UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

PATRICK GLAUBER OLIVEIRA DA SILVA

REESTRUTURAÇÃO DE UM PROCESSO DE DESENVOLVIMENTO COM A IMPLANTAÇÃO DE METODOLOGIAS ÁGEIS: UM ESTUDO DE CASO NO PROJETO DE EXTENSÃO NERDS

RUSSAS

2026

PATRICK GLAUBER OLIVEIRA DA SILVA

REESTRUTURAÇÃO DE UM PROCESSO DE DESENVOLVIMENTO COM A IMPLANTAÇÃO DE METODOLOGIAS ÁGEIS: UM ESTUDO DE CASO NO PROJETO DE EXTENSÃO NERDS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Marcos Vinicius de Andrade Lima

RUSSAS

2026

PATRICK GLAUBER OLIVEIRA DA SILVA

REESTRUTURAÇÃO DE UM PROCESSO DE DESENVOLVIMENTO COM A IMPLANTAÇÃO DE METODOLOGIAS ÁGEIS: UM ESTUDO DE CASO NO PROJETO DE EXTENSÃO NERDS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovado em: 16/01/2026

BANCA EXAMINADORA

Prof. Dr. Marcos Vinicius de Andrade Lima (Orientador)
Universidade Federal do Ceará (UFC)

Profa. Dra. Patrícia Freitas Campos de Vasconcelos
Universidade Federal do Ceará (UFC)

Profa. Dra. Marília Soares Mendes
Universidade Federal do Ceará (UFC)

Ao céu, pelo horizonte infinito; e às estrelas, pela luz na escuridão.

AGRADECIMENTOS

Aos meus amigos, que me acompanharam em toda esta jornada e sempre me ofereceram apoio incondicional.

Ao meu orientador, Prof. Dr. Marcos Vinicius de Andrade Lima, pela paciência, sabedoria e orientação fundamentais não apenas para a escrita deste trabalho, mas para toda a minha trajetória na Universidade Federal do Ceará (UFC).

Aos membros da banca examinadora, Prof. Dra Marília Soares Mendes e Prof. Dra. Patrícia Freitas Campos de Vasconcelos, pela disponibilidade e pelas valiosas contribuições para a avaliação e melhoria deste trabalho.

Aos membros do NERDS, por serem uma equipe incrível e pelo ambiente acolhedor. Foi um privilégio crescer ao lado de vocês.

Por fim, a todo o corpo docente da UFC – Campus Russas, por terem colaborado imensamente com meu desenvolvimento profissional e pessoal.

RESUMO

A complexidade inerente ao desenvolvimento de software contemporâneo, marcada pela volatilidade dos requisitos e pela necessidade de entregas frequentes de valor, impõe severas limitações às abordagens tradicionais de gerenciamento baseadas em escopo fixo. Nesse cenário, a gestão da produtividade e da qualidade torna-se um desafio crítico, especialmente em ambientes acadêmicos onde a rotatividade e a curva de aprendizado são elevadas. O presente trabalho teve como objetivo analisar o impacto da aplicação sistemática de práticas ágeis na produtividade de uma equipe de desenvolvimento em um projeto de extensão na Universidade Federal do Ceará. A metodologia adotada fundamentou-se nos *frameworks Scrum* e Kanban, integrando técnicas avançadas de engenharia de software, como a estimativa de esforço baseada em consenso via *Planning Poker*. Diferenciando-se de abordagens puramente qualitativas, este estudo empregou uma análise quantitativa rigorosa: os dados de produtividade foram submetidos a testes estatísticos inferenciais através da ferramenta JASP, permitindo validar a significância das variações de desempenho observadas ao longo das *sprints*. Os resultados demonstram que a transição de um modelo *ad hoc* para um processo ágil metrificado não apenas estabilizou o fluxo de entrega, mas também aumentou a clareza dos papéis e a satisfação da equipe. Conclui-se que a combinação de ritos ágeis com uma cultura orientada a dados e controle estatístico é determinante para mitigar incertezas e elevar a maturidade do processo de desenvolvimento, mesmo em contextos de times em formação.

Palavras-chave: produtividade; análise estatística; *planning poker*.

ABSTRACT

The inherent complexity of contemporary software development, marked by requirement volatility and the need for frequent value delivery, imposes severe limitations on traditional management approaches based on fixed scope. In this scenario, productivity and quality management pose a critical challenge, especially in academic environments where turnover rates and learning curves are high. This study aims to analyze the impact of the systematic application of agile practices on the productivity of a development team involved in a university extension project at the Federal University of Ceará. The adopted methodology was based on the Scrum and Kanban frameworks, integrating advanced software engineering techniques such as consensus-based effort estimation through Planning Poker. Distinguishing itself from purely qualitative approaches, this research employed a rigorous quantitative analysis: productivity data were submitted to inferential statistical tests using the JASP tool, enabling the validation of the statistical significance of performance variations observed throughout the sprints. The results demonstrate that the transition from an ad hoc model to a metrics-based agile process not only stabilized the delivery flow but also increased role clarity and team satisfaction. It is concluded that the combination of agile ceremonies with a data-driven culture and statistical control is decisive for mitigating uncertainties and raising the maturity of the development process, even in the context of newly formed teams.

Keywords: productivity; statistical analysis; planning poker.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

APA	*American Psychological Association*
GEX	Gestão da Extensão
IFPB	Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
LGPD	Lei geral de proteção de dados
NERDS	Núcleo Especializado em Reengenharia e Desenvolvimento de Software
PREX	Pró-Reitoria de Extensão
TCLE	Termo de Consentimento Livre e Esclarecido
UFC	Universidade Federal do Ceará
XP	*Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

Para Washizaki (2024), o desenvolvimento de software é um conjunto de atividades na área da computação em torno de criar, projetar, implementar, testar e evoluir softwares. Esses softwares são compostos por um conjunto de programas e documentação associada que executa um conjunto de funções para usuários ou outros sistemas (Gorton, 2011). Esse processo, como enfatizado por Pressman e Maxim (2021), geralmente é conduzido, principalmente, por programadores, engenheiros de software e desenvolvedores de sistemas inseridos em determinados contextos e projetos de desenvolvimento.

Dessa maneira, Washizaki (2024) destaca que softwares acabam por ser onipresentes e indispensáveis na sociedade moderna, fornecendo suporte em atividades simples, como organização de tarefas, escrita de notas pessoais, até atividades mais complexas, como definição de notas acadêmicas e auxílio a cirurgias médicas.

Porém, o desenvolvimento de software não pode ser visto como um processo simples. Como Martin (2020) demonstra, a definição do que será feito, como será feito e quem irá fazer, obedecendo todas as restrições que o contexto e o software a ser desenvolvido impõem, de modo que a qualidade do software não se perca e, principalmente, a produtividade do time de desenvolvimento não seja afetada, pode ser uma tarefa desafiadora.

Como uma forma de contornar a complexidade inerente na produção de software, Sommerville (2019) evidencia que a engenharia de software produziu técnicas, de modo a compreender os sistemas que devem ser desenvolvidos e suas limitações, e processos que pudessem guiar e gerenciar as equipes de desenvolvimento nessa tarefa, que não são, de forma alguma, limitados à indústria, mas que também podem ser aplicados em projetos de extensão, como é o caso do Núcleo Especializado em Reengenharia e Desenvolvimento de Software (NERDS).

O NERDS é um projeto de extensão da UFC Campus Russas, idealizado e coordenado pelo Prof. Marcos Vinicius de Andrade Lima, lançado em 14 de Abril de 2023, focado em desenvolvimento e reengenharia de software e na ideação e concepção de soluções inteligentes para órgãos e instituições públicas.

Uma dessas soluções é o sistema de Gestão da Extensão (GEX), um software voltado ao gerenciamento das ações de extensão, proporcionando a oferta de atividades aos membros da comunidade universitária e às pessoas externas à universidade. Além disso, o GEX também possibilita a organização de tarefas para os membros da própria ação de extensão.

As funcionalidades do GEX estão de acordo com as necessidades da Pró-Reitoria de Extensão (PREX) da UFC, que é o órgão responsável por planejar, coordenar e supervisionar as atividades de extensão universitária na instituição, que por si só, promovem a interação entre a universidade e a sociedade, aplicando o conhecimento acadêmico para atender às necessidades da comunidade e contribuir para o desenvolvimento social e cultural.

Contudo, apesar da ideia promissora, o desenvolvimento do GEX, que havia iniciado em abril de 2024, estava sujeito a um processo de desenvolvimento que não seguia um plano claro ou estrutura pré-definida, o que é comumente denominado como *ad hoc*.

Segundo Pressman e Maxim (2021), abordagens *ad hoc* tendem a gerar retrabalho e baixa previsibilidade, especialmente em projetos de médio e grande porte. Sommerville (2019) também destaca que, embora possam ser viáveis em contextos muito pequenos, tais práticas não escalam adequadamente quando há rotatividade de membros, que é um cenário comum em uma ação de extensão, ou necessidade de evolução contínua do software, gerando gargalos em relação à produtividade.

Nesse cenário, tornou-se necessária a adoção de uma metodologia mais estruturada, que garantisse maior controle do processo e alinhamento entre os participantes, o que motivou a introdução e investigação da abordagem ágil no projeto, especialmente neste contexto acadêmico.

Nesse sentido, este trabalho buscou analisar o impacto da adoção de metodologias ágeis no desenvolvimento do GEX, com foco na produtividade, na maturidade da equipe e na qualidade do processo. Para isso, utilizaram-se métricas como *story points* (coletados por meio da técnica *Planning Poker*) e o perfil técnico dos desenvolvedores envolvidos, mapeado por um questionário aplicado exclusivamente aos membros que participaram da transição antes e depois da adoção ágil (disponível no Apêndice A). Por fim, a análise levou em consideração diferentes recortes de tempo ao longo do desenvolvimento, que podem ser melhor visualizados na Seção 5.4.

Os participantes envolvidos neste trabalho podem ser divididos em dois grupos principais. O primeiro grupo foi composto por quatro integrantes da PREX, que participaram ativamente das reuniões com a equipe de desenvolvimento, validando o progresso do GEX e sugerindo melhorias e novas funcionalidades. Além disso, o Prof. Marcos Vinicius de Andrade Lima, coordenador do projeto, atuou como elo de comunicação entre a PREX e a equipe de desenvolvimento.

O segundo grupo correspondeu à equipe de desenvolvimento do NERDS, cuja

composição variou ao longo do projeto. Em diferentes momentos, a equipe variou de 4 a 34 membros, distribuídos em funções como desenvolvimento *front end*, desenvolvimento *back end*, DevOps, administração de banco de dados, análise de requisitos, análise de qualidade e gestão de projetos. Essa variação no tamanho e especialização da equipe refletiu a maturidade adquirida no processo de desenvolvimento.

No decorrer do projeto, foram realizadas diversas atividades relacionadas ao desenvolvimento do GEX. Entre elas, destacam-se o levantamento e a validação de histórias de usuário junto à PREX, o planejamento e acompanhamento das entregas pelo professor coordenador, a implementação do sistema pela equipe de desenvolvimento, além de reuniões periódicas de alinhamento e planejamento. Também foram conduzidos testes de funcionalidades e discussões sobre melhorias, bem como a adaptação do processo de trabalho para o uso de metodologias ágeis, o que contribuiu para a maturidade da equipe, evolução do produto, maior alinhamento com o cliente e adaptabilidade a mudanças.

Esta investigação se justificou pela relevância de analisar a adoção de metodologias ágeis em equipes acadêmicas de desenvolvimento e seu impacto em relação à produtividade, especialmente em contextos de alta rotatividade e interação direta com órgãos parceiros. Os resultados obtidos podem contribuir para a melhoria das práticas de ensino-aprendizagem em engenharia de software, além de oferecerem subsídios práticos para outros grupos que enfrentem desafios semelhantes.

Dessa forma, uma vez que é clara a importância social do estudo deste trabalho, surge uma questão: A reestruturação do processo de desenvolvimento, por meio da implantação de metodologias selecionadas de Engenharia de Software, pode aumentar a produtividade do time de desenvolvimento do NERDS em suas atividades? Para responder a essa questão, a seguir são especificados os objetivos (geral e específicos).

1.1 Objetivo Geral

Analisar o impacto da adoção de práticas de metodologias ágeis de Engenharia de Software no desenvolvimento do sistema da GEX, bem como a forma como essas práticas influenciaram o modo de trabalho da equipe.

1.2 Objetivos específicos

- Apresentar os processos de desenvolvimento de software adotados no NERDS.
- Identificar as fragilidades existentes nos processos atualmente empregados no NERDS.
- Listar técnicas e práticas que possam contribuir para a melhoria do desenvolvimento de software realizado no NERDS.
- Implantar as técnicas e práticas selecionadas, a fim de analisar a produtividade da equipe após sua adoção.

1.3 Organização do trabalho

Esse trabalho está organizado da seguinte maneira: na Seção 1, Introdução, são apresentados a contextualização do trabalho, a problemática, a questão de pesquisa e os objetivos geral e específicos; na Seção 2, Referencial Teórico, são abordados os princípios teóricos para o entendimento desta pesquisa; na Seção 3, Trabalhos Relacionados, é apresentada uma sucinta revisão bibliográfica em torno do tema deste trabalho; na Seção 4, Metodologia, é mostrada a descrição dos procedimentos metodológicos, a fim de alcançar os objetivos específicos descritos na Subseção 1.2; na Seção 5, Resultados, são apresentados os produtos extraídos da execução deste trabalho; e, por fim, na Seção 6, Conclusão, é apresentado o desfecho da execução de todo o estudo e sugestões para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Essa seção aborda os fundamentos teóricos necessários para a compreensão do trabalho, os quais envolvem o desenvolvimento e os processos de software, suas conexões com os modelos tradicionais e ágeis, o ciclo da Engenharia de Requisitos, a análise e o projeto de sistemas e sua relação com a concepção e o desenvolvimento da arquitetura de software, além de ferramentas para o gerenciamento eficaz de software.

2.1 Desenvolvimento de Software

Segundo Sommerville (2019), o desenvolvimento de software é descrito como um processo sistemático que envolve atividades para conceber, projetar, implementar, testar e manter sistemas de software, com o objetivo de transformar os requisitos dos usuários ou das organizações em produtos funcionais e eficientes, que atendam a determinados atributos de qualidade.

Ainda nessa linha de raciocínio, Pressman e Maxim (2021) destacam que, ao contrário da visão popular, o desenvolvimento de software está longe de se restringir ao processo de codificação, integrando também etapas como a gestão de requisitos e o projeto da arquitetura, que são essenciais para garantir a qualidade do produto final, ou seja, o software.

De maneira complementar, Mishra e Alzoubi (2023) destacam como a escolha da metodologia de desenvolvimento pode afetar o sucesso do processo, dependendo do contexto em que a equipe está inserida. Os autores concluem que, enquanto projetos que implementam metodologias ágeis apresentam uma taxa de falhas em torno de 10%, projetos que aplicam metodologias estruturadas, como o modelo cascata, podem alcançar taxas de falha de até 30%. Isso evidencia a importância da escolha apropriada da metodologia a ser utilizada.

De forma geral, Sommerville (2019) demonstra que os processos de desenvolvimento de software são estruturados em etapas, que podem variar conforme a metodologia adotada, mas que normalmente incluem especificação, desenvolvimento, validação e evolução.

Essas etapas, como Pressman e Maxim (2021) reforçam, são compostas por atividades, que representam o trabalho realizado; organizam-se em fluxos, que descrevem a ordem e dependência entre atividades; e geram artefatos, como documentos, modelos ou código-fonte, que são produtos intermediários ou finais do processo. Assim, com os elementos e conceitos chaves definidos, Sommerville (2019) demonstra que o primeiro passo para alcançar o sucesso

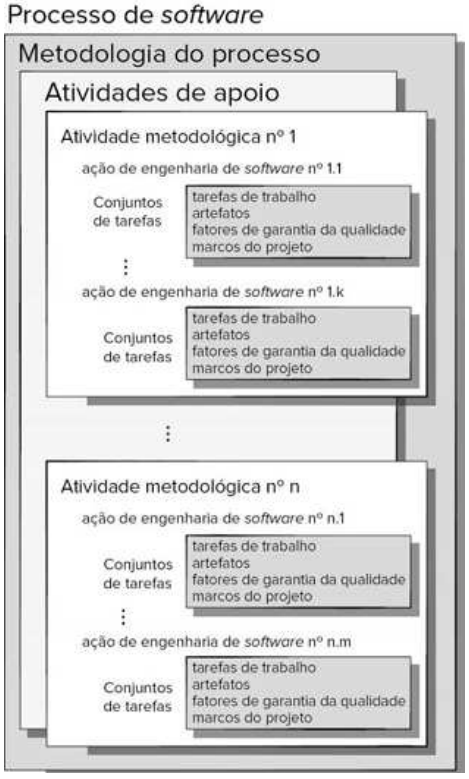
em um projeto, seja qual for, é a seleção adequada do processo de software que irá guiá-lo, descritos em detalhes a seguir.

2.2 Processos de software

Para que o desenvolvimento de qualquer software seja bem-sucedido, é necessário que haja um processo de software bem definido, o qual, segundo Sommerville (2019), consiste em um conjunto de atividades relacionadas que levam à produção de um sistema de software, visando à sua conformidade com os requisitos estabelecidos.

Conforme detalhado na Figura 1, o processo de software é encapsulado por uma metodologia, que por sua vez organiza um conjunto de atividades de apoio. Cada atividade se decompõe em uma ou mais ações de engenharia de software, que representam os passos técnicos a serem executados. Finalmente, cada ação se desdobra em conjuntos de tarefas, que são as unidades de trabalho que efetivamente produzem os artefatos, garantem a qualidade e definem os marcos do projeto. Essa estrutura hierárquica demonstra que existem diversos modelos de processos de software, os quais são aplicados em diferentes contextos de desenvolvimento.

Figura 1 – Modelo de processo genérico



Fonte: Pressman (2021, p. 86).

Conforme destaca Sommerville (2019), independentemente do modelo utilizado, seja em qual for o contexto que se encontra, todos eles compartilham algumas atividades em comum, sendo elas:

1. **Especificação**: a funcionalidade do software e as restrições sobre sua operação devem ser definidas.
2. **Desenvolvimento**: o software deve ser produzido para atender à especificação.
3. **Validação**: o software deve ser validado para garantir que atenda ao que o cliente deseja.
4. **Evolução**: o software deve evoluir para atender às mudanças nas necessidades dos clientes.

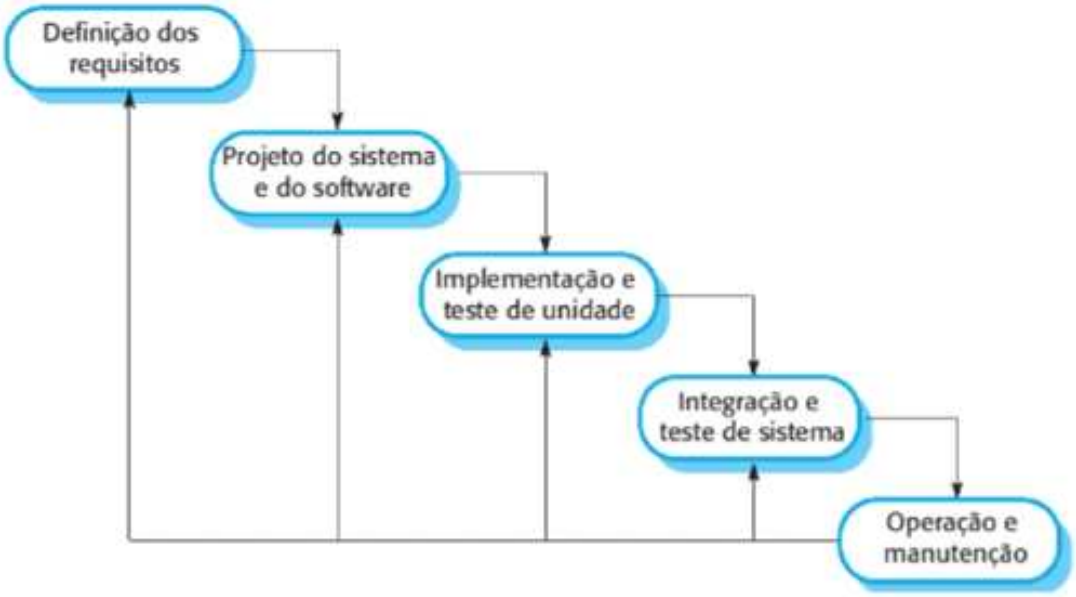
Contudo, apesar da grande variedade de modelos, Sommerville (2019) mostra que eles podem ser classificados em dois grandes grupos: os modelos prescritivos (ou tradicionais) e os modelos ágeis de desenvolvimento, apresentados em detalhes na próxima subseção.

2.2.1 Metodologias prescritivas ou tradicionais

Segundo Pressman e Maxim (2021), os modelos de processo prescritivos descrevem abordagens sistemáticas e estruturadas que definem um conjunto fixo de etapas e atividades necessárias para o desenvolvimento de um software. Sommerville (2019) acrescenta que esses modelos seguem uma sequência rígida, com fases bem delimitadas, proporcionando controle e previsibilidade ao processo de desenvolvimento. Apesar dessa previsibilidade, Pressman e Maxim (2021) mostram que os modelos prescritivos são pouco flexíveis a mudanças, uma vez que, após a definição do escopo, torna-se muito difícil retornar a etapas anteriores do processo.

Na Figura 2 é possível observar o funcionamento do modelo em cascata, considerado o precursor tanto dos modelos prescritivos quanto dos modelos de desenvolvimento de software como um todo. Suas etapas são executadas de forma sequencial, e o produto final é entregue apenas ao final do ciclo de desenvolvimento.

Figura 2 – Modelo em cascata



Fonte: Sommerville (2019, p. 33).

Entretanto, Guminski *et al.* (2023), ao investigarem as diferenças entre os modelos tradicionais e os modelos ágeis em projetos de desenvolvimento de software, destacam que a rigidez do modelo em cascata limita a adaptação a mudanças e resulta em *feedback* tardio, aumentando o risco de falhas.

Buscando superar as limitações do modelo em cascata, surgem os modelos iterativos, que adotam uma abordagem mais flexível e incremental. Nesse contexto, o modelo espiral, proposto por Boehm (1986), se destaca por permitir revisões constantes e um melhor controle de riscos ao longo do processo de desenvolvimento.

2.2.2 Metodologias ágeis

Segundo o Project Management Institute (PMI), as metodologias ágeis são abordagens flexíveis e iterativas para o desenvolvimento de software, que contrastam com os modelos tradicionais prescritivos (PMI, 2021). Elas foram criadas para lidar com a imprevisibilidade e a necessidade de adaptação em projetos de software, priorizando a entrega contínua de valor e a colaboração entre equipes. Em vez de seguir um plano rígido desde o início, os métodos ágeis permitem que as equipes respondam a mudanças e incorporem *feedback* contínuo durante o processo de desenvolvimento. Elas seguem os princípios das metodologias ágeis descritos em Beck *et al.* (2001), que são:

- Construir projetos em torno de indivíduos motivados, oferecendo suporte e confiança.
- Priorizar a satisfação do cliente por meio da entrega contínua de software funcional.
- Aceitar mudanças nos requisitos, mesmo que ocorram tardiamente no desenvolvimento.
- Entregar software funcionando frequentemente, de preferência em ciclos curtos (semanas ou meses).
- Promover colaboração próxima entre clientes e equipes ao longo do projeto.
- Valorizar a comunicação face a face, considerada a forma mais eficaz de troca de informações.
- Utilizar software funcional como principal medida de progresso.
- Garantir desenvolvimento sustentável, mantendo um ritmo constante.
- Manter atenção contínua à excelência técnica e bom design.
- Focar na simplicidade, realizando apenas o necessário.
- Incentivar equipes auto-organizadas, que tomam decisões de forma independente.
- Praticar reflexão e ajuste contínuos, com equipes revisando e melhorando processos regularmente.

Dentre as diversas metodologias ágeis, como *Extreme Programming* (XP), Kanban e *Feature Driven Development* (FDD), o *Scrum* se destaca como uma forte candidata a ser utilizada neste projeto. A escolha se justifica por sua comprovada resiliência em ambientes dinâmicos e com alta rotatividade de pessoal. Conforme destacado por Sutherland (2014), a estrutura do *framework* mitiga os riscos associados à perda de conhecimento através de artefatos transparentes, como o *Product Backlog*, que mantém o escopo e a priorização do trabalho visíveis a todos. Adicionalmente, Cohn (2010) reforça que seus ciclos curtos (*Sprints*) e eventos diários (*Daily Scrums*) aceleram a integração de novos membros, permitindo que eles agreguem valor rapidamente sem a necessidade de compreender todo o histórico do projeto.

2.2.2.1 Metodologia Scrum

Segundo Sutherland (2014), a metodologia *Scrum* é um *framework* de desenvolvimento criado por Jeff Sutherland e sua equipe no início dos anos 1990, que se mantém altamente popular e amplamente utilizado até a atualidade.

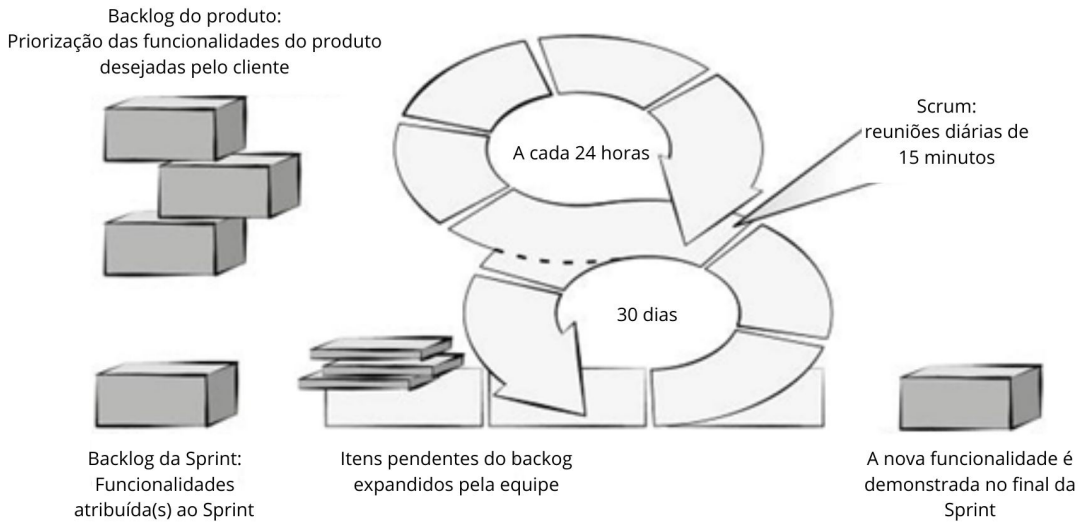
Pressman e Maxim (2021) caracterizam o *Scrum* como uma metodologia ágil que segue os princípios do Manifesto Ágil e é usada para orientar atividades de desenvolvimento dentro de um processo que incorpora as seguintes etapas metodológicas: requisitos, análise,

projeto, evolução e entrega.

Conforme ilustrado na Figura 3, o processo *Scrum* é cíclico e centrado em entregas incrementais. O ponto de partida é o *Backlog* do Produto, que contém a lista de todas as funcionalidades desejadas para o sistema. A cada ciclo, a equipe seleciona um subconjunto de itens dessa lista para formar o *Backlog* da *Sprint*. A equipe então entra em um período de trabalho focado e iterativo, a *Sprint* (representada na Figura 3 com um ciclo maior de 30 dias), durante o qual realiza reuniões diárias para sincronização (o ciclo menor de 24 horas). Ao final da *Sprint*, o objetivo é entregar uma nova funcionalidade ou um incremento de software funcional, e o processo se repete.

- ***Sprint:*** é um determinado período de tempo que as equipes de desenvolvimento se utilizam para focar no desenvolvimento de certa funcionalidade ou módulo do produto, apesar da Figura 3 representar como um período de 30 dias, a *Sprint* pode variar sua duração de 2 à 4 semanas.
- ***Backlog* do produto (ou *Product backlog*):** é uma lista que contém todas as funcionalidades, melhorias, correções e tarefas necessárias para o desenvolvimento do sistema como um todo.
- ***Backlog da Sprint*:** subconjunto de itens do *Backlog* Produto, em que os desenvolvedores irão realizar sua implementação durante a *Sprint*.
- **Reuniões diárias (ou *dailys*):** reuniões diárias da equipe de desenvolvimento, onde cada membro fala brevemente o que fez no dia anterior, o que pretende fazer no dia da reunião, e que impedimentos ele possa vir a ter.
- **Incremento (ou *increment*):** representa a nova versão funcional do produto ao final da *Sprint*, contendo todos os itens do *Backlog* da *Sprint* que foram concluídos.

Figura 3 – Fluxo do processo *Scrum*



Fonte: Adaptado de Pressman (2021, p. 128).

Embora o *Scrum* não possua fases formais de “Análise” ou “Projeto” como os modelos tradicionais, ele incorpora essas atividades em suas práticas iterativas. A Engenharia de Requisitos, conforme Cohn (2010) demonstra, ocorre de forma contínua através do gerenciamento e refinamento do *Product Backlog*. Já as atividades de Análise e Projeto, como Sutherland (2014) reforça, são realizadas pela equipe de desenvolvimento em tempo real durante cada *Sprint*, garantindo que o design evolua junto com o produto, em vez de ser fixado em uma fase inicial.

Dessa forma, fica claro que o *Scrum* tem como foco desenvolver software de maneira incremental, respeitando a duração prevista da *sprint*, ao mesmo tempo em que busca atender aos princípios estabelecidos no Manifesto Ágil. Na adoção de *Scrum*, outras metodologias podem ser adicionadas, como o Kanban e *Extreme Programming*, além de *design thinking*, detalhes a seguir.

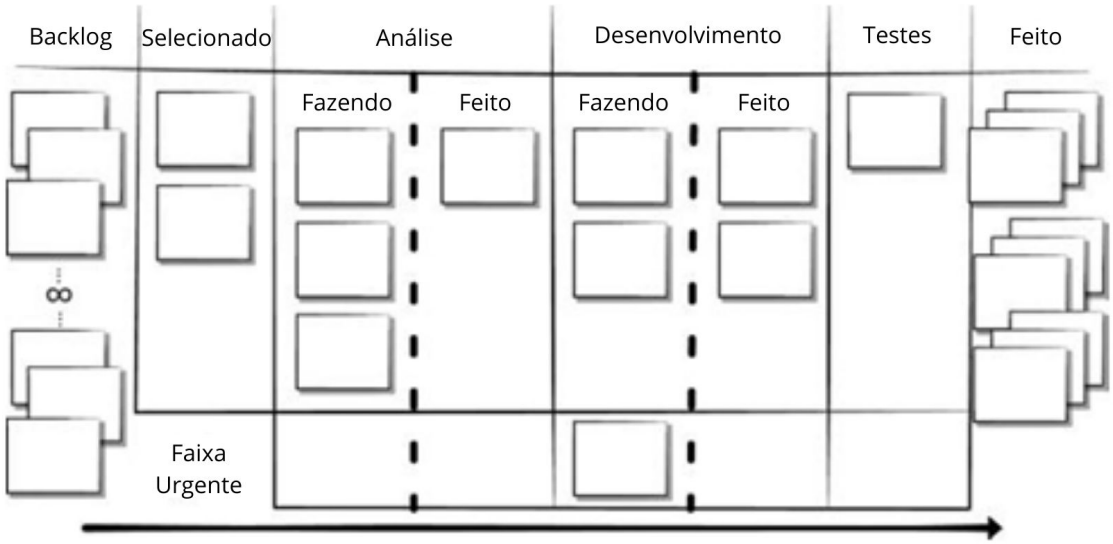
2.2.2.2 Metodologia Kanban

Ohno (1988) relata que, com o objetivo de eliminar desperdícios na linha de produção do Sistema Toyota, inspirou-se no modelo de reposição dos supermercados estadunidenses, onde o reabastecimento ocorre conforme o consumo. Baseado nessa lógica, desenvolveu o *Kanban*, um sistema visual de cartões para controle de fluxo que sinaliza o que deve ser produzido, a

quantidade necessária e o momento exato da produção.

Posteriormente, Anderson (2010) traz o conceito *Kanban* para fora do desenvolvimento industrial e mais voltado ao desenvolvimento de software e gestão de serviços, pegando os princípios propostos por Ohno, e adaptando para lidar com fluxo de trabalho de desenvolvimento de software, onde as atividades não são palpáveis como em fábricas, reforçando fortemente o uso de quadros visuais, conforme a Figura 4.

Figura 4 – Exemplo de quadro *Kanban*



Fonte: Adaptado de Pressman e Maxim (2021, p. 142).

Conforme a Figura 4 demonstra, o quadro é a ferramenta central do método. Nele, cada cartão representa uma unidade de trabalho que flui através de colunas que definem as etapas do processo, no exemplo:

- *Backlog*: Agrupa todas as tarefas existentes no projeto
- Selecionado: agrupa as tarefas selecionadas para iniciar sua implementação
- Análise: Dividida entre Fazendo e Feito, agrupa as tarefas que estão em fase de diagramação ou prototipação
- Desenvolvimento: Agrupa as tarefas que estão sendo desenvolvidas no momento
- Testes: Agrupa as tarefas que estão sendo testadas no momento
- Feito: Agrupa as tarefas que foram concluídas

O aspecto mais crucial do método *Kanban*, defendido por Anderson (2010), é a aplicação de limites de trabalho em progresso (*WIP Limits*), que é a definição de um número máximo de tarefas por coluna. Essa restrição força a equipe a focar na conclusão das tarefas ativas antes de iniciar novas, otimizando o fluxo e expondo gargalos no processo. Burrows (2014) aprofunda o conceito *Kanban*, e argumenta que sua implementação depende de uma cultura focada em valores, não em práticas, que são:

- **Transparência:** Tornar o trabalho e o processo visíveis.
- **Equilíbrio:** Balancear as diferentes demandas e capacidades do sistema.
- **Colaboração:** Incentivar a cooperação em vez de silos.
- **Foco no Cliente:** Orientar todo o trabalho para a entrega de valor ao cliente.
- **Fluxo:** Manter o trabalho fluindo de forma suave e contínua.
- **Liderança:** Encorajar a liderança em todos os níveis, não apenas na gerência.
- **Compreensão:** Buscar um entendimento compartilhado do trabalho e dos desafios.
- **Acordo:** Mover-se de uma cultura de comando para uma de consenso e acordo.
- **Respeito:** Respeitar as pessoas, suas funções e o processo atual.

Além disso, Hammarberg e Sunden (2014) reforçam que o *Kanban* não é limitado a apenas um nível organizacional, mas que também pode ser aplicado a diferentes níveis, como por exemplo nível estratégico e operacional, tornando a instituição que o aplica cada vez mais competitiva.

2.2.2.3 Metodologia Extreme Programming (XP)

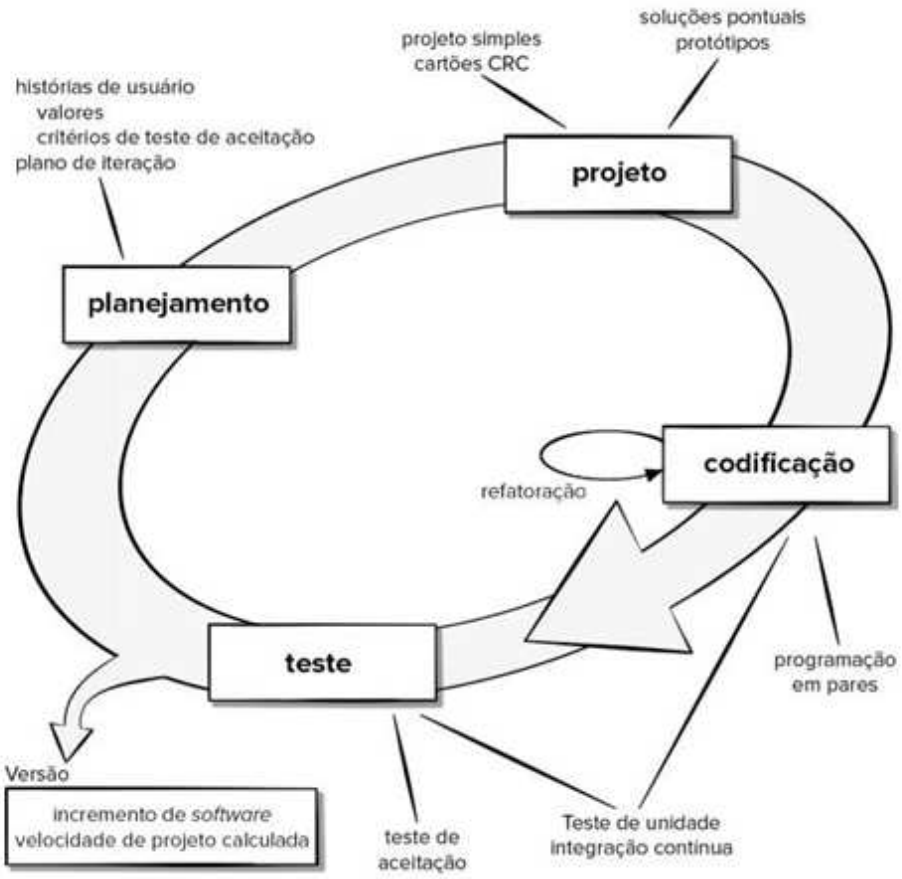
Beck (2004) descreve a metodologia *Extreme Programming* (XP) como um conjunto de valores, dentre eles a comunicação, *feedback*, simplicidade, coragem, respeito e práticas, que aprimorem comprovadamente o desenvolvimento de software, princípios, que auxiliem a aplicação prática do XP, e engajamento comunitário que possa fornecer auxílio a seus praticantes, Jeffries *et al.* (2000) complementam que o XP tem características que o diferencia e servem de base para outras metodologias, dentre elas:

- Ciclos de desenvolvimento curtos para *feedback* contínuo.
- Uma abordagem de planejamento incremental, adaptando-se à evolução do projeto.
- Flexibilidade na programação para atender às necessidades empresariais em mudança.
- Testes automatizados para monitorar o desenvolvimento e detectar defeitos precocemente.
- Comunicação oral e codificação para transmitir os designs do sistema.

- Processos de design evolutivo que persistem ao longo do ciclo de vida do software.
- Ênfase em indivíduos colaborativos e habilidosos.

Ademais, Beck (2004) enfatiza como a abordagem XP deve ser centrada no ser humano, reconhecendo as contribuições individuais e das relações humanas, buscando também a excelência técnica para aprimorar o trabalho em equipe e incentivando a vulnerabilidade como uma forma de fomentar a colaboração, e além disso, Jeffries *et al.* (2000) mostram como o XP evoluiu para permanecer adaptável a necessidades em constante mudanças, ser aplicável a equipes de quaisquer tamanhos, e como foca na melhoria da produtividade humana, sem regras rígidas para implementação e alcance dessa produtividade. Para operacionalizar esses valores e características, o XP propõe um processo de desenvolvimento iterativo e centrado em *feedback* constante, conforme ilustrado por Pressman e Maxim (2021) na Figura 5.

Figura 5 – Processo da Extreme Programming



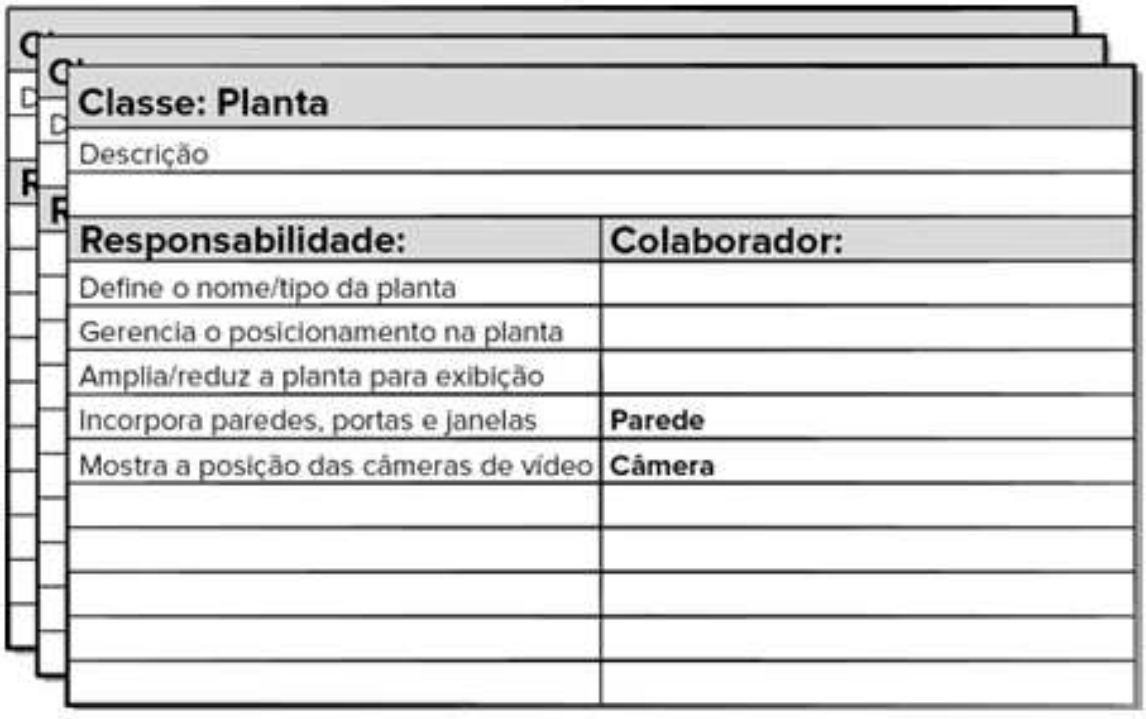
Fonte: Pressman e Maxim (2021, p. 138).

Conforme descrito na Figura 5, o XP passa por ciclos curtos de desenvolvimento, que envolve fases como planejamento, onde as necessidades dos usuários são descritas em histórias

de usuários, critérios de teste de aceitação são propostos, para definir se a entrega pode ser aceita, valores são definidos, para propor prioridades entre as histórias e um plano de iteração é feito para acompanhar o resto do ciclo.

A próxima etapa, a de projeto, segue a rigor o príncipio KISS (*keep it simple, stupid!*, ou seja, não complique!) que desestimula os desenvolvedores a projetar qualquer funcionalidade extra que seja, e prezar pelo mínimo necessário para o desenvolvimento do software. Além disso, o XP se utiliza dos cartões Classe-Responsabilidade-Colaborador (CRC), que consiste numa lista de fichas, onde cada ficha contém uma lista de responsabilidades no lado esquerdo, e uma lista de colaboradores correspondentes no lado direito, conforme o exemplo da Figura 6, que exemplifica uma classe denominada 'Planta', uma lista de responsabilidades que essa classe deve atender, e portanto, o desenvolvedor implementar, e uma lista de colaboradores, ou seja, outras classes que irão auxiliar no desenvolvimento da classe Planta, e que por vezes serão implementadas por outros desenvolvedores.

Figura 6 – Exemplo de cartões CRC



Fonte: Pressman e Maxim (2021, p. 330).

Na etapa de codificação, o código não é desenvolvido inicialmente, mas sim, um conjunto de testes unitários (que visam testar as menores unidade do software, ou seja, funções isoladas) a partir das histórias de usuário criadas na etapa de planejamento, passando

momentaneamente pela etapa de testes em um microciclo, uma vez feito, os desenvolvedores podem iniciar a codificação, se baseando nos testes unitários criados, garantindo uma maior qualidade ao código, e se utilizando da técnica de programação em pares, que consiste em dois desenvolvedores trabalhando de maneira conjunta em uma mesma funcionalidade, com o intuito de que supervisionem um ao outro, evitando possíveis erros. Uma vez que a funcionalidade é desenvolvida, ela é integrada ao trabalhos de outros pares de desenvolvedores e passa por processos de refatoração, ou seja, o código é refinado, com o objetivo de alcançar maiores níveis de qualidade.

Por fim, a etapa de testes se utiliza dos testes unitários criados, para compor os testes de regressão, que são um conjunto de testes que verifica se o software continua funcionando caso seja modificado, garantindo maior segurança ao desenvolvimento, e por último, nessa fase são realizados os testes de aceitação, que são testes realizados em conjunto com o cliente, com a intenção de revisar e validar as funcionalidades criadas neste ciclo de desenvolvimento e reunir informações para o próximo.

2.2.2.4 Design Thinking

Brown (2010) define o *Design Thinking* como uma abordagem de inovação centrada no ser humano, que se utiliza de um conjunto de ferramentas e mentalidades do design para integrar três pilares fundamentais: as necessidades das pessoas (o que é desejável), as possibilidades da tecnologia (o que é viável) e os requisitos para o sucesso do negócio (o que é viável financeiramente). Complementarmente, Norman (2013) reforça essa visão ao argumentar que o design centrado no ser humano começa com um profundo entendimento das necessidades reais e não articuladas dos usuários, focando em resolver o problema correto, em vez de apenas otimizar uma solução existente.

Ademais, Brown (2010) enfatiza que o *Design Thinking* não é apenas um processo, mas uma mentalidade (*mindset*) que deve permear a cultura da organização, sendo baseado em princípios-chave, dentre eles:

- **Empatia:** A capacidade de se colocar no lugar do usuário para entender seu contexto, dores e motivações.
- **Colaboração:** A integração de equipes multidisciplinares, unindo diferentes perspectivas (design, negócios, engenharia) para analisar um problema.
- **Experimentação:** Um viés para a ação, preferindo construir protótipos rápidos e de baixo

custo para testar ideias (o chamado "errar rápido para aprender cedo") em vez de longos ciclos de planejamento.

- **Otimismo:** A crença fundamental de que é possível criar uma solução melhor para qualquer problema.

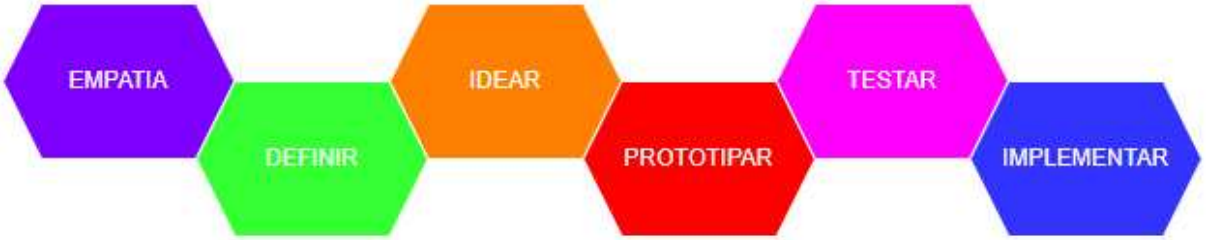
Para colocar essa mentalidade em prática, o *Design Thinking* se baseia em um processo de descoberta e desenvolvimento como Brown (2010) argumenta. Embora existam diferentes visualizações deste processo, ele é comumente estruturado em fases que guiam a equipe desde o entendimento profundo do problema até a entrega de uma solução viável.

A Figura 7 ilustra essas etapas, que, apesar de apresentadas sequencialmente para fins didáticos, compõem um ciclo altamente iterativo, onde a equipe pode e deve retornar a fases anteriores conforme novos aprendizados surgem, e que seguem as seguintes etapas.

- **Empatia (ou Imersão)**: É a fase de pesquisa, onde a equipe busca entender o problema do ponto de vista do usuário. Isso é feito através de técnicas de observação, entrevistas contextuais e outras formas de imersão no universo do usuário para ganhar empatia e coletar dados qualitativos.
- **Definir**: Com base nos dados coletados na fase de empatia, a equipe sintetiza os aprendizados para definir um ponto de vista (*Point of View - POV*) claro. O objetivo é enquadrar o problema de forma precisa e inspiradora, focando na necessidade humana que deve ser atendida
- **Idear**: Com um problema bem definido, a equipe entra em uma fase de divergência, onde o objetivo é gerar o maior volume possível de ideias e soluções potenciais, sem julgamento.
- **Prototipar**: Esta é a fase de "pensar com as mãos". A equipe seleciona as ideias mais promissoras da fase anterior e constrói protótipos de baixa fidelidade (que podem ser desenhos em papel, *storyboards*, *mockups* ou até mesmo encenações). O objetivo não é criar um produto final, mas sim algo tangível que possa ser usado para testar e validar suposições rapidamente.
- **Testar**: Os protótipos são apresentados aos usuários-alvo para coletar *feedback*. Esta fase é crucial e quase sempre leva a novas percepções que podem fazer a equipe retornar a fases anteriores, seja para refinar o protótipo, gerar novas ideias ou até mesmo redefinir o problema.
- **Implementar**: Esta é a fase onde a solução, já validada nos ciclos de prototipagem e teste, é efetivamente desenvolvida e entregue ao público. Esta etapa conecta o ciclo de design à

fase de produção, transformando o conceito validado em um produto ou serviço real.

Figura 7 – Etapas do Design Thinking



Fonte: Elaborada pelo autor.

Por fim, diversos autores têm explorado a sinergia e a integração entre o *Design Thinking* e as metodologias ágeis, incluindo o XP. Pesquisas como a de Sohaib *et al.* (2019) e Silva (2020) investigaram formas de integrar formalmente as atividades do *Design Thinking* ao processo do *Extreme Programming*. De forma similar, Higuchi e Nakano (2017) propuseram modelos combinados que utilizam o *Design Thinking* na fase de concepção e metodologias ágeis na fase de desenvolvimento.

A complementaridade, defendida por esses autores, reside na capacidade do *Design Thinking* em explorar o espaço do problema (garantindo que se está construindo a coisa certa através da empatia e validação), enquanto o XP oferece um processo disciplinado para construir a solução no espaço da solução (garantindo que se está construindo a coisa da forma certa com excelência técnica e ciclos rápidos). Nessa integração, o artefato gerado ao final do ciclo de *Design Thinking* (como um protótipo testado e um ponto de vista validado) serve como um input fundamental e bem fundamentado para a fase de planejamento do XP, alimentando a criação das primeiras histórias de usuário.

Independentemente do processo escolhido para a concepção do software, Pressman e Maxim (2021) afirmam que as práticas relacionadas à Engenharia de Requisitos, assim como as fases de Análise e Projeto de Sistemas, são essenciais para o sucesso do desenvolvimento.

2.3 Engenharia de requisitos

Para Vazques e Simões (2016), a Engenharia de Requisitos pode ser vista como uma subdisciplina da Engenharia de Software que consiste no uso sistemático e repetitivo de técnicas para cobrir atividades de obtenção, documentação e manutenção de um conjunto de requisitos que atendam aos objetivos propostos para o software, visando à sua qualidade. Pohl e

Rupp (2011) divide requisitos em duas grandes categorias: requisitos funcionais e requisitos não funcionais.

De acordo com Vazques e Simões (2016), os requisitos funcionais descrevem o comportamento que o software deve apresentar em termos de tarefas e serviços do usuário, sem considerar aspectos como o projeto da arquitetura do sistema, embora sejam profundamente afetados por estes.

Pressman e Maxim (2021) ressaltam que esses requisitos são geralmente especificados por frases relativamente curtas e diretas, acompanhadas de um identificador único e de um nível de prioridade. A seguir, alguns exemplos de requisitos funcionais:

- "**RF01 – Manter Cadastro de Usuários**: O sistema deve permitir o cadastro de novos usuários, possibilitando a inclusão, alteração, consulta e exclusão de dados pessoais e níveis de acesso."
- "**RF02 – Autenticação e Login**: O sistema deve permitir que usuários previamente cadastrados realizem a autenticação por meio de *e-mail* e senha para acessar as áreas restritas da aplicação."
- "**RF03 – Busca e Filtragem de Informações**: O sistema deve permitir que o usuário localize registros específicos (como tarefas, projetos ou usuários) através de uma barra de busca por palavras-chave e da aplicação de filtros combinados, como intervalo de datas, status ou categoria."

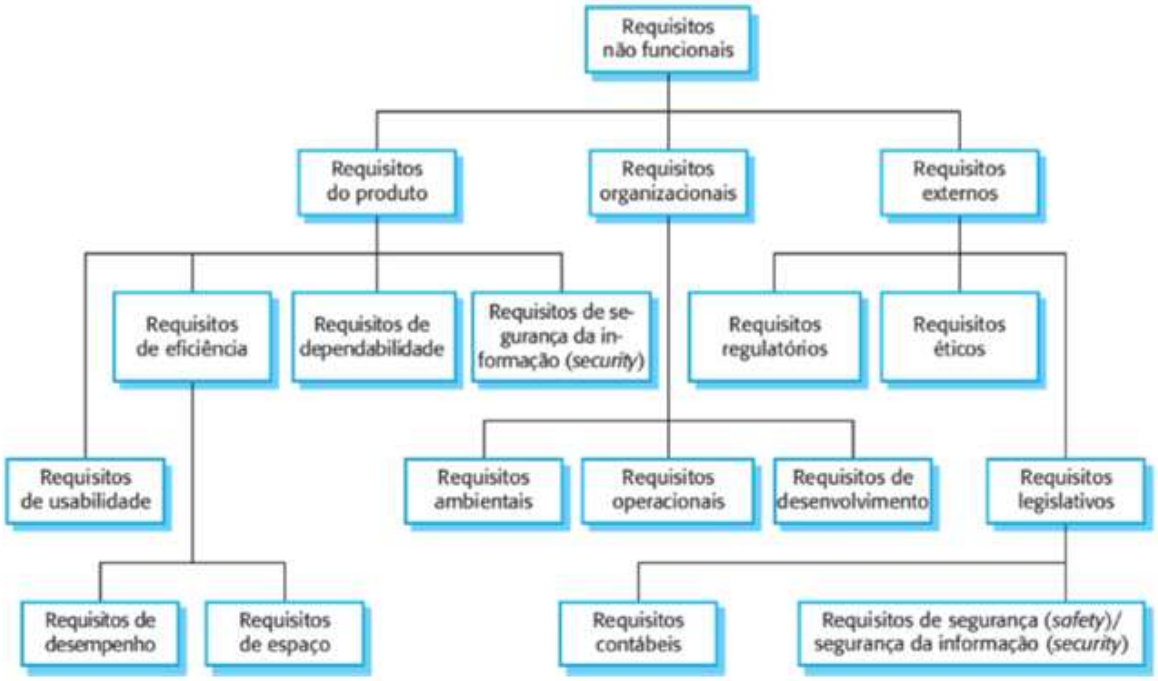
Seguindo a mesma linha de raciocínio, Vazques e Simões (2016) definem os requisitos não funcionais como limitações gerais aos requisitos funcionais, que complementam a especificação do software, definindo como o software deve ser feito, e não o que ele deve fazer.

Como destaca Sommerville (2019), esses requisitos estão sempre interligados a algum atributo de qualidade que o software deve possuir. A Figura 8 apresenta uma árvore de classificação dos requisitos não funcionais, evidenciando como esses requisitos podem ser organizados de acordo com sua origem e finalidade. No nível mais alto, os requisitos não funcionais são agrupados em requisitos do produto, requisitos organizacionais e requisitos externos. Os requisitos do produto dizem respeito às características de qualidade que o próprio sistema deve apresentar, incluindo eficiência, usabilidade, desempenho, uso de espaço, confiabilidade (*dependability*) e segurança da informação (*security*), as quais influenciam diretamente a experiência do usuário e o comportamento técnico do software. Já os requisitos organizacionais estão relacionados às políticas, processos e restrições da organização responsável pelo desenvolvimento e

operação do sistema, englobando aspectos ambientais, operacionais, contábeis e de desenvolvimento, como padrões, tecnologias e métodos a serem adotados. Por fim, os requisitos externos são aqueles impostos por fatores fora da organização, como requisitos regulatórios, legislativos e éticos, além de requisitos de segurança e proteção (*safety*), que garantem a conformidade legal e social do sistema. Essa classificação facilita a compreensão, a análise e a validação dos requisitos não funcionais, assegurando que diferentes dimensões de qualidade e restrição sejam adequadamente consideradas ao longo do ciclo de vida do software. A seguir, alguns exemplos de requisitos não funcionais:

- "**RNF01 – Desempenho**: O tempo de resposta do sistema para consultas simples ao banco de dados não deve exceder 2 segundos, considerando uma carga simultânea de até 50 usuários."
- "**RNF02 – Segurança**: Todas as senhas dos usuários devem ser armazenadas no banco de dados de forma criptografada."
- "**RNF03 – Portabilidade**: A interface do sistema deve ser responsiva, adaptando-se automaticamente a diferentes resoluções de tela para garantir a usabilidade em dispositivos móveis (smartphones e tablets) e desktops."

Figura 8 – Tipos de requisitos não funcionais



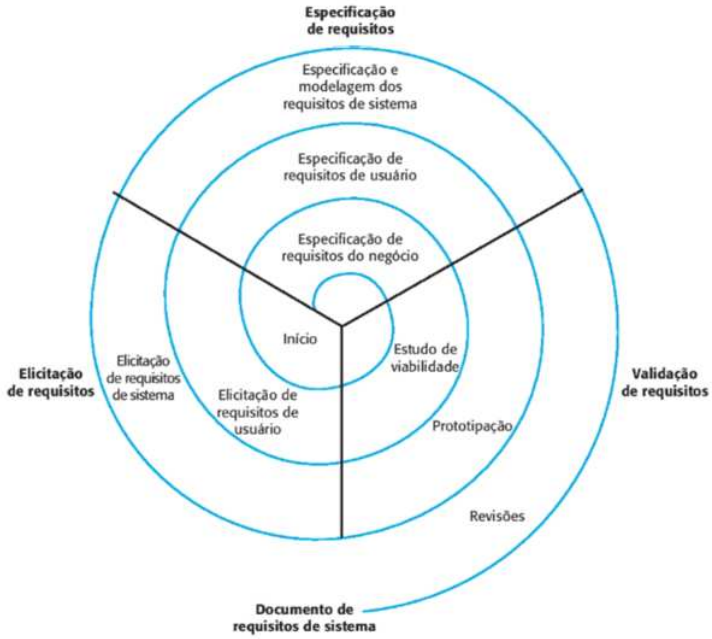
Fonte: Sommerville (2019, p. 92).

Como ressalta Sommerville (2019), esses requisitos são geralmente documentados em casos de uso, histórias de usuário (em metodologias ágeis) ou especificações técnicas. Sua implementação pode ser verificada por meio de testes funcionais, que confirmam se o software executa as ações esperadas (requisitos funcionais) e atuam em conjunto com uma série de restrições e fatores (requisitos não funcionais).

Parafraseando Pressman e Maxim (2021), a Engenharia de Requisitos abrange, de forma objetiva, a prática de elicitar, analisar, documentar e manter requisitos de software por meio de diferentes técnicas, sendo uma parte fundamental no desenvolvimento de software. Por meio dela, os desenvolvedores obtêm uma visão clara do que precisa ser realizado, além de acompanhar o progresso real ao longo do desenvolvimento, bem como esclarecer mal-entendidos entre a equipe e os clientes envolvidos.

A Figura 9, como detalhado por Sommerville (2019) ilustra o processo geral de Engenharia de Requisitos de forma iterativa e incremental, representado por círculos concêntricos que partem do início do projeto. No núcleo do processo encontram-se o estudo de viabilidade e os requisitos de negócio, que avaliam se o sistema é tecnicamente, economicamente e organizacionalmente viável, além de alinhar o software aos objetivos estratégicos da organização. A partir desse núcleo, o processo evolui para a elicitação e especificação dos requisitos de usuário, que descrevem as necessidades e expectativas dos usuários finais, frequentemente apoiadas por prototipação para facilitar o entendimento e a validação. Em seguida, os requisitos são refinados na especificação e modelagem dos requisitos de sistema, onde são detalhados de forma técnica, servindo como base para o desenvolvimento. Ao longo de todo o processo, ocorrem atividades contínuas de elicitação, validação e revisões, garantindo que os requisitos estejam corretos, completos e consistentes, culminando na produção do Documento de Requisitos de Sistema, que consolida formalmente os requisitos acordados entre as partes interessadas.

Figura 9 – Visão geral do processo de Engenharia de Requisitos



Fonte: Sommerville (2019, p. 95).

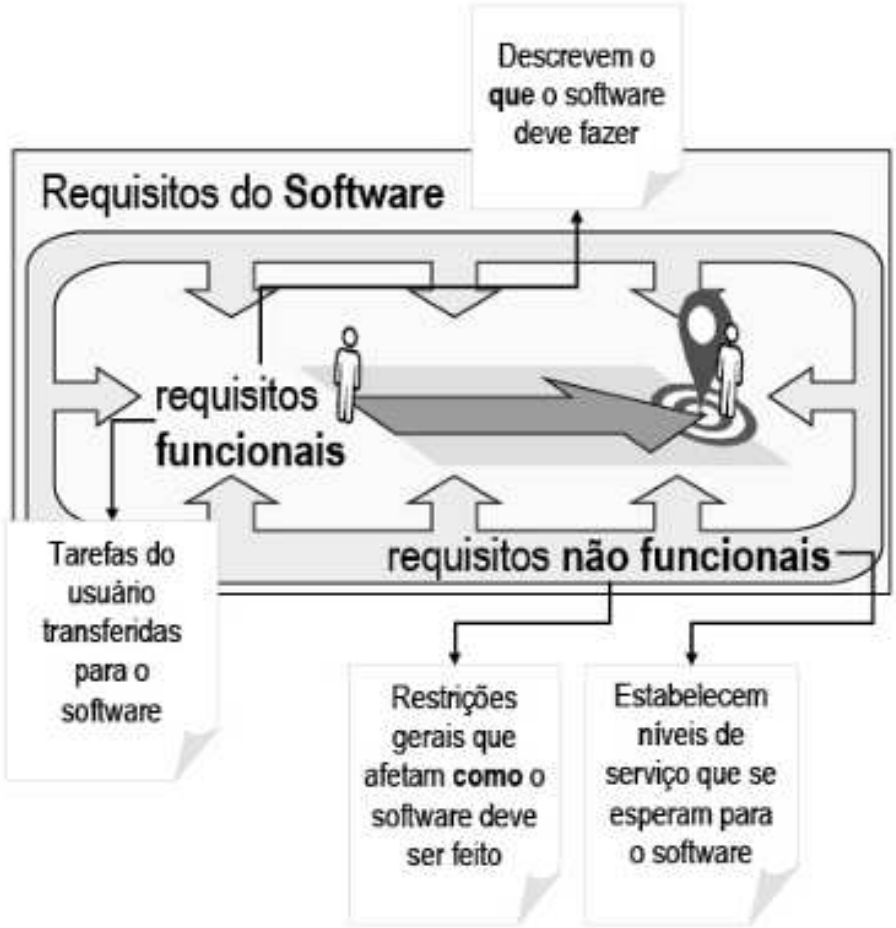
Segundo Wiegers e Beatty (2013), requisitos de software podem ser caracterizados como uma condição ou capacidade que deve ser alcançada ou possuída por algo, como um sistema no contexto do desenvolvimento de software, após a formalização em um documento.

Como destacam Pressman e Maxim (2021), os requisitos de um software podem ser divididos em dois grandes grupos: funcionais e não funcionais. A Figura 10, baseada em Vazques e Simões (2016), ilustra de forma clara a definição e a relação entre esses dois tipos de requisitos.

Conforme a Figura 10 demonstra, os requisitos funcionais descrevem diretamente “o que o software deve fazer”. Eles nascem das “tarefas do usuário transferidas para o software”, servindo como a base para os serviços e funcionalidades que o sistema irá oferecer. Em contrapartida, os requisitos não funcionais atuam como um complemento essencial, definindo o contexto e a qualidade da entrega. Eles estabelecem “restrições gerais que afetam como o software deve ser feito” e determinam os “níveis de serviço que se esperam para o software”, como desempenho, segurança e usabilidade.

A Figura 10 também evidencia, através das setas, que ambos os tipos de requisitos são interdependentes e se influenciam mutuamente para compor a visão completa dos "Requisitos do Software". Para que esse conjunto completo seja alcançado, é necessário passar pelo processo de elicitação e análise de requisitos.

Figura 10 – Exemplificação de requisitos funcionais e não funcionais



Fonte: Vazques e Simões (2016, p. 149).

2.3.1 Elicitação e análise de requisitos

Para Pressman e Maxim (2021), o processo de elicitação e análise de requisitos consiste em quaisquer atividades que possam levar à compreensão das necessidades das partes interessadas no sistema. Essas necessidades podem incluir funcionalidades, características, desempenho desejado, limitações de hardware, entre outros.

Como ilustrado e demonstrado por Sommerville (2019) na Figura 11, o processo de elicitação e análise de requisitos pode ser dividido em quatro fases genéricas, formando um eixo cíclico, que pendura enquanto o processo de desenvolvimento existir, elas são:

1. **Descoberta e compreensão dos requisitos**: processo de interação com as partes interessadas, no qual os requisitos são identificados e vagamente documentados.
2. **Classificação e organização dos requisitos**: processo de agrupamento dos requisitos conforme suas semelhanças e relações.

3. **Priorização e negociação dos requisitos**: processo em que os requisitos são priorizados e os conflitos entre as partes interessadas, decorrentes de diferentes visões de prioridade, são solucionados.
4. **Documentação dos requisitos**: fase em que os requisitos são documentados de forma informal para servirem como entrada para a próxima atividade.

Figura 11 – Processo de elicitação e análise de requisitos

Fonte: Sommerville (2019, p. 97).

Conforme destacam Pohl e Rupp (2011), existem diversas técnicas que permitem a elicitação de requisitos em diferentes cenários, como por exemplo entrevistas, questionários, análise de documentos, *workshops*, *brainstormings*, prototipação, etnografia, entre outros. Dentre as mais relevantes, estão:

- **Entrevista**: A entrevista é uma das técnicas mais tradicionais e eficazes para a engenharia de requisitos, baseando-se na comunicação direta entre o analista e as partes interessadas. Segundo Sommerville (2019), essa abordagem permite não apenas a coleta de informações explícitas, mas também a compreensão do contexto e das nuances do domínio do problema, podendo variar de conversas abertas e não estruturadas até interrogatórios formais baseados em roteiros pré-definidos.
- **Questionário**: O uso de questionários é indicado quando é necessário obter informações de um grande volume de pessoas ou quando as partes interessadas estão geograficamente dispersas. Conforme explicam Wiegers e Beatty (2013), essa técnica é econômica e

eficiente para validar hipóteses ou quantificar preferências, embora tenha como limitação a impossibilidade de esclarecer dúvidas em tempo real, exigindo que as perguntas sejam formuladas com extrema clareza para evitar ambiguidades nas respostas.

- ***Brainstorming***: O *brainstorming* é uma técnica de dinâmica de grupo voltada para a geração rápida de ideias e soluções inovadoras, incentivando a criatividade livre de julgamentos imediatos. De acordo com Pressman e Maxim (2021), o objetivo principal dessa sessão é explorar o maior número possível de alternativas para requisitos ou problemas do sistema, promovendo a colaboração entre a equipe técnica e os clientes antes que qualquer filtro de viabilidade seja aplicado.
- **Arqueologia de Sistemas**: A arqueologia de sistemas, frequentemente associada à análise documental ou engenharia reversa, consiste na extração de requisitos a partir do exame de sistemas legados, códigos-fonte antigos ou documentações pré-existentes. Como aponta Hunt e Thomas (2000), essa técnica é crucial quando o conhecimento tácito sobre as regras de negócio se perdeu com a saída de funcionários antigos, cabendo ao desenvolvedor "escavar" a lógica implementada no software atual para compreender o comportamento esperado na nova versão.
- **Etnografia**: A etnografia é uma técnica observacional onde o analista imerge no ambiente de trabalho do usuário para compreender como as tarefas são realmente executadas, e não apenas como são descritas formalmente. Segundo Viller e Sommerville (1999), essa abordagem é valiosa para identificar requisitos sociais e organizacionais sutis que raramente são verbalizados em entrevistas, pois permite visualizar os "gargalos" reais e as adaptações que os usuários fazem em sua rotina diária.

Por fim, independentemente da técnica de elicitação utilizada, é necessário realizar a especificação dos requisitos levantados, de modo que o resultado seja relevante para a construção do sistema.

2.3.2 Especificação de requisitos

Para Sommerville (2019), a especificação de requisitos é o processo de documentar os requisitos elicitados em um documento formal. Idealmente, esses requisitos devem ser claros, inequívocos, fáceis de entender, completos e consistentes, como os exemplos a seguir:

- O sistema deve permitir que o usuário faça login utilizando e-mail e senha
- O sistema deve suportar até 1.000 usuários simultâneos

Perceba como os objetivos estabelecidos em cada um dos requisitos é de fácil entendimento, e como cada um deles é completamente independente, sem gerar conflitos ou inconsistências com o outro.

Porém, como afirmam Pressman e Maxim (2021), as diferentes prioridades e visões das partes interessadas no sistema podem gerar conflitos entre os requisitos elicitados na etapa anterior.

Ademais, Sommerville (2019) destaca diversas maneiras de documentar requisitos: a linguagem natural, adequada para comunicação com os clientes; a linguagem estruturada, utilizada para comunicar-se com os desenvolvedores; além de modelos gráficos e especificações matemáticas, conforme ilustrado pelo Quadro 1.

Quadro 1 – Classificação das notações para especificação de requisitos

Notação	**Descrição**
Sentenças em linguagem natural.	Os requisitos são escritos usando frases numeradas em linguagem natural. Cada frase expressa um requisito.
Linguagem natural estruturada.	Os requisitos são escritos em linguagem natural em um formulário ou *template*. Cada campo fornece informações sobre um aspecto do requisito.
Notação gráfica.	Modelos gráficos, suplementados por anotações em texto, são utilizados para definir os requisitos funcionais do sistema. São utilizados com frequência os diagramas de casos de uso e de sequência da UML.
Especificações matemáticas.	Essas notações se baseiam em conceitos matemáticos como as máquinas de estados finitos ou conjuntos. Embora essas especificações inequívocas possam reduzir a ambiguidade em um documento de requisitos, a maioria dos clientes não compreende uma especificação formal. Eles não conseguem averiguar se ela representa o que desejam e relutam em aceitar essa especificação como um contrato do sistema.

Fonte: Adaptada de Sommerville (2011, p. 104).

Ademais, as metodologias ágeis fazem uso das chamadas histórias de usuário como

uma técnica fundamental para a especificação dos requisitos. Por meio delas, Pressman e Maxim (2021) mostra a busca pela descrição das necessidades do usuário de forma simples, colaborativa e facilmente compreensível.

2.3.2.1 Histórias de usuário

Uma história de usuário, como definem Vazques e Simões (2016), é uma breve declaração que descreve algo que o sistema deve fazer para o usuário. Geralmente, uma história de usuário segue o seguinte *template*, especificado pela primeira vez em Cohn (2004):

- Como <ator da ação>
- Quero <realizar uma ação>
- Para que <motivo da ação ser realizada>

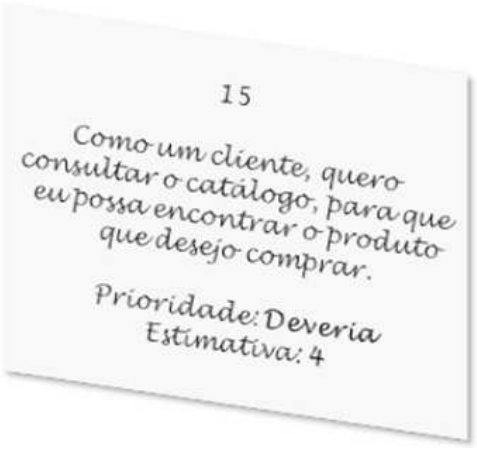
Conforme exemplificado na Figura 12, onde se é possível identificar qual o ator da ação (cliente), a ação a ser realizada (consultar catálogo) e o motivo pelo qual ela deve ser realizada (encontrar o produto que se deseja comprar), além de outras informações adicionais, como a prioridade que define o quão urgente é a implementação dessa história de usuário, e a estimativa, que define o quão grande e demorada pode ser a implementação.

Além disso, conforme descreve Cohn (2004), a história pode conter partes integrantes complementares, como os critérios de aceitação, que são as condições determinantes para validar a implementação de uma história de usuário, e os cenários de uso, responsáveis por descrever a execução em diferentes situações e os resultados esperados.

Ademais, o uso de histórias de usuário foi introduzido como uma unidade de funcionalidade do XP e atualmente integra outras abordagens ágeis, como o *Scrum*. Conforme descrito por Vazques e Simões (2016), a história deve ser compreensível e apresentar valor na perspectiva do cliente, ser testável e breve o suficiente para ser implementada em uma única iteração, cuja duração varia geralmente entre duas e quatro semanas. Essas características tornam as histórias de usuário favoráveis e aplicáveis a esta pesquisa.

Independentemente da abordagem escolhida para realizar a especificação dos requisitos, seja ela tradicional ou ágil, é fundamental que esses requisitos passem por um processo de validação.

Figura 12 – Exemplo de história de usuário



Fonte: Vazques e Simões (2016, p. 307).

2.3.3 Validação de requisitos

Para Pressman e Maxim (2021), a validação de requisitos é o processo de conferir se os requisitos definem o que as partes interessadas realmente querem. Essa etapa é bastante importante, pois corrigir erros nesta fase custa muito menos do que em fases posteriores do desenvolvimento. Segundo Sommerville (2019), diferentes tipos de conferências devem ser realizadas no documento de requisitos. Essas conferências incluem:

- **Conferência de validade**: confere se os requisitos refletem as reais necessidades dos usuários do sistema.
- **Conferência da consistência**: verifica se os requisitos no documento não entram em conflito entre si.
- **Conferência da completude**: o documento de requisitos deve incluir as funções e as restrições pretendidas pelo usuário do sistema.
- **Conferência do realismo**: verifica se, a partir do conhecimento das tecnologias existentes, é possível implementar os requisitos propostos, levando em consideração o orçamento e o cronograma.
- **Verificabilidade**: verifica se os requisitos do sistema são verificáveis, ou seja, se podem ser testados.

No entanto, para que a verificabilidade e o realismo sejam assegurados, é fundamental a aplicação de métricas de requisitos. De acordo com Wiegers e Beatty (2013), requisitos vagos ou subjetivos impossibilitam a criação de casos de teste assertivos e a medição do sucesso do projeto. A utilização de métricas quantitativas atua como um critério de aceitação, transfor-

mando expectativas abstratas em dados mensuráveis. Essas métricas podem estar associadas a desempenho (tempo de resposta), confiabilidade (tempo médio entre falhas), volume (capacidade de dados) ou usabilidade (tempo de aprendizado).

Para diferenciar um requisito mal especificado (subjetivo) de um requisito bem especificado (metrificado), é necessário comparar exemplos práticos, conforme visto a seguir:

- **Requisito Vago:** O sistema deve ser fácil de usar.
- **Requisito Vago:** O software deve ser rápido.
- **Requisito Vago:** O software deve enviar notificações ao usuário.

Perguntas como: "Como o sistema será fácil de utilizar?", "Que critério será usado para determinar se ele é fácil ou não de usar?", "Que velocidade o sistema deve alcançar?" e "Essa velocidade será a mesma para todas as operações?" evidenciam a falta de critérios objetivos. Como enfatiza Pressman e Maxim (2021), essa ambiguidade pode levar a desentendimentos entre a equipe de desenvolvimento e à insatisfação do cliente. Contudo, ao aplicar a definição de métricas proposta por Wiegers e Beatty (2013), é possível tornar requisitos mal especificados em requisitos verificáveis:

- **Requisito Metrificado:** 95% dos usuários devem completar o cadastro em menos de três minutos sem consultar o manual de ajuda (Métrica de Usabilidade).
- **Requisito Metrificado:** O sistema deve processar 1000 transações bancárias em menos de dez segundos, utilizando no máximo 90% da CPU (Métrica de Desempenho e Eficiência).
- **Requisito Metrificado:** O sistema deve enviar *e-mails* de confirmação para os usuários dentro de um minuto após a finalização do cadastro (Métrica de Tempo de Resposta).

Note que agora são estabelecidas métricas claras, como uma margem de usuários (95%) e um tempo limite (três minutos), que delimitam o quão "fácil" deve ser a utilização do sistema. Também são definidos volume e intervalo de tempo (1000 transações em dez segundos), além do custo máximo de hardware permitido.

Sommerville (2019) ressalta que a definição dessas métricas ajuda o desenvolvedor a eliminar dúvidas sobre o funcionamento esperado, levando a softwares mais confiáveis e com maior taxa de sucesso na validação.

Com isso, uma vez que os requisitos foram validados, é necessário estimar o esforço necessário para que eles sejam implementados de maneira efetiva no software.

2.3.4 Estimativa de esforço (Planning poker)

A estimativa de esforço em projetos ágeis difere das abordagens tradicionais por focar na complexidade relativa das tarefas e não apenas no tempo cronológico. A técnica mais difundida para esse fim é o *Planning Poker*, uma abordagem baseada em consenso descrita por Cohn (2005) e criada originalmente por Grenning (2002). O objetivo principal é evitar o "viés de ancoragem" e promover a discussão técnica entre a equipe, utilizando a inteligência coletiva para dimensionar os requisitos.

A dinâmica da técnica depende diretamente da escala de pontuação adotada nas cartas. Ao contrário de uma escala linear (1, 2, 3, 4, 5...), o *Planning Poker* utiliza progressões que refletem a incerteza inerente ao desenvolvimento de software: quanto maior o requisito, menor é a precisão da estimativa. Segundo Cohn (2005), existem diferentes escalas aplicáveis, sendo as mais comuns:

- **Sequência de Fibonacci Original**: Segue a progressão matemática clássica (0, 1, 2, 3, 5, 8, 13, 21, 34...). Embora matematicamente elegante, pode gerar uma falsa sensação de precisão em números altos (por exemplo, a distinção entre 21 e 20 pontos é irrelevante na prática).
- **Potências de 2**: Utiliza uma escala de progressão geométrica (1, 2, 4, 8, 16, 32...). É útil para forçar decisões mais binárias sobre o tamanho das tarefas, mas oferece menos opções de granularidade intermediária.
- **Sequência de Fibonacci Modificada (Escala *Scrum*)**: É a variação mais utilizada na indústria e a adotada neste trabalho. A sequência comumente aceita é (0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100).

A preferência pela Escala *Scrum* (Fibonacci Modificada) se justifica pela Lei dos Grandes Números aplicada à incerteza. Conforme explica Rubin (2012), ao arredondar o 21 para 20 e o 34 para 40, a equipe assume explicitamente que, em níveis elevados de complexidade, a precisão detalhada é impossível. Se uma tarefa é tão grande que vale "40 pontos", não importa se ela levaria 38 ou 42 horas; ela é simplesmente "muito grande" e provavelmente deve ser quebrada em itens menores. O processo de estimativa ocorre durante a cerimônia de Planejamento da *Sprint*:

1. O *Product Owner* apresenta um item do *Backlog*.
2. A equipe discute brevemente as implicações técnicas.
3. Cada membro seleciona uma carta da escala escolhida e a mantém virada para baixo.

4. As cartas são reveladas simultaneamente.

Se houver convergência, a estimativa é aceita. Se houver discrepância (ex: um desenvolvedor joga ’3’ e outro ’13’), ocorre uma discussão para alinhar o entendimento sobre a complexidade, seguida de nova votação. Esse rito garante que a pontuação final represente o consenso técnico da equipe, considerando a escala de incerteza apropriada para o projeto.

Dessa forma, após a validação, definição das métricas e mensuração do esforço, torna-se indispensável convertê-los em uma representação visual e estruturada do software. Esse processo ocorre por meio das atividades de análise e projeto de sistemas, que têm como objetivo detalhar a arquitetura, os componentes e a organização das funcionalidades que serão implementadas.

2.4 Análise e projeto de sistemas

Para Larman (2005), analisar um sistema é um processo focado em compreender o problema e identificar os requisitos que o software deve atender, sem se preocupar com soluções técnicas ou implementações. Essa é a etapa na qual os analistas realizam um estudo detalhado dos requisitos levantados na fase de elicitação.

Enquanto isso, na fase de projeto, como destacado por Pressman e Maxim (2021), determina-se “como” o sistema funcionará para atender aos requisitos, de acordo com os recursos tecnológicos existentes. Nessa fase, são adicionadas as chamadas “restrições de tecnologia” a qualquer modelo concebido na fase de análise.

Conforme visto em Bezerra (2014), a fase de projeto pode ser dividida em suas atividades principais: o projeto da arquitetura, ou projeto de alto nível, e o projeto detalhado, ou projeto de baixo nível.

Aqui surge uma relação interessante entre a projeção da arquitetura de software e a fase de projeto em si. Note que, embora a arquitetura seja um aspecto do projeto, ela não tem foco no nível de detalhe que o projeto se propõe a alcançar.

Independentemente da arquitetura de software adotada no desenvolvimento, torna-se essencial realizar um gerenciamento eficiente de todo o processo. Esse gerenciamento adequado é fundamental para garantir que o produto final atenda aos requisitos definidos, alcance os objetivos propostos e seja bem-sucedido em seu contexto de uso.

2.5 Gerenciamento de Software

De acordo com Sommerville (2019), o gerenciamento de projetos de software é uma disciplina essencial que envolve o planejamento, monitoramento e controle das atividades de desenvolvimento, visando garantir que o produto seja entregue dentro do cronograma e orçamento previstos, mantendo os padrões de qualidade exigidos. Esta atividade é necessária porque a engenharia de software é sujeita a restrições organizacionais e econômicas, e não apenas técnicas.

Diferente de projetos em áreas como a construção civil ou manufatura, o gerenciamento de software lida com um produto intangível e altamente volátil. Pressman e Maxim (2021) destaca que essa invisibilidade torna o progresso difícil de ser medido com precisão, exigindo dos gestores um esforço constante de comunicação e controle de mudanças para evitar o fracasso do projeto.

Segundo o Project Management Institute (2021), a gestão tradicional é fundamentada no conceito da "Tripla Restrição". Este modelo postula que a qualidade do projeto é restringida pelo equilíbrio entre três variáveis interdependentes:

- **Escopo:** Define todo o trabalho necessário, e apenas o necessário, para entregar o produto com as funcionalidades e características especificadas.
- **Tempo:** Refere-se ao cronograma planejado e aos prazos estabelecidos para a conclusão das atividades do projeto.
- **Custo:** Corresponde ao orçamento aprovado e aos recursos financeiros necessários para a execução do projeto.

Uma alteração em qualquer um desses vértices impacta inevitavelmente os outros e, por consequência, a qualidade final do produto (Kerzner, 2017). No entanto, em ambientes de desenvolvimento moderno, onde os requisitos mudam com frequência, a tentativa de "congelar" essas variáveis no início do projeto mostrou-se ineficaz para muitos contextos. Conforme aponta Pressman e Maxim (2021), projetos reais raramente seguem o fluxo sequencial proposto por esse modelo, uma vez que a incerteza é inerente ao processo de software e o bloqueio de mudanças frequentemente resulta na entrega de um produto obsoleto ou inadequado às necessidades do cliente.

Diante desse cenário de incerteza, Highsmith (2009) argumenta que o gerenciamento de projetos precisou evoluir de um modelo de "comando e controle" para um modelo de "liderança e colaboração". Nasce assim a gestão ágil, que inverte a lógica do triângulo tradicional:

em vez de fixar o escopo e tentar estimar tempo e custo, as abordagens ágeis fixam o tempo e o custo e permitem que o escopo varie para maximizar o valor entregue ao cliente.

Essa mudança de paradigma exige que a gestão abandone planos rígidos em favor de um planejamento adaptativo, focado em entregas frequentes e adaptação rápida às mudanças do negócio.

Uma vez ciente da importância de um gerenciamento efetivo em equipes de desenvolvimento de software, é necessário abordar o custo de retrabalhos e manutenções que possam vir a ocorrer, a partir de escolhas que resultem em softwares de menor qualidade.

2.5.1 Dívida técnica

O conceito de dívida técnica foi introduzido originalmente por Cunningham (1992) como uma metáfora financeira para descrever o custo implícito de retrabalho causado pela escolha de uma solução fácil e rápida no curto prazo, em detrimento de uma abordagem melhor que levaria mais tempo. Conforme explica Cunningham (1992), assim como em uma dívida financeira, o código desenvolvido sob essas condições gera "juros": o esforço extra que a equipe terá que despender no futuro para manter ou evoluir o software. Se essa dívida não for paga (através da refatoração), os juros acumulam-se até que o desenvolvimento estagne.

Segundo Fowler (2009), a dívida técnica não é necessariamente o resultado de código ruim ou incompetência. Frequentemente, ela é uma ferramenta estratégica de gerenciamento, onde a equipe opta conscientemente por lançar uma funcionalidade mais cedo para ganhar mercado, aceitando o compromisso de refatorar o código posteriormente. Fowler (2009) categoriza a dívida em quatro quadrantes, baseados na intenção (deliberada ou inadvertida) e na prudência (prudente ou imprudente).

No entanto, o acúmulo descontrolado dessa dívida pode levar a consequências graves para o projeto. McConnell (2004) alerta que, se a dívida não for "paga", os juros compostos se manifestam na forma de maior rigidez do sistema, onde pequenas alterações exigem esforços desproporcionais. Isso resulta em uma queda progressiva na velocidade da equipe, aumento na taxa de defeitos e, em casos extremos, na insolvência técnica, onde o custo de manutenção supera o valor gerado pelo software.

No contexto de equipes ágeis, Kruchten *et al.* (2012) destaca que a dívida técnica deve ser visível e gerenciada. Itens de dívida técnica devem ser incluídos no *Product Backlog* e priorizados junto com novas funcionalidades, garantindo que a produtividade e a qualidade

do software se mantenham sustentáveis a longo prazo. Sendo assim, é necessário conseguir mensurar o quão produtiva a equipe de desenvolvimento está sendo, e que armadilhas podem vir dessa medição.

2.6 Mensuração da produtividade em Engenharia de Software

A mensuração da produtividade em engenharia de software é um desafio persistente, dada a natureza intangível e intelectual do desenvolvimento de sistemas. Segundo Fenton e Bieman (2014), métricas tradicionais baseadas em volume, como Linhas de Código (LOC), mostram-se insuficientes para capturar a complexidade e a entrega de valor real.

No contexto das metodologias ágeis, o foco da mensuração desloca-se para o fluxo e a entrega de funcionalidades. Cohn (2005) define a Velocidade, entendida como a soma dos *Story Points* entregues em uma iteração, como a métrica primária para planejamento.

Entretanto, a simples coleta desses dados não é suficiente para comprovar hipóteses científicas ou validar melhorias de processo. Para determinar se uma variação na produtividade é significativa ou fruto do acaso, torna-se necessário o uso de estatística inferencial. De acordo com Wohlin *et al.* (2012), a experimentação em engenharia de software exige rigor na análise de dados, demandando ferramentas computacionais capazes de executar testes de hipótese complexos. No mercado e na academia, diversas ferramentas de análise quantitativa são utilizadas para esse fim, com destaque para:

- **SPSS (*Statistical Package for the Social Sciences*)**: Uma das ferramentas mais tradicionais e robustas do mercado. Embora extremamente potente, possui alto custo de licenciamento e uma interface considerada complexa para usuários não estatísticos.
- **R (Linguagem R)**: Um ambiente de software livre para computação estatística e gráficos. É o padrão-ouro para flexibilidade e poder de análise, porém exige conhecimentos avançados de programação para a criação de *scripts* de tratamento de dados.
- **JASP (*Jeffrey's Amazing Statistics Program*)**: Uma ferramenta de código aberto desenvolvida pela Universidade de Amsterdã, que tem ganhado ampla adoção na pesquisa acadêmica recente. Segundo Love *et al.* (2019), o JASP se diferencia por oferecer uma interface gráfica intuitiva e amigável, permitindo a execução de testes estatísticos complexos, como ANOVA e Kruskal-Wallis, sem a necessidade de programação.

O JASP destaca-se por democratizar a análise estatística, oferecendo suporte tanto para a estatística frequentista (clássica) quanto para a estatística bayesiana. Sua capacidade

de gerar relatórios visuais dinâmicos e tabelas formatadas no padrão *American Psychological Association* (APA), facilita a interpretação dos resultados por engenheiros de software, permitindo validar com rigor matemático se as flutuações de produtividade observadas em uma equipe são estatisticamente relevantes ou apenas ruído no processo.

3 TRABALHOS RELACIONADOS

Esta seção apresenta uma breve revisão bibliográfica pertinente ao tema deste trabalho, contextualizando-o dentro do panorama contemporâneo de desenvolvimento de software. Para isso, são analisados estudos anteriores que abordam problemas similares, com ênfase em suas metodologias, contribuições e limitações. O objetivo é estabelecer um diálogo crítico entre essas produções e a presente pesquisa, destacando:

- **O percurso investigativo**: como os trabalhos foram selecionados (ex.: bases acadêmicas, critérios de inclusão/exclusão) e quais lacunas ou demandas da área eles revelaram.
- **Análise comparativa**: similaridades e diferenças em relação a esta pesquisa, especialmente em abordagens, resultados e escopo.
- **Posicionamento deste trabalho**: de que forma esta pesquisa avança, complementa ou diverge das soluções propostas anteriormente.

A partir da questão de pesquisa central: "A reestruturação do processo de desenvolvimento ao implantar metodologias selecionadas de Engenharia de Software pode aumentar a produtividade do time de desenvolvimento do NERDS em suas atividades?", foram selecionadas as seguintes bases de dados: Google Scholar, SBC OpenLib e ACM Digital Library. Foi elaborada a seguinte *string* genérica de busca, adaptada para cada plataforma conforme o Quadro 2. Foram incluídos apenas trabalhos lançados entre 2020 a 2025, redigidos em língua portuguesa ou inglesa, com texto completo disponível gratuitamente, e que abordassem diretamente a aplicação de metodologias ágeis no contexto de engenharia de software.

É de interesse também que estes trabalhos tivessem ênfase em métricas quantitativas em relação a produtividade ou gestão de equipes, fossem executados em um contexto acadêmico e abordassem o impacto gradual da expansão dos times em seus estudos.

Em contrapartida, foram excluídos estudos puramente teóricos sem validação empírica, trabalhos duplicados entre as bases de dados, bem como pesquisas aplicadas a outras áreas do conhecimento (como manufatura ou marketing) que não dialogassem com os desafios específicos do desenvolvimento de software.

("agile methodologies" OR "software engineering methodologies" OR "metodologias ágeis" OR "metodologias de engenharia de software") AND ("developer productivity" OR "produtividade do desenvolvedor") AND ("case study" OR "estudo de caso") AND ("metrics" OR "métricas" OR "performance measurement" OR "medição de desempenho")

Quadro 2 – *Strings* de busca utilizadas nas bases de dados

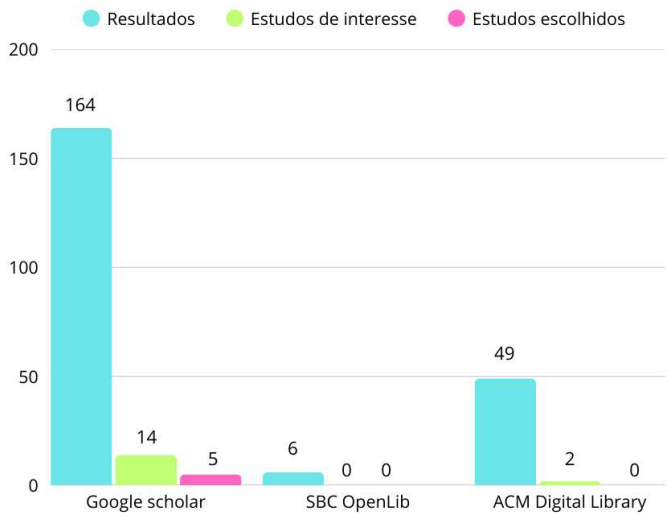
Base de Dados	***Strings* de Busca**
Google Scholar	("agile methodologies" OR "software engineering methodologies" OR "metodologias ágeis" OR "metodologias de engenharia de software") AND ("developer productivity" OR "produtividade do desenvolvedor") AND ("case study" OR "estudo de caso") AND ("metrics" OR "métricas" OR "performance measurement" OR "medição de desempenho")
ACM Digital Library.	("agile methodologies" OR "software engineering methodologies") AND ("developer productivity" OR "team productivity") AND ("case study" OR "empirical study") AND ("metrics" OR "performance measurement")
SBC OpenLib.	metodologias ágeis OR produtividade OR estudo de caso OR métricas OR agile methodologies OR productivity OR case study OR metrics

Fonte: Elaborado pelo autor.

Dessa maneira, foram selecionados os seguintes estudos, conforme ilustrado pelo Gráfico 1. A busca inicial no Google Scholar resultou em 164 artigos. Após uma filtragem primária pela leitura dos títulos e resumos, e a aplicação dos critérios de exclusão (eliminando, em sua maioria, estudos aplicados a outras áreas ou puramente teóricos sem validação empírica), apenas 11 foram considerados relevantes. Desses, após a leitura integral, 5 foram escolhidos para análise aprofundada por atenderem plenamente aos critérios de inclusão, especialmente por apresentarem métricas quantitativas e aderência ao contexto acadêmico.

Na SBC OpenLib, a pesquisa retornou 6 artigos, mas nenhum foi selecionado para aprofundamento, pois, após a leitura preliminar, constatou-se que não se adequavam ao escopo prático da pesquisa. Por fim, a busca na ACM Digital Library retornou 49 resultados, dos quais apenas 2 demonstraram interesse preliminar. Contudo, ambos foram descartados na fase de leitura completa por não dialogarem diretamente com os desafios específicos de expansão de times e métricas propostos neste trabalho. A seguir, os 5 trabalhos finais selecionados são detalhados.

Gráfico 1 – Resultados da busca por estudos semelhantes



Fonte: Elaborado pelo autor.

3.1 Implementação de metodologias ágeis para o gerenciamento de projetos em uma indústria

Gama *et al.* (2020) descrevem um estudo de caso sobre a implementação de metodologias ágeis para o gerenciamento de projetos em uma indústria metalúrgica, onde o departamento de TI da empresa, responsável pelo desenvolvimento e manutenção dos softwares administrativos próprios, enfrentava problemas como o levantamento de requisitos restrito à visão do usuário, resultando em explicações superficiais e projetos que nem sempre atendiam às necessidades dos solicitantes, sendo frequentemente abandonados após a entrega. Além disso, as estimativas de prazo geravam grandes atrasos nas entregas, e a ausência de uma metodologia específica de gerenciamento de projetos fazia com que as atividades fossem realizadas de forma não padronizada, baseando-se na experiência e disponibilidade dos programadores.

Dessa forma, o objetivo foi realizar um estudo de caso no departamento de TI para propor e implantar uma metodologia ágil de gerenciamento de projetos, utilizando práticas de *Design Thinking* e *Scrum*.

O estudo incluiu dois experimentos com equipes pequenas para validar a metodologia. A metodologia proposta combinou o *Design Thinking* com o método ágil *Scrum*, resultando em um *framework* sob medida para a empresa. O processo, que anteriormente contava com as fases “Não iniciado”, “Em andamento” e “Concluído”, foi expandido para incluir as fases “Em

planejamento" (imersão profunda) e "Em homologação" (prototipação), subdividindo a fase "Em andamento" em "Em planejamento", "Em desenvolvimento" e "Em homologação".

A metodologia auxiliou no planejamento das demandas, apoiando o desenvolvimento e gerenciamento de projetos de software. As reuniões diárias mostraram-se essenciais para manter o foco e a motivação, e a alternância de responsabilidades nos testes (em que um desenvolvedor programa e outro testa) ajudou na identificação precoce de erros. A estimativa de tempo via *Planning Poker* foi bem aceita e garantiu maior assertividade no planejamento.

De maneira semelhante, este trabalho busca analisar a implementação da metodologia *Scrum* em um ambiente real de desenvolvimento, reestruturando o processo, que antes era *ad hoc*, para um modelo firmemente baseado nos conceitos ágeis, utilizando fases personalizadas como "A fazer", "Fazendo", "Testando", "Concluído", "Reaberto" e "Bloqueado".

3.2 Aplicação do Kanban no gerenciamento de projeto de software remoto em uma empresa de *e-grocery* no contexto pós-pandemia Covid-19: um estudo de caso

Caetano (2023) investigou a aplicação da metodologia Kanban em uma equipe de desenvolvimento de software remoto, em uma empresa de e-grocery, no contexto pós-pandemia da COVID-19.

A pandemia de COVID-19, iniciada em 2020, forçou uma mudança abrupta no modelo de trabalho, impulsionando a adoção do trabalho remoto em diversos setores, incluindo o mercado de tecnologia. Essa transição gerou desafios significativos para empresas e equipes de desenvolvimento, tais como problemas de comunicação, redução do engajamento, dificuldades na entrega contínua e na gestão da visibilidade e prazos dos projetos. Diante desse cenário, metodologias ágeis como o Kanban surgiram como potenciais soluções para mitigar esses desafios e otimizar a eficiência e as estimativas de entrega em equipes remotas.

Caetano (2023) optou por investigar a aplicação do Kanban em uma equipe que adotou o modelo remoto devido à pandemia e o manteve no período pós-pandemia. A implementação do Kanban trouxe diversos benefícios para a equipe remota, como maior visibilidade do trabalho, facilitando a identificação de gargalos e bloqueios. A metodologia também contribuiu para a melhoria da capacidade de entregas da equipe e aumentou a agilidade por meio do sistema de fila puxada. Conforme define Anderson (2010), trata-se de um mecanismo onde novas tarefas são iniciadas apenas quando existe capacidade disponível na etapa seguinte, limitando o trabalho em progresso e evitando sobrecargas. A maioria dos participantes da pesquisa considerou o

Kanban útil na gestão de demandas e prioridades, recomendando sua adoção para outras equipes remotas. Em termos de satisfação, a maioria dos participantes demonstrou estar satisfeita ou muito satisfeita com a aplicação do Kanban no modelo de trabalho remoto.

Dessa forma, este trabalho também busca avaliar a implementação de uma metodologia ágil, porém em um contexto em que o cenário de desenvolvimento era até então *ad hoc*, utilizando métricas para avaliação da produtividade da equipe.

3.3 *Improving the Software Development Process in a Software Development Team - A Case Study*

Thebe (2020) conduziu um estudo de caso focado na melhoria do processo de desenvolvimento de software em uma equipe que enfrentava dificuldades com a visibilidade do progresso e a garantia da qualidade do código.

O cenário analisado envolvia uma equipe que, embora utilizasse algumas práticas de desenvolvimento, carecia de uma estrutura formal que garantisse a integração contínua e a transparência das atividades. A falta de ritos definidos gerava problemas de comunicação e dificultava o rastreamento de impedimentos, impactando a satisfação dos *stakeholders*. Para mitigar esses problemas, a pesquisa acompanhou a transição de um modelo de trabalho menos estruturado para a adoção formal das cerimônias do *Scrum*, com o objetivo de organizar o fluxo de trabalho e melhorar a colaboração interna.

Os resultados obtidos por Thebe (2020) indicaram que a formalização dos processos trouxe benefícios tangíveis para a qualidade do produto final. Observou-se que, apesar de as reuniões e ritos do *Scrum* consumirem tempo produtivo da equipe, sendo este um assunto frequentemente debatido, o ganho em visibilidade e a redução de erros no código compensaram esse custo. A pesquisa concluiu que a satisfação da equipe aumentou devido à clareza das metas, e a qualidade do software foi elevada.

Nesse contexto, este trabalho dialoga com a pesquisa de Thebe (2020) ao investigar se, em um cenário de rápida expansão de equipe, o custo temporal dos ritos ágeis também é compensado pela governança, ou se impacta negativamente a velocidade de entrega individual.

3.4 Monitoramento de métricas de qualidade e produtividade em projetos ágeis de software através da integração de dados extraídos de ferramentas de gestão e testes

Ramos (2022) desenvolveu uma pesquisa aplicada ao contexto do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB), investigando como mensurar a qualidade e a produtividade em projetos de software que não visam necessariamente o lucro financeiro, mas sim a entrega de valor social e acadêmico.

Ramos (2022) identificou que, em ambientes institucionais e acadêmicos, a gestão de projetos de software muitas vezes sofre com a falta de indicadores precisos que atestem a maturidade do processo. Diante disso, foi proposto e validado um modelo de monitoramento que integra dados extraídos automaticamente de ferramentas de gestão e ferramentas de teste estático. O objetivo era correlacionar o nível de maturidade das práticas ágeis adotadas pela equipe com a qualidade final do produto entregue.

A investigação de Ramos (2022) demonstrou que existe uma correlação direta entre a maturidade do processo de desenvolvimento e a estabilidade das entregas. O estudo evidenciou que equipes que seguem rigorosamente os ritos de monitoramento e qualidade tendem a ter uma produtividade mais previsível e menos sujeita a falhas críticas. O estudo concluiu que a automação da coleta de métricas é fundamental para a governança em projetos públicos.

Dessa forma, este trabalho se assemelha à proposta de Ramos (2022) ao utilizar métricas quantitativas em um projeto de extensão universitária, buscando validar se a estruturação do processo atua como fator determinante para a organização de uma equipe em crescimento.

3.5 Reestruturação da metodologia ágil *Scrum* numa empresa de pequeno porte

Cunha e Barbosa (2022) realizaram um estudo de caso voltado para a reestruturação dos processos de uma equipe de desenvolvimento em uma empresa de pequeno porte, que enfrentava sérios gargalos de produtividade devido à desorganização interna.

O cenário anterior à intervenção era caracterizado pelo caos produtivo: a equipe não possuía clareza sobre as prioridades, a comunicação era falha e o volume de entregas era considerado baixo para a capacidade do time. O estudo descreveu um ambiente onde a falta de metodologia resultava em desperdício de esforço e baixa motivação. A intervenção proposta consistiu na implantação rigorosa da metodologia *Scrum*, aliada ao uso de ferramentas de gerenciamento de projetos para dar transparência ao fluxo de tarefas.

Os resultados apresentados por Cunha e Barbosa (2022) foram expressivos quantitativamente. Após a reestruturação e a fase de adaptação cultural da equipe, a produtividade saltou de uma média de 8 atividades entregues para 33 atividades no mesmo intervalo de tempo. O estudo concluiu que a metodologia ágil, quando bem aplicada e suportada por ferramentas adequadas, atua como um catalisador de produtividade, organizando o esforço coletivo e eliminando o tempo ocioso gerado pela indecisão.

Diante disso, este trabalho utiliza o estudo de Cunha e Barbosa (2022) como contraponto comparativo, analisando se esse aumento de produtividade bruta se sustenta quando a variável relacionada ao tamanho da equipe aumenta drasticamente, ou se o custo de coordenação impõe limites a esse crescimento.

3.6 Comparação

A análise conjunta dos trabalhos selecionados evidencia que a adoção de metodologias ágeis se consolidou como uma estratégia fundamental para a governança de processos de software, transcendendo as barreiras de nicho ou modelo de trabalho. Enquanto Caetano (2023) e Gama *et al.* (2020) demonstram a eficácia da gestão visual e da mudança cultural em extremos opostos, como o trabalho remoto pós-pandemia e a indústria metalúrgica, Cunha e Barbosa (2022) e Thebe (2020) reforçam que a reestruturação de ritos é vital para garantir a qualidade das entregas e a satisfação dos envolvidos.

Nesse contexto, destaca-se a contribuição de Ramos (2022), que valida o monitoramento de métricas em ambientes institucionais sem fins lucrativos, aproximando-se da realidade observada no projeto de extensão analisado. Todavia, a presente pesquisa se diferencia ao investigar especificamente o impacto da rápida expansão da equipe sobre a produtividade individual através de uma abordagem estatística inferencial. Diferentemente dos estudos focados majoritariamente na satisfação qualitativa ou na entrega final do produto, este trabalho isola variáveis para compreender matematicamente o custo de coordenação inerente ao ganho de escala.

Para sintetizar essas relações e posicionar esta pesquisa frente ao estado da arte, o Quadro 3 compara os estudos mencionados. A comparação foi estruturada com base em quatro critérios fundamentais, derivados diretamente da questão de pesquisa e do cenário do NERDS:

- **Adoção de Metodologia Ágil**: Critério base para verificar a implantação de processos ágeis como ferramenta de gestão.

- **Contexto Acadêmico**: Necessário para identificar se a literatura aborda as peculiaridades de projetos universitários, como a alta rotatividade de alunos e o foco no aprendizado.
- **Uso de Métricas Quantitativas**: Utilizado para separar estudos baseados puramente em percepção qualitativa (satisfação) daqueles que, assim como este trabalho, utilizam dados numéricos para validação empírica.
- **Análise de Escala da Equipe**: O principal diferencial investigado nesta pesquisa, visando identificar estudos que abordem o impacto produtivo do aumento expressivo e rápido no número de desenvolvedores.

Quadro 3 – Comparação dos trabalhos relacionados

Trabalho	**Adoção de Metodologia Ágil**	**Contexto Acadêmico**	**Uso de Métricas Quantitativas**	**Análise de Escala da Equipe**
Caetano (2023)	X			
Gama et al. (2020)	X			
Thebe (2020)	X			
Cunha e Barbosa (2022)	X		X	
Ramos (2022)	X	X	X	
Este trabalho	**X**	**X**	**X**	**X**

Fonte: Elaborado pelo autor.

A partir do Quadro 3, evidenciam-se os pontos que diferenciam esta pesquisa das demais. Fica claro que, enquanto a maioria dos autores foca em validar se a metodologia melhorou a satisfação da equipe ou aumentou o número total de entregas, nenhum deles abordou especificamente o desafio da escala. O grande diferencial deste trabalho é preencher exatamente essa lacuna, utilizando dados estatísticos para demonstrar o comportamento da produtividade individual quando um time de desenvolvimento cresce de forma acelerada, cenário este que os estudos de caso em equipes estáveis não chegaram a cobrir.

4 METODOLOGIA

Esta seção apresenta a descrição dos procedimentos metodológicos adotados neste trabalho, ressaltando a importância da pesquisa científica, o paradigma e a abordagem escolhidos. Além disso, são detalhados o processo de coleta de dados e as etapas necessárias para a conclusão da pesquisa. Por fim, aborda-se a forma como os aspectos éticos foram tratados ao longo do estudo.

4.1 Pesquisa científica

Para Lakatos e Marconi (2017), a pesquisa científica pode ser descrita como um procedimento formal que exige pensamento reflexivo, requer tratamento científico e se constitui no caminho para conhecer a realidade ou descobrir verdades parciais. De maneira complementar, Wazlawick (2020) observa que a pesquisa científica pode ser caracterizada quanto à sua natureza, objetivos e procedimentos técnicos. Já Creswell (2009) mostra como uma pesquisa pode ser categorizada quanto à sua abordagem e reforça que o método científico é uma poderosa ferramenta para o claro entendimento do mundo.

Esta pesquisa se dedicou à apresentação dos processos de desenvolvimento adotados pelo projeto de extensão NERDS, à identificação de fraquezas nesses processos, à listagem de técnicas e práticas que poderiam melhorar o desenvolvimento e, por fim, à implantação de técnicas e processos de Engenharia de Software adaptados ao contexto dos participantes do projeto, utilizando-se de *Story Points* e do perfil dos participantes do projeto a cada recorte temporal como métrica para avaliação da produtividade.

4.2 Caracterização da pesquisa

Para Creswell (2009), o pragmatismo é um paradigma filosófico e epistemológico que serve de base para a pesquisa de métodos mistos, mas que também pode orientar pesquisas qualitativas ou quantitativas de forma prática e flexível. Morgan (2007) complementa que o pragmatismo foca nas consequências da pesquisa, nas perguntas formuladas e na utilidade dos resultados. Ele enfatiza o que funciona, em vez do que é absolutamente verdadeiro ou falso. Este foi o paradigma adotado neste trabalho, devido à sua grande flexibilidade e ao fato de não se restringir a absolutos.

Alinhado a esse paradigma, este trabalho se configurou como um estudo de caso.

Conforme Yin (2015), o método de estudo de caso é relevante ao investigar um fenômeno social contemporâneo (neste caso, o projeto NERDS) de forma ampla e profunda, permitindo compreender sua complexidade. Segundo Merriam (1998), o estudo de caso é uma "descrição holística e intensiva" de um "fenômeno limitado" (a sua unidade social), caracterizando-se por ser descritivo e heurístico (ajudando a clarear o entendimento).

O NERDS é uma ação de extensão universitária focada no desenvolvimento e reengenharia de software. O seu corpo de participantes é composto por estagiários e voluntários dos cursos de Ciência da Computação e Engenharia de Software de diferentes semestres da UFC campus Russas. As atividades centrais do projeto envolvem desenvolvimento e a manutenção de softwares para repartições públicas, com o objetivo de facilitar as atividades administrativas em torno da esfera educacional. Este contexto, com suas dinâmicas e participantes específicos, constitui a unidade de análise ("o caso") desta investigação.

Para avaliar o impacto da implementação de metodologias e práticas ágeis dentro deste caso, o estudo adotou uma abordagem quantitativa, caracterizando-se como uma pesquisa explicativa. Foi utilizado um delineamento quasi-experimental, com base em recortes temporais (antes e depois da intervenção). A produtividade, mensurada por *story points*, foi comparada entre o período pré-intervenção (funcionamento *ad hoc*) que vai de Abril de 2024 a Outubro de 2024 e o período pós-intervenção que vai de Novembro de 2024 a Dezembro de 2025, controlando-se estatisticamente por variáveis relacionadas ao perfil dos participantes envolvidos em diferentes momentos do projeto.

4.3 Coleta de dados

Os dados para a progressão da pesquisa foram coletados, primeiramente, por meio de diálogos e reuniões com os membros e a coordenação do projeto, a fim de identificar e esclarecer o funcionamento do processo de desenvolvimento utilizado até então. Adicionalmente, também foram realizadas reuniões com os membros da PREX, com o objetivo de detalhar o funcionamento do GEX e estabelecer tarefas para o time de desenvolvimento.

Foi também aplicado um questionário de satisfação, disponível no APÊNDICE A, aos participantes que estiveram antes do período de pré-intervenção e participaram das mudanças e implantação no processo de desenvolvimento. Cabe ressaltar que foi necessário utilizar um Termo de Consentimento Livre e Esclarecido (TCLE), com o intuito de fornecer clareza quanto aos objetivos da pesquisa e informar aos participantes que os dados fornecidos foram utilizados

de forma confidencial, assegurando o anonimato do respondente.

Além disso, foram realizadas rodadas de *Planning Poker* com os membros do projeto, utilizando o software *PlanITpoker*[1] para a atribuição de pontos a cada tarefa presente no *backlog* do GEX, com o propósito de obter uma métrica clara e confiável para determinar a grandeza das tarefas do projeto.

4.4 Etapas da pesquisa

A metodologia adotada nesta pesquisa baseia-se em conceitos e práticas da Engenharia de Software, com ênfase nas áreas de Engenharia de Requisitos e Gerenciamento de Projetos, seguindo o fluxo representado na Figura 13.

A subseção 4.4.1 trata das atividades voltadas à identificação de fraquezas no processo de desenvolvimento do NERDS, bem como à enumeração de técnicas e metodologias que possam auxiliar no enfrentamento dessas fragilidades.

Na subseção 4.4.2, é detalhado o processo de levantamento das tarefas realizadas antes da aplicação de qualquer técnica ou metodologia da Engenharia de Software, tendo como artefato de saída a lista de tarefas concluídas com seus respectivos *Story Points*.

Durante a subseção 4.4.3, é descrito o processo de implantação das técnicas e práticas selecionadas na etapa inicial. Além disso, na subseção 4.4.4, é abordado como ocorrerá a especificação das tarefas para o time de desenvolvimento, sendo os *Story Points* dessas tarefas os principais artefatos de saída.

Além disso, na subseção 4.4.5, foi realizado uma pesquisa de satisfação, por meio de um questionário, disponível no APÊNDICE A, com os membros que estiveram antes e depois do período de intervenção, com o objetivo de levantar mais dados quantitativos relacionados a satisfação, além dos que serão extraídos por meio de experimentação com os dados advindos dos *Story Points* de cada tarefa concluída e o perfil do grupo de participantes que estiveram em volta a sua conclusão.

Por fim, a subseção 4.4.6 apresenta o processo de extração e análise dos resultados, levando em consideração os *Story Points* obtidos na segunda e quarta etapas, o perfil dos membros da equipe de desenvolvimento nos quatro recortes temporais diferentes utilizados neste trabalho, que são:

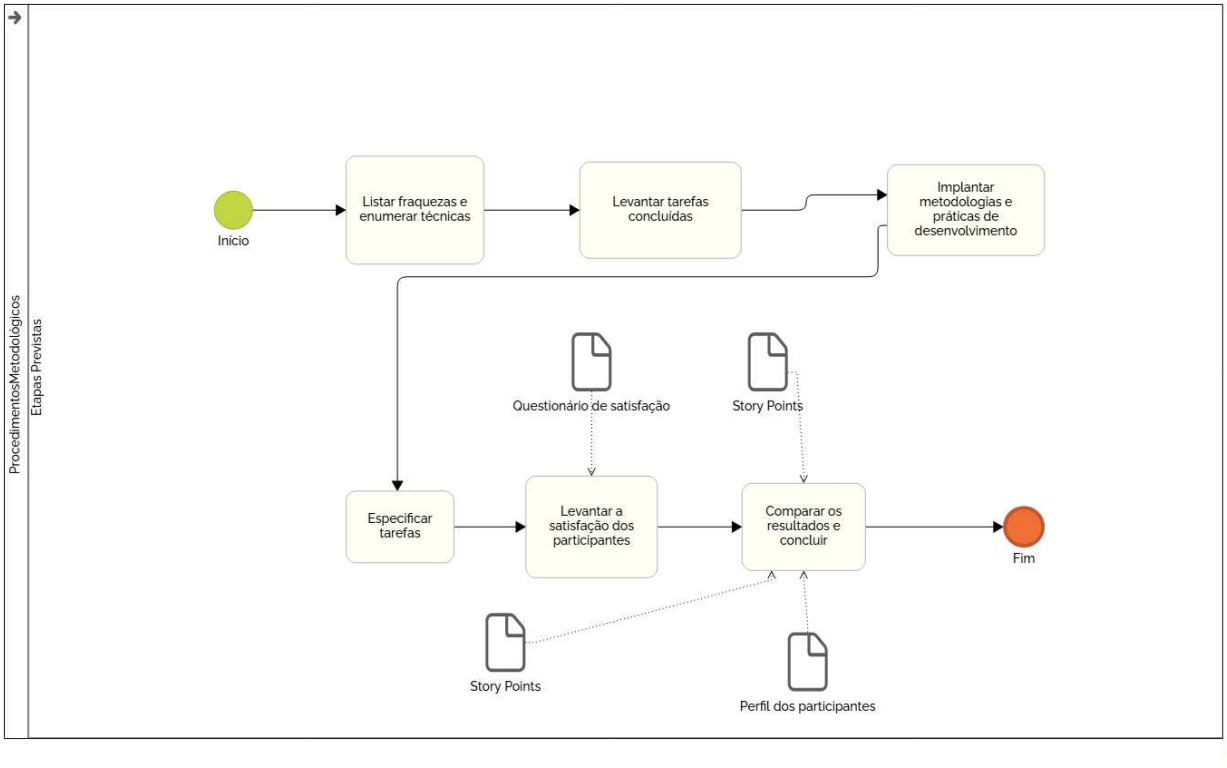
- Antes da aplicação da metodologia ágil: Abril de 2024 a Outubro de 2024

[1] Disponível em: <https://planitpoker.com>

- Semestre 2024.2: Novembro de 2024 a Fevereiro de 2025
- Semestre 2025.1: Março de 2025 a Junho de 2025
- Semestre 2025.2: Agosto de 2025 a Novembro de 2025

De maneira complementar, a pesquisa de satisfação quantitativa serviu como uma forma secundária de extração dos resultados, além do experimento principal, com o intuito de se obter resultados mais completos e amplos.

Figura 13 – Etapas da pesquisa



Fonte: Elaborada pelo autor.

4.4.1 Listar fraquezas e enumerar técnicas

Para alcançar os objetivos 1, 2 e 3 descritos na Seção 1.2, foi necessário dialogar com a coordenação e os membros do time de desenvolvimento do NERDS, com o intuito de identificar quais processos estão sendo empregados, suas fraquezas inerentes e o que pode ser aprimorado, considerando o contexto em que os participantes estão inseridos.

Adicionalmente, foram listadas e selecionadas metodologias, como as exemplificadas na Seção 2.2, com o claro objetivo de mitigar as fraquezas identificadas. Da mesma forma, foi preciso enumerar e selecionar técnicas de Engenharia de Software, como as abordadas na Subsessão 2.3, para a definição de requisitos que, posteriormente, foram transformados em

tarefas atribuídas à equipe de desenvolvimento. Esta etapa foi realizada em novembro de 2024.

4.4.2 Levantar tarefas concluídas

Também foi necessário identificar e especificar todas as tarefas que foram concluídas antes de qualquer mudança no processo de desenvolvimento. Para isso, foi preciso, mais uma vez, dialogar com a coordenação e os membros do projeto, a fim de levantar todas as atividades que foram plenamente realizadas.

Cabe destacar que foi necessário realizar rodadas de *Planning Poker* com os membros do projeto, com o objetivo de obter a respectiva pontuação de cada tarefa realizada, facilitando a posterior comparação com os resultados obtidos após a aplicação das técnicas propostas. Esta etapa foi realizada em julho de 2025.

4.4.3 Implantar metodologias e práticas de desenvolvimento

Antes de realizar qualquer implantação, foi necessário preparar os membros do projeto, tornando-os conscientes das metodologias ou práticas selecionadas na subseção 4.4.1. Para isso, foram realizadas reuniões de preparação e alinhamento, com o objetivo de explicar e exemplificar o novo processo de desenvolvimento adotado pela equipe do NERDS.

Além disso, foi fundamental escolher uma ferramenta confiável de gerenciamento de projetos, que possa auxiliar tanto a equipe quanto a própria pesquisa ao longo da execução. Por fim, o novo processo de desenvolvimento foi plenamente implantado. Esta etapa foi realizada em novembro de 2024.

4.4.4 Especificar tarefas

Com o novo processo de desenvolvimento em plena execução, tornou-se necessário especificar as tarefas que constituíram o *backlog* do GEX, as quais guiaram o time de desenvolvimento em suas atividades.

Cabe ressaltar que, uma vez implantado o novo processo, essa etapa deve ser executada continuamente até o término do trabalho, tendo seu início em novembro de 2024. O artefato gerado, composto pelos *Story Points* de cada tarefa, foi utilizado para a obtenção dos resultados e a elaboração das conclusões.

4.4.5 Levantar a satisfação dos participantes

Foi divulgado um questionário de satisfação com os participantes que estiveram antes e depois da implantação adaptada do novo processo de desenvolvimento, com o objetivo claro de extrair informações quantitativas com os membros que passaram pela mudança, e portanto, contribuem de maneira significativa ao expor sua opinião.

4.4.6 Comparar os resultados e concluir

Para alcançar o objetivo 4 desta pesquisa, descrito na seção 1.2, foi necessário utilizar os *Story Points* obtidos nas etapas 2 e 4, o perfil de cada grupo de participantes a cada período de tempo, e, por fim, considerar o recorte temporal, anterior e posterior à aplicação do novo processo de desenvolvimento, assim como seu grupo de participantes.

Para isso, foi utilizada a ferramenta JASP[2], um programa gratuito e de código aberto para análise estatística, desenvolvido pela Universidade de Amsterdã, que foi fundamental para a obtenção dos resultados e para a elaboração das conclusões da pesquisa.

4.5 Aspectos éticos da pesquisa

Este trabalho respeita os princípios éticos conforme a Lei nº 13.709/2018, também conhecida como Lei geral de proteção de dados (LGPD), que visa proteger os direitos fundamentais de liberdade, privacidade e a livre formação da personalidade de cada indivíduo.

Todos os participantes foram informados sobre os objetivos da pesquisa, assim como sobre a utilização dos dados coletados, assegurando sua participação de forma voluntária e anônima. Para isso, será apresentado um Termo de Consentimento Livre e Esclarecido (TCLE), disponível no APÊNDICE A, que explica os objetivos da pesquisa e garante o direito de recusa ou desistência a qualquer momento, sem quaisquer consequências.

Os dados obtidos foram utilizados exclusivamente para fins acadêmicos, mantidos sob sigilo e tratados de forma agregada, sem identificação individual dos participantes. Não foram coletadas informações sensíveis ou que permitam a identificação direta das pessoas envolvidas. Todos os procedimentos seguiram os princípios éticos em pesquisa, pautados na responsabilidade, transparência e respeito aos participantes.

[2] Disponível em: <https://jasp-stats.org/>

5 RESULTADOS

Esta seção apresenta a descrição da análise dos resultados obtidos durante a execução do trabalho, com ênfase na listagem de fraquezas encontradas no processo de desenvolvimento do NERDS entre abril e outubro de 2024, listagem de técnicas que possam superar essas fraquezas, levantamento de todas as atividades concluídas antes e após a intervenção em novembro de 2024 até setembro de 2025, descrição detalhada da metodologia adotada após o período *ad hoc*, e por fim, o resultado da análise quantitativa de tarefas realizadas em cada período de tempo no desenvolvimento do GEX, junto à análise quantitativa em relação à satisfação dos membros que estiveram no período antes e após intervenção.

5.1 Fraquezas do processo de desenvolvimento do NERDS

Durante o mês de novembro de 2024, foram realizadas diversas conversas com o coordenador do projeto, o professor Marcos Vinicius, e o time de desenvolvimento, com o objetivo de identificar fraquezas no processo de desenvolvimento até então utilizado, que serão aprofundadas a seguir.

5.1.1 Fraquezas relacionadas ao planejamento

O primeiro grupo de fraquezas imediatamente identificado foi relacionado à capacidade de planejamento que o projeto tinha até então, que o afetava a longo prazo e pontualmente, sendo os seguintes tópicos:

- **Baixa previsibilidade**: Sem métricas, estimativas baseadas em fontes confiáveis e um planejamento formal, era muito difícil estabelecer quando determinada entrega poderia ficar pronta no contexto que o projeto se encontrava
- **Gerenciamento de escopo**: Sem um *backlog* priorizado, novas funcionalidades eram aceitas para desenvolvimento de maneira desordenada, sem uma compreensão clara do objetivo por trás do desenvolvimento do GEX.
- **Reatividade acima da proatividade**: Sem um processo claro, a equipe acabava mais por corrigir problemas ou retornar a funcionalidades vagamente definidas, ao invés de planejar com clareza quais seriam os próximos passos.

Uma vez identificadas as fraquezas em relação a planejamento, podemos focar em um conjunto de fraquezas menos óbvias ao time de desenvolvedores: o quão bem eles conseguem

administrar o conhecimento em relação ao que o GEX se propõe a fazer.

5.1.2 Fraquezas relacionadas à gestão do conhecimento

O segundo grupo de fraquezas identificado, foi relacionado a como o conhecimento em relação ao funcionamento do GEX, em termos de regras, acordos e outras condições eram administradas e compartilhadas pelo time, com isso, surgiram os seguintes tópicos:

- **Conhecimento tribal**: O conhecimento sobre como o sistema funcionava ficava centralizado em alguns indivíduos que escreviam os códigos, que por vezes não coincidiam. Se esses indivíduos saem do projeto, o conhecimento vai embora junto com eles.
- **Documentação insuficiente**: A única documentação existente no projeto, era o próprio código do GEX em si, o que poderiam gerar dúvidas, confusões e dificultar a entrada de novos membros.
- **Curva de aprendizagem elevada**: Novos integrantes precisariam explorar o sistema a fundo e ler o código de ponta a ponta, antes de fato poderem ser produtivos.

Uma vez adquirida a consciência do conhecimento fragmentado que até então o projeto tinha, é necessário analisar como todos os pontos citados podem afetar a qualidade do software que estava sendo desenvolvido.

5.1.3 Fraquezas relacionadas à qualidade de software

Sommerville (2019) define a qualidade de software como sua adequação ao uso, que no contexto do GEX engloba professores, alunos e qualquer tipo de pessoa interessada nas atividades universitárias. Dessa maneira, os principais problemas identificados em relação a qualidade são:

- **Dívida técnica exponencial**: Como a prioridade era sempre a entrega, sem documentação suficiente, o retrabalho de voltar em funcionalidades já existentes poderia ser muito demorado e prejudicial
- **Ausência de critérios de aceitação**: Sem critérios bem definidos, quaisquer tipos de funcionalidades poderiam ser aceitas ao sistema, levando a possibilidade de comprometer a satisfação que o usuário poderia ter ao utilizar a aplicação.
- **Testes insuficientes**: Sem um esquema de testes estruturado, o sistema acabava por ter mau funcionamento em diversas partes, aumentando ainda mais o retrabalho em torno do desenvolvimento.

Por fim, é necessário identificar um tipo de problema mais íntimo aos membros do projeto, que é como eles se comunicam entre si, e como se posicionam em relação ao desenvolvimento.

5.1.4 Fraquezas relacionadas à comunicação

O último grupo de problemas identificados, foi relacionado à comunicação, como eles compartilham o conhecimento do que deve ser feito e o que cada membro está fazendo no momento. Dessa maneira, foram identificados os seguintes pontos:

- **Vácuos de informação**: A comunicação ocorria de maneira informal, fazendo com que parte da equipe não soubesse o que a outra estava fazendo, gerando ainda mais retrabalho
- **Indefinição de papéis**: Não estava claro quais eram as responsabilidades de cada membro dentro do projeto, alguns alegavam que atuavam como "faz-tudo", o que acabava por gerar conflitos.

Ademais, é necessário listar quais técnicas já definidas na literatura podem superar essas fraquezas, e com isso, montar um modelo de processo de desenvolvimento adaptado ao contexto da ação de extensão que o GEX está.

5.2 Identificação das metodologias para mitigação das fraquezas

Para solucionar as fraquezas diagnosticadas na Seção 5.1, é necessário reunir técnicas, metodologias e práticas que atuem especificamente sobre cada um dos pontos citados. O objetivo é que, quando integradas, essas abordagens estabeleçam um processo maduro e sustentável para o time de desenvolvimento do NERDS. O primeiro passo consiste em definir qual metodologia servirá de alicerce para o projeto, estabelecendo o macro-processo antes de abordar as práticas específicas.

5.2.1 Metodologias de Desenvolvimento

Existem diversas metodologias aplicáveis a times de desenvolvimento de software. Contudo, considerando o contexto em que o NERDS se encontrava, duas abordagens se destacam como fundamentais:

- ***Scrum***: Ao definir ciclos iterativos claros (*Sprints*), o *Scrum* endereça diretamente os problemas de gerenciamento e planejamento de longo prazo descritos na Seção 5.1.1.

Além disso, fornece a base estrutural para a definição de responsabilidades, mitigando a indefinição de papéis citada na Seção 5.1.4.

- **Kanban**: Focado na visualização do fluxo de trabalho, o Kanban auxilia na resolução dos tópicos da Seção 5.1.1 ao tornar explícitas as atividades em andamento e organizar o escopo. Adicionalmente, ao evidenciar "quem está fazendo o quê", combate os silos de informação descritos na Seção 5.1.4.

Uma vez definido o arcabouço metodológico, é necessário listar técnicas que resolvam problemas específicos que o escopo macro da metodologia, por si só, pode não detalhar suficientemente.

5.2.2 Técnicas para gestão de escopo e planejamento

O gerenciamento eficiente do escopo e um planejamento claro dos próximos passos são essenciais para o sucesso do software. As seguintes técnicas foram selecionadas para auxiliar o NERDS a superar esses obstáculos:

- **Histórias de Usuário (*User Stories*)**: Ao padronizar a documentação focando no "quem", "o que" e "por que", as histórias de usuário combatem a fragmentação do conhecimento (Seção 5.1.2). Simultaneamente, mitigam o risco de dívida técnica (Seção 5.1.3) e fornecem o insumo inicial para resolver a desorganização do planejamento (Seção 5.1.1).
- ***Planning Poker***: Como o produto final desta técnica são tarefas pontuadas com base em sua complexidade relativa, o *Planning Poker* auxilia nos pontos da Seção 5.1.1 ao fornecer estimativas consensuais e mais confiáveis para o time de desenvolvimento.

Com o gerenciamento de escopo e o planejamento endereçados já bem estabelecidos, volta-se a atenção para a garantia da qualidade e a disseminação do conhecimento técnico entre os membros do projeto.

5.2.3 Técnicas para qualidade de software e gestão do conhecimento

Pressman e Maxim (2021) definem a qualidade de software como a conformidade com requisitos explícitos e implícitos. Para atingir esse patamar, é necessário estabelecer práticas visíveis a todos os participantes do projeto:

- **Critérios de Aceitação**: Cada história de usuário deve incluir critérios objetivos que determinam se a implementação é satisfatória. O objetivo principal é estabelecer regras claras para o sistema, auxiliando na gestão do conhecimento (Seção 5.1.2) e suprindo a

ausência de critérios de qualidade (Seção 5.1.3).

- **Revisão por Pares (*Code Review*)**: Ao garantir que pelo menos duas pessoas inspecionem um trecho de código (o autor e o revisor), essa técnica combate diretamente o "conhecimento tribal" citado na Seção 5.1.2, promovendo a propriedade coletiva do código.
- **Testes**: Verificar a conformidade do código construído em relação à documentação é crucial para elevar a qualidade do software, mitigando as falhas apontadas na Seção 5.1.3.

Por fim, é imperativo garantir que o time mantenha uma comunicação fluida e esteja alinhado sobre o andamento do projeto, com o objetivo de estabelecer uma comunicação clara e objetiva entre seus membros.

5.2.4 Técnicas de comunicação

Para o bom andamento do projeto, a comunicação deve ser transparente tanto sobre o estado atual quanto sobre as perspectivas futuras. Para isso, destacam-se as seguintes cerimônias (originárias do *Scrum*):

- ***Daily Scrum***: Reuniões diárias curtas onde os membros compartilham o progresso, o planejamento do dia e eventuais impedimentos. Que em projetos onde os membros têm diferentes horários, pode ser realizada por meio de mensagens de textos por diferentes meios de comunicação, como *Discord* e *WhatsApp*. É a técnica central para resolver as falhas de comunicação citadas na Seção 5.1.4.
- ***Sprint Review***: Reunião realizada ao final de uma *Sprint* para inspecionar o incremento do produto. É ideal para alinhar o conhecimento sobre o que foi produzido (Seção 5.1.2) e validar as entregas.
- ***Sprint Retrospective***: Reunião focada na inspeção do processo de trabalho, e não do produto. É o momento fundamental para discutir melhorias na dinâmica da equipe, atacando os conflitos e indefinições da Seção 5.1.4.

Com esse embasamento, torna-se possível descrever em detalhes o modelo de desenvolvimento adaptado adotado pelo NERDS para a construção do GEX entre novembro de 2024 e novembro de 2025, evidenciando como cada prática foi orquestrada para sanar as fraquezas da Seção 5.1.

5.3 O novo processo de desenvolvimento

Com o objetivo de solucionar as fraquezas identificadas na Seção 5.1, e impulsionar a produtividade dentro do time, a metodologia adotada para o desenvolvimento do GEX foi o *Scrumban*, que combina a estrutura do *Scrum* com o fluxo de tarefas contínuas do Kanban

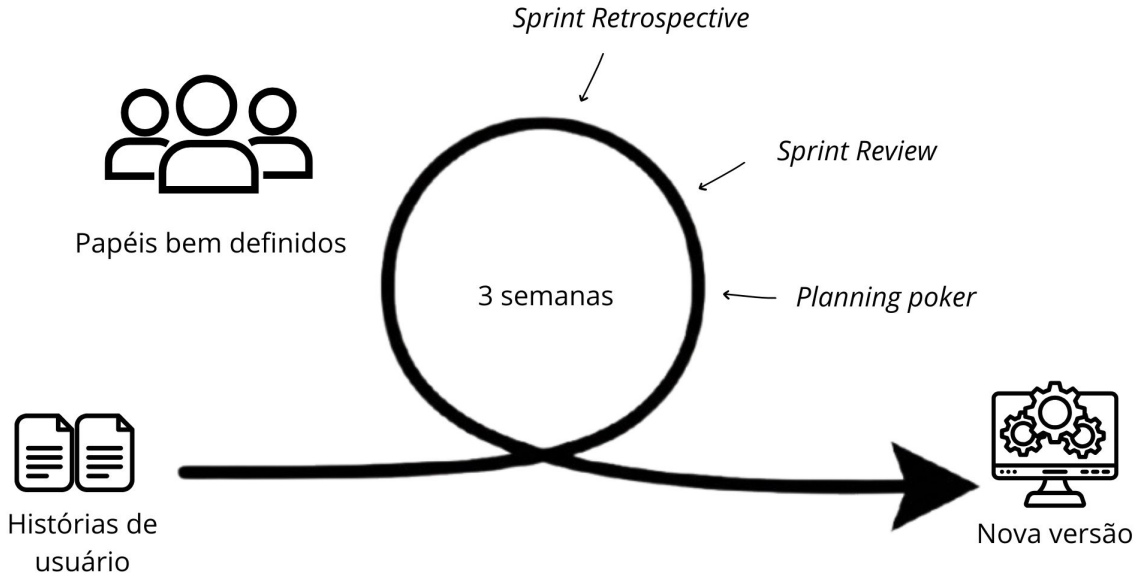
5.3.1 O lado Scrum

Na Figura 14 é possível visualizar a estrutura do processo de desenvolvimento utilizada pelo GEX, onde um ciclo começa com um conjunto de histórias de usuários do *backlog* do produto, cada uma delas com critérios de aceitação bem definidos e elaborados pelos times de requisitos e qualidade, que são trabalhadas pelo time de desenvolvimento durante um período de 3 semanas.

Durante esse intervalo, ocorrem alguns eventos, o primeiro deles, o *Planning Poker*, onde são reunidos alguns membros do time, que juntos representem diferentes partes do todo, como *backend*, *frontend*, design e qualidade, em que histórias de usuário são debatidas e atreladas pontos a elas para o próximo ciclo.

Também, ao final da *Sprint*, são realizados dois eventos, o *Sprint review* e o *Sprint retrospective*, no primeiro, é feita uma visão geral do que ocorreu na *Sprint*, assim como são debatidos quais serão os próximos itens a serem desenvolvidos para o próximo ciclo, já na segunda, é debatido o processo de ciclo desenvolvimento em si, o que deu certo, o que não deu certo e coleta de sugestões, com o objetivo de aperfeiçoar ainda mais o processo utilizado. Ao final, uma nova versão do GEX é desenvolvida por um time com papéis bem estabelecidos.

Figura 14 – Fluxo do processo *Scrum* no GEX



Fonte: Elaborada pelo autor.

Com o esqueleto claro do *Scrum*, já é possível visualizar como o fluxo de trabalho do Kanban pode entrar e contribuir para refinar ainda mais o ciclo de vida, ao especificar a ordem das tarefas e responsabilidades de cada indivíduo dentro do projeto.

5.3.2 O lado Kanban

Na Figura 15 é detalhado como funciona o fluxo Kanban utilizado no GEX, representado pela Figura 16. Tudo começa com a necessidade de uma nova tarefa, que caso ela não tenha sido trabalhada antes, a tarefa vai para a coluna "*Backlog*", onde sua história de usuário associada é trabalhada pelo time de requisitos, mas caso ela já tenha sido trabalhada, ela vai para a coluna "Reabertura", onde sua história de usuário associada é reprojetada para atender as novas necessidades.

Uma vez que a atuação do time de requisitos se encerra, a tarefa vai para as mãos do time de Design, que elabora o protótipo daquela nova ou alterada funcionalidade, uma vez acabada, ela vai para o time de desenvolvimento, inicialmente para a coluna "Pronto para Dev" (abreviada na figura como "P. para dev"), que representa tarefas cuja escrita de código já está pronta para ser iniciada, e assim, uma vez que começa, a tarefa vai para a coluna "Fazendo".

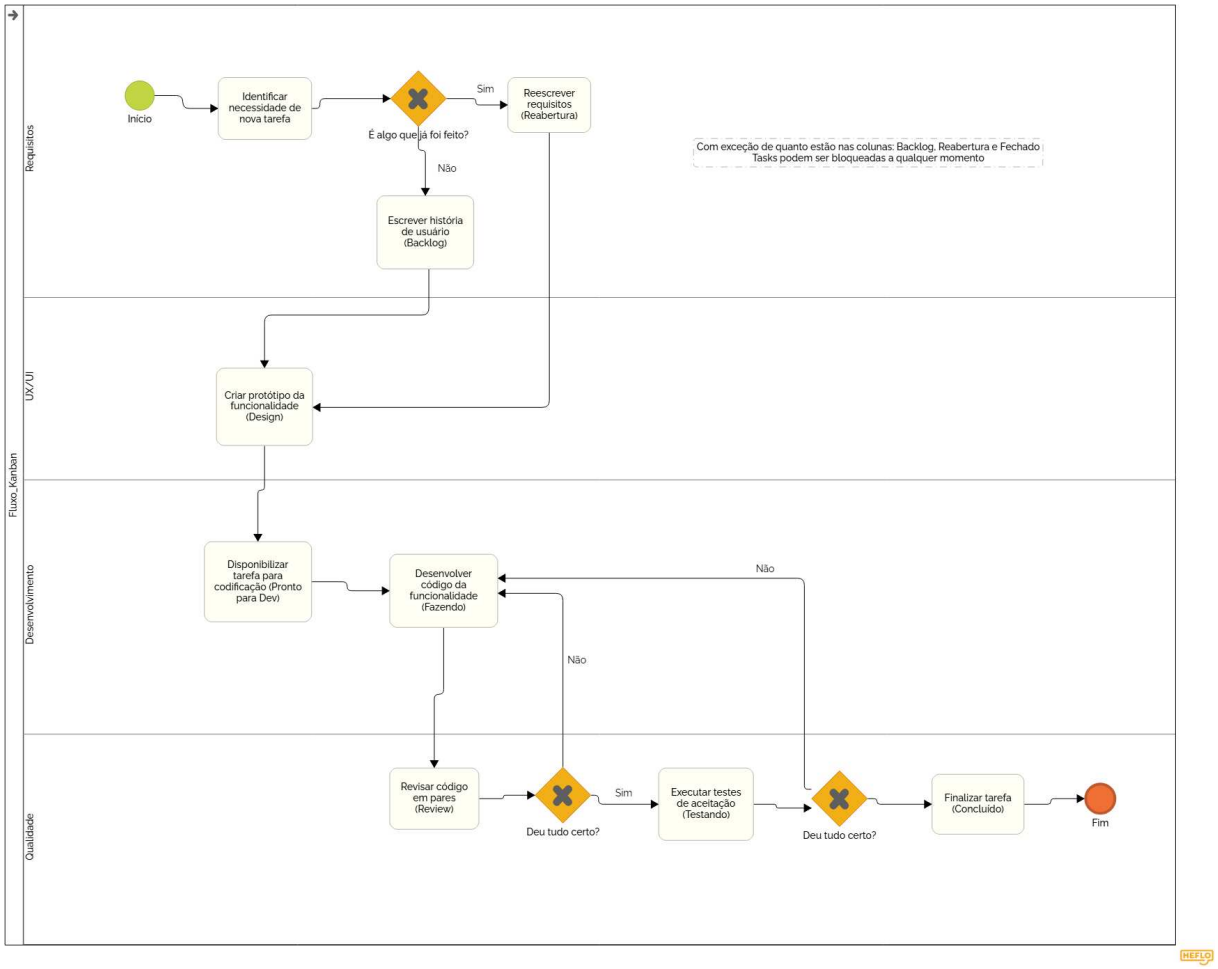
Assim que o código é terminado, é realizada uma revisão por um outro desenvolvedor ou membro do time de qualidade, caso o código esteja atendendo os critérios de aceitação

descritos em sua história de usuário, ele passa para a coluna “Testando”, se não, volta para a coluna “Fazendo”.

Na coluna “Testando”, os analistas de qualidade do projeto vão testar o código em diferentes cenários, auxiliados pelos critérios de aceitação da história de usuário associada àquele código, caso o código passe nos testes, a tarefa é dada como feita e movida para a coluna “Fechado”, se não, ela retorna para a coluna “Fazendo”.

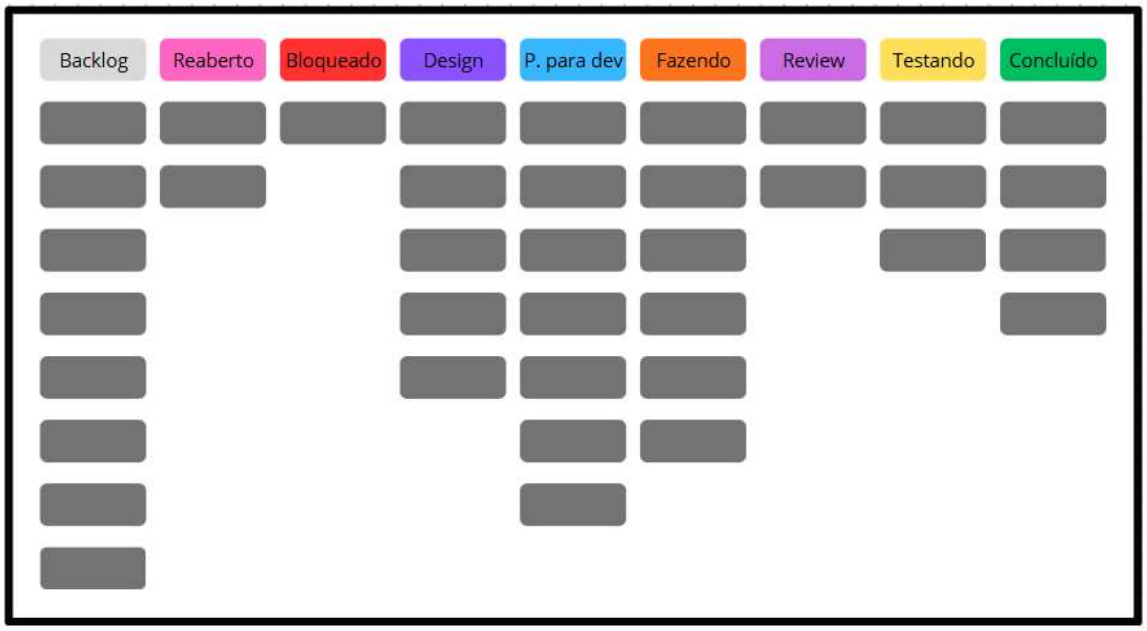
Cabe ressaltar, que uma tarefa, a menos que esteja na coluna “*Backlog*” ou “Reabertura” ou “Fechado”, pode ser bloqueada a qualquer momento por diversos motivos, como alteração na história de usuário, descoberta tardia de pré-requisitos, entre outros.

Figura 15 – Fluxo do processo Kanban no GEX



Fonte: Elaborada pelo autor.

Figura 16 – Quadro Kanban do GEX



Fonte: Elaborada pelo autor.

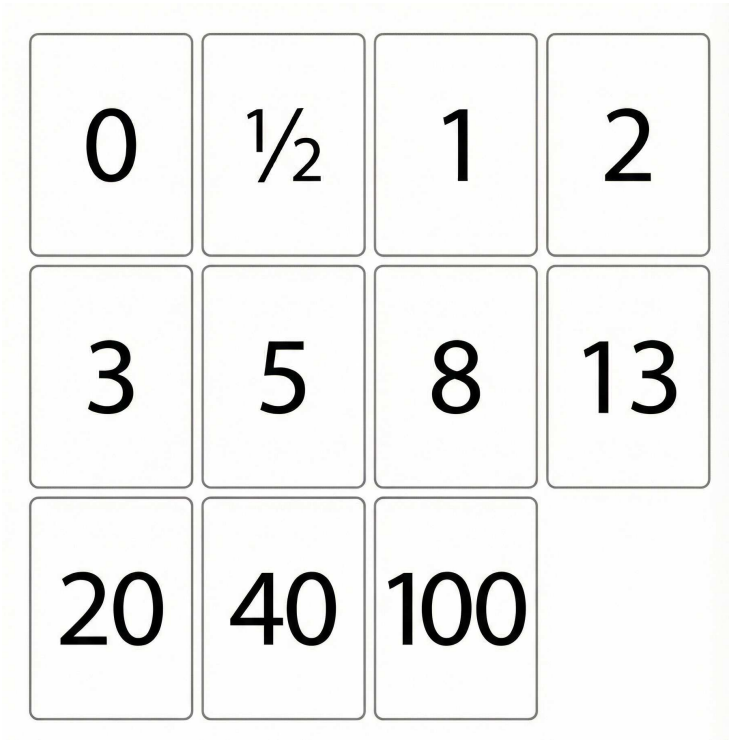
Com a metodologia de desenvolvimento bem estabelecida, já é possível alavancar todas as diferentes tarefas feitas com diferentes grupos nos diferentes períodos de desenvolvimento que o GEX possuiu.

5.4 Tarefas realizadas entre novembro de 2024 a novembro de 2025

Diversas tarefas foram distribuídas para o time de desenvolvimento entre os períodos de novembro de 2024 a novembro de 2025, e a pontuação de cada tarefa foi determinada por meio da técnica *Planning poker* em encontros pontuais com o time de desenvolvimento, a escala utilizada pode ser encontrada na Figura 17, onde cada pontuação tem o determinado significado:

- **0**: Item tão pequeno que não vale a pena realizar estimativa
- **1/2**: Item muito pequeno.
- **1, 2 e 3**: Itens pequenos.
- **5 e 8**: Itens médios.
- **13, 20 e 40**: Itens grandes.
- **100**: Item muito grande.

Figura 17 – Escala *Scrum*



Fonte: Elaborada pelo autor.

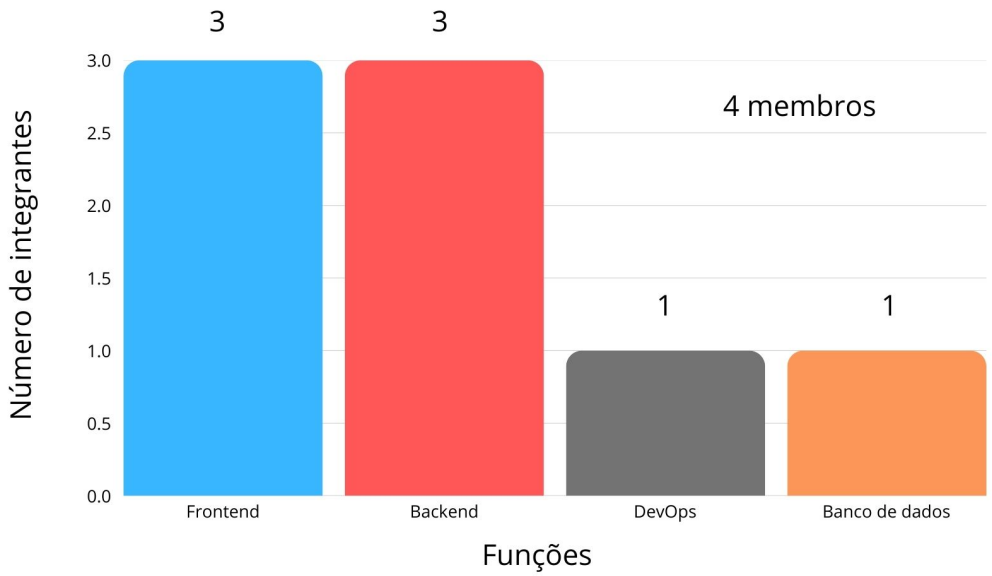
Os dados que caracterizam o perfil de cada um dos participantes e a pontuação de cada tarefa realizada pode ser encontrada no APÊNDICE B. Dessa maneira, as subseções seguintes descrevem o perfil do grupo de participantes e as tarefas realizadas por eles para cada um dos períodos de tempos observados e analisados durante a execução do trabalho.

5.4.1 Período pré-ágil

No intervalo entre abril de 2024 e outubro de 2024, não havia uma estrutura formal de desenvolvimento para o software GEX. Observa-se no Gráfico 2 que, embora houvesse apenas quatro participantes ativos, existiam oito funções distribuídas na equipe. Isso indica um acúmulo de responsabilidades, sugerindo que os membros atuavam de forma generalista e sem clareza sobre seus papéis específicos além da codificação.

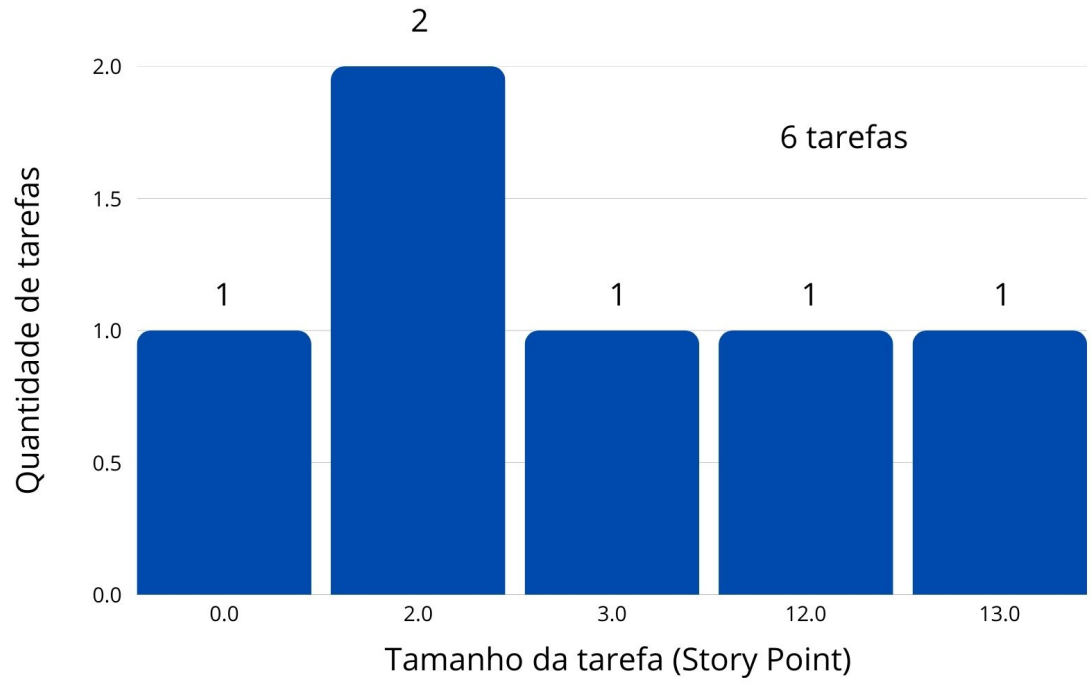
De maneira complementar, o Gráfico 3 evidencia uma acentuada variação na complexidade das entregas, oscilando entre tarefas triviais e complexas (baseado nos Story Points). Apesar do baixo volume de atividades, essa disparidade aponta para dificuldades iniciais do time na identificação do escopo e na decomposição eficiente das tarefas em unidades menores de trabalho.

Gráfico 2 – Perfil dos participantes durante o período pré-ágil



Fonte: Elaborado pelo autor.

Gráfico 3 – Tarefas realizadas durante o período pré-ágil



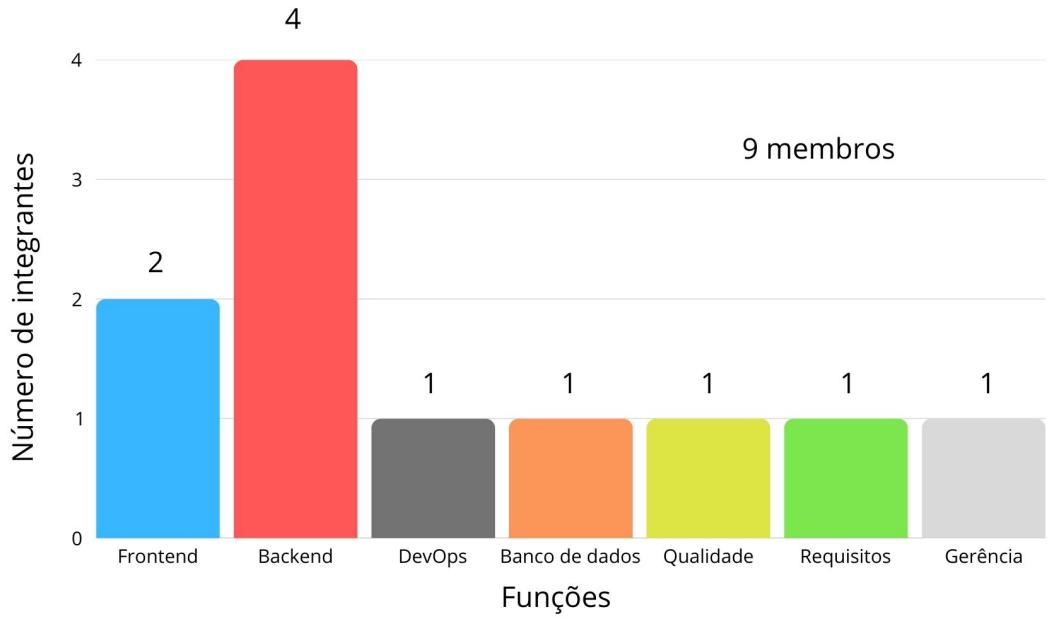
Fonte: Elaborado pelo autor.

Após esse período, iniciou-se a transição para a metodologia ágil. Nessa nova fase, o processo evoluiu e amadureceu à medida que foi executado. Nas subseções seguintes, são detalhados os resultados obtidos após essa mudança.

5.4.2 Semestre 2024.2

Neste ciclo, que vai de novembro de 2024 a fevereiro de 2025, a equipe expandiu para nove integrantes, como mostrado pelo Gráfico 4, marcando a primeira especialização formal de papéis. Diferentemente do período anterior, houve a introdução das figuras do Analista de Qualidade, Analista de Requisitos e Gerente de Projeto, permitindo que os desenvolvedores focassem na implementação.

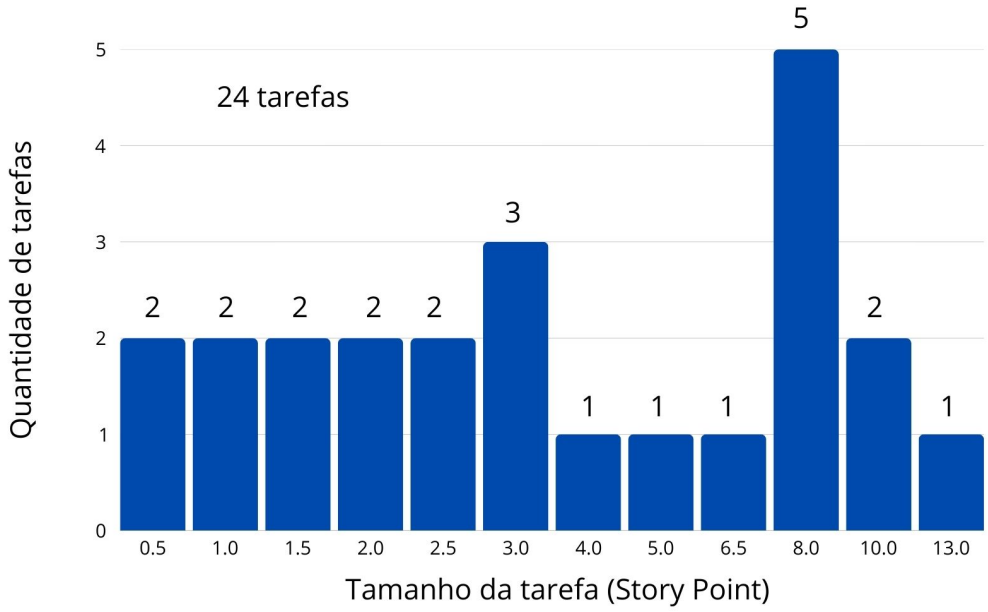
Gráfico 4 – Perfil dos participantes durante o semestre 2024.2



Fonte: Elaborado pelo autor.

Essa reorganização refletiu-se nos resultados: este foi o período de maior produtividade relativa do projeto. O Gráfico 5 demonstra uma concentração de entregas na faixa de 8 *Story Points*. Embora 8 pontos seja considerado uma tarefa de alta complexidade no *Scrum*, a capacidade do time de entregar múltiplos itens desse tamanho sugere que a especialização dos papéis removeu impedimentos operacionais, permitindo um fluxo de trabalho contínuo

Gráfico 5 – Tarefas realizadas durante o semestre 2024.2



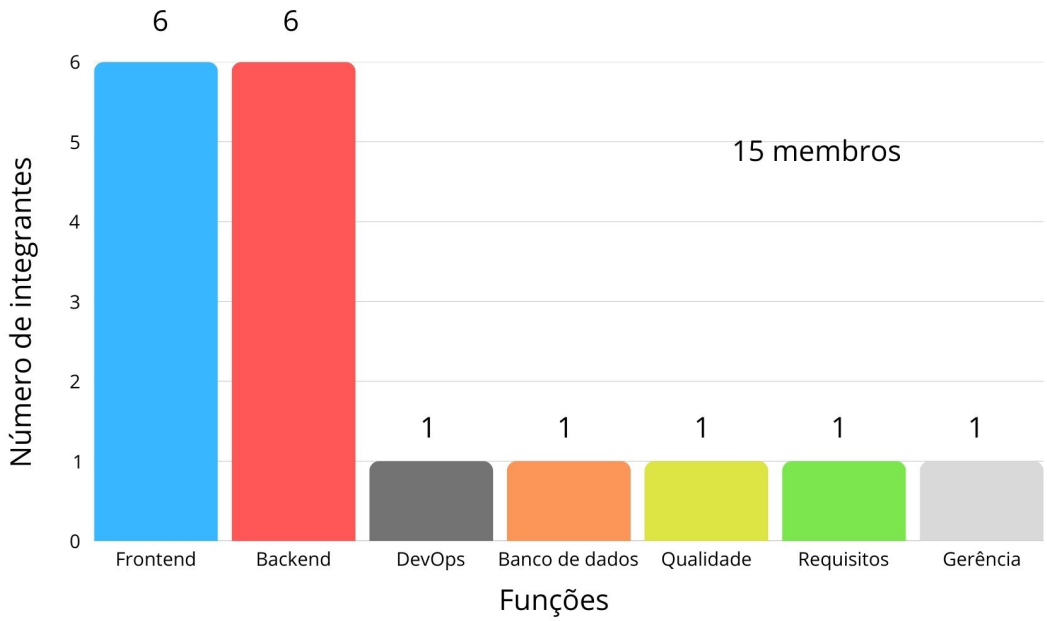
Fonte: Elaborado pelo autor.

5.4.3 Semestre 2025.1

No primeiro semestre de 2025, que foi de março a junho, a equipe cresceu para 15 participantes, com um forte aporte de novos desenvolvedores *front-end* e *back-end* como mostrado pelo Gráfico 6. Contudo, o aumento numérico não se traduziu em aumento proporcional de entrega.

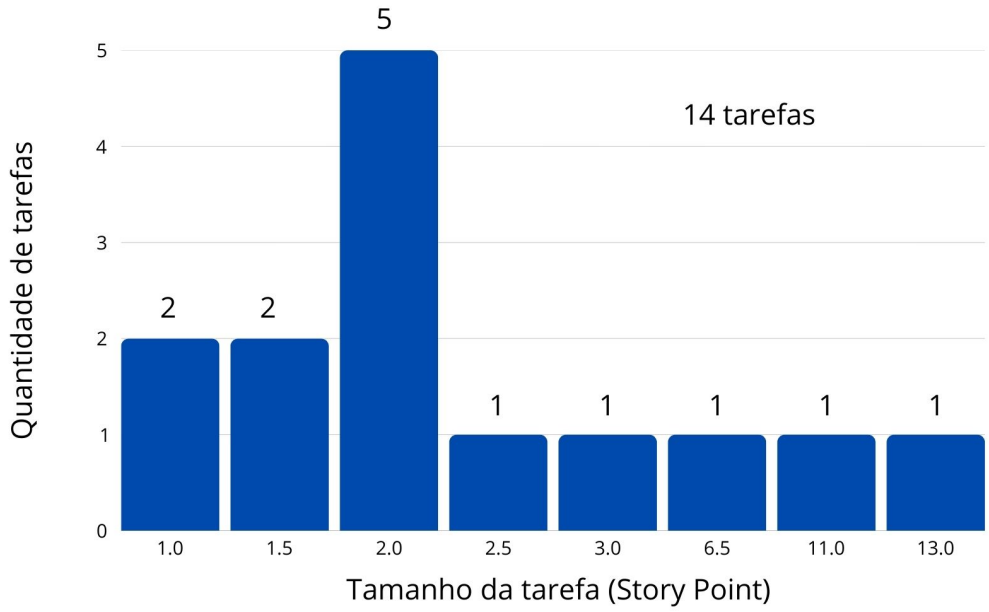
A análise da distribuição de tarefas no Gráfico 7 revela um fenômeno de "atomização": a maior frequência de entregas ocorreu em tarefas de pontuação mínima (2 pontos). Simultaneamente, a produtividade individual sofreu uma queda abrupta. Isso sugere que o esforço de integração de novos membros e a fragmentação excessiva do escopo geraram um custo de coordenação superior à capacidade de produção.

Gráfico 6 – Perfil dos participantes durante o semestre 2025.1



Fonte: Elaborado pelo autor.

Gráfico 7 – Tarefas realizadas durante o semestre 2025.1

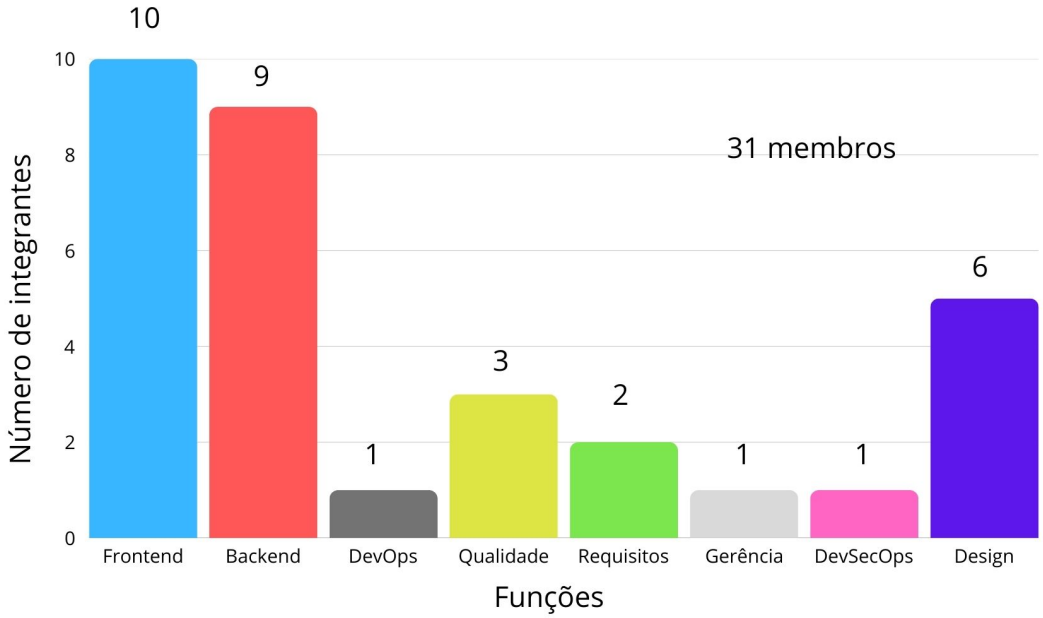


Fonte: Elaborado pelo autor.

5.4.4 Semestre 2025.2

O último período analisado compreende um recorte temporal de dois meses (setembro a novembro de 2025) de um fluxo de trabalho contínuo. No Gráfico 8, a equipe atingiu seu ápice numérico com 31 integrantes, introduzindo uma equipe design e reforçando a equipe de qualidade.

Gráfico 8 – Perfil dos participantes durante o semestre 2025.2

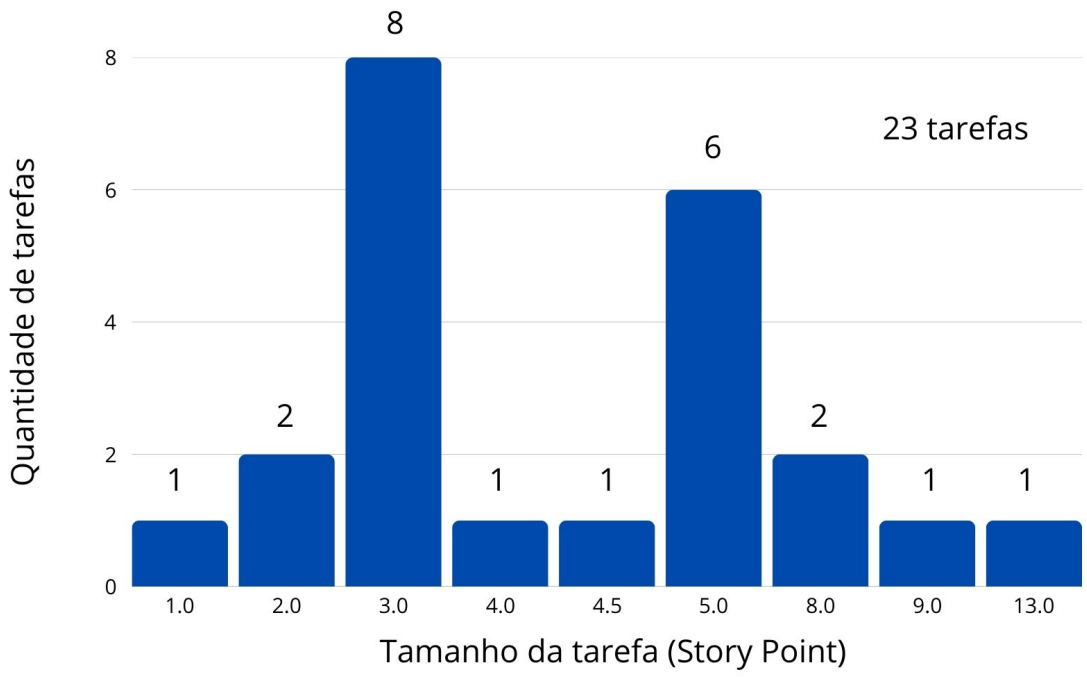


Fonte: Elaborado pelo autor.

Um dado revelador deste período é a 'densidade' da entrega. Mesmo sendo um intervalo de observação curto (apenas dois meses), o time acumulou 105.5 *Story Points*, o que representa um volume quase idêntico ao do semestre 2024.2 inteiro (112.5 pontos em quatro meses). Isso indica que, embora a eficiência individual tenha oscilado, a capacidade absoluta de entrega da equipe escalada é extremamente alta quando opera em regime normal.

A distribuição de tarefas no Gráfico 9 reflete essa estabilidade operacional. A concentração de entregas na faixa de 3 a 5 pontos não foi uma reação a um prazo curto, mas sim a manifestação do padrão de trabalho consolidado do time. Com uma equipe tão numerosa trabalhando simultaneamente, a padronização das tarefas em tamanhos médios (evitando os extremos de complexidade) surge naturalmente como a forma mais segura de manter o fluxo contínuo de integração e entrega.

Gráfico 9 – Tarefas realizadas durante o semestre 2025.2



Fonte: Elaborado pelo autor.

Dessa forma, ver o histórico de desempenho durante esses períodos não é suficiente, por isso, foi realizada uma análise quantitativa utilizando o software JASP, descrita na Seção seguinte.

5.5 Análise quantitativa com JASP

Foi realizada uma análise quantitativa utilizando o software JASP, utilizando os dados extraídos ao longo trabalho, que foram divididos em *ID* (número único), *Story Point* e Recorte temporal, como pode ser visualizado no APÊNDICE C, além disso, também foi realizado um teste de hipótese e validação da métrica, ambas detalhadas nas subseções seguintes.

5.5.1 Análise estatística

Na Tabela 1, os *Story Points* foram analisados em função de cada recorte temporal do projeto. A linha *Valid* (Válido) indica o número total de tarefas analisadas, sendo o recorte de 2024.2 o mais volumoso e o período pré-ágil o menor, condizente com a fase inicial do projeto. A linha *Missing* confirma a integridade da base de dados, indicando que não houve perda de informações em nenhum dos recortes.

Observando a linha *Mode* (Moda), nota-se que o período 2024.2 se sobressai com a entrega frequente de tarefas maiores (8 pontos). Isso ocorre em contraste com os demais períodos, onde a moda se estabiliza em tarefas de menor complexidade (entre 2 e 3 pontos).

Na linha *Median* (Mediana), o período pré-ágil apresenta o valor 2.5. Quando comparado à sua média elevada, isso indica uma distribuição assimétrica, onde poucas tarefas muito grandes distorceram a média para cima. Já o período 2025.2, com mediana 4, demonstra uma capacidade de entrega robusta e consistente, mesmo considerando o curto intervalo de tempo.

A linha *Mean* (Média) reforça essa leitura: o período pré-ágil detém a maior média, reflexo não da eficiência, mas da ausência de decomposição das tarefas complexas. Em contrapartida, 2025.1 apresenta a menor média, evidenciando uma "atomização" do escopo e maior maturidade do time na quebra de requisitos.

A linha *Std. Deviation* (Desvio Padrão) é um dos indicadores mais importantes desta análise. O período pré-ágil apresenta o maior valor, sinalizando a imprevisibilidade e a falta de padrão nas entregas iniciais. Por outro lado, o período 2025.2 registra o menor desvio de todos, comprovando estatisticamente a evolução do time na padronização e na constância da execução do trabalho.

Nas linhas *Shapiro-Wilk* e P-value of Shapiro-Wilk (P-valor de Shapiro-Wilk), verifica-se a normalidade da distribuição dos dados. O P-valor (*P-value*) inferior a 0.05 nos períodos 2024.2, 2025.1 e 2025.2 confirma que a distribuição dos *Story Points* não é normal, o que é esperado para a escala discreta de Fibonacci. O período pré-ágil é o único com valor acima de 0.05, porém, trata-se de um falso positivo estatístico decorrente do tamanho reduzido da amostra (apenas 6).

Por fim, as linhas *Minimum* (Mínimo) e *Maximum* (Máximo) delimitam o alcance das pontuações. É relevante notar que todos os recortes, inclusive o pré-ágil, têm como teto o valor 13. Isso sugere que existe um limite prático de complexidade aceitável pelo time; tarefas superiores a esse valor tendem a ser quebradas ou rejeitadas, independentemente da maturidade da equipe.

Tabela 1 – Resultados estatísticos

	Story Point			
Métrica	**2024.2**	**2025.1**	**2025.2**	**Pré-Ágil**
Válidos	24	14	23	6
Ausentes	0	0	0	0
Moda[a]	8,000	2,000	3,000	2,000
Mediana	3,000	2,000	4,000	2,500
Média	4,688	3,643	4,587	5,333
Desvio Padrão	3,608	3,805	2,708	5,645
Shapiro-Wilk	0,891	0,664	0,833	0,795
Valor-p (Shapiro-Wilk)	0,014	< 0,001	0,001	0,053
Mínimo	0,500	1,000	1,000	0,000
Máximo	13,000	13,000	13,000	13,000

[a] A moda é calculada assumindo que as variáveis são discretas.

Fonte: Elaborada pelo autor.

No Gráfico 10 é possível visualizar a dispersão dos dados e identificar valores discrepantes (*outliers*). A altura da "caixa" colorida representa o Intervalo Interquartil (IQR), ou seja, onde se concentram 50% das tarefas centrais do semestre.

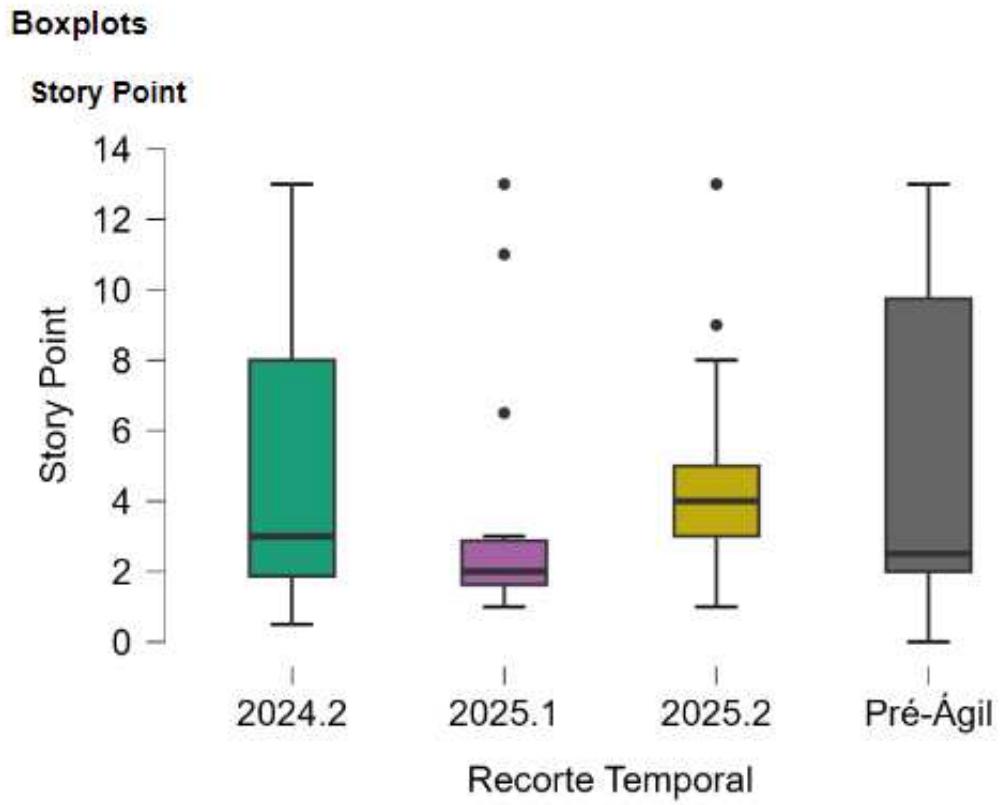
Durante o período pré-ágil, a caixa cinza é a mais "alongada" verticalmente. Isso indica uma amplitude interquartil gigantesca. O terceiro quartil (topo da caixa) está quase no 10, enquanto o primeiro quartil (base) está no 2, indicando que o time não tinha critério. Uma tarefa "padrão" poderia variar em muito seu esforço e complexidade. Não havia padrão de tamanho, refletindo a falta de metodologia.

Durante o recorte de 2024.2, a caixa ainda é grande, indicando que, apesar da alta produtividade desse semestre, havia muita variação no tamanho das entregas. A linha da mediana (traço preto dentro da caixa) está em 3, mas a caixa se estende até 8, significando que o time entregava muito, mas ainda oscilava entre tarefas pequenas e grandes com frequência.

Já no período de 2025.1 a caixa é mais "achatada" e posicionada na parte inferior (mediana em 2). Isso confirma visualmente uma maior fragmentação do escopo. Além disso, há vários "pontinhos" acima da caixa, indicando *outliers* (valores discrepantes), o que mostra que o normal do time era fazer tarefas minúsculas (caixa achatada). Quando aparecia uma tarefa de 8 ou 13 pontos, ela era uma exceção, fugindo completamente do padrão de trabalho daquele recorte.

Por último, no período 2025.2 A caixa está mais compacta indicando pouca variação, mas, diferentemente de 2025.1, ela está posicionada mais acima, centrada na mediana 4, o que evidência que o time encontrou um ponto de equilíbrio. As tarefas não são nem trivialmente pequenas, nem caoticamente grandes. A concentração de 50% dos dados entre 3 e 5 pontos mostra que o time aprendeu a padronizar o trabalho.

Gráfico 10 – Distribuição de tarefas



Fonte: Elaborado pelo autor.

Dessa forma, para realizar uma análise inferencial adequada, optou-se pelo teste de Kruskal-Wallis. Segundo Field (2020), este teste é a alternativa não paramétrica à ANOVA de um fator, sendo indicado precisamente quando os dados não atendem ao pressuposto de normalidade e envolvem três ou mais grupos independentes. No contexto deste trabalho, essa escolha justifica-se pela distribuição não normal dos *Story Points* e pela comparação entre quatro recortes temporais distintos com tarefas independentes.

5.5.2 Teste de hipótese e validação da métrica

Para verificar se as diferenças observadas entre os semestres são estatisticamente relevantes, foram aplicados o teste de Levene (1960), para avaliar a homogeneidade da variância, e o teste de Kruskal e Wallis (1952), indicado para a comparação de postos e medianas em distribuições não paramétricas.

Na Tabela 2, é observado o resultado do teste de Levene, que apresentou um valor p = 0.014, ou seja, um valor menor que 0.05. Esse resultado rejeita a hipótese de homogeneidade das variâncias. Em termos de gestão de projetos, isso prova estatisticamente que o processo de trabalho mudou: a variabilidade das entregas do time não é a mesma do início, corroborando a tese de aumento de previsibilidade (menor variância) observado no Gráfico 10.

Tabela 2 – Teste de Levene

Verificação de Pressupostos			
F	**gl1**	**gl2**	**p**
3,821	3,000	63,000	0,014

Fonte: Elaborada pelo autor.

Por outro lado, o teste não-paramétrico de Kruskal-Wallis, mostrado na Tabela 3 apresentou p = 0.238, valor acima de 0.05, indicando que não há diferença significativa nas medianas dos *Story Points* entre os períodos, confirmando que a complexidade média das tarefas permaneceu estável ao longo de todo o projeto. Ou seja, as variações de produtividade não ocorreram porque as tarefas ficaram "mais fáceis" ou "mais difíceis", mas sim devido a fatores em torno à equipe, como tamanho, comunicação e metodologia aplicada e com base na triangulação entre os dados de produtividade individual, distribuição de esforço e testes estatísticos, podem ser observados três pontos:

- **A Troca de Eficiência por Robustez**: No início (2024.2), o time era pequeno e altamente eficiente (16.1 pontos/dev), mas operava com alta variância e dependência de grandes entregas individuais. Com o crescimento da equipe para 31 membros em 2025.2, a produtividade individual caiu para cerca de 5.0 pontos/dev, refletindo o efeito previsto do custo de coordenação em grandes grupos de desenvolvimento.
- **O Ganho de Previsibilidade**: A análise estatística comprova que, em troca dessa redução de velocidade individual, o projeto ganhou previsibilidade. A redução do Desvio Padrão para 2.708 no recorte de 2025.2 e o resultado do teste de Levene confirmam que o time

atual entrega com uma consistência que não existia nos períodos anteriores.

- **Maturidade Metodológica**: O teste de Kruskal-Wallis valida que a régua de medição (*Story Points*) se manteve íntegra. Portanto, a concentração final de tarefas na faixa de 3 a 5 pontos é um indicador de maturidade: o time grande aprendeu a fatiar problemas complexos em unidades de trabalho padronizadas, permitindo fluxo contínuo mesmo com uma equipe numerosa.

Tabela 3 – Teste de Kruskal-Wallis

Fator	Estatística	gl	p
Recorte Temporal	4,222	3	0,238

Fonte: Elaborada pelo autor.

Em suma, a produtividade do time evoluiu de um modelo instável e veloz para um modelo maduro, previsível e escalável, capaz de absorver novos membros, mantendo a entrega constante, ainda que a um custo operacional unitário mais elevado.

Com uma noção robusta de como o projeto foi afetado quantitativamente, é interessante visualizar como o projeto foi afetado em torno da satisfação dos membros que estavam antes e depois da aplicação da metodologia ágil, e também com aqueles que entraram logo após a aplicação.

5.6 Análise quantitativa em relação à satisfação

No Quadro 4 são apresentadas as cinco perguntas mais relevantes do questionário em relação com a pesquisa, no APÊNDICE D é possível visualizar em detalhes todas as respostas que o questionário obteve

O Quadro 4 destaca os cinco itens do questionário que melhor elucidam a correlação entre a percepção da equipe e os dados estatísticos de produtividade apresentados anteriormente. No APÊNDICE D é possível visualizar em detalhes todas as respostas que o questionário obteve. A análise desses itens permite compreender a visão dos envolvidos na adoção da metodologia ágil em larga escala, com os seguintes pontos:

- **Gestão de Mudanças e Flexibilidade**: O item referente à gestão de mudanças obteve concordância unânime e máxima. Este dado qualitativo é a explicação fundamental para o desempenho observado no semestre 2025.2, onde a equipe entregou 105 *Story Points* em apenas dois meses. A capacidade do processo ágil de absorver repriorizações sem

paralisar o desenvolvimento foi percebida por todos os membros como o maior ganho da metodologia.

- **Organização e Previsibilidade**: A alta pontuação neste item valida diretamente o teste de Levene e a redução do Desvio Padrão observados na subseção 5.5.2. A equipe confirma, através de sua percepção subjetiva, que o ambiente de trabalho deixou de ser caótico para se tornar previsível. O contraste com a abordagem *ad hoc* anterior é evidente, especialmente para os membros mais antigos do projeto.
- **Clareza de Papéis**: Em uma equipe que cresceu para 31 integrantes, o risco de sobreposição de funções e confusão hierárquica é alto. Contudo, a média de 4.71 indica que a estruturação em núcleos especializados (Qualidade, *Design*, *Front-end*, *Back-end*) foi bem-sucedida. Os participantes sentem segurança em suas atribuições, o que justifica a manutenção da estabilidade do fluxo de trabalho mesmo com a entrada massiva de novos membros.
- **O Paradoxo da Produtividade Pessoal**: Observa-se uma queda na avaliação quando o tema é a produtividade individual. Este indicador qualitativo corrobora a métrica quantitativa que apontou uma redução na entrega de pontos (de 16.1 para 5.0). Os desenvolvedores reconhecem implicitamente que os ritos da metodologia (reuniões, documentação, alinhamentos), embora necessários para a organização coletiva, consomem tempo que anteriormente era dedicado exclusivamente à codificação.
- **O Desafio da Comunicação**: O item com a menor avaliação refere-se à comunicação. Apesar da clareza dos papéis, o custo de coordenação e a troca de informações em um grupo numeroso permanecem como os principais gargalos. Este dado sugere que, embora o ágil tenha resolvido problemas de escopo e organização, a complexidade comunicacional aumenta exponencialmente com a escala, impactando a satisfação da equipe neste quesito específico.

Quadro 4 – As 5 perguntas-chave para compreensão do impacto da metodologia

Dimensão	**Pergunta**	**Média***	**Interpretação**
Processo	A metodologia ágil torna mais fácil gerir mudanças de prioridades e escopo.	5,00	Flexibilidade total
Processo	A previsibilidade e a organização são melhores com o ágil do que com abordagem *ad hoc*.	4,57	Redução do caos
Papéis	Tenho clareza sobre meu papel e minhas responsabilidades na equipe.	4,71	Especialização eficaz
Satisfação	A metodologia ágil tem contribuído para a minha produtividade pessoal.	3,85	Custo operacional
Satisfação	Estou satisfeito(a) com a comunicação e a troca de informações na equipe.	3,42	Gargalo de escala

*Escala Likert de 1 a 5, onde 5 representa “Concordo Totalmente”.
Fonte: Elaborada pelo autor.

Os dados confirmam que a metodologia ágil foi decisiva para instituir governança e previsibilidade, garantindo organização e clareza de papéis fundamentais para a expansão da equipe. Contudo, essa estruturação evidenciou um custo organizacional significativo, onde o aumento da coordenação impactou negativamente a fluidez da comunicação e a percepção de produtividade individual. Conclui-se que o modelo adotado priorizou a sustentabilidade e a consistência das entregas coletivas em detrimento da velocidade bruta de desenvolvimento.

6 CONCLUSÃO

O presente trabalho propôs-se a analisar o impacto da adoção de metodologias ágeis na produtividade e na qualidade do processo de desenvolvimento de software em um projeto de extensão universitária sujeito a uma rápida expansão de equipe. Por meio da triangulação entre métricas quantitativas de esforço (*Story Points*), testes estatísticos de hipótese e a percepção dos colaboradores, foi possível traçar um diagnóstico preciso sobre a maturidade do processo de engenharia de software no projeto.

Os resultados obtidos demonstram que a implementação do ágil não atuou como um acelerador linear de produção individual, mas sim como um instrumento indispensável de governança e escalabilidade. A análise estatística descritiva revelou uma redução drástica na variabilidade das entregas: o Desvio Padrão, que atingiu 5.645 no período pré-ágil (caracterizado por alta imprevisibilidade), caiu para 2.708 no último ciclo analisado. Esse dado, corroborado pelo teste de Levene ($p < 0.05$), comprova matematicamente que a equipe transitou de um modelo de trabalho instável e dependente de esforços individuais para um sistema padronizado e previsível.

Entretanto, essa conquista de previsibilidade cobrou seu preço em termos de eficiência. Observou-se que, enquanto a equipe crescia para 31 membros, a produtividade individual recuou de patamares superiores a 16 pontos (em equipes pequenas) para uma média de 5 pontos por desenvolvedor. O teste de Kruskal-Wallis ($p > 0.05$) foi crucial para isolar a variável de complexidade, confirmando que as tarefas não se tornaram mais difíceis; foi o custo de coordenação e comunicação inerente à escala que diluiu a velocidade individual.

Sob a ótica qualitativa, a pesquisa de satisfação validou essa dicotomia. A unanimidade dos participantes quanto à facilidade de gestão de mudanças (média 5.0/5.0) e a alta percepção de organização explicam como o projeto foi capaz de absorver repriorizações e entregar 105 pontos em um curto intervalo no semestre 2025.2. Por outro lado, a insatisfação latente com a comunicação e a percepção neutra sobre a produtividade pessoal confirmam que os ritos metodológicos, embora vitais para a coesão do grupo, geraram uma sobrecarga cognitiva e burocrática perceptível, especialmente para os membros veteranos que vivenciaram a agilidade desestruturada do passado.

Ressaltam-se, contudo, as limitações inerentes a este estudo de caso. A natureza acadêmica do projeto, composto majoritariamente por estudantes em regime de voluntariado, restringe a generalização direta dos resultados para ambientes corporativos de alta pressão

comercial. Adicionalmente, a utilização de *Story Points* como métrica principal carrega um componente de subjetividade nas estimativas, o que pode introduzir vieses na comparação direta entre períodos com equipes de maturidade técnica distinta.

Conclui-se, portanto, que a hipótese de que a metodologia ágil aumentaria a organização foi confirmada, mas a expectativa de que ela manteria a mesma velocidade de desenvolvimento individual foi refutada pela realidade da escala. O sucesso do projeto não residiu na manutenção de uma velocidade vertiginosa, mas na construção de uma estrutura resiliente capaz de suportar a especialização de papéis e a entrada massiva de novos membros sem colapsar.

6.1 Lições aprendidas

A vivência prática da transição de um modelo *ad hoc* para uma estrutura ágil no NERDS proporcionou aprendizados valiosos, que transcendem os dados estatísticos coletados. Dentre as principais lições aprendidas, destacam-se:

- **O dilema entre agilidade e escala:** Compreendeu-se na prática que adicionar mais desenvolvedores a um projeto não resulta em entregas proporcionais de imediato. A escala exige um nível de formalização e comunicação que, inevitavelmente, desacelera a velocidade individual de codificação em prol da organização coletiva.
- **A importância de combater o "conhecimento tribal":** Em um contexto acadêmico marcado por alta rotatividade, depender da memória de indivíduos provou-se um risco crítico. A adoção de critérios de aceitação rigorosos e a padronização do fluxo de trabalho tornaram-se as principais salvaguardas para garantir a continuidade do GEX a longo prazo, facilitando a entrada de dezenas de novos membros.
- **Ritos ágeis não são "balas de prata":** Verificou-se que a simples introdução de cerimônias não resolve problemas técnicos profundos se a equipe não compreender o valor por trás da prática. A pesquisa como um todo evidenciou que a gestão do projeto precisa calibrar constantemente o nível de burocracia para evitar a fadiga e a desmotivação técnica da equipe.

6.2 Trabalhos futuros

Para trabalhos futuros, sugere-se a investigação de ferramentas de automação que possam reduzir o "custo de rito" identificado na pesquisa qualitativa, buscando mitigar os

gargalos de comunicação sem sacrificar a governança conquistada. O legado deste estudo é a demonstração de que, em projetos de software em expansão, a previsibilidade é uma métrica de sucesso superior à velocidade bruta, pois é ela que garante a sustentabilidade do desenvolvimento a longo prazo.

REFERÊNCIAS

ANDERSON, D. J. **Kanban: Mudança Evolucionária de Sucesso para seu Negócio de Tecnologia**. [s.l.]: Blue Hole Press, 2010.

BECK, K. **Extreme Programming Explained: Embrace Change**. 2. ed. [s.l.]: Addison-Wesley Professional, 2004.

BECK, K.; BEEDLE, M.; BENNEKUM, A. van; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R.; KERN, J.; MARICK, B.; MARTIN, R. C.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; THOMAS, D. **Manifesto for Agile Software Development**. 2001. Disponível em: https://agilemanifesto.org/. Acesso em: 18 jun. 2025.

BEZERRA, E. **Princípios de análise e projetos de sistemas com UML**. [s.l.]: Campus, 2014.

BOEHM, B. A spiral model of software development and enhancement. **ACM SIGSOFT Software Engineering Notes**, v. 11, n. 4, p. 14–24, 1986.

BROWN, T. **Design Thinking: Uma Metodologia Poderosa Para Decretar o Fim Das Velhas Ideias**. Rio de Janeiro: Alta Books, 2010. Tradução de: "Change by Design".

BURROWS, M. **Kanban from the Inside: Understand the Kanban Method, connect it to what you already know, introduce it with impact**. [s.l.]: Blue Hole Press, 2014.

CAETANO, L. F. **Aplicação do Kanban no gerenciamento de projeto de software remoto em uma empresa de e-grocery no contexto pós-pandemia COVID-19: um estudo de caso**. Monografia (Trabalho de Conclusão de Curso (Graduação em Engenharia de Software)) — Universidade Federal do Ceará, Russas, 2023.

COHN, M. **User Stories Applied for Agile Software Development**. [s.l.]: Addison-Wesley, 2004.

____. **Agile Estimating and Planning**. Upper Saddle River, NJ: Prentice Hall Professional, 2005.

____. **Succeeding with Agile: Software Development Using Scrum**. [s.l.]: Addison-Wesley Professional, 2010. (Addison-Wesley Signature Series). ISBN 9780321579362.

CRESWELL, J. W. **Research Design: Qualitative, Quantitative, and Mixed Methods Approaches**. 3. ed. Thousand Oaks: SAGE Publications, 2009.

CUNHA, P. S.; BARBOSA, A. C. G. Reestruturação da metodologia ágil scrum numa empresa de pequeno porte. 2022. Trabalho de Conclusão de Curso/Artigo.

CUNNINGHAM, W. The wycash portfolio management system. In: ACM. **Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)**. [s.l.], 1992. v. 4, p. 29–30.

FENTON, N.; BIEMAN, J. **Software Metrics: A Rigorous and Practical Approach**. 3. ed. Boca Raton, FL: CRC Press, 2014.

FIELD, A. **Discovering statistics using IBM SPSS statistics**. 5. ed. [s.l.]: SAGE Publications Limited, 2020.

FOWLER, M. **Technical Debt Quadrant**. 2009. Disponível em: https://martinfowler.com/bliki/TechnicalDebtQuadrant.html. Acesso em: 15 jan. 2025.

GAMA, R. da; REZENDE, D. de; PEREIRA, B. Implementação de metodologias ágeis para o gerenciamento de projetos em uma indústria. In: **Anais do Congresso Brasileiro de Engenharia de Produção (ENEGEP)**. [s.l.]: [s.n.], 2020.

GRENNING, J. Planning poker or how to avoid analysis paralysis while release planning. **Renaissance Software Consulting**, 2002. Disponível em: https://wingman-sw.com/papers/PlanningPoker-v1.1.pdf. Acesso em: 15 jan. 2025.

GUMINSKI, R.; DOHN, K.; OLOYEDE, O. Advantages and disadvantages of traditional and agile methods in software development projects – case study. **Scientific Journal of Poznań University of Technology**, 2023.

HAMMARBERG, M.; SUNDEN, J. **Kanban in Action**. [s.l.]: Manning Publications, 2014.

HIGHSMITH, J. **Agile Project Management: Creating Innovative Products**. 2. ed. Upper Saddle River, NJ: Addison-Wesley Professional, 2009.

HIGUCHI, M. M.; NAKANO, D. N. Agile design: a combined model based on design thinking and agile methodologies for digital games projects. **Revista de Gestão e Projetos - GeP**, v. 8, n. 2, p. 109–126, 2017.

HUNT, A.; THOMAS, D. **The Pragmatic Programmer: From Journeyman to Master**. Boston, MA: Addison-Wesley Professional, 2000.

JEFFRIES, R.; ANDERSON, A.; HENDRICKSON, C. **Extreme Programming Installed**. [s.l.]: Addison-Wesley, 2000.

KRUCHTEN, P.; NORD, R. L.; OZKAYA, I. Technical debt: From metaphor to theory and practice. **IEEE Software**, IEEE, v. 29, n. 6, p. 18–21, 2012.

KRUSKAL, W. H.; WALLIS, W. A. Use of ranks in one-criterion variance analysis. **Journal of the American statistical Association**, Taylor & Francis, v. 47, n. 260, p. 583–621, 1952.

LAKATOS, E. M.; MARCONI, M. d. A. **Fundamentos de Metodologia Científica**. 8. ed. São Paulo: Atlas, 2017.

LARMAN, C. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development**. 3. ed. [s.l.]: Prentice Hall, 2005.

LEVENE, H. Robust tests for equality of variances. In: **Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling**. Palo Alto, CA: Stanford University Press, 1960. p. 278–292.

LOVE, J.; SELKER, R.; MARSMAN, M.; JAMIL, T.; DROPMANN, D.; VERHAGEN, J.; LY, A.; GRONAU, Q. F.; ŠMÍRA, M.; EPSKAMP, S.; MATZKE, D.; ROUDER, J. N.; MOREY, R. D.; WAGENMAKERS, E.-J. Jasp: Graphical statistical software for common statistical designs. **Journal of Statistical Software**, UCLA, Department of Statistics, v. 88, n. 1, p. 1–17, 2019.

MARTIN, R. C. **Clean Agile: Back to Basics**. 1. ed. Boston, MA: Pearson, 2020. ISBN 978-0135781869.

MCCONNELL, S. **Code Complete**. 2. ed. Redmond, WA: Microsoft Press, 2004.

MERRIAM, S. B. **Qualitative research and case study applications in education**. San Francisco: Jossey-Bass, 1998.

MISHRA, A.; ALZOUBI, Y. A. Structured software development versus agile software development: a comparative analysis. **Journal of Computer Information Systems**, v. 63, n. 1, p. 1–12, 2023.

MORGAN, D. L. Paradigms lost and pragmatism regained: Methodological implications of combining qualitative and quantitative methods. **Journal of Mixed Methods Research**, SAGE Publications, v. 1, n. 1, p. 48–76, 2007.

NORMAN, D. **The Design of Everyday Things: Revised and Expanded Edition**. New York: Basic Books, 2013.

OHNO, T. **Toyota Production System: Beyond Large-Scale Production**. [s.l.]: Productivity Press, 1988.

POHL, K.; RUPP, C. **Requirements Engineering Fundamentals**. [s.l.]: Rocky Nook, 2011.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**. 10. ed. [s.l.]: McGraw-Hill, 2021.

Project Management Institute. **Agile Practice Guide**. [s.l.]: Project Management Institute, 2021.

RAMOS, R. A. d. L. **Monitoramento de métricas de qualidade e produtividade em projetos ágeis de software através da integração de dados extraídos de ferramentas de gestão e testes**. Dissertação (Dissertação de Mestrado) — Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB), João Pessoa, 2022.

RUBIN, K. S. **Essential Scrum: A Practical Guide to the Most Popular Agile Process**. Upper Saddle River, NJ: Addison-Wesley, 2012.

SILVA, I. F. da. Describing the design thinking and extreme programming activities during a technology innovation academic workshop. **Innovation & Management Review**, Emerald Publishing Limited, v. 17, n. 4, p. 421–435, 2020.

SOHAIB, O.; SOLANKI, H. R.; DHALIWA, N.; HUSSAIN, W.; ASIF, M. Integrating design thinking into extreme programming. **Journal of Ambient Intelligence and Humanized Computing**, Springer, v. 10, p. 2485–2492, 2019.

SOMMERVILLE, I. **Engenharia de software**. [s.l.]: Pearson, 2019.

SUTHERLAND, J. **Scrum: the art of doing twice the work in half the time**. [s.l.]: Crown Business, 2014.

THEBE, L. **Improving the Software Development Process in a Software Development Team: a Case Study**. Dissertação (Master's Thesis) — Aalto University, Helsinki, 2020.

VAZQUES, C. E.; SIMõES, G. S. **Engenharia de requisitos: Software orientado ao negócio**. [s.l.]: Novatec, 2016.

VILLER, S.; SOMMERVILLE, I. Social analysis in the requirements engineering process: from ethnography to method. In: IEEE. **Proceedings of the IEEE International Symposium on Requirements Engineering**. Limerick, Ireland, 1999. p. 6–13.

WASHIZAKI, H. (Ed.). **Guia para o Corpo de Conhecimento de Engenharia de Software (Guia SWEBOK), Versão 4.0**. [s.l.]: IEEE Computer Society, 2024. Disponível em: https://www.swebok.org. Acesso em: 11 jun. 2026.

WAZLAWICK, R. S. **Metodologia de Pesquisa para Ciência da Computação**. 3. ed. Rio de Janeiro: Elsevier, 2020.

WIEGERS, K.; BEATTY, J. **Software requirements**. 3. ed. [s.l.]: Microsoft Press, 2013.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in Software Engineering**. Berlin, Heidelberg: Springer Science & Business Media, 2012.

YIN, R. K. **Estudo de caso: planejamento e métodos**. 5. ed. Porto Alegre: Bookman, 2015.

APÊNDICE A – QUESTIONÁRIO DE SATISFAÇÃO

Esse questionário visa coletar informações quantitativas em relação aos participantes que estiveram antes e depois do período de intervenção e reestruturação do processo de desenvolvimento do GEX.

Questionário de satisfação - TCC

Olá!
Sua opinião é muito importante. Este questionário faz parte da execução do Trabalho de Conclusão de curso do estudante Patrick Glauber Oliveira Da Silva sob a orientação do Prof. Dr. Marcos Vinicius de Andrade Lima.

Nosso objetivo é coletar seu feedback sobre a reestruturação do processo de desenvolvimento do sistema da Gestão da extensão (GEX). Suas respostas nos ajudarão a entender o que funcionou bem, o que pode ser melhorado e como otimizar os processos no futuro.

Levará apenas alguns minutos para preenchê-lo, e suas respostas são **anônimas** e serão utilizadas **apenas para fins acadêmicos**.

Em caso de dúvidas, entre em contato pelo número: 88 994029725 (Patrick Glauber - Analista de Requisitos)

* Indica uma pergunta obrigatória

1. **Termo de Consentimento Livre e Esclarecido (TCLE)** *

Você está sendo convidado(a) a participar de um questionário como parte de um **estudo sobre os impactos e melhorias percebidas após a aplicação de métodos ágeis** no projeto.

A sua participação é **voluntária**, e você pode optar por **não responder** ou **interromper sua participação a qualquer momento**, sem nenhum prejuízo.

As respostas serão utilizadas **apenas para fins de estudo** e **não identificarão você pessoalmente**. Os dados coletados serão tratados de forma confidencial, com o único propósito de analisar os resultados da adoção dos métodos ágeis no projeto.

Ao continuar e enviar suas respostas, você declara que:
Leu este termo;
Compreendeu seus objetivos;
Concorda, de forma livre e esclarecida, em participar deste estudo.

Marcar apenas uma oval.

- ◯ Declaro que li e concordo com os termos acima.
- ◯ Não concordo com os termos e, portanto, não participarei do estudo.

Perfil do respondente

2. Nome completo

_____

3. Em qual semestre você está? *

Marcar apenas uma oval.

- ◯ Primeiro semestre
- ◯ Segundo semestre
- ◯ Terceiro semestre
- ◯ Quarto semestre
- ◯ Quinto semestre
- ◯ Sexto semestre
- ◯ Sétimo semestre
- ◯ Oitavo semestre
- ◯ Outro: _____

4. Qual o seu nível de experiência em projetos de software (considerando atuação acadêmica, profissional, freelance, estágios, projetos pessoais ou voluntariado)? *

Marcar apenas uma oval.

- ◯ Iniciante - Tenho pouca ou nenhuma experiência prática em programação, conheço apenas conceitos básicos.
- ◯ Aprendiz - Já participou de algumas atividades práticas em projetos, consigo realizar tarefas simples com auxílio de colegas mais experientes.
- ◯ Intermediário - Atuo de forma mais autônoma em tarefas de média complexidade, já entendo papéis e artefatos de metodologias ágeis (Scrum, Kanban, etc).
- ◯ Avançado - Planeja e executo tarefas de forma autônoma, participa ativamente de práticas ágeis (Ex: Sprint review, Planning, etc).
- ◯ Mentor - Contribuo na definição e melhoria do processo de desenvolvimento, tenho domínio técnico e metodológico (Scrum, Kanban, TDD, versionamento, integração contínua etc.).

5. Qual sua área de atuação? *

Marque todas que se aplicam.

- [ ] Desenvolvimento front end
- [ ] Desenvolvimento back end
- [ ] Garantia de qualidade
- [ ] DevOps
- [ ] Engenharia de requisitos
- [ ] Gerenciamento de projetos
- [ ] Administração de banco de dados
- [ ] Outro: _____

6. Em que semestre entrou no projeto NERDS? *

Marcar apenas uma oval.

- ◯ 2023.1
- ◯ 2023.2
- ◯ 2024.1
- ◯ 2024.2
- ◯ 2025.1
- ◯ 2025.2
- ◯ Outro: _____

Processos e fluxo de trabalho

Em cada afirmação a seguir, selecione a opção que melhor representa sua percepção.

7. A definição, visibilidade e prioridade das tarefas no projeto NERDS são claras. *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

8. O fluxo de trabalho ágil (ex: sprints, ciclos de feedbacks, reuniões de alinhamento) contribui para a entrega de valor de forma mais rápida e contínua. *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

9. A previsibilidade e a organização do trabalho são melhores com a metodologia ágil do que com uma abordagem ***ad hoc***. *

ad hoc refere-se a um processo de desenvolvimento que é feito de forma improvisada ou sob demanda, sem um planejamento prévio extenso e estruturado

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

10. A metodologia ágil torna mais fácil gerir mudanças de prioridades e escopo dentro do projeto. *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Totalmente
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

11. O quadro de tarefas no Clickup é eficaz para o trabalho diário. *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

Papéis e responsabilidades

Em cada afirmação a seguir, selecione a opção que melhor representa sua percepção.

12. Eu tenho clareza sobre meu papel e minhas responsabilidades dentro da equipe. *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

13. Eu recebo suporte e colaboração adequados dos outros membros da equipe para meus objetivos e desafios. *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

14. Sinto que o grupo entende e respeita os papéis de cada participante.

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

15. Sinto que consigo contribuir com ideias e sugestões para melhorar o processo de desenvolvimento.

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

16. Eu sou alocado para tarefas apropriadas para o meu nível de desenvolvimento. *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

Satisfação geral e impacto pessoal

Em cada afirmação a seguir, selecione a opção que melhor representa sua percepção.

17. Estou satisfeito(a) com a implantação da metodologia ágil. *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

18. A metodologia ágil tem contribuído para a minha produtividade pessoal. *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo
- Não concordo nem discordo
- Discordo
- Totalmente

19. Hoje, compreendo melhor como meu papel se encaixa no processo de desenvolvimento como um todo.

Marcar apenas uma oval.

- Concordo totalmente
- Concordo
- Não concordo nem discordo
- Discordo
- Discordo totalmente

20. Considero que o ambiente de trabalho no grupo é motivador e colaborativo.

Marcar apenas uma oval.

- Concordo totalmente
- Concordo
- Não concordo nem discordo
- Discordo
- Discordo totalmente

21. Estou satisfeito(a) com a comunicação e a troca de informações dentro da equipe.

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo
- ◯ Não concordo nem discordo
- ◯ Discordo
- ◯ Discordo totalmente

APÊNDICE B – COMPILADO DE TAREFAS CONCLUÍDAS ENTRE NOVEMBRO DE 2024 A NOVEMBRO DE 2025

Esse documento visa compilar todas as tarefas realizadas no período de execução do TCC, com o objetivo de fornecer todos os dados possíveis para o leitor.

Compilado de tarefas feitas

Antes da aplicação da metodologia ágil:

Data: Abril de 2024 a Outubro de 2024

Quadro 5 - Participantes antes da aplicação da metodologia ágil

Participante	Funções
Participante 1	Desenvolvedor front end, Desenvolvedor back end, Administrador de banco de dados
Participante 2	Desenvolvedor front end
Participante 4	Desenvolvedor back end, DevOps
Participante 3	Desenvolvedor front end, Desenvolvedor back end

Fonte: Elaborado pelo autor.

Quadro 6 - Tarefas realizadas antes da aplicação da metodologia ágil

Tarefas	Story Point
Tarefa 1	13
Tarefa 2	3
Tarefa 3	12
Tarefa 4	2
Tarefa 5	2
Tarefa 6	0

Fonte: Elaborado pelo autor.

Semestre 2024.2

Data: Novembro de 2024 a Fevereiro de 2025

Quadro 7 - Participantes durante o semestre 2024.2

Participante	Funções
Participante 1	Desenvolvedor back end, administrador de banco de dados
Participante 2	Desenvolvedor front end
Participante 4	DevOps
Participante 5	Analista de qualidade
Participante 9	Desenvolvedor front end
Participante 12	Desenvolvedor back end
Participante 13	Analista de requisitos, Gerente de projeto
Participante 14	Desenvolvedor back end
Participante 15	Desenvolvedor back end

Fonte: Elaborado pelo autor.

Quadro 8 - Tarefas realizadas durante o semestre 2024.2

Tarefas	Story Points
Tarefa 7	5
Tarefa 8	6.5
Tarefa 9	1
Tarefa 10	2.5
Tarefa 11	1
Tarefa 12	0.5
Tarefa 13	0.5
Tarefa 14	2.5
Tarefa 15	1.5
Tarefa 16	3
Tarefa 17	2
Tarefa 18	8
Tarefa 19	13

Tarefa 20	2
Tarefa 21	1.5
Tarefa 22	3
Tarefa 23	8
Tarefa 24	8
Tarefa 25	4
Tarefa 26	10
Tarefa 27	10
Tarefa 28	3
Tarefa 29	8
Tarefa 30	8

Fonte: Elaborado pelo autor.

Semestre 2025.1

Data: Março de 2025 a Junho de 2025

Quadro 9 - Participantes durante o semestre 2025.1

Participante	Funções
Participante 1	Desenvolvedor back end, administrador de banco de dados
Participante 2	Desenvolvedor front end
Participante 3	Desenvolvedor back end
Participante 4	DevOps
Participante 5	Analista de qualidade
Participante 6	Desenvolvedor front end
Participante 7	Desenvolvedor front end
Participante 8	Desenvolvedor front end
Participante 9	Desenvolvedor front end

Participante 10	Desenvolvedor front end
Participante 11	Desenvolvedor back end
Participante 12	Desenvolvedor back end
Participante 13	Analista de requisitos, Gerente de projeto
Participante 14	Desenvolvedor back end
Participante 15	Desenvolvedor back end

Fonte: Elaborado pelo autor.

Quadro 10 - Tarefas realizadas durante o semestre 2025.1

Tarefas	Story Points
Tarefa 31	2
Tarefa 32	1,5
Tarefa 33	2
Tarefa 34	1
Tarefa 35	6.5
Tarefa 36	2,5
Tarefa 37	1,5
Tarefa 38	3
Tarefa 39	2
Tarefa 40	11
Tarefa 41	13
Tarefa 42	1
Tarefa 43	2
Tarefa 44	2

Fonte: Elaborado pelo autor.

Semestre 2025.2

Data: Setembro de 2025 a Novembro de 2025

Quadro 11 - Participantes durante o semestre 2025.2

Participante	Funções
Participante 13	Analista de requisitos, Gerente de projeto
Participante 16	Analista de requisitos
Participante 11	Desenvolvedor back end
Participante 17	Desenvolvedor back end
Participante 12	Desenvolvedor back end
Participante 3	Desenvolvedor back end
Participante 18	Desenvolvedor back end
Participante 19	Desenvolvedor back end
Participante 14	Desenvolvedor back end
Participante 15	Desenvolvedor back end
Participante 20	Desenvolvedor back end
Participante 1	DevSecOps
Participante 4	DevOps
Participante 21	Desenvolvedor front end
Participante 22	Desenvolvedor front end
Participante 2	Desenvolvedor front end
Participante 6	Desenvolvedor front end
Participante 23	Desenvolvedor front end
Participante 7	Desenvolvedor front end
Participante 24	Desenvolvedor front end
Participante 25	Desenvolvedor front end
Participante 26	Desenvolvedor front end
Participante 27	Desenvolvedor front end
Participante 5	Analista de qualidade
Participante 28	Analista de qualidade
Participante 29	Analista de qualidade

Participante 30	Designer
Participante 31	Designer
Participante 32	Designer
Participante 33	Designer
Participante 34	Designer

Fonte: Elaborado pelo autor.

Quadro 12 - Tarefas realizadas durante o semestre 2025.2

Tarefas	Story Points
Tarefa 45	13
Tarefa 46	2
Tarefa 47	8
Tarefa 48	5
Tarefa 49	5
Tarefa 50	5
Tarefa 51	3
Tarefa 52	3
Tarefa 53	5
Tarefa 54	3
Tarefa 55	3
Tarefa 56	8
Tarefa 57	2
Tarefa 58	3
Tarefa 59	3
Tarefa 60	4
Tarefa 61	5
Tarefa 62	5
Tarefa 63	3
Tarefa 64	4.5

Tarefa 65	3
Tarefa 66	9
Tarefa 67	1

Fonte: Elaborado pelo autor.

APÊNDICE C – TABULAÇÃO DOS DADOS PARA ANÁLISE QUANTITATIVA

Esse Quadro visa reunir os dados tabulados para análise qualitativa realizada na subseção 5.5.

Quadro 13 - Relação entre *story points* e recortes temporais

ID Tarefa	Story Points	Recorte Temporal
1	13	Pré-Ágil
2	3	Pré-Ágil
3	12	Pré-Ágil
4	2	Pré-Ágil
5	2	Pré-Ágil
6	0	Pré-Ágil
7	5	2024.2
8	6,5	2024.2
9	1	2024.2
10	2,5	2024.2
11	1	2024.2
12	0,5	2024.2
13	0,5	2024.2
14	2,5	2024.2
15	1,5	2024.2
16	3	2024.2
17	2	2024.2
18	8	2024.2
19	13	2024.2
20	2	2024.2
21	1,5	2024.2
22	3	2024.2
23	8	2024.2
24	8	2024.2
25	4	2024.2
26	10	2024.2
27	10	2024.2
28	3	2024.2
29	8	2024.2
30	8	2024.2
31	2	2025.1
32	1,5	2025.1
33	2	2025.1
34	1	2025.1
35	6,5	2025.1
36	2,5	2025.1
37	1,5	2025.1
38	3	2025.1

ID Tarefa	Story Points	Recorte Temporal
39	2	2025.1
40	11	2025.1
41	13	2025.1
42	1	2025.1
43	2	2025.1
44	2	2025.1
45	13	2025.2
46	2	2025.2
47	8	2025.2
48	5	2025.2
49	5	2025.2
50	5	2025.2
51	3	2025.2
52	3	2025.2
53	5	2025.2
54	3	2025.2
55	3	2025.2
56	8	2025.2
57	2	2025.2
58	3	2025.2
59	3	2025.2
60	4	2025.2
61	5	2025.2
62	5	2025.2
63	3	2025.2
64	4,5	2025.2
65	3	2025.2
66	9	2025.2
67	1	2025.2

Fonte: Elaborado pelo autor.

APÊNDICE D – RESPOSTAS DO QUESTIONÁRIO DE SATISFAÇÃO

Esse documento visa reunir as respostas de cada uma das quinze perguntas do formulário do APÊNDICE A

1. Processo e Fluxo de Trabalho

Questão: A definição, visibilidade e prioridade das tarefas no projeto NERDS são claras.

Opção de Resposta	Frequência (%)
Concordo totalmente	85.7%
Concordo	14.3%
Não concordo nem discordo	0.0%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: O fluxo de trabalho ágil (ex: sprints, ciclos de feedbacks) contribui para a entrega de valor mais rápida.

Opção de Resposta	Frequência (%)
Concordo totalmente	71.4%
Concordo	28.6%
Não concordo nem discordo	0.0%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: A previsibilidade e a organização do trabalho são melhores com a metodologia ágil.

Opção de Resposta	Frequência (%)
Concordo totalmente	57.1%
Concordo	42.9%
Não concordo nem discordo	0.0%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: A metodologia ágil torna mais fácil gerir mudanças de prioridades e escopo.

Opção de Resposta	Frequência (%)
Concordo totalmente	100.0%
Concordo	0.0%
Não concordo nem discordo	0.0%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: O quadro de tarefas no Clickup é eficaz para o trabalho diário.

Opção de Resposta	Frequência (%)
Concordo totalmente	28.6%
Concordo	57.1%
Não concordo nem discordo	14.3%
Discordo	0.0%
Discordo totalmente	0.0%

2. Papéis e Responsabilidades

Questão: Eu tenho clareza sobre meu papel e minhas responsabilidades dentro da equipe.

Opção de Resposta	Frequência (%)
Concordo totalmente	71.4%
Concordo	28.6%
Não concordo nem discordo	0.0%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: Eu recebo suporte e colaboração adequados dos outros membros da equipe.

Opção de Resposta	Frequência (%)
Concordo totalmente	28.6%
Concordo	57.1%
Não concordo nem discordo	14.3%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: Sinto que o grupo entende e respeita os papéis de cada participante.

Opção de Resposta	Frequência (%)
Concordo totalmente	28.6%
Concordo	42.9%
Não concordo nem discordo	28.6%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: Sinto que consigo contribuir com ideias e sugestões para melhorar o processo.

Opção de Resposta	Frequência (%)
Concordo totalmente	42.9%
Concordo	28.6%
Não concordo nem discordo	28.6%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: Eu sou alocado para tarefas apropriadas para o meu nível de desenvolvimento.

Opção de Resposta	Frequência (%)
Concordo totalmente	71.4%
Concordo	14.3%
Não concordo nem discordo	14.3%
Discordo	0.0%
Discordo totalmente	0.0%

3. Satisfação e Impacto Pessoal

Questão: Estou satisfeito(a) com a implantação da metodologia ágil.

Opção de Resposta	Frequência (%)
Concordo totalmente	57.1%
Concordo	42.9%
Não concordo nem discordo	0.0%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: A metodologia ágil tem contribuído para a minha produtividade pessoal.

Opção de Resposta	Frequência (%)
Concordo totalmente	28.6%
Concordo	28.6%
Não concordo nem discordo	42.9%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: Hoje, compreendo melhor como meu papel se encaixa no processo de desenvolvimento.

Opção de Resposta	Frequência (%)
Concordo totalmente	71.4%
Concordo	14.3%
Não concordo nem discordo	14.3%
Discordo	0.0%
Discordo totalmente	0.0%

Questão: Considero que o ambiente de trabalho no grupo é motivador e colaborativo.

Opção de Resposta	Frequência (%)
Concordo totalmente	42.9%
Concordo	14.3%
Não concordo nem discordo	28.6%
Discordo	14.3%
Discordo totalmente	0.0%

Questão: Estou satisfeito(a) com a comunicação e a troca de informações dentro da equipe.

Opção de Resposta	Frequência (%)
Concordo totalmente	14.3%
Concordo	28.6%
Não concordo nem discordo	42.9%
Discordo	14.3%
Discordo totalmente	0.0%