, pontos negativos significativos podem também ser identificados nesse trabalho, os quais destacam lacunas bastante familiares quando se trata da inserção de métodos ágeis em ambientes com projetos legados:

- Falta de estratégias de teste que se alinhem com a linha de tempo ágil;
- Caos na execução de *sprints*, principalmente em um primeiro momento de mudança;
- Perda de referências para prazos definidos de maneira não ágil;

[2] Ritos ágeis são configurados como cerimônias padrões a serem realizadas em projetos onde existe a aplicação de metodologias ágeis (SILVA; SELBACH; MAURER, 2015). Exemplos de ritos são as reuniões de planejamentos e as retrospectivas de *Sprint*.

- Perda de qualidade na obtenção e refinamento de requisitos.

Finalmente, o trabalho apresenta como resultados o levantamento dos principais pontos de estresse dentro da aplicação de um método ágil e tenta mostrar através de uma discussão geral como os pontos críticos podem ser contornados. Assim, este projeto está alinhado com o trabalho de Saeeda, Ahmad e Gustavsson (2023) quando surgem questões sobre a mitigação de problemas relativos a distribuição de tarefas e sobre a participação de integrantes da equipe em diferentes momentos de um período de desenvolvimento. A boa estruturação de períodos de desenvolvimento e a manutenção da qualidade dos requisitos por meio de boas práticas de elicitação e definição de ritos mandatários para o desenvolvimento também serão tratadas na proposta deste projeto.

3.3 Práticas, Restrições e Adoção de Práticas de Software em Pequenas Empresas

Small Software Companies (SSCs) são um modelo de empresas que possuem como uma de suas principais características a interação direta e sem camadas com o cliente e a comunicação direta e efetiva na maioria dos casos. Quando apresentadas a um mercado cada vez mais globalizado e com desafios que requerem práticas que são cada vez mais distantes de sua realidade, tais empresas podem enfrentar desafios para fazer os ajustes necessários. O trabalho de Tuape et al. (2022) aborda um tema mais refinado e com aplicações diretas ao suporte de desenvolvimento em empresas de pequeno porte. Assim, por meio de questionários e entrevistas, o trabalho busca levantar quais as principais dificuldades que as empresas enfrentam na adoção dos métodos ágeis, que muitas vezes é a saída mais indicada para a atualização das práticas de trabalho das SSCs. Como pontos positivos do trabalho podem ser citados:

- Melhoria do engajamento dos clientes e *stakeholders*;
- Diminuição do retrabalho em tarefas em consequência do maior detalhamento e maior interação entre envolvidos;
- Questionário abrangente em termos de questões sobre entregas, qualidade, documentação e envolvimento de equipe.

Ainda, como pontos negativos, podem ser observados:

- Falta de uma solução prática ou opções proveitosas para os desafios encontrados quando tais não foram solucionados com a implementação dos métodos ágeis;
- Amostra de pessoas participantes em apenas uma cultura de empresa;

- Apesar de apresentar ganhos em termos de engajamento com os colaboradores, o trabalho ainda peca ao definir como os ritos ágeis devem ser definidos e como a fixação de tais poderia gerar ganho.

O trabalho de Tuape et al. (2022) entrega como contribuição final um estudo para uso direto que gera engajamento direto dos colaboradores e que busca transformar práticas de pequenos negócios em atividades mais robustas e à prova de interferências do ambiente. Além disso, como consequência, gera maior atenção aos pedidos do cliente em relação as práticas que costumam ser fixas para empresas que estão nesse porte. O trabalho também mostra como a implementação de novas práticas pode ser uma forma de elevar a qualidade do software criado, o que pode levar as empresas para novos patamares financeiros e de qualidade.

3.4 Entendendo Software Colaborativo por meio de Entrevistas

O trabalho de Constantino et al. (2020) apresenta uma visão de como abordagens colaborativas podem ser vantajosas em relação ao desenvolvimento de forma isolada. Ao se afastar de abordagens mais comuns, as quais focam em modelos de colaboração e ferramentas de suporte, o trabalho apresenta as motivações para um desenvolvimento colaborativo. Além disso, também apresenta os desafios de se criar um ambiente que sustente esse processo e como acontece a colaboração entre equipes e indivíduos. Em um mundo de equipes de desenvolvimento espalhadas por diversas localizações, se faz necessário o refinamento de práticas de documentação e colaboração entre todos os envolvidos em um projeto. Como forma de abordagem e coleta de dados, foram realizadas entrevistas com diversos desenvolvedores e *stakeholders* que apontaram as diferentes características e preocupações que surgem ao se distribuir um time que colabora em prol de um projeto.

O trabalho apresenta pontos positivos interessantes, tais como:

- Apresentação clara das diferentes contribuições que são feitas por diferentes membros de uma equipe;
- As principais mudanças necessárias de ambiente e *mindset* em um projeto;
- Apontamento das dificuldades não técnicas atreladas ao processo.

Ainda, como pontos negativos, são apresentados:

- A falta da apresentação de soluções que auxiliem a solucionar os problemas além das características técnicas de um projeto;

- O projeto foca bastante no indivíduo através de seu questionário aplicado, como consequência, uma visão estrutural com viés focado na equipe ou empresa na qual o indivíduo está inserido é deixado de lado.

O trabalho de Constantino et al. (2020) se relaciona com este trabalho pois aborda temas como: (1) o desenvolvimento com colaboração; (2) a temporização de tarefas; (3) a dissolução de tarefas em tarefas mais palpáveis, ou seja, que sejam de menor complexidade e de escopo mais simples; (4) a documentação de conhecimento sobre o projeto; e, por fim, (5) o subsequente compartilhamento de conhecimento sobre o projeto. A partir desse ponto, este projeto pode complementar o trabalho de Constantino et al. (2020) ao propor uma estrutura de documentação, compartilhamento e estruturação de tarefas e artefatos de um projeto de software.

3.5 Metodologias Híbridas: Percepções de Praticantes do Scrum

O trabalho de Feitosa Ivan Machado (2023) visa processos de software ágeis e orientados à planos podem ser utilizados em paralelo com o Scrum para criar abordagens que atendam a diferentes modelos de negócios e dar maior abertura para possibilidades de caminhos e estratégias a serem utilizadas dentro do ciclo de desenvolvimento de um software. A partir de um questionário, são levantadas questões sobre a definição de propostas híbridas de desenvolvimento de software. O trabalho apresenta, juntamente com os entrevistados, visões de como tais abordagens híbridas podem usar da flexibilidade do Scrum para tirar melhor proveito de abordagens orientadas a planos de longo prazo. Além disso, são discutidos pontos como a documentação de um projeto híbrido e como o uso de metodologias orientadas à planos são vistos por diferentes partes interessadas em um projeto. Por fim, o trabalho entrega diversas opções de como essa mescla de abordagens pode ser feita e define de forma geral como o processo de uso híbrido de metodologias pode ser interpretado e aplicado, mostrando seus benefícios.

O trabalho entrega pontos positivos relevantes, a saber:

- Abertura e discussão clara de ideias para ajudar a melhorar o processo de integração de metodologias de desenvolvimento;
- Boa visão sobre a tradução da literatura para aplicação de conceitos na prática.

Além disso, também podem ser apontados pontos negativos conforme as descritas a seguir:

- Falta de exploração da visão do cliente sobre o impacto das metodologias híbridas nas entregas realizadas;

- Falta apresentação de soluções para uma das principais dificuldades encontradas, que é a abordagem de tarefas com complexidade elevada.

Por fim, este trabalho se conecta ao trabalho de Feitosa Ivan Machado (2023) pelo fato de ambos abordarem a necessidade de se padronizar processos e organizar as estruturas de projetos ágeis, priorizando e dividindo atividades, além de mapear de maneira direta as tarefas de um projeto, o que pode ajudar na diminuição da complexidade de entendimento e divisão de atividades.

3.6 Considerações Finais

Os trabalhos relacionados descritos neste capítulo formam a base de conhecimento sobre o estado atual do desenvolvimento de software com as plataformas de apoio e a integração de métodos ágeis em ambientes que passam por transformações constantes.

A importância de mapear os trabalhos relacionados é grande, dado o contexto do trabalho atual, pois dá uma visão mais ampla para mostrar que é possível aplicar metodologias ágeis em processos de desenvolvimento de software. Isso mostra que é possível integrar equipes que utilizam outras metodologias às práticas ágeis e que as práticas ágeis podem ser um catalisador de melhorias em entregas, tornando-ás mais robustas, mapeáveis e com maior integração entre a equipe. Além disso, é possível notar em todos os pontos citados, como o mapeamento de atividades, integração do time de desenvolvimento e aplicação de metodologias ágeis as possível falhas que podem ocorrer, quando esses conceitos são inseridos sem a preparação correta. A adaptação e o entendimento a uma nova metodologia de entregas, a comunicação e a dispersão de conhecimento dentro de um ambiente em constante mudança pode ser algo que venha a gerar problemas para desenvolvedores, gerentes e *stakeholders*.

A Tabela 1 apresenta um resumo sobre os objetivos, aplicações, vantagens e desvantagens dos trabalhos relacionados. Essa tabela apresenta as seguintes informações:

- **Finalidade e Aplicação**: apresenta os objetivos e contexto estudado pelo trabalho;
- **Técnicas Utilizadas**: apresenta a abordagem utilizada pelos colaboradores para identificar, estudar e criar soluções para o tema em foco;
- **Vantagens**: apresenta as principais características positivas dentro da abordagem utilizada assim como os principais ganhos encontrados no ataque ao tema em foco;
- **Desvantagens**: apresenta os pontos em que o trabalho em foco deixa de se aprofundar ou as facetas do trabalho poderiam ser apresentadas e exploradas de maneiras mais clara ou com resultados mais claros e significantes.

Tabela 1 – Resumo dos trabalhos relacionados.

Trabalho	Finalidade e Aplicação	Técnicas Utilizadas	Vantagens	Desvantagens
Bjørn-Hansen et al. (2019)	Estudar e levantar adversidades em desenvolvimento multi-plataforma.	Aplicação de questionário que aborda os temas relevantes ao estudo.	Integração entre plataformas, relativa facilidade para integrar testes e suporte em abundância.	Perda de desempenho considerável, baixa integração de aspectos visuais e integração de documentação.
Saeeda, Ahmad e Gustavsson (2023)	Estudo, avaliação e aplicação de metodologias ágeis pareadas com softwares de apoio em contextos de software em larga escala.	Estudo de casos de empresas que aplicaram metodologias ágeis como base de processos de desenvolvimento.	Melhoria na rotação de membros de uma equipe, maior número de iterações dentro de um *backlog* estruturado e maior alinhamento entre equipes seguindo ritos ágeis.	Caos na execução de *sprints*, definição de prazos conturbada, perda de qualidade na obtenção e refinamento de requisitos.
Tuape et al. (2022)	Aplicação de métodos ágeis com foco em restrições e características de empresas de software de pequena escala.	Aplicação de questionário e realização de entrevistas com colaboradores inseridos em ambientes que receberam as mudanças abordadas no trabalho.	Melhoria do engajamento, diminuição do retrabalho e maior detalhamento e Questionário abrangente.	Falta de uma solução prática para os desafios encontrados, pequena amostra de pessoas participantes e o trabalho peca ao definir como os ritos ágeis geram ganho.
Constantino et al. (2020)	Levantamento e Avaliação de mudanças de práticas e atividades em empresas de software que adotam metodologias que permitem desenvolvimento colaborativo.	Aplicação de questionário e realização de entrevistas com colaboradores inseridos em contextos de desenvolvimento colaborativo.	Apresentação clara das diferentes contribuições de uma equipe, apresentação das mudanças de práticas e apontamento das dificuldades não-técnicas.	Falta da apresentação de soluções que auxiliem a solucionar os problemas não-técnicos, foco restrito no indivíduo, falta de visão estrutural na equipe ou empresa.
Feitosa Ivan Machado (2023)	Avaliar a aplicação de metodologias orientadas a plano juntamente com métodos ágeis em contextos de desenvolvimento de software.	Aplicação de Questionário e avaliação das respostas obtidas.	Proposta de melhorias palpáveis, boa correlação entre literatura e métodos aplicados.	Visão unilateral dos impactos e visões, falta de soluções para distribuição de tarefas complexas.
Plan and Go	Modelar um fluxo de controle para atividades de software ágeis.	Modelagem e estudos de casos através de aplicação direta.	Abrangência de visões sobre um período de desenvolvimento e fluxo de requisitos refinado.	Necessidade de maior visão dos entregáveis e de sua construção além dos desenvolvedores.

Além das informações dos trabalhos relacionados, a Tabela 1 apresenta informações deste trabalho, sendo assim, indica as suas principais características e a sua abrangência dentro dos contextos abordados por outros trabalhos. Assim, este projeto pretende tratar sobre: (*i*) modelos de documentação de requisitos dentro de um período de desenvolvimento; (*ii*) a criação de um ambiente que controle a as atividades já firmadas para codificação; e, por fim, (*iii*) a facilitação de obtenção de novos requisitos, planejando quando e como esses novos requisitos serão levantados e refinados. Desse modo, espera-se que seja gerado um fluxo de desenvolvimento mais controlado que não gere retrabalho, mudança de requisitos e que apresente prazos mais aceitáveis.

4 Metodologia de Desenvolvimento

Este capítulo apresenta o desenvolvimento do trabalho, que começa pela definição dos principais conceitos a serem aplicados, seguido pela definição de parâmetros a serem utilizados, a idealização do fluxo de software e finaliza com a construção e apresentação do protótipo que demonstra de forma visual o fluxo de uso do processo desenvolvido.

Este capítulo está estruturado conforme descrito a seguir. A Seção 4.1 descreve como são vistas as principais características de um período de desenvolvimento de software e como os blocos de tarefas necessários podem ser alocados. Em seguida, nas Seções 4.2, 4.3 e 4.4 são abordados os papéis assumidos pelos integrantes base de uma equipe de desenvolvimento. Por sua vez, a Seção 4.5 mostra como podem ser geradas diferentes documentações a partir das etapas de desenvolvimento de uma solução de software e, por fim, a Seção 4.6 concatena as ideias propostas neste capítulo para a definição de um modelo de processo em que os atores tenham clareza de sua atuação no desenvolvimento de software.

4.1 Configuração de *Sprint*

A definição das configurações de *sprint* tem impacto direto na velocidade evolução progressiva de um projeto de software (ASGHAR, 2016). Desse modo, podem ser destacados alguns aspectos que podem ser isolados e estudados para que a escolha dos parâmetros corretos que maximizem a agilidade e os fluxos dos planos de ação tomados durante o período de desenvolvimento. Os principais aspectos identificados e sua delimitação para o projeto seguindo referências em trabalhos correlatos podem ser observador a seguir:

- **Duração do total da *sprint***: segundo Asghar (2016) um processo de desenvolvimento de software bem planejado tem o poder de ajudar a manter e elevar a qualidade e padrões definidos como base para o produto desejado. Na aplicação desses processos dentro de metodologias ágeis como Kanban e Scrum é sempre visto como um *design* ótimo a avaliação de duas principais características para definir duração de *sprints* e períodos de desenvolvimento em geral. Como pode ser visto no trabalho de Asghar (2016), as duas características que mais influenciam a definição da duração de uma *sprint* são:
  - **O custo atrelado à tarefa**: Uma tarefa pode ser estimada em custos de acordo com seu tamanho ou complexidade. O tamanho de uma tarefa na maioria das vezes é dado pela estimativa de tempo que um analista técnico provê ao analisar todas as atividades necessárias para entregar tal tarefa com 100% de suas

funcionalidades em funcionamento. Já a complexidade em geral é estimada de acordo com a quantidade de pessoas necessárias que serão envolvidas, a quantidade de dados a serem trabalhados e a quantidade de processamento que o sistema deve realizar para entregar os resultados desejados.

Metodologias ágeis em geral buscam dividir para conquistar em relação a divisão de atividades, logo, períodos de desenvolvimento extensos não são esperados. Com a diminuição dos períodos de tempo para desenvolvimento, como consequência o particionamento de tarefas muito complexas ou de tamanho elevado.

O uso de sistemas de pontuação de atividades como visto por Dubois (2013), pode ser de grande ajuda para parametrizar a definição de limites para complexidade e tamanho de tarefas a serem incluídos em uma *sprint*. Esses sistemas tem como base a definição de valores chave para cada tipo de atividade em cada módulo de desenvolvimento que a atividade tem ações planejadas. Após a estimativa técnica da atividade ser realizada e dada pelos desenvolvedores ou gerentes técnicos, é realizada a soma dos pontos atribuídos para uma tarefa, caso tal soma ultrapasse um valor delimitado como limite para uma única tarefa, é visto como necessária a reestruturação e/ou divisão de tal atividade em sub-atividades que tenham uma melhor distribuição;

  - **O ritmo de entregas definido**: a entrega constante de valor é um dos pilares das metodologias ágeis. Dar visibilidade para o cliente sobre a evolução da ferramenta é primordial para gerar *feedback* constante e abrir as portas para processos de *discovery* que irão agregar no crescimento do software. Logo, a definição de metas e *deadlines* para realizar entregas são fator determinante no momento de estimar a duração de uma *sprint*;

- **Validação técnica de requisitos**: segundo Asghar (2016) é necessária a existência de um período dentro dos prazos de desenvolvimento que seja destinado a avaliação e encaixe de novos requisitos dentro do contexto do projeto, esse período pode ser identificado e particionado de tal forma:

  - **Validação de requisitos levantados**: avaliação dos novos requisitos pela equipe de produto buscando entendimento de como e quando as necessidades levantadas se encaixam no contexto do projeto;
  - **Avaliação de *backlog***: busca encontrar o melhor contexto e conjunto de funcionalidades em que um novo requisito se encaixa;
  - **Priorização de funcionalidades**: a equipe de produto busca elencar o nível de prioridade sobre outras tarefas que o novo requisito impõe, além de identificar atividades que devem obrigatoriamente ser desenvolvidas antes ou depois do requisito em foco;

- **Período de estudo de requisitos**:
  - **Análise por parte do desenvolvedor**: o desenvolvedor de software tem a possibilidade de avaliar os requisitos levantados e seus respectivos objetivos. A partir dessa avaliação, o desenvolvedor consegue ser parte atuante no processo gamificação que indicará os níveis de complexidade e tamanho das atividades de desenvolvimento necessárias;
  - **Levantamento de dúvidas e soluções possíveis**: após a análise dos artefatos anteriores, o desenvolvedor consegue com essa gama de conhecimento em mãos, traçar um plano de ação que inclui uma linha do tempo e a discretização das atividades necessárias para obter sucesso na criação dos entregáveis que compõem o requisito em destaque;
- **Desenvolvimento**:
  - **Criação de subtarefas**: esse passo requer atenção individual dos colaboradores. A criação das tarefas em um software de acompanhamento de desenvolvimento se faz crucial para que as estimativas de tempo que cada tarefa possui para sua conclusão se encaixem em uma linha do tempo coesa e que faça sentido para todos os integrantes. Uma descrição e estimativa correta de tarefas gera maior entendimento dos processos de criação e integração de artefatos desenvolvidos;
  - **Documentação de código**: em paralelo ao desenvolvimento dos artefatos de código, se faz necessária a documentação dos blocos de informações criados. Uma boa documentação de software gera menor curva de aprendizado para terceiros além de ser fonte de informação para implementações futuras;
- **Aplicação testes de software**:
  - **Estudos dos requisitos definidos**: a fim de criar um plano de teste que seja abrangente em todos os aspectos da solução desenvolvida é necessário que a equipe de qualidade realize um estudo sobre a funcionalidade proposta e que tal seja analisada em seus pontos funcionais e de integração com o sistema já existente. Assim, os testes criados terão cobertura ampliada e melhorias sobre o requisito já levantado podem ser apontados perante a realização dos testes;
  - **Criação de planos de teste**: em geral, a criação do plano de teste é realizada com ajuda de software próprio integrado ao ambiente de desenvolvimento, a criação dos testes deve ser totalmente integrada ao sistema em funcionamento e busca encontrar falhas estruturais e replicar o uso do sistema com a nova funcionalidade inserida;
- ***Deploy* das soluções desenvolvidas**:

– **Alinhamento do ritmo de entregas e *feedback***: com o objetivo de delimitar os entregáveis necessários e acordados o ritmo das entregas deve ser alinhado buscando estar inserido dentro da extensão de *sprint* e, por consequência, os *feedbacks* e possíveis melhorias recebidos devem ser alocados como tarefas futuras também dentro de um contexto de tempo que compreenda uma *sprint* de desenvolvimento e análise.

A definição do tempo de *sprint* ideal como descrita por Saeeda, Ahmad e Gustavsson (2023) pode variar de 1 a 4 semanas, dependendo das variáveis citadas. Ao procurar determinar um balanço ideal entre complexidade, agilidade e flexibiliza no período de levantamento de requisitos futuros, desenvolvimento e testes é visto como um número ideal a duração de *sprint* se estendendo por duas semanas ou 10 dias trabalhados (na maioria das empresas) (SAEEDA; AHMAD; GUSTAVSSON, 2023). Esse período de tempo é identificado como o *sweet spot* de duração pois dá liberdade e flexibilidade para criação e desenvolvimento de tarefas, além de sempre existir tempo hábil para alinhamentos e mudanças em planos de ação que venham a ocorrer. O que vai de encontro ao objetivo dos métodos ágeis que buscam prover ambientes aptos a mudanças em seus fluxos.

A *sprint* de estudo é definida por Branislav Ruzicky (2018) como o período dentro da *sprint* atual em que requisitos em levantamento passam por um processo de discussão, avaliação de prototipação e validação com usuário. A *sprint* de estudo busca levantar dúvidas e impeditivos para que a equipe possa encontrar de maneira prévia ao desenvolvimento, tratativas que solucionem possíveis obstáculos. Dentro dessa *sprint*, podem ser identificados os papéis e atividades desenvolvidas por cada membro da equipe, o que torna, dessa forma, sua participação mais clara.

A partir dos conceitos estabelecidos anteriormente, agora pode ser definido um um *roadmap* de atividades necessárias para que uma *sprint* seja preparada e executada com segurança, agilidade e qualidade. A Figura 1 mostra um *roadmap* ideal que tem como objetivo preparar os temas a serem trabalhados em uma *sprint* de estudo, definir os métodos de solução dos problemas levantados, documentar, desenvolver, testar e entregar as funcionalidades atacadas.

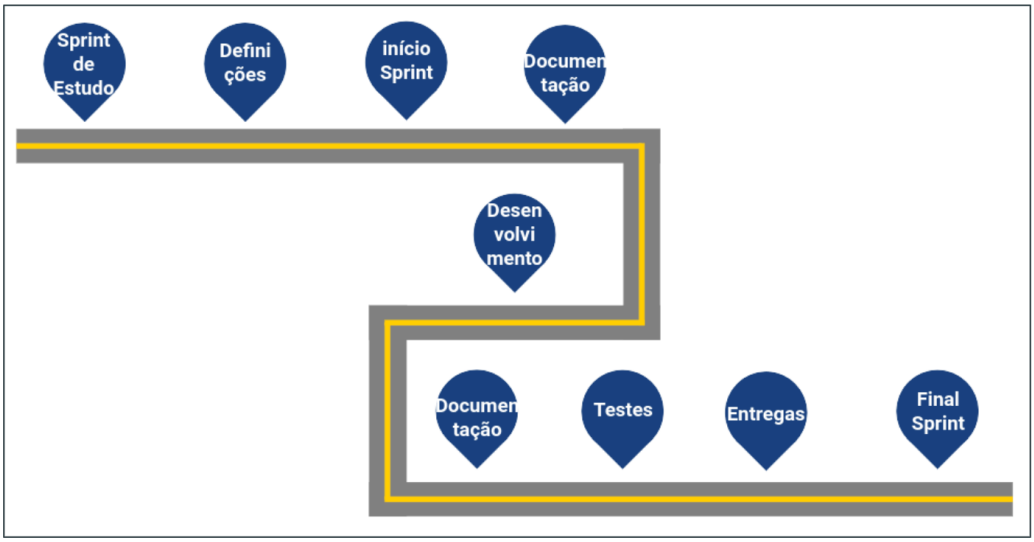

Figura 1 – *Roadmap* de planejamento, execução e entrega de *sprints*.

4.2 Papel e Atuação do Desenvolvedor

Dentro do fluxo configurado para a *sprint* como um todo, o desenvolvedor irá atuar nas seguintes etapas e das seguintes formas:

- **Período de análise de requisitos**: análise do problema, do requisito levantando como solução e de sua complexidade;
- **Validação de requisitos levantado**s: Verificação das soluções apresentadas e de sua completude;
- **Criação de subtarefas**: análise das atividades necessárias e inserção de tais em uma plataforma de acompanhamento;
- **Integração entre módulos**: criação dos artefatos necessários para interligar diferentes bloco de implementação;
- **Aplicação dos planos de teste**: aplicação dos testes criados pela equipe de qualidade para verificação da cobertura provida pela implementação base;
- **Correção de *bugs***: a correção de *bugs* reportados após testes do desenvolvedor e da equipe de qualidade.

Dentro desse fluxo, o desenvolvedor é visto como o ponto central e deve estar envolvido em atividades que preparam, desenvolvem e certificam a qualidade de um entregável. Segundo Heeager (2009), o entendimento completo de um requisito e dos artefatos por parte do desenvolvedor faz com que sejam prevenidos *bugs*, mudanças de abordagens em desenvolvimento e que a cobertura de uma solução seja a maior possível resultando em menor possibilidade de atividades futuras de correção e melhoria.

A Figura 2 apresenta o diagrama que mostra como o desenvolvedor atua dentro do fluxo de levantamento de requisitos até a entrega da solução pensada com destaque para as suas áreas de atuação. O diagrama foi criado utilizando a ferramenta Microsoft Visio com base em um de seus modelos de fluxo de atividades.

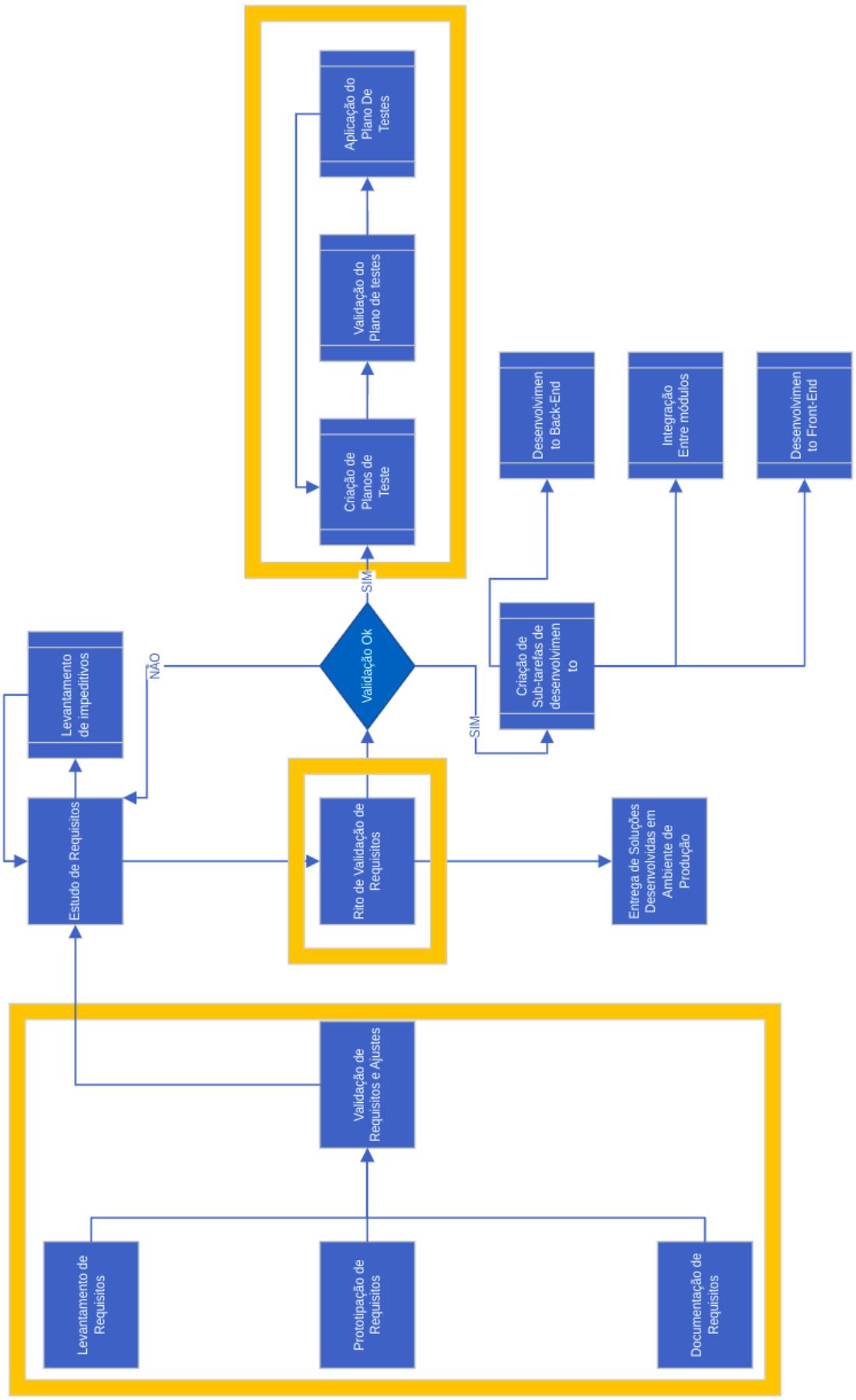

Figura 2 – Apresentação do processo de software com destaque nos papéis de *design*.

4.3 Papel e Atuação da Pessoa de Design

Dentro do fluxo configurado para a *sprint* como um todo, o desenvolvedor irá atuar nas seguintes etapas e das seguintes formas:

- **Levantamento de dúvidas e soluções possíveis**: levantamento de possibilidades de solução para um problema ou requisito a ser trabalhado, essa abordagem pode ser feita por meio de diversos métodos, como, por exemplo, a prototipação em baixo nível;
- **Validação de requisitos levantados**: verificação da abrangência e possibilidade de implementação de um protótipo de solução encontrado;
- **Priorização de funcionalidades**: juntamente com a equipe de produto, nesse ponto, a equipe de *design* avalia a ordem e prioridade de inserção de uma funcionalidade em um *backlog sprint*;
- **Validação de *bugs***: avaliação dos entregáveis criados com foco na análise visual e sua aderência à proposta inicialmente apresentada.

A Figura 3 apresenta o diagrama que mostra como a pessoa responsável pelo levantamento de propostas de *design* atua dentro do fluxo de levantamento de requisitos até a entrega da solução planejada com destaque para suas áreas de atuação. O diagrama foi criado utilizando a ferramenta Microsoft Visio com base em um de seus modelos de fluxo de atividades.

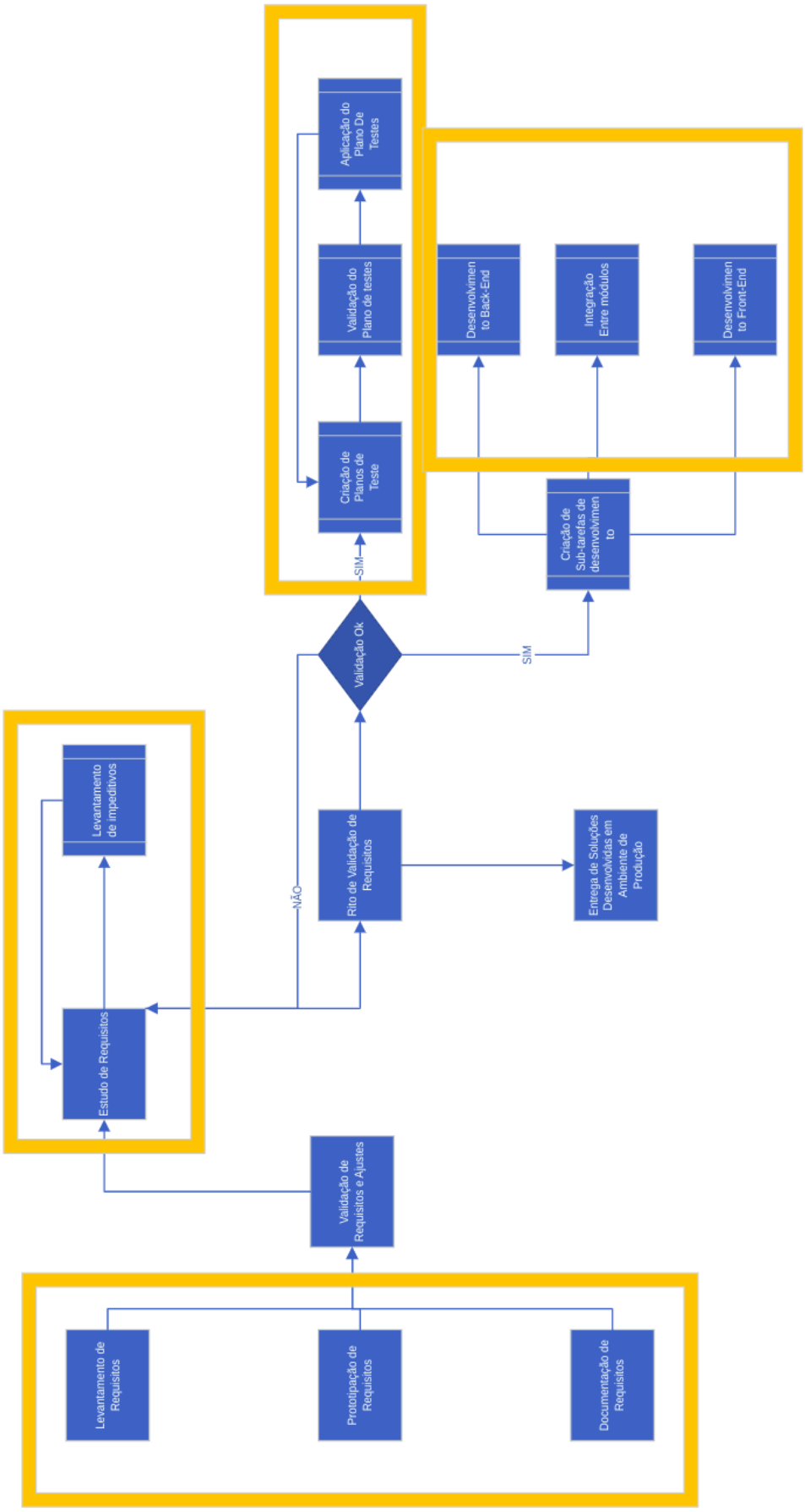

Figura 3 – Apresentação do processo de software com destaque para os papéis de desenvolvimento.

4.4 Papel e Atuação da Pessoa de Produto

Deve-se definir como o analista de requisitos, *Product Owner* ou *Product Manager*[1] atuam em cada etapa descrita para a estruturação da *sprint* quais seus entregáveis e qual seu papel na estruturação de novas *sprints* de maneira técnica conforme descrito a seguir:

- **Levantamento de dúvidas e soluções possíveis**: levantamento de possibilidades de solução para um problema ou requisito a ser trabalhado, essa abordagem pode ser feita por meio de diversos métodos, como, por exemplo, por meio de entrevistas ou observação de contextos;
- **Validação de requisitos levantados**: verificação da abrangência e possibilidade de implementação de um protótipo de solução encontrado.
- **Priorização de funcionalidades**: juntamente com a equipe de *design*, aqui a equipe de *design* avalia a ordem e prioridade de inserção de uma funcionalidade em um *backlogsprint*;
- **Criação de subtarefas**: a equipe de *design* realiza a inserção das tarefas necessárias para construir e validar os protótipos de interface que cobrem o requisito em foco;
- **Alinhamento do ritmo de entregas e *feedback***: definição de reuniões e alinhamentos que pautam datas e cronogramas de entregas, tais definições são enquadradas dentro do período de *sprint* anteriormente detalhado.

A Figura 4 apresenta o diagrama que mostra como a pessoa responsável pelo levantamento e documentação de requisitos é vista dentro do fluxo de levantamento de requisitos até a entrega da solução pensada com destaque para as suas área de atuação. O diagrama foi criado utilizando a ferramenta Microsoft Visio com base em um de seus modelos de fluxo de atividades.

[1] *Product Owner* e *Product Manager* são, respectivamente, segundo Oikkanen, Hyrynsalmi e Paasivaara (2023) responsáveis pela manutenção de escopo de projeto e realização de entregas em prazos delimitados dentro de um projeto de *software*

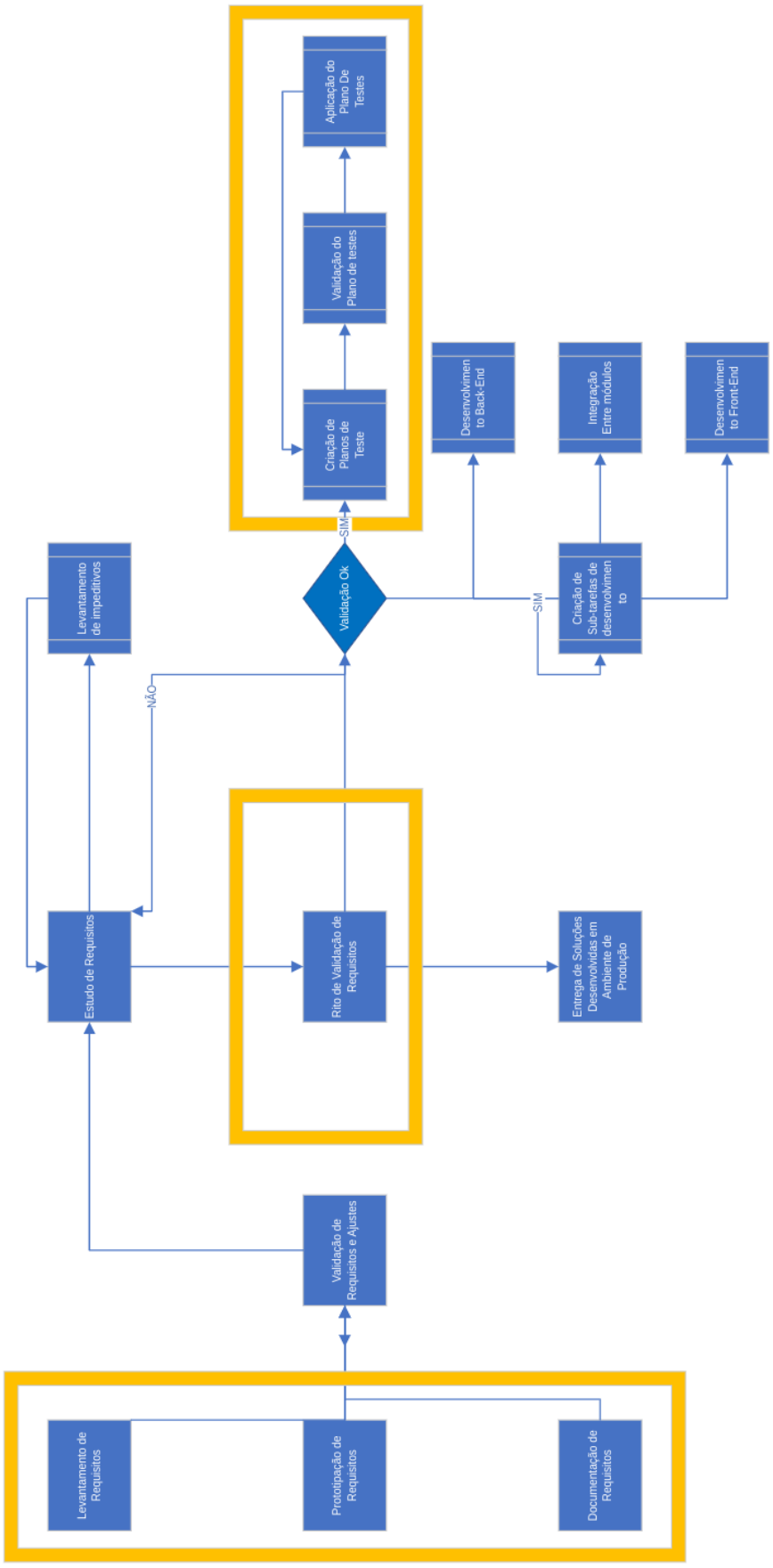

Figura 4 – Apresentação do processo de software com destaque nos papéis de levantamento e refinamento de requisitos.

4.5 Documentação

Segundo Heeager (2009), as etapas de desenvolvimento devem intrinsecamente gerar documentos que descrevam os passos a fim de imputar ganho e agregar conhecimento a um projeto de software. Perante a atuação de cada ator dentro das fases de desenvolvimento, podem ser também levantados os documentos gerados para cada etapa do processo de gerenciamento e desenvolvimento de software, dentro das etapas previamente abordadas, podem ser criadas as relações entre a etapa do processo descrita e a documentação gerada. Assim, podem ser relacionadas:

- **Elicitação de requisitos**: devem ser gerados os artefatos iniciais de um requisito levantado com objetivo de descrever as ideias iniciais de um entregável, suas características técnicas e visuais além das regras de negócio criadas e afetadas pelo requisito. Os principais documentos gerados aqui são:
  - **História de usuário base**: é uma documentação de baixo nível sem integração com protótipos ou funcionalidades terceiras, tendo em foco as características dos dados a serem exibidos e tratados, além do fluxo de uso;
  - **Protótipo de Baixa fidelidade**: que descreve as funcionalidades base de uma solução de forma a levantar dúvidas e melhorias em uma ideia inicial;
- **Validação técnica e com cliente de propostas iniciais**: a validação técnica parte do pressuposto de que as regras e limites de um requisitos foram bem definidos, assim, líderes técnicos podem avaliar a possibilidade de desenvolvimento A validação com cliente deve em seguida firmar o entendimento dos artefatos criados e validados previamente com a equipe de desenvolvimento, e fornecer sinal verde para desenvolvimento. Nesse ponto, devem ser gerados os seguintes documentos:
  - **Protótipo de alta nível**: que explora todas as possibilidade de um fluxo de uso real da solução levantada, além de integrar tal interface com todo o sistema já existente;
  - **História de usuário**: validada e integrada com o protótipo finalizado e suas interfaces, além de unir a solução com as funcionalidades existentes no sistema;
- **Rito de validação prévio ao desenvolvimento**: deve ser utilizado como um momento para equipe analisar prazos e definir métricas a serem alcançadas dentro do período de desenvolvimento:
  - Ata de reunião que cita tópicos abordados identificando possíveis obstáculos e ritos necessários para que o desenvolvimento da solução cubra todos os aspectos do requisito delimitado;

- **Plano de teste**: a identificação e criação de testes deve apresentar resultados em um plano de ação para testar e validar as funções e limitações de uma solução desenvolvida:
  - Plano de testes que identifica todos os passos do fluxo de uso, tratamento de dados e fases de *input/output* de dados da solução desenvolvida. Além disso, o plano de testes documentados também deve abordar a interação da nova funcionalidade com o sistema já existente;
- **Documentação da entrega de soluções**: a entrega de soluções desenvolvidas deve ser realizada com o objetivo de demonstrar o uso da nova funcionalidade resultando em documentos que descrevam características observadas no uso do entregável e o nível de satisfação do cliente:
  - Ata de reunião citando e elaborando tópicos sobre a reunião entregue, possíveis melhorias futuras encontradas após uso da solução e requisitos posteriores que complementam a funcionalidade entregue.

Após definir todos os documentos gerados por meio das etapas do processo um fluxo ideal pode ser idealizado onde cada documento gerado deve ser criado e entregue com a finalidade de mapear as importantes descobertas, conhecimentos adquiridos e processos realizados ao longo de um processo de criação de software. A Figura 5 mostra o fluxo de processo definido juntamente com os documentos a serem gerados com o avanço de um requisito.

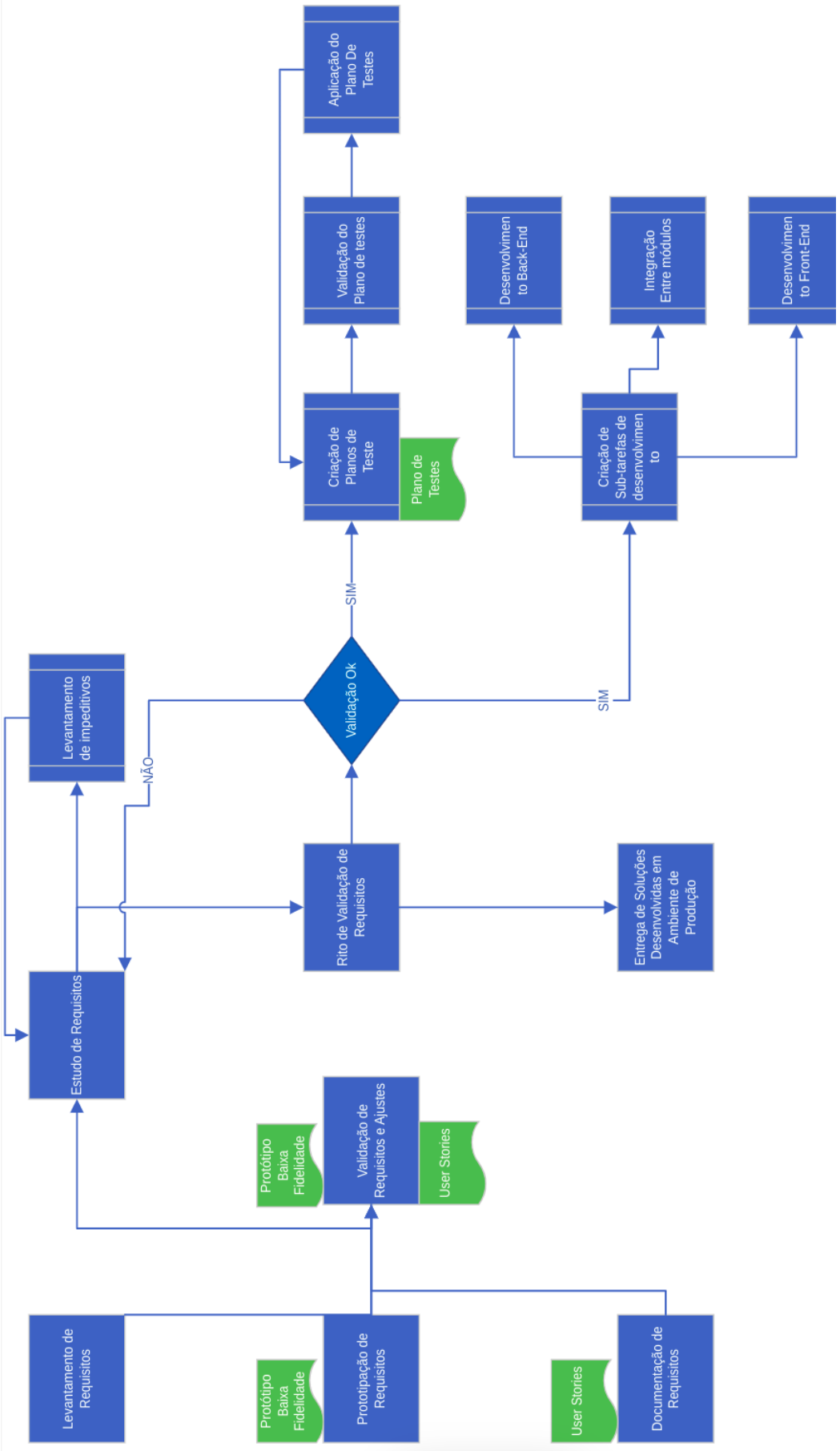

Figura 5 – Fluxo de documentação gerada a cada etapa do processo de levantamento e solução de um requisito.

4.6 Considerações Finais

Considerando as ideias apresentadas pudemos desenvolver aqui as características de um processo onde são consideradas facetas de um projeto de software e que geram clareza sobre os papéis das pessoas atuantes no fluxo de desenvolvimento de um produto de *software*. Com a definição dos papéis pode ser definida uma estratégia de participação e de entregas que variam desde as histórias de usuários parciais, histórias de usuários finalizadas, artefatos de prototipação, testes e documentação necessária.

Mapear os papéis e atividades dos participantes em um processo de desenvolvimento de *software* é de suma importância para que sejam claras e objetivas as ações a serem tomadas por cada um deles. Definir com clareza escopos de participação e entregas gera maior engajamento e interação entre membros da equipe. Como apontado por Branislav Ruzicky (2018), uma equipe que entende seus papéis e entende a necessidade de colaborar constantemente está mais próxima de entregar soluções mais robustas, integradas e com qualidade. Somado a isso, uma definição sensata para o período de desenvolvimento e testes, a equipe passa a viver em um ambiente em que a clareza de objetivos e prazos faz com que todos os integrantes estejam dispostos a contribuir com as soluções buscadas.

5 Estudo de Caso Simulado

Este capítulo apresenta um estudo de caso simulado que é construído a partir de um problema real. Assim, os conceitos e práticas do modelo de processo proposto são aplicados no sentido de apresentar uma possível solução simulada para esse problema, em razão desse fato, neste trabalho esse estudo foi definido como estudo de caso simulado. Além disso, uma análise inicial é realizada para avaliar a tratativa dessa solução. Para exemplificar o uso e a aplicação do fluxo de acompanhamento do projeto descrito no capítulo anterior, aqui também é descrito e analisado o desenvolvimento de um protótipo de software de acompanhamento do levantamento de requisitos, bem como do acompanhamento de tarefas e da documentação de artefatos.

Este capítulo está estruturado conforme descrito a seguir. A Seção 5.1 apresenta o contexto e o problema idealizado para aplicação da solução proposta neste trabalho. Na Seção 5.2 são descritos os principais dados e indicadores a serem considerados no processo de avaliação da solução. Em seguida, a Seção 5.3 mostra como a aplicação do modelo desenhado, juntamente com a visualização de sua aplicação por meio de um protótipo podem ajudar a solucionar o caso em estudo. Por fim, a Seção 5.4 apresenta os resultados esperados e obtidos com a aplicação do fluxo de processo idealizado, além de suas limitações dentro de seu respectivo contexto.

5.1 O Problema e sua Contextualização

Dentro do contexto do trabalho aqui desenvolvido, um dos principais impeditivos existentes é a dificuldade em se manter requisitos levantados e validados em paralelo com as tarefas e documentações necessárias. Dessa forma, isso é necessário para que o requisito seja levado em diante no seu processo de *discovery* e posterior desenvolvimento. Assim, a partir da análise de um projeto real, é possível projetar as soluções propostas nesse capítulo e avaliar o comportamento e os resultados passíveis de serem obtidos.

O projeto aqui explorado é localizado dentro do contexto de uma empresa de desenvolvimento de software mineira situada na cidade de Belo Horizonte. Essa empresa é focada em otimização de processos logísticos e metalúrgicos. Por meio dos parâmetros de entrada naturais ou obtidos através de sistemas terceiros, o software proposto recebe, analisa, processa, otimiza as tarefas e transmite uma cadeia de eventos a ser realizada para cada unidade de demanda recebida. Dessa forma, constitui-se um modelo de software conhecido como “Gêmeo Digital”.

5.2 Principais Dados e Indicadores

Por meio do diagrama da Figura 6, gerado utilizando a ferramenta Microsoft Visio com base em um de seus modelos de fluxo de atividades, é possível identificar as etapas do processo otimizado e as principais informações recebidas pela equipe de desenvolvimento. Na etapa inicial pode ser visto o recebimento de dados de diversas fontes sendo essas fontes digitais ou de documentos feitos a mão. Em seguida as informações recebidas são armazenadas e recebem uma análise de validação para que, dentre as informações recebidas sejam identificadas as demandas válidas para o processo. Finalmente pode ser visualizada a otimização das demandas validadas e sua posterior transmissão para *softwares* terceiros ou a geração de indicadores de performance e resultados de análise.

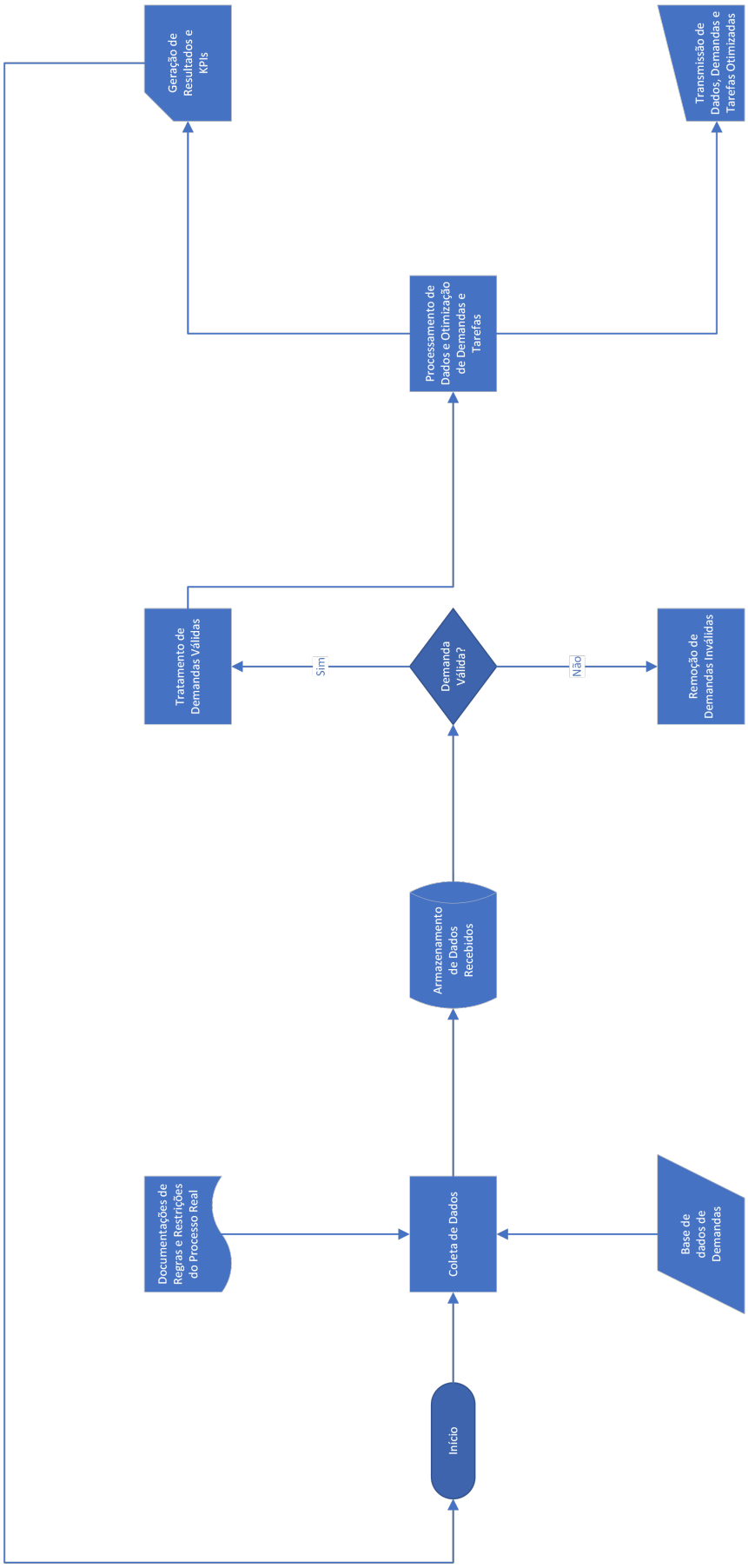

Figura 6 – Fluxograma de Processo para otimização de demandas.

Diante desse cenário, por meio da identificação das etapas do processo é possível destrinchar as tarefas de cada equipe de desenvolvimento e como essas tarefas se encaixam em um ciclo de desenvolvimento de software.

Dentro de cada etapa, as equipes de requisitos, *design* e desenvolvimento sempre buscam propor as seguintes perguntas:

- O que estamos recebendo?
- Como vamos exibir as informações recebidas?
- Como o usuário irá interagir com essas informações?
- O que pode ser otimizado e melhorado para obtenção de melhores resultados?
- Como vamos exibir esses resultados?

As perguntas descritas anteriormente são primordiais para que o entendimento do objetivo do desenvolvimento do projeto esteja claro e que um plano de ação coerente para cada funcionalidade criada seja traçado da melhor maneira possível.

Nesse cenário, será analisado o fluxo em que há a necessidade do recebimento de um novo dado de configuração. Esse dado deve ser exibido em interface, o dado pode ser modificado duplicado ou excluído para atender a algum parâmetro, além de impactar na otimização dos processos gerais. Nesse fluxo todas as equipes possuem atuação e devem realizar compartilhamento das informações e das abordagens escolhidas em cada desafio.

5.3 Solução: Aplicação de Conceitos e Práticas

Em primeiro lugar, a equipe de requisitos necessita identificar a disponibilidade das informações necessárias. Como essa disponibilidade pode não ser imediata, aqui se encaixa o fluxo da *sprint* de estudo que configura um período de avaliação de requisitos anterior ao desenvolvimento, assim, uma janela de tempo hábil é criada para que os dados necessários para o processo de desenvolvimento sejam disponibilizados. Também são mapeadas tarefas para a equipe de requisitos que envolvem análise de infraestrutura, reuniões de habilitação de dados por meio de credenciais necessárias, por fim, alinhamentos sobre a completude e cobertura dos dados necessários são mapeados. Além disso, em atividades paralelas a equipe de estrutura de dados realiza a construção da infraestrutura necessária para armazenamento das informações recebidas e sua caracterização. Nesse momento a existência do software de apoio se faz necessária para que seja possível documentar os artefatos gerados, aqui os artefatos em foco são as histórias de usuário e as tarefas atreladas as tais histórias.

As histórias de Usuários são modeladas utilizando um modelo padrão, geralmente utilizado no mercado de desenvolvimento atual, no qual são levantados itens como o autor de uma ação, sua necessidade e a finalidade de tal ação. Dessa forma, foi possível desenvolver a seguinte tela exibida na Figura 7 para acompanhamento de tais artefatos. A figura apresenta a tela de controle e cadastro de histórias de usuário. Aqui são apresentadas duas histórias que possuem informações obrigatórias como:

- Título;
- Descrição do problema a ser atacado;
- Hipótese de solução;
- Critérios de aceite a serem atingidos na solução idealizada;
- Links para protótipos de tela;
- Identificador da história de usuário visualizada.

Figura 7 – Apresentação das histórias de usuário iniciais para levantamento de requisitos.

Na Figura 8 é identificada uma tela de sistema chamada de “Requisitos em Desenvolvimento”, que demonstra a relação das tarefas em andamento, mais especificamente, uma tela que apresenta tarefas ligadas a um requisito já finalizado e em desenvolvimento. A tela mostra as tarefas atreladas a cada história de usuário que fazem parte dos requisitos em desenvolvimento na *sprint* atual, cada tarefa com suas características e identificadores que serão posteriormente descritos.

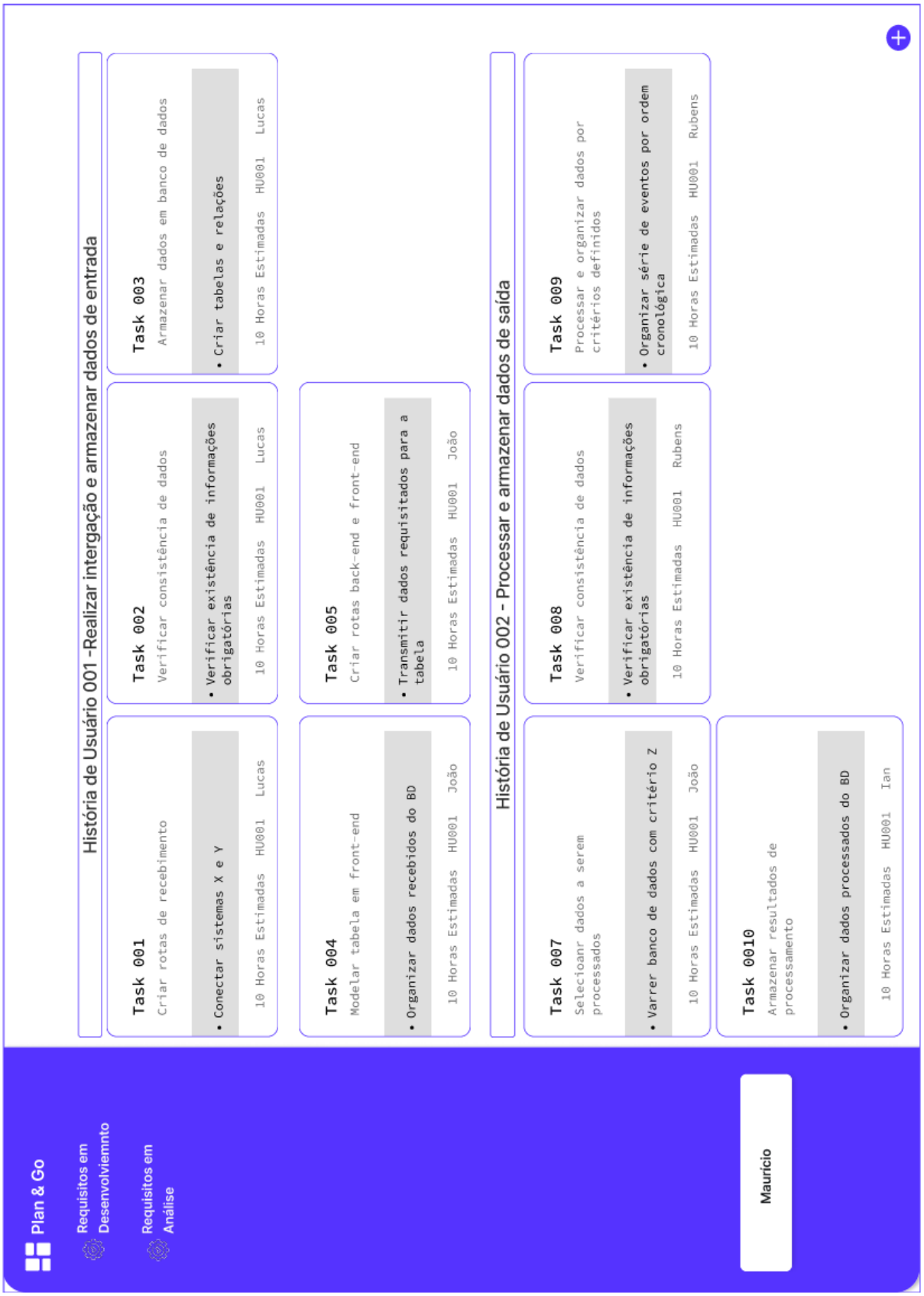

Figura 8 – Apresentação das histórias de usuário em desenvolvimento e as tarefas atreladas a elas.

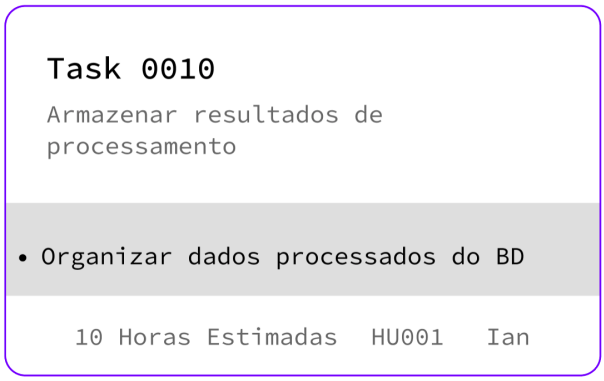

Figura 9 – Cartão de mapeamento de principais informações que caracterizam uma tarefa.

Em uma análise mais direta dos itens, é possível identificar o modelo de cartão que pode ser utilizado para rastrear informações relativas a uma tarefa ligada a uma história de usuário. A Figura 9 apresenta o modelo criado para esse cartão. O cartão apresenta respectivamente o identificador da tarefa dentro do projeto, seu objetivo principal, a principal ação a ser tomada para alcançar tal objetivo, a estimativa em horas para sua finalização, o identificado da história de usuário a qual a tarefa está atrelada e por fim a pessoa responsável por tal tarefa.

Com esses artefatos criados é possível formular um processo em que as necessidades do projeto surgem, são avaliadas, transformadas em requisitos, os requisitos são distribuídos em histórias de usuário e, finalmente, as tarefas são criadas para realizar a criação de uma solução para o requisito.

Com o Recebimento dos dados, a equipe de requisitos agora passa a trabalhar ao lado da equipe de *design*. Os atores também analisam os dados recebidos e identificam a melhor maneira das informações em foco serem disponibilizadas em tela para que o usuário possa, dessa forma, interagir com elas. O objetivo dessa etapa é gerar uma visualização de alto nível que possa ser apresentada aos *stakeholders* para ser validada. Esse processo é cíclico já que as propostas podem receber dicas de melhorias e modificações a cada rodada de validação.

Após a avaliação e apresentação das propostas de solução para os requisitos apresentados é necessária a aprovação dos mesmos por parte do time técnico. Sendo aprovadas, os atores relativos ao fluxo de desenvolvimento técnico podem começar a atuar em suas tarefas que atacam os requisitos levantados. Ao receber em mãos a documentação inicial de requisitos como histórias de usuários, protótipos de alto nível e exemplos de dados a serem trabalhados, os integrantes da eqiupe podem agora avaliar abordagens possíveis para desenvolver as propostas criadas. Os dados agora podem passar pela validação de regras de negócio e criação de regras que realizem a filtragem das informações, para que o conjunto final atenda as necessidades da aplicação. O objetivo de tal etapa é principalmente apontar impeditivos e necessidades extras para que uma solução encontrada

possa ser desenvolvida em sua totalidade. Nesse ponto, estão envolvidas pessoas como desenvolvedores de diversos módulos como *backend*, *frontend*, gerentes técnicos de dados e de otimização.

Após a avaliação e validação técnica, é importante que as propostas visuais e de documentação sejam revistas. A passagem dos artefatos por outras equipes sempre agrega valor e expande o horizonte de uma solução, assim, pode gerar novas informações que são cruciais para o entendimento de uma proposta e sua abordagem final de desenvolvimento. Assim, é possível avaliar a necessidade de documentar os alinhamentos e definições que foram acordadas para o desenvolvimento da solução, dessa forma, uma tela do protótipo que habilita a possibilidade de atrelar a um requisito diferentes informações como exemplos de dados a serem processados e explicações técnicas sobre algoritmos utilizados pode ser criada.

Com a definição final do escopo de desenvolvimento sendo totalmente validada junto aos clientes e as partes interessadas, a etapa de criação das soluções pode ser iniciada, juntamente com novos atores envolvidos, mais especificamente, os integrantes da equipe que abordam as tarefas de testes e qualidade. Sendo definida a solução, a equipe consegue definir quais aspectos devem ser observados e estressados em uma bateria de testes que possa apontar falhas e divergências do escopo inicial definido para a solução. Então, as informações levantadas são documentadas em software próprio para criação de testes, mas também podem ser atreladas documentações às histórias de usuário mapeadas no Plan And Go.

Em um fluxo natural, todo processamento gera resultados e, novamente, é necessário interpretar e exibir os dados que são chamados de "dados de saída". O fluxo de levantamento de propostas agora se inicia a partir do desenvolvimento de uma interface em que os dados resultantes do processamento são exibidos de diversas formas diferentes. Assim, artefatos e propostas com a presença de gráficos temporais e relacionando as grandezas relativas às características dos dados processados devem ser criados nessa nova rodada.

A geração de dados de saída passa também pela geração de documentação que descreva os dados processados, além do ferramental utilizado para se alcançar tais dados de saída. As Figuras 10 e 11 apresentam respectivamente os modelos de telas de documentação criados para armazenar informações relativas ao desenvolvimento de *sprints* passadas, artefatos e recursos técnicos e padrões de design escolhidos para serem utilizados em um sistema.

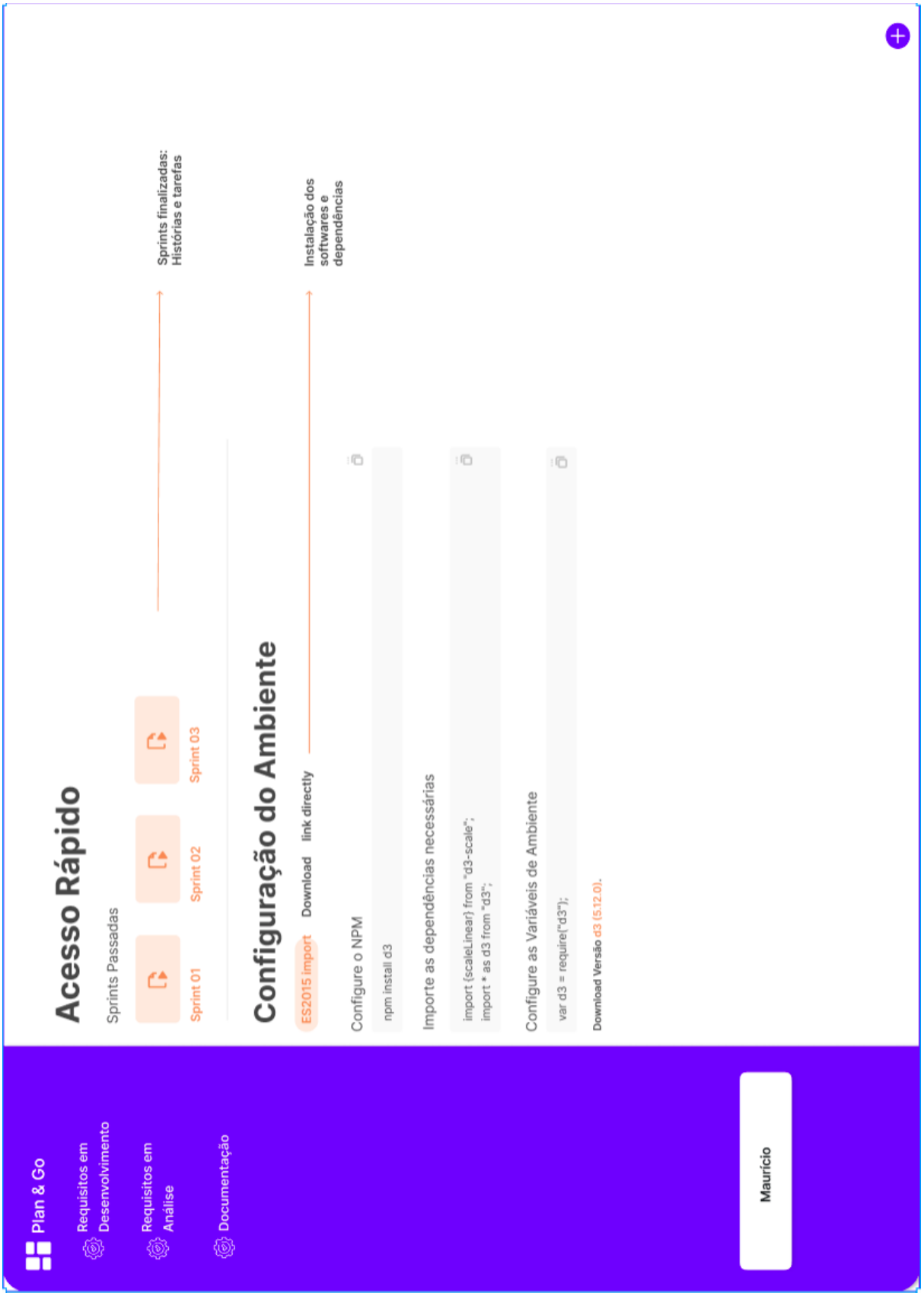

Figura 10 – Tela de documentação de recursos e *Sprints* passadas.

Figura 11 – Tela de documentação de padrões de design escolhidos para um sistema em desenvolvimento.

Mais uma vez, perante a criação de artefatos e sua aceitação pelo cliente, as soluções são apresentadas à equipe técnica que novamente avalia as características de implementação e abordagens possíveis para a entrega do resultado. Com a passagem dos artefatos pela validação técnica, a equipe de validação de qualidade e testes pode atuar também na criação de testes que possam focar na interface de resultados aprovada.

Com as etapas de levantamento de requisitos, prototipação e documentação de soluções e criação de testes finalizada por todos as equipes para todos os processos citados (recebimento de dados, manipulação e processamento de informações e exibição de resultados), agora é possível inserir tais atividades em uma *sprint* de desenvolvimento que permita que a funcionalidade seja desenvolvida. Assim, é necessário ser sensível a complexidade definida para cada tarefa no momento da validação. Dessa forma, pode ser necessária a distribuição do processo de desenvolvimento e aplicação de testes em várias *sprints*, seguindo, por exemplo, uma tratativa de testes isolados do período de desenvolvimento, isso pode ser apresentado da seguinte forma:

- ***Sprint* 1**:
  - Desenvolvimento do recebimento e processamento de dados;
  - Desenvolvimento da exibição de dados recebidos e armazenados;
  - Aplicação de testes de recebimento, processamento e exibição de dados.
- ***Sprint* 2**:
  - Desenvolvimento de interface de resultados obtidos;
  - Criação de testes para a exibição de resultados obtidos;
  - Aplicação de testes de exibição de resultados.

5.4 Considerações Sobre o Estudo

A partir da utilização do fluxo descrito, pode-se avaliar os possíveis resultados por etapas e em relação aos atores envolvidos. Em primeiro lugar, é notável que a melhor definição das atividades em cada etapa resulta na geração um fluxo de informações mais completo e detalhado. O surgimento da informação é mapeado e ela segue em uma cadeia fixa onde é avaliada, interpretada e incrementada por diversos participantes. Também permite que exista um fluxo cíclico de atividades que faz com que seja possível a reavaliação constante de tratativas e propostas criadas.

A documentação dos processos também pode se tornar mais clara e a transmissão do conhecimento adquirido pela equipe pode se tornar mais compartimentalizada, assim, o repasse de pequenos blocos de informação se faz possível, o que pode impactar na curva

de aprendizado de um novo integrante da equipe ou usuário do sistema. Já, no âmbito dos atores envolvidos, fica em foco como a criação de suas tarefas atreladas a cada etapa pode se tornar uma atividade bem mais clara, o contexto e a atuação de cada um fica exposto desde o surgimento de um requisito que se transformará em uma funcionalidade.

Uma maior clareza no processo, pode gerar melhor comunicação entre os membros de uma equipe, faz com que a documentação de suas tarefas se torne algo fixo e coloca o profissional em um estado em que sua atuação se torna independente do restante da equipe, logo se outros atores encontrarem impeditivos, o processo como um todo não será interrompido. Por fim, o fluxo apresentado pode definir com clareza os entregáveis de cada etapa do processo, pois, passa por protótipos de alto e baixo nível, histórias de usuário, testes de interface e documentação de código. Dessa forma, fica claro quais e quando cada artefato deve ser iniciado, refinado e entregue por seus respectivos responsáveis.

6 Conclusão

Neste trabalho foi desenvolvido uma proposta de fluxo de desenvolvimento de software juntamente com um layout de protótipo que simula a aplicação desse fluxo em um projeto de real e seu respectivo estudo de caso simulado. O fluxo de atividades foi inicialmente baseado em um trabalho anteriormente desenvolvido pelo autor, expandindo sua abrangência e abordando suas principais características em paralelo aos conceitos de levantamento, documentação e repasse de requisitos através de um software de apoio, características essas consolidadas no mercado como uma base forte para a manutenção e evolução de um projeto. O protótipo que simula o fluxo de desenvolvimento foi desenvolvido utilizando a ferramenta Figma, que é uma ferramenta de prototipagem para geração de fluxos de visualização de telas em diversos formatos.

A fundamentação teórica se deu a partir de conceitos de gerenciamento de software e de processos de software que abordam metodologias ágeis e processos de levantamento, refinamento, desenvolvimento e entrega de produtos com necessidades atreladas a criação de tarefas de prototipação documentação e codificação. Por sua vez, os trabalhos relacionados mostraram a capacidade que sistemas de gerenciamento e suporte de software tem de ajudar a manter um projeto com um bom nível de rastreabilidade. Além disso, os trabalhos relacionados como um fluxo de software com atividades bem definidas, ajuda todos os integrantes de uma equipe a desenvolver e entender de maneira clara suas tarefas.

A partir da revisão da literatura, foi possível gerar fluxos de atividades para cada um dos atores, sendo pequenas equipes ou indivíduos isolados. Esses fluxos de tarefas mostram como integrantes participam, contribuem e entregam suas atividades, assim, dá maior clareza ao processo e gera uma sensação que objetivos em curto, médio e longo prazo são alcançáveis. A partir dessas definições, foi possível desenvolver um fluxo geral de atividades que contempla um processo cíclico de levantamento e construção de requisitos. A ideia foi transcrita em fluxogramas que demonstram a evolução de um requisito na cadeia do processo de software idealizado. Finalmente, para aplicar o fluxo definido foi formulado um estudo de caso onde foi descrita a aplicação das etapas dos fluxogramas criados e também foram demonstradas como essas etapas poderiam ser encaixadas dentro de um software real de acompanhamento de desenvolvimento de software através de uma proposta de layout de protótipo de telas interativo.

O desenvolvimento do trabalho aponta que a criação de um fluxo que atenda o acompanhamento de um requisito desde sua concepção até sua entrega atende a uma lacuna vista nos trabalhos relacionados estudados. Os trabalhos relacionados abrangem, em grande parte de suas abordagens, apenas um dos blocos explorados pelo trabalho

aqui proposto. Desses blocos podem ser citados: (1) o levantamento do requisitos; (2) sua estruturação técnica; (3) a criação de tarefas atreladas a seu desenvolvimento; e (4) o acompanhamento até a entrega. Assim, a principal contribuição aqui proposta é a criação de um ciclo de atividades dentro de uma *sprint* de desenvolvimento e um estudo de requisitos que pretende deixar clara a visão e evolução dos entregáveis desejados.

6.1 Limitações e Trabalhos Futuros

O fluxo de processo não busca estruturar um *backlog* completo de projeto, na proposta deste trabalho são abordados os períodos de desenvolvimento chamados de *sprints* e que são um período menor de desenvolvimento em comparação quando com o que que se busca mapear em um *backlog* de longo prazo para projetos de grande porte ou com longa duração.

Em relação à proposta de layout de protótipo, não é objetivo do trabalho abordar temas gerais associados à softwares de acompanhamento como o Microsoft DevOps, por exemplo. O acompanhamento do *deploy* de uma solução, O mapeamento de *bugs* e o gerenciamento de pessoal não entraram no escopo do projeto aqui descrito. Com isso o trabalho acaba se limitando a mapear *sprints* isoladas sem o rastreio de requisitos contínuos, alocação de horas para partcipantes da equipe ou métricas de sucesso para *deploy* das soluções.

Outros modelos de aplicações do fluxo também poderiam ser explorados, como sua integração com diferentes tipos de computação. Aqui é abordado apenas o conceito do desenvolvimento Web com aplicações de otimização. Usos relacionados a Internet das coisas, computação em nuvem e sistemas embarcados necessitam de abordagens que modificariam a estruturação do fluxo de projeto definido, utilizando assim de sua adaptabilidade mas que não foram explorados no trabalho atual.

Podem ser consideradas melhorias em duas etapas do trabalho. Por exemplo, na etapa de formulação do fluxo de atividades podem ser inseridas ramificações que cubram com maior profundidade às tarefas relacionadas, à qualidade de software e ao desenvolvimento de código. Essas ramificações podem ser melhor exploradas e ter seus artefatos melhor descritos de maneira mais aprofundada para gerar maior valor e rastreabilidade para o processo de *software* identificado, agregando mais informações que são importantes a médio e longo prazo.

Outro ponto a ser considerado para trabalhos futuros é a integração e adaptação do modelo de processo criado para que ele interaja com softwares e práticas externas. Como exemplo, pode ser citado o software Microsoft DevOps[1] que emprega práticas de *pipelines* para entrega e integração constantes. Mapear as atividades necessárias para implantar

[1] <https://azure.microsoft.com/pt-br/solutions/devops>

tais práticas e rastrear de forma cruzada entre o Plan And Go e o Microsoft DevOps é uma tarefa que pode ser explorada para trazer maior robustez ao modelo criado.

Já na construção da proposta de layout do protótipo, podem ser realizadas melhorias na escolha de cores e modelos que facilitem a visualização, essa melhoria depende de um estudo de acessibilidade e a participação de pessoas com conhecimento profundo em experiência do usuário ao utilizar produtos digitais. Também podem ser realizadas melhorias de adaptação de documentação para que o sistema seja personalizável para cada caso em que for colocado em uso. Dessa forma, adaptar campos de informações para tarefas e para histórias de usuários são conceitos que podem ser melhorados e tratados em uma nova área de configurações gerais do sistema.

Por fim, com a intenção de ter uma visão real do uso de um modelo como o aqui desenvolvido, um dos trabalho futuros que poderia ser explorados seria a realização de pesquisas com usuários para captar suas percepções sobre a aplicação do modelo e como ele se adapta ao dia-dia de uso em um contexto real, levantando dificuldades enfrentadas e melhorias possíveis.

Referências

ACUNA, J. C. S. A hci technique for improving requirements elicitation. *Information and Software Technology*, v. 1, 2019. Acesso em: 01 feb. 2024. Citado na página 13.

ANAND, D. M. D. R. V. Popular agile methods in software development: Review and analysis. *School of Information Technology and Engineering*, 2016. Acesso em: 05 jun. 2023. Citado na página 19.

ASGHAR, A. T. A. R. Role of requirements elicitation & prioritization to optimize quality in scrum agile development. *International Journal of Advanced Computer Science and Applications*, v. 1, n. 1, 2016. Acesso em: 04 jan. 2024. Citado 2 vezes nas páginas 33 e 34.

BIøRN-HANSEN, A. et al. An empirical study of cross-platform mobile development in industry. *Hindawi Wileyl*, Mobile Technology Lab, Department of Technology, Kristiania University College, v. 1, n. 1, p. 1–13, 2019. Acesso em: 18 jun. 2019. Citado 3 vezes nas páginas 24, 25 e 31.

BRANISLAV RUZICKY, O. The change in management style during the course of a project from the classical to the agile approach. *Journal of Competitiveness*, p. 38–53, 2018. Acesso em: 05 jun. 2023. Citado 6 vezes nas páginas 13, 20, 22, 23, 36 e 47.

CHO, J. J. An exploratory study on issues and challenges of agile software development with scrum. *Utah State University*, 2019. Acesso em: 01 feb. 2024. Citado na página 12.

CONSTANTINO, K. et al. Understanding collaborative software development: an interview study. *ICGSE '20: Proceedings of the 15th International Conference on Global Software Engineering*, p. 55–65, 09 2020. Citado 3 vezes nas páginas 28, 29 e 31.

DUBOIS, G. T. D. J. Understanding gamification mechanisms for software development. *ESEC/FSE 2013: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, v. 1, n. 1, 2013. Acesso em: 04 jan. 2024. Citado na página 34.

FEITOSA IVAN MACHADO, R. S. P. M. T. M. Hybrid software development with scrum: Perceptions of brazilian software practitioners. *SBES '23: Proceedings of the XXXVII Brazilian Symposium on Software Engineering*, p. 226–235, 09 2023. Citado 3 vezes nas páginas 29, 30 e 31.

GOGUEN, C. L. J. A. Techniques for requirements elicitation. *International Journal of Computer Applications*, Oxford University Computing Lab, v. 1, n. 1, 2015. Acesso em: 04 jun. 2023. Citado 3 vezes nas páginas 21, 22 e 23.

HAJJDIAB, A. S. T. H. Adopting agile software development: Issues and challenges. *International Journal of Managing Value and Supply Chains*, 2011. Acesso em: 18 set. 2023. Citado na página 12.

HEAGNEY, J. Fundamentals of porject management. Amacom, v. 5, 2016. Citado na página 25.

HEEAGER, P. A. N. L. T. Agile software development and its compatibility with a document-driven approach? a case study. *ACIS Proceedings*, v. 1, p. 11, 12 2009. Citado 2 vezes nas páginas 37 e 44.

MORI FABIO PATERNO, F. F. G. Design requirements for web applications to affect the end user emotional state. *International Conference on Human-Computer Interaction*, v. 1, 2015. Acesso em: 01 feb. 2024. Citado na página 14.

MOURA, M. Diretrizes da engenharia de requisitos para projetos de softwares embarcados. *Encontro de Saberes - PIVIC*, v. 1, 2016. Acesso em: 20 feb. 2024. Citado na página 17.

MOURA, M. Plan and go: Um sistema para controle de micro atividades em projetos de software. 2019. Citado na página 13.

OIKKANEN, T.; HYRYNSALMI, S.; PAASIVAARA, M. How does the role of a product owner relate to the role of a software product manager? *LUT University Journal*, v. 1, 2023. Acesso em: 21 feb. 2024. Citado na página 42.

PAASIVARA, C. L. M. Deepening our understanding of communities of practice in large-scale agile development. *2014 Agile Conference*, 07 2014. Citado 2 vezes nas páginas 12 e 14.

PATIL, L. K. S. Embedded software - issues and challenges. *SAE International*, v. 1, 2010. Acesso em: 20 feb. 2024. Citado na página 14.

RASHEED, A. Requirement engineering challenges in agile software development. *Mathematical Problems in Engineering*, 2021. Acesso em: 19 feb. 2024. Citado na página 12.

SACHDEVA, S. Scrum methodology. *International Journal Of Engineering And Computer Science*, v. 1, 2016. Acesso em: 20 feb. 2024. Citado 2 vezes nas páginas 14 e 20.

SAEEDA, H.; AHMAD, M. O.; GUSTAVSSON, T. Challenges in large-scale agile software development projects. *Association for Computing Machinery.*, 2023. Acesso em: 24 set. 2023. Citado 6 vezes nas páginas 12, 22, 26, 27, 31 e 36.

SILVA, T.; SELBACH, M.; MAURER, F. Usability evaluation practices within agile development. *Annual Hawaii International Conference on System Sciences*, v. 1, 2015. Acesso em: 21 feb. 2024. Citado na página 26.

SOARES, M. dos S. Metodologias Ágeis extreme programming e scrum para o desenvolvimento de software. *Universidade Presidente Antônio Carlos*, University of Canterbury, v. 1, n. 1, 2004. Acesso em: 04 jun. 2023. Citado na página 20.

SOUSA, N. M. C. V. Aline de O. Prototyping usability and user experience: A simple technique to agile teams. *SBQS '19: Proceedings of the XVIII Brazilian Symposium on Software Quality*, Oxford University Computing Lab, v. 1, n. 1, 2019. Acesso em: 04 jan. 2024. Citado na página 23.

TUAPE, M. et al. Software practice in small software companies: Development context constraints on process adoption. *ESSE '22: Proceedings of the 2022 European Symposium on Software Engineering*, p. 1–9, 10 2022. Citado 3 vezes nas páginas 27, 28 e 31.

YOUSUF, M. A. M. Comparison of various requirements elicitation techniques. *International Journal of Computer Applications*, Baba Ghulam Shah Badshah University, v. 1, n. 1, 2015. Acesso em: 04 jun. 2023. Citado 2 vezes nas páginas 20 e 21.