Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Reengenharia do Aplicativo Mia Ajuda: Uma Abordagem Orientada a TDD e DDD

Autores: Mateus Gomes do Nascimento e Vinícius de Sousa Saturnino

Orientador: Prof. Dr. Maurício Serrano

Brasília, DF

2024

Mateus Gomes do Nascimento e Vinícius de Sousa Saturnino

Reengenharia do Aplicativo Mia Ajuda: Uma Abordagem Orientada a TDD e DDD

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Maurício Serrano

Coorientadora: Prof[a]. Dr[a]. Milene Serrano

Brasília, DF

2024

Mateus Gomes do Nascimento e Vinícius de Sousa Saturnino

Reengenharia do Aplicativo Mia Ajuda: Uma Abordagem Orientada a TDD e DDD/ Mateus Gomes do Nascimento e Vinícius de Sousa Saturnino. – Brasília, DF, 2024-

99 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Maurício Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2024.

1. Test-Driven Development (TDD). 2. Domain-Driven Design (DDD). I. Prof. Dr. Maurício Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Reengenharia do Aplicativo Mia Ajuda: Uma Abordagem Orientada a TDD e DDD

CDU 02:141:005.6

Mateus Gomes do Nascimento e Vinícius de Sousa Saturnino

Reengenharia do Aplicativo Mia Ajuda: Uma Abordagem Orientada a TDD e DDD

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 10 de Julho de 2024:

Prof. Dr. Maurício Serrano
Orientador

Profª. Drª. Milene Serrano
Coorientadora

Profª. Drª. Elaine Venson
Convidado 1

M.Sc Rafael Fazzolino P. Barbosa
Convidado 2

Brasília, DF
2024

Este trabalho é dedicado a todos que, ao longo de nossa trajetória acadêmica, acreditaram continuamente em nosso sucesso e potencial e sempre nos apoiaram.

Agradecimentos

Primeiramente, expressamos nossos agradecimentos às nossas famílias, que consistentemente nos ofereceram apoio em todos os desafios. Eles não apenas nos incentivaram a perseguir nossos sonhos e objetivos, mas também nos proporcionaram uma educação de qualidade ao longo de toda a nossa vida. Esses esforços culminaram em nossa aprovação na prestigiada Universidade de Brasília.

Agradecemos também nossas companheiras, bem como nossos amigos, pelo apoio e incentivo que nos foi dado não somente durante o período de desenvolvimento do TCC, mas em todas as etapas durante nossa formação acadêmica.

Por fim, agradecemos aos docentes da Universidade de Brasília, especificamente do campus Gama, que nos proporcionaram um ensino de qualidade e excelência, nos capacitando para sermos profissionais de alta qualidade, em especial, aos professores Maurício Serrano e Milene Serrano, nossos orientadores, que nos auxiliaram durante esse desafio que culminou em uma ótima experiência.

Resumo

A manutenção evolutiva de um *software* é um processo importante para mantê-lo alinhado às necessidades dos *stakeholders*, bem como às demandas do mercado. Este processo é um desafio complexo, principalmente, no contexto do desenvolvimento *mobile*. Isso ocorre devido à rápida e constante evolução do cenário, o que torna o processo altamente dinâmico. A escolha do *design* é de grande importância, bem como a correta escolha das técnicas de programação que serão utilizadas na implementação do *software*. Aplicações que demandam uma urgência no prazo de conclusão costumam priorizar a agilidade na implementação, muitas vezes ignorando as boas práticas de desenvolvimento definidas pela comunidade da Engenharia de *Software*. Nesse contexto, há margem para implantação de muitas melhorias nos produtos de *software* produzidos. Este trabalho teve como intuito a condução de um estudo exploratório, orientado a provas de conceito, promovendo a reengenharia de um aplicativo de *software* existente, e utilizando uma abordagem que combina *Test Driven Development* (TDD) e *Domain-Driven Design* (DDD). Para que seja possível a reengenharia do aplicativo, fez-se necessária a engenharia reversa do mesmo, procurando compreender suas funcionalidades e particularidades qualitativas, uma vez que o aplicativo encontra-se em atendimento a um público alvo específico, apesar da inerente dificuldade em evoluí-lo. O presente trabalho aplicou uma abordagem orientada a testes, por meio do processo cíclico do TDD, visando corrigir possíveis problemas antes mesmo da nova implantação do aplicativo. Adicionalmente, usando *design* orientado a domínio, obteve-se maior valor semântico na nova proposta de desenvolvimento do aplicativo. Considerando o domínio a razão do negócio existir, compreendendo ideias, conhecimentos e processos de negócio, centrar-se nele tende a permitir maior imersão dos envolvidos. Os resultados obtidos cumpriram com os objetivos traçados para este trabalho, validando a pertinência das técnicas utilizadas e verificando a testabilidade facilitada e adequada modelagem de domínio após a reengenharia do aplicativo móvel em estudo.

Palavras-chave: Reengenharia. Engenharia Reversa. Aplicações *Mobile*. Desenvolvimento Orientado a Testes. *Design* Orientado a Domínio.

Abstract

The evolutionary maintenance of software is a crucial process to keep it aligned with the stakeholders' needs and market demands. This is particularly challenging in the context of mobile development due to the rapid and constant evolution of the scenario, making the process highly dynamic. The choice of design and the selection of programming techniques for software implementation are of great importance. Applications with tight deadlines often prioritize agility in implementation, sometimes neglecting the good development practices defined by the Software Engineering community. In this context, there is room for implementing many improvements in the produced softwares. This work aims to conduct an exploratory study, focused on proof of concepts, promoting the reengineering of an existing software application. It utilizes an approach that combines Test Driven Development (TDD) and Domain-Driven Design (DDD). Reengineering the application requires reverse engineering to understand its functionalities and qualitative peculiarities. The application serves a specific target audience, and despite the inherent difficulty in evolving it, it is believed that this difficulty arises from the development being carried out without adherence to Software Engineering best practices. This work applied a test-driven approach, using the TDD cycle process, to address potential issues before the application's new deployment. Additionally, using Domain-Driven Design, the software achieved greater semantic value in the new application development proposal. Focusing on the business domain, understanding ideas, knowledge, and business processes tends to allow greater immersion of those involved. Consequently, this may indicate greater coherence with the real needs of users and, therefore, greater success when the solution is made available for use. Such improvements are desired in the specific application under consideration. The results obtained met the objectives set for this work, validating the relevance of the techniques used and verifying the facilitated testability and adequate domain modeling after the reengineering of the mobile application under study.

Key-words: Reengineering. Reverse Engineering. Mobile Applications. Test Driven Development. Domain-Driven Design.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
BaaS	*Backend as a Service*
BDD	*Behavior Driven Development*
BPMN	*Business Process Modeling Notation*
BSON	*Binary Serialized Object Notation*
CD	*Continuous Delivery*
CI	*Continuous Integration*
CPU	*Central Processing Unit*
DDD	*Domain-Driven Design*
GRASP	*General Responsibility Assignment Software Pattern*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
NoSQL	*Not Only Structured Query Language*
POC	*Proof of Concept*
SDK	*Software Development Kit*
SQL	*Structured Query Language*
TDD	*Test Driven Development*
UI	*User Interface*
URL	*Uniform Resource Locator*
UX	*User Experience*
VM	*Virtual Machine*

Sumário

1 Introdução

Este capítulo tem como propósito apresentar o contexto existente na área de atuação desse trabalho, buscando inserir conceitos que auxiliem na compreensão da proposta como um todo. Neste trabalho, aborda-se o desenvolvimento de aplicativos móveis como domínio. Quanto à área de atuação, concentra-se na Engenharia de *Software*, mais precisamente na Manutenção Evolutiva. Também será apresentada uma justificativa para esclarecer os principais motivos por trás da realização do estudo proposto. A seguir, apresenta-se a questão de pesquisa, bem como os objetivos, incluindo o objetivo geral e os objetivos específicos. Por fim, será descrita a estrutura da monografia em termos de capítulos.

1.1 Contexto

O presente trabalho compreende, dentre outros aspectos, estudos na etapa de manutenção evolutiva. Nesse cenário, há necessidade de manter o *software* alinhado com os anseios dos *stakeholders*, bem como de acompanhar o mercado. Isso representa um desafio complexo devido à constante evolução das tecnologias, especialmente no contexto de dispositivos móveis, onde as mudanças ocorrem de maneira mais intensa (AL., 2019).

O aplicativo (MIAAJUDA, 2020) que foi desenvolvido em um contexto acadêmico, com o envolvimento de professores, alunos voluntários e bolsistas (graduandos de Engenharia de *Software* e de áreas afins). O aplicativo possui cunho social, com a finalidade de conectar os usuários que necessitam de apoio - atuando como uma plataforma que conecta pessoas que precisam de ajuda com voluntários dispostos a oferecer apoio, seja ele de natureza material ou emocional (MIAAJUDA, 2020). O aplicativo surgiu em 2020, durante os primeiros meses da pandemia de Covid-19, quando o isolamento social era a principal medida de segurança. Nesse período desafiador, a demanda por soluções que ajudassem a população aumentou consideravelmente. Isso levou os desenvolvedores a criar o aplicativo rapidamente, utilizando tecnologias com as quais já estavam familiarizados. No entanto, devido à urgência da situação, o desenvolvimento do aplicativo não atendeu aos critérios considerados como boas práticas da área.

Atualmente, o aplicativo tem passado por constantes evoluções, através de Trabalhos de Conclusão de Curso, Iniciações Científicas, Projetos de Pesquisa e Extensão, dentre outros. No contexto desse trabalho, evoluiu-se o aplicativo no intuito de incorporar novas orientações, sendo essas: *Test-Driven Development* (VERNON, 2016) e *Domain-Driven Design* (DDD) (BECK, 2022). No primeiro caso, objetivou-se a facilitação, principalmente da testabilidade. No segundo caso, objetivou-se prioritariamente um código orientado ao

domínio, o que, naturalmente, tende a melhorar a coesão, e até mesmo diminuir o acoplamento, dentre outros ganhos ((GUDWIN, 2010); (MATTOS; DOLL; ALMEIDA, 2010)).

Segundo Gudwin (2010), alta coesão - em particular no nível de código - é algo desejado, pois permite, por exemplo, compreender mais facilmente sobre o que uma classe ou um método trata, uma vez que seus nomes estão coerentes com suas responsabilidades. Já o baixo acoplamento, segundo Mattos, Doll e Almeida (2010) - também no nível de código - é algo desejado, pois permite, por exemplo, incorporar uma nova funcionalidade sem ter de rever, reescrever, ou ajustar o código como um todo, uma vez que as interdependências entre classes, módulos e subsistemas estão adequadamente especificadas.

O TDD (*Test-Driven Development*) consiste em uma abordagem de desenvolvimento de *software*, na qual é adotada como prática a escrita de testes unitários antes de qualquer linha de código relacionada ao projeto. Essa prática visa aumentar a qualidade do código e a confiabilidade do *software* (BECK, 2022). Já o DDD (*Domain-Driven Design*) consiste em uma abordagem de *design* de *software* que possui concentração na modelagem e na organização do código considerando o domínio do problema que o *software* visa resolver. É amplamente utilizado para o desenvolvimento de sistemas complexos e com alta qualidade. O DDD tem como principal objetivo, alinhar o código com o domínio do problema, o conjunto de regras, os conceitos e a lógica, sendo esses elementos inerentes quando se almeja uma adequada especificação/implementação do *software* (VERNON, 2013).

Para que fosse possível orientar-se por TDD, por ser algo que demanda planejar e criar os testes primeiramente, antes mesmo de qualquer linha de código, houve necessidade de realizar a Reengenharia do Aplicativo Mia Ajuda, uma vez que ele já se encontra desenvolvido. Segundo Cagnin (2005), a reengenharia consiste em uma técnica de análise crítica e reformulação de algo existente - nesse caso, um *software* - do início, ao invés de optar por desenvolver apenas melhorias. Entretanto, deve-se ter em mente a necessidade de considerar o que já foi/está desenvolvido. Afinal, trata-se de algo em funcionamento, com público alvo definido. Sendo assim, ocorreu uma demanda prioritária, que consistiu em realizar a Engenharia Reversa do aplicativo existente. De acordo com Chikofsky e Cross (1990), entende-se por Engenharia Reversa o processo pelo qual, a partir do código, ou seja, um insumo de baixo nível de abstração, recupera-se artefatos e insumos de maior nível de abstração, como por exemplo requisitos e regras de negócio. Tal levantamento foi relevante para a realização da Reengenharia do Aplicativo.

O Aplicativo Mia Ajuda já tem público alvo: Comunidades e vizinhanças que desejam promover a solidariedade e a colaboração entre seus membros. Encontra-se implantado e hospedado na Play Store, e atende as necessidades imediatas dos usuários. Entretanto, evoluir o aplicativo tornou-se uma tarefa pouco provável (PEREIRA; SOUZA, 2023), em especial, devido ao alto acoplamento, baixa coesão, dificuldade inerente de garantir tes-

tabilidade, dentre outros.

Ao realizar a Reengenharia do Aplicativo Mia Ajuda, possibilitou a incorporação de boas práticas de testabilidade de forma mais facilitada, além de adequações em termos de modularização e ajustes no código, no intuito de aumentar a coesão e reduzir o acoplamento, orientando-se pelo domínio e por técnicas de programação (ex. *Clean Code* (MARTIN, 2013)), já estudadas pela comunidade especializada em DDD.

1.2 Justificativa

Técnicas de programação e demais boas práticas mencionadas anteriormente (ex. testes) melhoram o *software* em diferentes aspectos, em especial mantendo maior coesão e menor acoplamento, além de facilidades em termos de manutenção evolutiva (BECK, 2022). Entretanto, ainda segundo Beck (2022), o uso dessas boas práticas demanda maiores esforço, capacitação e tempo da equipe responsável por prover o *software*. Sendo assim, comumente, são práticas negligenciadas em projetos com prazos curtos e demandas urgentes.

Os testes de *software* são fundamentais para, por exemplo, minimizar a existência de *bugs*, conferindo ainda maior segurança de que antigas funcionalidades não serão desfeitas na implementação de novas, a menos que esse seja o anseio. Porém, apenas os testes não garantem tudo em um *software*, também é necessária uma boa modelagem da aplicação, o que está diretamente ligado ao *design* da mesma (VERNON, 2013).

De acordo com Martin (2000) existem alguns sintomas que podem evidenciar que o *design* do *software* não está correto para a aplicação, sendo eles: rigidez, fragilidade, imobilidade e viscosidade. Segundo autores renomados da área, tal como Larman (2012), muitos destes sintomas estão atrelados aos princípios clássicos da Engenharia de *Software* de coesão e acoplamento, ambos conhecidos como GRASPs (*General Responsibility Assignment Software Patterns*). Caso não estejam bem implementados em um *software*, ainda com base no mesmo autor, essa situação pode incorrer em complicações até mesmo quando pequenas mudanças e correções precisam ser realizadas no *software*. Isso, praticamente, inviabiliza realizar atualizações no *software*.

No contexto do aplicativo (MIAAJUDA, 2020), no qual existe um alto acoplamento e uma baixa coesão na implementação de seus componentes arquiteturais e submódulos, é possível verificar os sintomas de um *design* que pode ser adequado/refinado para servir corretamente para o contexto atual da aplicação. Aliado à melhor testabilidade, que pode ser atingida com o uso da técnica do TDD, o DDD faz com que o *design* do *software* reflita exatamente suas funcionalidades, tornando muito pertinente e desejada a reengenharia do aplicativo (MIAAJUDA, 2020) orientando-se por essas duas abordagens (VERNON, 2013).

No intuito de conduzir esse trabalho, foram estabelecidas Questões de Pesquisa e de Desenvolvimento. Visando um cunho mais investigativo, ocorreu a necessidade de pesquisa, considerando que o trabalho demanda estar embasado na literatura especializada, em especial envolvendo os conceitos de Engenharia Reversa, Reengenharia, TDD, DDD e Técnicas de Programação para Aplicativos Móveis já existentes, e que se encontram na etapa de Manutenção Evolutiva do ciclo de vida de um *software*. Entretanto, ocorreu a necessidade adicional de realizar Engenharia Reversa e, na sequência, a Reengenharia do Aplicativo Mia Ajuda, com base em TDD, DDD e Técnicas de Programação investigadas, justificando a definição da Questão de Desenvolvimento.

1.3 Questões de Pesquisa e de Desenvolvimento

Este trabalho orientou-se pela seguinte questão de pesquisa:

Quais são as principais recomendações da Engenharia de *Software* no que diz respeito aos processos de Engenharia Reversa e Reengenharia de Aplicativos Móveis existentes, tomando como base TDD, DDD e Técnicas de Programação?

Responder essa questão demandou, em um primeiro momento: estabelecer o perfil de aplicativo móvel alvo da pesquisa, e conhecer sobre as melhores práticas adotadas para cada viés de orientação (TDD, DDD e Técnicas de Programação) aplicado aos processos de Engenharia Reversa e Reengenharia. Na sequência, combinou-se esses levantamentos em uma abordagem única, revelando de forma clara sobre os resultados obtidos, e respondendo, portanto, a Questão de Pesquisa estabelecida.

Demais detalhes sobre o perfil do aplicativo móvel alvo e as melhores práticas adotadas constam descritos(as) ao longo desta monografia.

Tendo em vista conferir um viés mais aplicado ao trabalho, e considerando a necessidade de evolução do aplicativo Mia Ajuda, considerou-se a seguinte Questão de Desenvolvimento:

É possível aplicar as principais recomendações da área, acordadas nos levantamentos de cunho investigativo, no aplicativo Mia Ajuda (ou em parte representativa dele)?

No intuito de responder à Questão de Desenvolvimento, realizou-se a Engenharia Reversa e, na sequência, a Reengenharia do aplicativo Mia Ajuda (ou de parte representativa dele) orientando-se por TDD, DDD e Técnicas de Programação. Os resultados desse viés aplicado estão expostos nesta monografia, bem como em repositório hospedado no GitHub (2023). Entende-se por parte representativa do aplicativo uma ou mais *feature(s)* de relevância, a serem apresentadas mais adiante nessa monografia.

1.4 Objetivos

A fim de responder às questões anteriormente apresentadas, foram estabelecidos um Objetivo Geral e alguns Objetivos Específicos. Estes objetivos podem ser vistos nas próximas subseções.

1.4.1 Objetivo Geral

Reengenharia de um aplicativo móvel existente, visando testabilidade facilitada e adequada modelagem de domínio, sendo esse processo apoiado em práticas que possam ser usadas em aplicativos móveis de cunho similar.

1.4.2 Objetivos Específicos

- Estudo sobre Engenharia Reversa de Aplicativos Móveis que se encontram na etapa de Manutenção Evolutiva;
- Estudo sobre Reengenharia de Aplicativos Móveis que se encontram na etapa de Manutenção Evolutiva;
- Levantamento sobre TDD;
- Levantamento sobre DDD;
- Levantamento sobre Técnicas de Programação;
- Documentação das principais recomendações da Engenharia de *Software* acordadas nos estudos e levantamentos realizados;
- Aplicação das principais recomendações no Aplicativo Mia Ajuda, e
- Exposição dos resultados obtidos.

1.5 Organização da Monografia

Esta monografia está estruturada da seguinte forma:

- Capítulo 2 - Referencial Teórico: Exposição dos referenciais teóricos utilizados para embasamento e elaboração desse trabalho, com destaque para etapa de Manutenção Evolutiva, Aplicativos Móveis, Engenharia Reversa, Reengenharia de *Software*, TDD (*Test Driven Development*), DDD (*Domain-Driven Design*) e Técnicas de Programação;

- Capítulo 3 - Capítulo 3 - Suporte Tecnológico: Apresentação das principais tecnologias utilizadas ao longo da concretização desse trabalho, mencionando os *frameworks* e ambientes para desenvolvimento e gerenciamento (ex. versionamento e hospedagem), assim como recursos inerentes à comunicação entre os autores e à elaboração dessa monografia;

- Capítulo 4 - Metodologia: Exibição do plano metodológico seguido pelos autores para o gerenciamento do trabalho num âmbito geral, destacando a classificação da pesquisa, e os métodos para levantamento bibliográfico, desenvolvimento e análise dos resultados;

- Capítulo 5 - Engenharia Reversa e Reeengenharia do Mia Ajuda: Apresentação os processos de engenharia reversa e reengenharia do aplicativo Mia Ajuda, detalhando-os em termos de origem da ideia; perfil de dispositivo móvel (alvo de interesse desse projeto); insumos elaborados ao longo do projeto via provas de conceito, dentre outros aspectos;

- Capítulo 6 - Análise de Resultados: Apontamentos sobre os resultados obtidos ao longo do projeto, analisando esses resultados com base em métricas e ferramentas, e

- Capítulo 7 - Conclusão: Detalhamento sobre as considerações fnais do projeto, bem como sobre os trabalhos futuros propostos para evolução do mesmo.

2 Referencial Teórico

Este capítulo irá tratar dos conceitos teóricos necessários para a realização deste Trabalho de Conclusão de Curso. Este trabalho teve como objetivo gerar contribuições para as áreas de *Design* de *Software* e Técnicas de Programação em Desenvolvimento *Mobile*.

Primeiramente, são abordados os temas de *Design* de *Software* 2.1 e Técnicas de Programação 2.2, que são áreas de interesse do trabalho por englobarem o DDD (*Domain Driven Design*) e o TDD (*Test Driven Development*). Após isso, o capítulo detalha o *Story Slicing* 2.3, tema importante para a divisão das tarefas que foram implementadas na parte prática do trabalho. Depois, são passados pelos temas de Desenvolvimento *Mobile* 2.4, Engenharia Reversa 2.5 e Reengenharia 2.6, por serem temas inerentes ao trabalho no intuito de contribuir com uma aplicação móvel. Por fim, são feitas Considerações Finais 2.7 sobre o capítulo, repassando os principais pontos abordados durante o mesmo.

2.1 *Design* de *Software*

2.1.1 Definição

Segundo Martin (2019), o *Design* de *Software* e a Arquitetura de *Software* não possuem diferença alguma e, por definição, são detalhamentos orientados a um conjunto de princípios, normas e técnicas utilizadas no desenvolvimento de um sistema, sendo um processo fundamental, seja ele uma solução web ou um aplicativo. Na ausência desses detalhamentos, o *software* acaba ficando sucetível a diversos problemas futuros, tanto em relação à manutenção, quanto à evolução.

O *design* de *software* é, também, um processo que envolve revisões contínuas. Visto que, à medida que o *software* é desenvolvido e testado, podem surgir necessidades de ajustes no *design* inicial. Assim como uma ponte bem projetada deve ser capaz de suportar cargas e resistir ao tempo, um *design* de *software* eficaz é essencial para criar sistemas eficientes de forma confiável e escalável.

2.1.2 Princípios do *Design*

Construir um sistema pode ser uma tarefa desafiadora, tendo em vista que não existe uma arquitetura pré definida para atender todo e qualquer tipo de solução. Cabe ao arquiteto de *software* analisar e decidir qual caminho seguir para um *design* de qualidade.

O desenvolvimento de *software* é uma área que está em constante evolução, e a

qualidade do código desempenha um papel vital no seu sucesso. Segundo Martin (2019), a única maneira de otimizar esse desenvolvimento é escrever um código limpo. Isso ressalta a importância dos princípios de *design* de *software* na criação de código limpo e de alta qualidade. Com isso, alguns princípios acabaram surgindo no contexto do *design* de *software*, estes são conhecidos como SOLID (MARTIN, 2019).

O acrônimo SOLID simboliza cinco princípios de *design* de *software* que foram criados para desenvolver sistemas de *software* de uma maneira mais compreensível, concisa, flexível e de fácil manutenção, são eles:

- SRP - Princípio da Responsabilidade Única (*Single Responsibility Principle*): Estabelece que uma classe ou módulo deve ter apenas uma razão para mudar. Em outras palavras, uma classe deve ter uma única responsabilidade bem definida. Isso promove o *design* de componentes de software mais coesos e facilita a manutenção, uma vez que as alterações afetam apenas uma área específica;
- OCP - Princípio do Aberto/Fechado (*Open/Closed Principle*): Afirma que um componente de *software* deve estar aberto para extensão, mas fechado para modificação. Isso significa que novas funcionalidades podem ser adicionadas sem alterar o código existente, promovendo sistemas flexíveis e facilmente expansíveis;
- LSP - Princípio da Substituição de Liskov (*Liskov Substitution Principle*): Enfatiza que as subclasses devem ser substituíveis por suas classes base sem afetar o comportamento do programa. Isso garante que a herança seja utilizada de forma coerente e que as classes derivadas mantenham o mesmo contrato das classes base;
- ISP - Princípio da Interface Segregada (*Interface Segregation Principle*): Recomenda que as interfaces sejam específicas para as necessidades dos clientes, evitando interfaces monolíticas. Isso ajuda a evitar a implementação de métodos desnecessários e promove a coesão nas classes que implementam essas interfaces, e
- DIP - Princípio da Inversão de Dependência (*Dependency Inversion Principle*): Sugere que os módulos de alto nível não devem depender diretamente dos módulos de baixo nível, mas sim de abstrações. Isso promove a flexibilidade e a reutilização de componentes, permitindo que diferentes implementações sejam facilmente substituídas.

2.2 Técnicas de Programação

2.2.1 Visão Geral

A sociedade atual está cada vez mais orientada pela tecnologia. Nesse cenário, a tecnologia possui um papel extremamente importante na criação de soluções de *software*,

bem como aplicativos, sistemas e plataformas que impulsionam a vida digital e o mercado. Porém, desenvolver um *software* eficaz e confiável é um grande desafio. Isso requer conhecimento técnico, bom planejamento e aplicação de métodos adequados para atingir os principais objetivos relacionados à qualidade de *software*.

As técnicas de programação são abordagens utilizadas no desenvolvimento de *software* para solucionar problemas, organizar o código e criar sistemas que atendam as necessidades do mercado. Elas visam aprimorar a qualidade, a manutenibilidade e a eficiência do código. Dessa forma, tanto o desenvolvimento quanto as equipes tornam-se mais eficientes.

Existem diversas técnicas de programação dentro da Engenharia de *Software*, cada uma com sua particularidade. A compreensão e a aplicação de forma correta dessas técnicas é de extrema importância em um cenário onde a tecnologia está em constante evolução e demanda cada dia mais por soluções de *software* no mercado em diversas áreas.

2.2.2 TDD (*Test Driven Development*)

Uma técnica de programação comum na Engenharia de *Software* é o TDD (*Test Driven Development*), que é uma abordagem que busca, principalmente, otimizar o desenvolvimento e aumentar a confiabilidade do código. Essa abordagem visa a escrita dos testes unitários antes do desenvolvimento de qualquer linha de código funcional, sendo um processo dividido em três etapas:

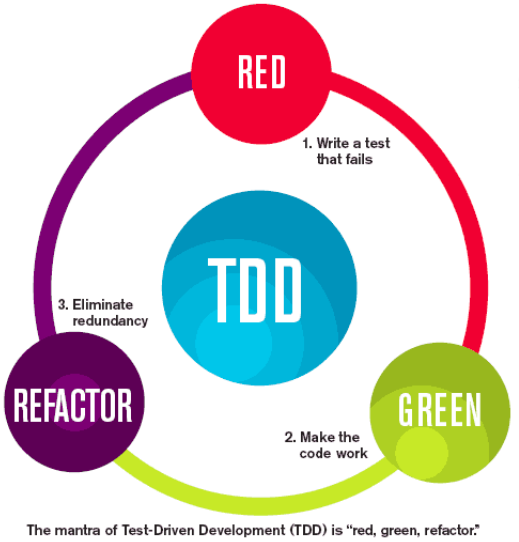
- Escrita do teste (*Red*)- Nessa etapa, o foco é escrever os testes unitários relacionados à funcionalidade, *feature* ou *user story* a ser implementada, sempre visando cobrir o maior número de casos possíveis. Porém, como essa funcionalidade ainda não foi escrita, o teste resulta em falha, por isso o termo *Red* (Vermelho na tradução);

- Escrita do código (*Green*)- Nessa etapa, o foco é a escrita da funcionalidade em si para que os testes sejam atendidos e resultem em sucesso, fazendo com que o *Red* se torne *Green* conforme o esperado, e

- Refatoração do código (*Refactor*)- Nessa etapa, após o código e seus respectivos teste funcionais, o foco é o aprimoramento do código visando uma melhor estrutura e organização. Dessa forma, é garantida sua maior eficácia, confiabilidade e manutenibilidade.

Tais etapas podem ser visualizadas na Figura 1 (DEV, 2023).

Figura 1 – Ciclo do *Test-Driven Development*



FONTE: (DEV, 2023)

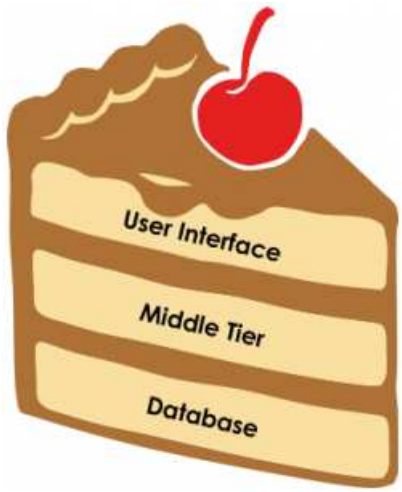
2.3 *Story Slicing*

O *Story Slicing*, ou *Story Splitting*, é essencialmente a divisão das histórias de usuário em níveis de granularidade menores, criando subtarefas programáveis. Com isso, é possível gerar valor para os usuários com rapidez, possuindo adequada familiaridade com o ciclo do desenvolvimento ágil. Além disso, a técnica de desenvolvimento do TDD pode se beneficiar desta metodologia, com os níveis de granularidade menores gerados como blocos de funcionalidades, as quais serão implementadas em suas etapas (BECK, 2022). Existem duas abordagens para o *Story Slicing*, sendo elas o fatiamento horizontal e vertical.

O fatiamento vertical é quando as histórias de usuário são subdivididas verticalmente, de maneira com que os blocos menores ainda resultem em uma funcionalidade que além de ser funcional, possa ser demonstrada, sendo assim útil para os usuários. De maneira geral, pode-se pensar que essa abordagem utiliza de várias camadas técnicas para a implementação de uma fatia. Um exemplo dessa abordagem é a definição de uma subtarefa que, para ser concluída, necessita de implementações na camada gráfica, no banco de dados e nas configurações da aplicação, sendo essas diferentes camadas técnicas do *software* (PARADIGM, 2024).

Já no fatiamento horizontal, pensa-se na divisão baseada nas camadas técnicas existentes na aplicação, podendo assim combinar com as habilidades específicas do time de desenvolvimento. Nesta abordagem, cada subtarefa irá interagir apenas com a camada técnica definida para a mesma. Um exemplo de utilização desta abordagem é a definição de subtarefas de uma história de usuário em que uma fatia implementa apenas a parte do

Figura 2 – *Story Slicing* Vertical



FONTE: (PARADIGM, 2024)

banco de dados; enquanto outra implementa a parte gráfica da mesma história de usuário original. Com a utilização desta funcionalidade, as subtarefas, na maioria das vezes, só irão agregar valor para os usuários quando elas integrarem entre si (PARADIGM, 2024).

Visto as características das abordagens vertical e horizontal, pode-se perceber que a primeira é mais interessante quando o objetivo é entregar pequenos blocos de funcionalidade que agreguem valor aos usuários, e que também interajam com diversas camadas técnicas do projeto. Enquanto a segunda, torna-se mais interessante quando o time de desenvolvimento possui habilidades mais especializadas, e que cada funcionalidade é pensada para interagir apenas com uma camada técnica do *software*.

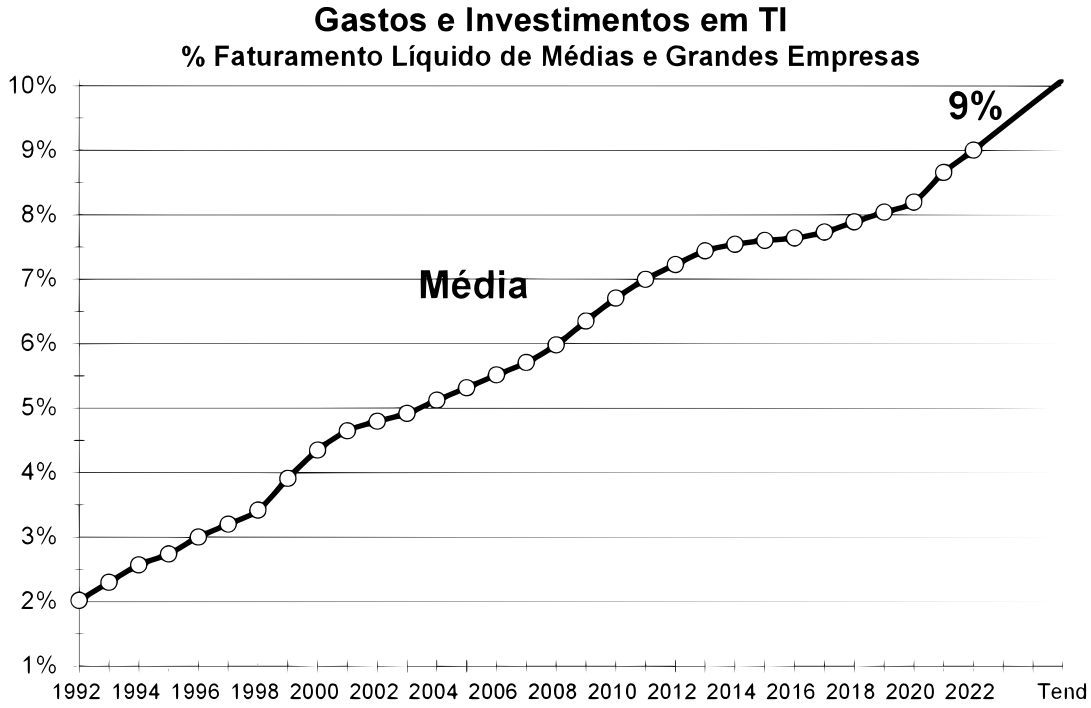
2.4 Desenvolvimento *Mobile*

O uso exponencial de dispositivos móveis nos últimos anos tem sido notável no cotidiano da população em todas as faixas etárias, abrangendo desde a comunicação até tarefas mais complexas, conforme apontado por dados da (FGV, 2023). No Brasil, o registro de, aproximadamente, 464 milhões de dispositivos digitais, incluindo computadores, *notebooks* e *tablets*, destaca-se especialmente pelos 249 milhões de *smartphones* em uso, o que equivale a mais de um dispositivo móvel por habitante, conforme dados também fornecidos pela (FGV, 2023). Essa proliferação de dispositivos móveis tem gerado diversas consequências, como mencionado por (COUTINHO, 2014).

Paralelamente, observa-se um aumento significativo nos investimentos em equipes de Tecnologia da Informação nas empresas, com um custo anual médio de aproximadamente 52 mil reais por usuário, conforme evidenciado na Figura 3.

Diante desse cenário e do considerável avanço das tecnologias, percebe-se uma

Figura 3 – Média de Gastos de Investimentos em TI



FONTE: (FGV, 2023)

clara tendência de crescimento de soluções direcionadas ao público de dispositivos móveis. Destacam-se aplicativos voltados para serviços de entrega, comunicação por meio de bate-papo e reuniões *online*. Nesse contexto, o conceito de *mobile first* ganha destaque, indicando que o desenvolvimento dessas soluções tem como prioridade atender, majoritariamente, aos dispositivos móveis.

Nos últimos anos, o desenvolvimento *mobile* tem testemunhado uma evolução significativa. O destaque ocorre com relação ao crescimento do desenvolvimento multiplataforma, impulsionado por *frameworks* como React Native e Flutter (FREITAS, 2022). Essas ferramentas permitem a criação eficiente de aplicativos a partir de um único código-base, agilizando o processo de desenvolvimento.

Garantir a qualidade em desenvolvimento *mobile* é importante, sendo possível com testes de unidade e integração (SANTOS et al., 2015), fundamental para assegurar a estabilidade dos componentes do aplicativo. Além disso, existem os testes de aceitação do usuário (UI/UX), que se concentram na experiência do usuário final (MELO, 2023). Por fim, o monitoramento contínuo após o lançamento do aplicativo permite uma resposta às falhas, contribuindo para melhorias sucessivas na qualidade do *software*.

2.5 Engenharia Reversa

2.5.1 Definição

De acordo com Eilam (2011), a engenharia reversa é o processo de extrair conhecimento a partir de algo feito pelo ser humano. Este conceito existe desde antes dos computadores e da tecnologia moderna, e se assemelha muito à pesquisa científica clássica, tendo como diferença o objeto de estudo, que é algo produzido pelo homem, e não um fenômeno natural.

A engenharia reversa, tradicionalmente, desmonta produtos físicos para descobrir os segredos de seu *design*. Um exemplo disso é a prática comum entre as pessoas de, por curiosidade, desmontar algum produto eletrônico, como o rádio, para descobrir o funcionamento do mesmo. O conhecimento adquirido com esta prática normalmente é utilizado para construir produtos similares ou melhores (EILAM, 2011).

A evolução da tecnologia vem dificultando cada vez mais esta prática. Os produtos digitais modernos estão com peças cada vez menores, o que torna mais difícil a descoberta de funcionamentos a partir da desmontagem destes dispositivos. O surgimento dos produtos de *software* também trouxe outra dificuldade para esta prática, que por não serem físicos, são impossíveis de serem desmontados. Ainda sim, é possível realizar a engenharia reversa em produtos de *software* (EILAM, 2011).

2.5.2 Engenharia Reversa em *Software*

Assim como acontece com os produtos físicos na engenharia reversa tradicional, as aplicações de *software* também podem ser "desmontadas" a fim de descobrir seu funcionamento e seu *design*. Este processo acontece de forma virtual, envolvendo apenas ferramentas computacionais e o raciocínio lógico. (EILAM, 2011).

A engenharia reversa na Engenharia de *Software* demanda habilidades e entendimento sobre computadores e desenvolvimento de *software*. Existem algumas técnicas que compõem este processo, sendo elas: quebra de código, resolução de quebra-cabeças, programação e análise lógica. Estas técnicas podem ser aplicadas em um arquivo binário de uma aplicação existente, a fim de descobrir suas funcionalidades, ou ainda diretamente no código fonte (EILAM, 2011).

É válido dizer que, na indústria de *software*, a principal motivação para a realização da engenharia reversa para descobrir os segredos de um sistema é a de desenvolver aplicações competidoras. Porém, dependendo da complexidade da aplicação, torna-se um processo muito custoso, não valendo a pena despender recursos e tempo para isso. Nestes casos, desenvolver a aplicação do zero, elicitando os requisitos e funcionalidades necessárias, torna-se um processo mais fácil e mais barato (EILAM, 2011).

Na área de *hardware*, é comum realizar projetos novos a partir da engenharia reversa em produtos finalizados. Já na área de *software*, a engenharia reversa pode ser utilizada com o intuito de apoiar a manutenção, a evolução ou até a substituição de um sistema já implementado. Neste caso, o objetivo principal deixa de ser o de produzir uma aplicação competidora, e passa a ser de aprimorar um sistema no qual o código fonte esteja disponível para a equipe desenvolvedora (CAGNIN, 2005).

A engenharia reversa, quando utilizada com o objetivo de analisar um sistema em funcionamento, é um processo que realiza a abstração em alto nível dos componentes e relacionamentos do projeto. Ela pode ser classificada como redocumentação ou recuperação de projeto. Na redocumentação, a engenharia reversa preocupa-se em criar ou melhorar a documentação sobre o sistema analisado. Já na recuperação de projeto, o objetivo é compreender completamente o sistema, entendendo o que o sistema faz, de qual maneira e o porquê de realizar desta forma, para posteriormente, aplicar a reengenharia do sistema (CAGNIN, 2005).

2.6 Reengenharia

2.6.1 Visão Geral

Segundo Chikofsky e Cross (1990), existem seis termos que são relacionados à reengenharia de *software*, sendo eles: engenharia avante, engenharia reversa, redocumentação, recuperação do projeto, reestruturação e reengenharia. Ao introduzir estes termos, os autores tiveram como objetivo racionalizar e padronizar termos que já estavam sendo utilizados na área de Engenharia de *Software*. Estes termos são apresentados a seguir:

A engenharia avante consiste na implementação dos requisitos de projeto e modelos lógicos que são abstraídos numa fase anterior à fase de desenvolvimento. A engenharia reversa tem o objetivo de analisar um sistema legado, possuindo duas classificações, a de redocumentação e a de recuperação do projeto, como foi explicado na seção 2.5.2. A reestruturação trata-se de manter a mesma funcionalidade de um sistema, porém, com uma outra implementação, sendo um processo também conhecido como refatoração. Por fim, a reengenharia consiste na renovação de um sistema já implementado (CHIKOFSKY; CROSS, 1990).

A reengenharia de *software* é o processo de analisar e alterar o sistema, mantendo os requisitos da aplicação original, mas aplicando novas abordagens de desenvolvimento. Normalmente, o primeiro passo para se realizar a reengenharia, é a realização da engenharia reversa, para conhecer o funcionamento e *design* do produto, para depois realizar a engenharia avante ou reestruturação, com o objetivo de re-implementar o sistema de outra forma (CAGNIN, 2005).

Um produto de *software* é algo que passa por constantes evoluções, para satisfazer novas necessidades de seus usuários e também para resolver *bugs* [1] existentes no sistema. Estas constantes atividades de manutenção que os desenvolvedores fazem, juntamente com a falta de atualizações na documentação do *software*, muitas vezes degradam não só o código fonte, como também o projeto como um todo. Estes sistemas degradados, que são difíceis de serem mantidos e evoluídos, mas ainda sim estão em funcionamento, são chamados de sistemas legados (CAGNIN, 2005).

2.6.2 Reengenharia em Sistema Legado

No contexto do desenvolvimento de *software*, muitas vezes os desenvolvedores devem lidar com código legado ao invés de desenvolver novos códigos. Existem diversos tipos de projetos legados, mas uma característica que normalmente aparece em comum é a dificuldade de realizar a manutenção e evolução do código fonte. Esta dificuldade está atrelada a algumas propriedades dos projetos legados, como o tempo de existência do projeto, alta quantidade de linhas de código, mudanças dentro do time de desenvolvimento e a má documentação da aplicação (BIRCHALL, 2016).

De acordo com Sneed (1995), a reengenharia em projetos legados possui quatro objetivos principais, sendo eles: melhorar a manutenibilidade; migrar o sistema; alcançar maior confiabilidade, e aumentar a escalabilidade. Existem diversas formas para alcançar estes objetivos, como dividir o sistema em módulos menores ou até migrar para uma infraestrutura mais adequada. Ao realizar a reengenharia, as funcionalidades originais são preservadas, mas implementadas em uma nova arquitetura, sendo uma grande vantagem desta técnica.

Assim como existem diversos tipos de sistema legado com diversos tipos de problemas e características, também existem diversas formas de se conduzir a reengenharia de *software*. Pensando nisso, Rosenberg e Hyatt (1996) propõe quatro tipos de abordagens para se realizar a reengenharia, sendo elas: a "*big bang*", a incremental, a evolutiva e a híbrida. Estas abordagens de reenharia de *software* serão abordadas a seguir.

- A abordagem "*big bang*", assim como o nome sugere, é feita de uma forma que o sistema legado é totalmente trocado por um sistema novo, como pode ser evidenciado na Figura 4(a). A vantagem de se utilizar esta abordagem é a de não precisar ter a compatibilidade entre componentes do sistema anterior com o sistema novo. Já as desvantagens são que esta técnica não é apropriada para grandes sistemas, e também as eventuais alterações no código fonte legado que foram feitas

[1] Segundo o Dicionário Escolar da Língua Portuguesa DICIO, *Bug* significa "Falha ou erro num programa informático, dispositivo, *site* etc., que impede o seu funcionamento correto ou causa um mau desempenho"(DICIO, 2020)

durante o desenvolvimento do novo sistema devem ser implementadas novamente (ROSENBERG; HYATT, 1996).

- A abordagem incremental é feita de forma gradativa, substituindo aos poucos os componentes do sistema legado por componentes novos, por meio de versões do novo sistema, como pode ser evidenciado na Figura 4(b). As vantagens desta técnica são que a reengenharia em blocos menores é mais veloz e mais fácil de se perceber falhas, e também é possível de se disponibilizar versões para os usuários, conforme os componentes são trocados, podendo ocorrer a validação por meio de testes de aceitação. As desvantagens desta técnica são a de existir a necessidade de um grande controle no ambiente e configuração para os dois sistemas coexistirem, e também por deixar a arquitetura do sistema menos maleável, pois nesta técnica apenas o interior dos componentes são alterados, não havendo uma mudança no *design* da aplicação (ROSENBERG; HYATT, 1996).

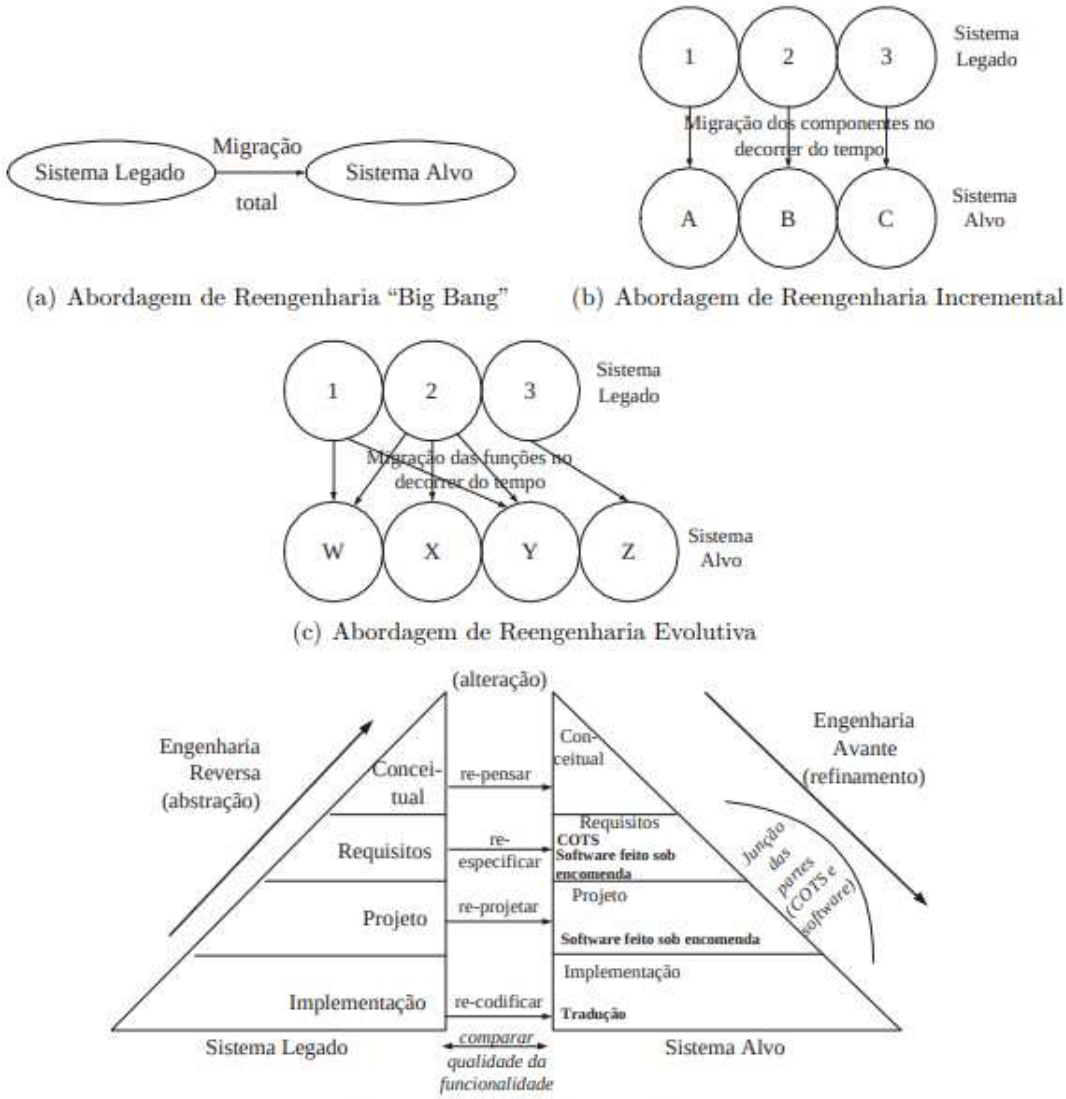
- A abordagem evolutiva, assim como na abordagem incremental, também ocorre de forma gradativa, com a diferença de migrar os componentes baseado em funcionalidades em comum, e não apenas na estrutura, como pode ser visto na Figura 4(c). A grande vantagem desta técnica é a centralização de funcionalidades semelhantes. Já a desvantagem é o de exigir muito esforço em agrupar as funcionalidades em comum que estão espalhadas pelos diversos componentes do sistema legado (ROSENBERG; HYATT, 1996).

- A abordagem de reengenharia híbrida realiza a migração do sistema legado a partir de abstrações e métodos específicos, como pode ser visto na Figura 4(d). Nesta técnica, o código fonte é utilizado para a criação de um novo projeto para o sistema, no qual são elicitados os requisitos necessários e também é identificado quais funcionalidades do projeto legado estão obsoletas e não precisam constar no novo projeto. E por fim, é realizada a codificação deste novo projeto (ROSENBERG; HYATT, 1996).

2.7 Considerações Finais do Capítulo

Este capítulo preocupou-se em mostrar uma visão geral sobre os pontos de maior relevância que compõem a fundamentação teórica deste trabalho. Primeiramente, foram apresentadas conceituações e detalhamentos sobre *Design* de *Software*, para apresentar este conceito da Engenharia de *Software* e para mostrar a importância de uma boa escolha de *design* para uma aplicação de qualidade. Também é falado sobre o DDD, que é o modelo de *design* escolhido para o novo sistema, implementado para o caso do Mia Ajuda ao longo desse trabalho.

Figura 4 – Abordagens de Reengenharia



(a) Abordagem de Reengenharia "Big Bang"

(b) Abordagem de Reengenharia Incremental

(c) Abordagem de Reengenharia Evolutiva

(d) Abordagem de Reengenharia Híbrida

FONTE: (CAGNIN, 2005)

Depois, há a abordagem do tema de Técnicas de Programação, na qual é feita uma conceituação e uma explicação da importância da utilização de boas técnicas na implementação de um sistema de *software*, para garantir a qualidade e testabilidade do mesmo. Além disso, também é apresentado o TDD, que é uma técnica utilizada na implementação da aplicação prática deste trabalho.

Outro tópico abordado foi o de Desenvolvimento *Mobile*, que está relacionado ao aplicativo MiaAjuda (2020), aplicativo este que é o alvo das contribuições deste trabalho. Neste tópico, são abordadas questões sobre o mercado *mobile* e também sobre o desenvolvimento de aplicativos móveis.

Por fim, são apresentados os conceitos de Engenharia Reversa e Reengenharia, dois

tópicos que conversam entre si, visto que o primeiro passo para a reengenharia de *software* é a própria engenharia reversa. Nestes tópicos, são feitas definições e também considerações sobre como estes conceitos são importantes na hora de revitalizar um sistema legado e melhorar sua manutenibilidade.

3 Suporte Tecnológico

Este capítulo tem como principal objetivo apresentar as principais ferramentas e tecnologias utilizadas na elaboração desse trabalho. De início, serão citadas as ferramentas relacionadas ao desenvolvimento, tais como o *framework* Node.js e as ferramentas de virtualização de ambientes Docker e o Jest para desenvolvimento de testes automatizados. Em seguida, serão apresentadas as ferramentas de apoio ao desenvolvimento, por exemplo, o MongoDB, que é um sistema de gerenciamento de bancos de dados NoSQL (*Not Only SQL*). Outro exemplo é o SonarQube, que é uma ferramenta de análise de qualidade código que possibilita ter acesso a diversas métricas. Tem-se ainda o GitHub, utilizado para o gerenciamento e armazenamento de código, e o Draw.io para elaboração de diagramas. Por fim, encontram-se as Considerações Finais do Capítulo.

3.1 Ferramentas de Desenvolvimento

3.1.1 Node

Como principal ferramenta de desenvolvimento *back-end*, tem-se o Node [1], um *framework open-source* desenvolvido em JavaScript, que é bastante popular em soluções de grandes empresas como Netflix, LinkedIn, Uber, PayPal e Mozilla (KINSTA, 2023). Nesse cenário, é importante citar, também, o *Express* [2] que é um outro *framework open-source* construído com base no Node, e que simplifica o desenvolvimento de aplicações *web* e APIs (*Application Programming Interfaces*), fornecendo uma camada de abastração sobre as funcionalidades básicas do Node.

Esse *framework* tem como principais características, o seu roteamento que permite a definição de rotas para URLs específicas, facilitando seu mapeamento. A possibilidade de implementação de *Middlewares*, que são componentes ou funções que podem ser executadas em tempo de requisição HTTP, facilitando a execução de tarefas como autenticação e validação.

Além disso, podem ser citadas algumas vantagens como: gerenciamento de requisições e respostas; suporte aos vários métodos HTTP; facilidade de integração; escalabilidade e desempenho; relevância no mercado atual, e tamanho da sua comunidade, que acaba sendo de extrema importância para a evolução do *framework*.

[1] https://nodejs.org. Acessado pela última vez em 31/10/2023.
[2] https://expressjs.com/. Acessado pela última vez em 31/10/2023.

3.1.2 Jest

O Jest [3] é um *framework* de testes automatizados amplamente utilizado na comunidade JavaScript, principalmente, no *back-end* de projetos que utilizam Nest [4] e Node [5]. É uma ferramenta que foi desenvolvida pelo Facebook, e visa facilitar o processo de escrita de testes unitários e de integração.

Entre suas principais características, podem ser citadas sua facilidade de escrita onde os testes acabam ficando mais descritivos e legíveis; possibilidade de utilização de *mocks* para simulação de resultados, retornos e comportamentos específicos; possibilidade de verificar a porcentagem de cobertura, tanto total do código, quanto total de cada arquivo, e fácil integração com ferramentas de Integração e Deploy Contínuo (respectivamente em inglês, *Countinuos Integration* (CI) e *Continuous Deploy* (CD)) para a garantir que os testes sejam sempre executados quando houver uma alteração no código.

3.1.3 Docker

O Docker [6] é uma ferramenta *open-source* que visa facilitar o desenvolvimento, a execução e a implantação de um *software* através de contêineres. Os contêineres são ambientes isolados, onde todas as dependências do projeto estão empacotadas. Isso garante que, em todas as etapas, o ambiente seja o mesmo, isento de qualquer tipo de falha por incompatibilidade de versão ou algo do gênero.

Como principais características, podem ser citadas: portabilidade; eficiência, comparada às *Virtual Machines* (VMs); facilidade de orquestração; facilidade de subir um container de determinado *framework* ou linguagem específica, e segurança e cofiabilidade empregadas na implantação do Docker em um projeto.

3.1.4 MongoDB

O MongoDB [7] é um banco de dados NoSQL de código livre e orientado a documentos. Assim como se espera de um banco de dados NoSQL, o MongoDB possui uma alta flexibilidade no armazenamento de dados, utilizando um formato chamado BSON, que é uma representação binária de dados. Isso ocorre de forma diferenciada em relação aos bancos de dados SQL, os quais armazenam dados em tabelas de linhas ou colunas. Com o armazenamento em representação binária, as aplicações podem interagir com o banco de dados por meio do formato JSON, sendo esse um formato muito utilizado atualmente.

[3] https://jestjs.io. Accesado pela última vez em 31/10/2023
[4] https://nestjs.com. Acessado pela última vez em 31/10/2023.
[5] https://nodejs.org. Acessado pela última vez em 31/10/2023.
[6] https://www.docker.com. Accesado pela última vez em 31/10/2023
[7] https://www.mongodb.com. Accesado pela última vez em 20/06/2024

Um exemplo de um documento JSON descrevendo uma figura histórica pode ser visto na Figura 5.

Figura 5 – Exemplo de JSON em MongoDB

FONTE: (MONGODB, 2023)

Bancos de dados de documento são altamente flexíveis, o que permite que os documentos possam ser salvos parcialmente completos, e também podendo ter diversas estruturas. Nos bancos de dados SQL, é possível realizar a indexação das colunas, para melhorar o desempenho nas operações do banco. O mesmo pode ser feito no MongoDB, mas ao invés de colunas, a indexação acontece nos campos do documento.

O MongoDB é muito popular atualmente, por possuir um fácil armazenamento de dados tanto estruturados como não estruturados, o que é possível por se tratar de um banco de dados de documentos. A possibilidade de armazenar os dados no formato JSON e por dispensar a necessidade de normalizar os dados também são propriedades muito atrativas para os desenvolvedores.

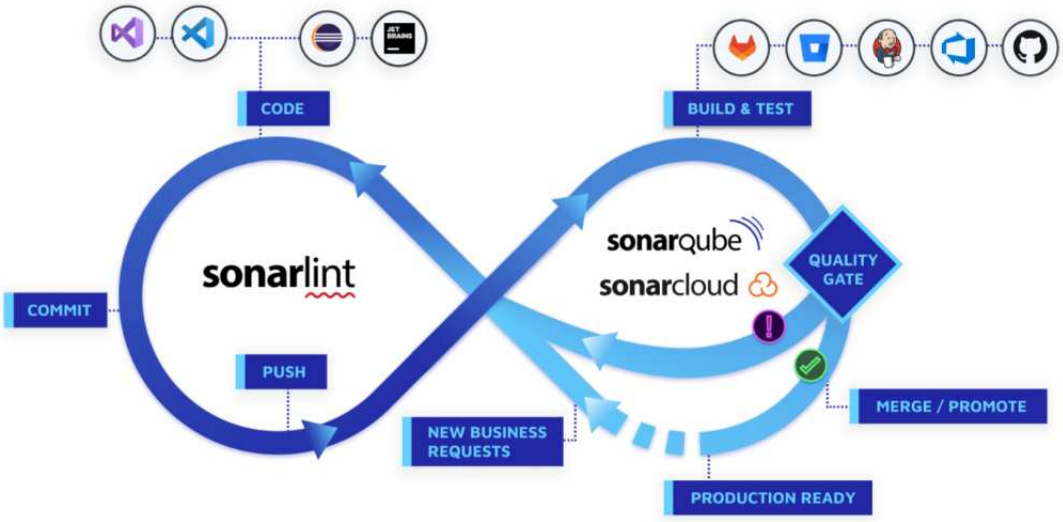
Além da flexibilidade no armazenamento, o MongoDB também possui uma arquitetura muito escalável, o que permite que pequenas máquinas trabalhem juntas para criar sistemas velozes e lidar com altos volumes de dados. Com isso, o MongoDB traz muitos benefícios para aplicações que precisam armazenar e interagir com uma alta carga de dados.

3.1.5 SonarQube

O SonarQube [8] é uma ferramenta auto gerenciável e automática de análise de código, que ajuda na implementação das boas práticas de *Clean Code* em um projeto de *software*. Esta ferramenta pode ser inserida no *workflow* de uma aplicação para gerar uma análise contínua de problemas no código fonte e sugestões de melhoria. O SonarQube possui suporte para uma grande variedade de linguagens de programação e pode ser integrado com *pipelines* de CI e plataformas de DevOps, com o objetivo de garantir que a aplicação sempre esteja de acordo com os padrões de qualidade estabelecidos.

[8] https://docs.sonarsource.com/sonarqube/latest. Accesado pela última vez em 31/10/2023

Figura 6 – SonarQube *Workflow*



FONTE: (SONARQUBE, 2023)

A Figura 6 demonstra como o SonarQube pode ser utilizado no processo de desenvolvimento de *software*. Esta ferramenta pode ser integrada ao CI/CD de um projeto a fim analisar os *Pull Requests* criados, e com isso, toda nova submissão de código terá sua qualidade analisada. Com essa análise de *Pull Requests*, é possível configurar um portão de qualidade, que, a partir de critérios de qualidade determinados, delimita se o código fonte submetido está de acordo com os padrões de *Clean Code* e está apto a ir para produção, ou se ele deve voltar para a etapa de desenvolvimento.

A análise que o SonarQube realiza no projeto colabora muito com a avaliação de qualidade do mesmo. Com os dados fornecidos pela ferramenta, o time de desenvolvimento pode facilmente realizar correções no código a fim de aumentar sua qualidade, e também resolver possíveis vulnerabilidades presentes na implementação.

3.2 Ferramentas de Apoio

3.2.1 GitHub

O GitHub [9] é uma plataforma muito popular de hospedagem de projetos de *software* utilizando o Git [10], permitindo que desenvolvedores do mundo todo contribuam em projetos privados ou públicos. O Git é uma ferramenta de código livre e gratuita para versionamento de código fonte, que possui diversas funcionalidades para o desenvolvimento colaborativo de *software*.

[9] https://github.com. Accesado pela última vez em 30/06/2024
[10] https://git-scm.com. Accesado pela última vez em 31/10/2023

3.2.2 Draw.io

O Draw.io [11] é um *software* online e grátis para a modelagem de diagramas. Esta ferramenta foi utilizada para criar os diversos tipos de diagramas presentes na *Engenharia de Software*, como os diagramas UML, que são muito populares na fase de planejamento de produtos de *software*.

3.3 Considerações Finais do Capítulo

Neste capítulo, foram apresentadas as principais ferramentas de desenvolvimento e de apoio utilizadas no desenvolvimento desse trabalho. A Tabela 1 tem o objetivo de fornecer um resumo dos principais suportes tecnológicos.

Tabela 1 – Suporte Tecnológico

Nome	**Descrição**	**Versão**	**Link**
Node	*Framework* para criação de APIs.	20.9.0	<https://nodejs.org/>
Jest	*Framework* para criação de testes automatizados.	29.7.0	<https://jestjs.io/>
Docker	Ferramenta de virtualização de ambientes.	24.0.0	<https://www.docker.com/>
MongoDB	Banco de dados NoSQL para armazenamento de dados.	7.0	<https://www.mongodb.com/>
SonarQube	Ferramenta de análise e controle de qualidade de código fonte.	10.0	<https://www.sonarsource.com/>
GitHub	Plataforma de hospedagem de projetos de *software*.	-	<https://github.com/>
Draw.io	Ferramenta de modelagem de diagramas.	-	<https://www.drawio.com/>

FONTE: Autores

[11] https://www.drawio.com. Accesado pela última vez em 31/10/2023

4 Metodologia

Este capítulo aborda a metodologia do trabalho, que foi utilizada para o desenvolvimento teórico e prático do mesmo. No primeiro momento, será abordada a Classificação da Pesquisa 4.1, considerando a Abordagem 4.1.1, a Natureza 4.1.2, os Objetivos 4.1.3 e os Procedimentos 4.1.4 da pesquisa. Na sequência, serão abordadas as metodologias que fundamentam este trabalho, em sua parte teórica e prática, sendo elas: Metodologia Investigativa 4.2, Metodologia Orientada a Provas de Conceito 4.3, Metodologia de Desenvolvimento 4.4 e Metodologia de Análise de Resultados 4.5. Além disso, também serão apresentados o Fluxo de Atividades/Subprocessos 4.6 e o Cronograma de Atividades/Subprocessos 4.7, que descrevem o escopo e as atividades definidas para a primeira e segunda etapas do TCC. Por último, serão apresentadas as Considerações Finais do Capítulo 4.8, recapitulando as seções abordadas neste capítulo.

4.1 Classificação da Pesquisa

De acordo com Gerhardt e Silveira (2009), a pesquisa científica é a atividade principal da Ciência. Ela utiliza métodos científicos de forma sistemática para explicar ou solucionar problemas. Os autores Gerhardt e Silveira (2009) ainda classificam uma pesquisa em quatro termos, sendo eles: abordagem, natureza, objetivos e procedimentos.

4.1.1 Abordagem

A abordagem de pesquisa deste trabalho pode ser classificada como **qualitativa** e **quantitativa**. A abordagem qualitativa tem o objetivo de aprofundar o conhecimento sobre um tema, utilizando de dados não-métricos. Logo, ela não busca quantificar valores, mas sim explicar o porquê das coisas através destes dados. Em contrapartida, a abordagem quantitativa busca quantificar seus resultados, através da análise de dados brutos que são coletados por ferramentas sistemáticas e neutras (GERHARDT; SILVEIRA, 2009).

Este trabalho busca analisar e quantificar dados brutos que podem ser coletados através da ferramenta SonarQube, sendo eles: quantidade de *bugs*; complexidade ciclomática; segurança do código, dentre outros. Porém, além desta análise quantitativa, o trabalho também busca uma abordagem qualitativa ao analisar dados não-métricos, como por exemplo, o desempenho da aplicação após a reengenharia do código fonte em comparação com o projeto original.

4.1.2 Natureza

Este trabalho pode ter sua natureza classificada como uma **pesquisa aplicada**. Este tipo de pesquisa busca gerar conhecimentos a partir de uma aplicação prática, e assim, solucionar problemas específicos (GERHARDT; SILVEIRA, 2009). A aplicação prática deste trabalho busca responder as questões levantadas nas Questões de Pesquisa e de Desenvolvimento 1.3.

4.1.3 Objetivos

Este trabalho tem seus objetivos classificados no grupo da **pesquisa exploratória**. Esta pesquisa exploratória tem o intuito de trazer uma maior familiaridade com o problema, a fim de o tornar mais compreensível ou de formular hipóteses a partir dele (GERHARDT; SILVEIRA, 2009). Esta abordagem de objetivos está alinhada com o problema descrito na seção de Justificativa 1.2 do trabalho.

4.1.4 Procedimentos

Em relação aos procedimentos deste trabalho, pode-se citar a **pesquisa bibliográfica**, definida pelos autores Gerhardt e Silveira (2009), que consiste no levantamento de referências teóricas já publicadas no meio científico, como artigos e livros, a fim de conhecer o que já foi estudado sobre o tema. Esta pesquisa compõe a Metodologia Investigativa 4.2 do trabalho.

Este trabalho também conta com uma pesquisa de **provas de conceito**, que compõe a Metodologia Orientada a Provas de Conceito 4.3. Com essa abordagem, é possível descrever desafios que evidenciam as capacidades de um *software*, e verificam se o mesmo está cumprindo as necessidades dos clientes e usuários (PRASANNA et al., 2021).

4.2 Metodologia Investigativa

De acordo com Gil et al. (2002), um trabalho científico inicia-se com uma pesquisa bibliográfica, que permite com que os autores conheçam o tema, que já foi definido, por meio de fontes científicas que já o abordaram antes. Assim, após a definição do tema deste trabalho, realizou-se um levantamento teórico na base científica Google Scholar, que é uma base consolidada, que permite a busca não só em artigos, mas também em livros, revistas e outras fontes científicas, fornecendo uma busca mais abrangente. A Tabela 2 contém as *strings* de busca que foram utilizadas neste levantamento, assim como a quantidade de fontes retornadas.

Tabela 2 – *Strings* de Busca

Base de dados	***String***	**Quantidade**
Google Scholar	'reverse engineering'	4.610.000
Google Scholar	'reengineering'	414.000
Google Scholar	'test-driven development (TDD)'	13.500
Google Scholar	'mobile development'	5.940.000
Google Scholar	'software design'	8.900.000
Google Scholar	'reengineering' AND 'reverse engineering'	60.500
Google Scholar	'reengineering' AND 'TDD' OR 'DDD'	3.300
Google Scholar	'TDD' AND 'DDD'	11.000
Google Scholar	'software design' AND 'mobile development'	4.740.000
Google Scholar	'TDD' AND 'DDD'	11.000

FONTE: Autores

4.2.1 Critérios de Seleção

A fim de filtrar as principais fontes a partir do material obtido no levantamento bibliográfico, tornou-se necessário realizar uma análise exploratória. Essa análise foi feita pelos autores, a partir da leitura do resumo e das palavras-chave dos trabalhos, seguindo os seguintes critérios de seleção:

- Abordar engenharia reversa no contexto da *Engenharia de Software*;
- Abordar reengenharia de *software*;
- Abordar desenvolvimento utilizando a técnica TDD e/ou orientado ao DDD, e
- Disponíveis gratuitamente.

Houve certa dificuldade em refinar o material obtido na Tabela 2 com os critérios de seleção definidos. Esta dificuldade pode ser explicada primeiramente pelo fato dos temas de Engenharia Reversa e Reengenharia não serem exclusivos da Engenharia de *Software*. Muitas fontes os abordam em contextos da manufatura e de outras áreas, não se relacionando com o contexto deste trabalho, de utilizar estas práticas em um produto de *software* já existente.

Outra dificuldade encontrada foi a falta de exemplos práticos da utilização da técnica de programação TDD juntamente com o desenvolvimento orientado ao DDD. Embora ambos tenham uma alta popularidade na comunidade de *software*, eles não se

tratam de uma associação óbvia, por atacarem áreas distintas de um produto de *software* (testes e domínio).

As dificuldades encontradas foram contornadas por meio da alta quantidade de materiais retornados pelas *strings* de busca definidas na Tabela 2, o que possibilitou o refinamento dos resultados obtidos e a seleção de fontes para compor a base bibliográfica deste trabalho. Os trabalhos escolhidos foram:

- *Test driven development: By example* (BECK, 2022);
- *Implementing domain-driven design* (VERNON, 2013);
- *Refactoring* (FOWLER, 2018);
- *Design principles and design patterns* (MARTIN, 2000), e
- *Reverse engineering and design recovery: A taxonomy* (CHIKOFSKY; CROSS, 1990).

Adicionalmente, torna-se necessário um outro estudo, que busca atender aos desafios definidos nas provas de conceito. O processo investigativo deste estudo é o mesmo que foi realizado anteriormente, seguindo os mesmos passos de pesquisa bibliográfica. Este subprocesso pode ser visto na Figura 7, que ilustra, utilizando a modelagem BPMN, as atividades presentes na metodologia investigativa.

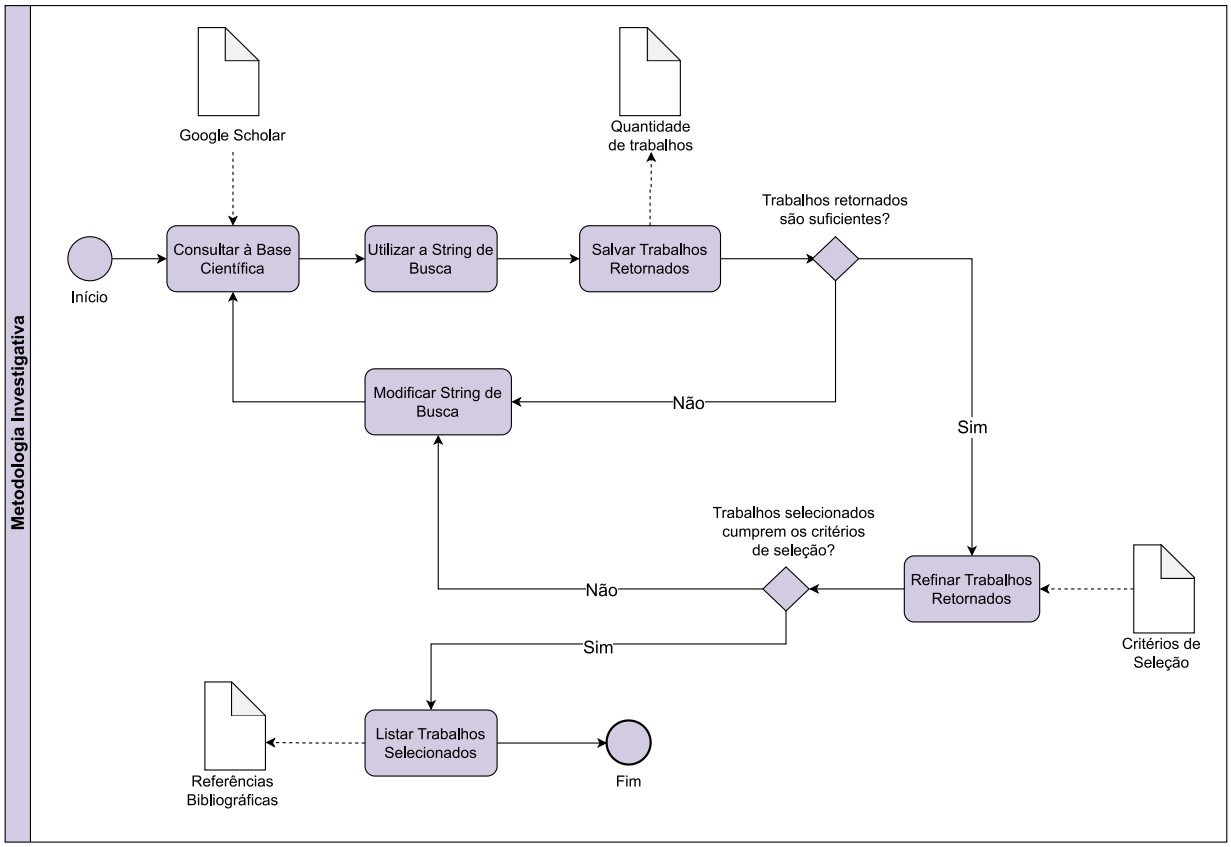
4.3 Metodologia Orientada a Provas de Conceito

No contexto do desenvolvimento de *software*, o termo Prova de Conceito (em inglês, *Proof of Concept* - POC), é uma ferramenta que pode ser utilizada para demonstrar as capacidades de um *software*, assim como sua viabilidade e se o mesmo atende as necessidades dos clientes e dos usuários. As provas de conceito podem ser aplicadas em diversas áreas, como *marketing* e medicina. Já no domínio da *Engenharia de Software*, elas seguem um processo específico que pode utilizar o desenvolvimento de *hardware*, *websites* ou outro tipo de *software* que implemente um conceito (PRASANNA et al., 2021).

Nesta metodologia, o primeiro passo é estabelecer uma etapa inicial, também chamada de Definição do Desafio. Esta etapa possui o objetivo de fornecer uma compreensão clara do desafio que será abordado na prova de conceito, e pode ser feita por meio exclusivo de elementos textuais, ou também pode ser complementada com imagens e códigos fonte de exemplo (SILVA, 2023).

Em seguida, vem a etapa do Estudo da Literatura, que tem o objetivo de fundamentar a solução para o desafio proposto por meio do embasamento teórico. Nesta etapa,

Figura 7 – Fluxograma da Metodologia Investigativa



FONTE: Autores

é feito um estudo investigativo, com um olhar teórico e técnico, a fim de levantar conceitos e tecnologias adequadas para a futura solução, de forma que a mesma possua muita coerência e solidez (SILVA, 2023).

Com o Estudo da Literatura realizado e pronto para fundamentar a solução, é possível realizar a apresentação da mesma. Na etapa de Apresentação da Solução, são utilizados elementos ilustrativos, como imagens, códigos fonte e outros elementos, a fim de fornecer mais clareza para expor esta solução. Uma solução deve atender o desafio proposto de forma satisfatória, não tendo a responsabilidade de o resolver de forma completa e definitiva, mas sim seguir as orientações definidas pela literatura especializada (SILVA, 2023).

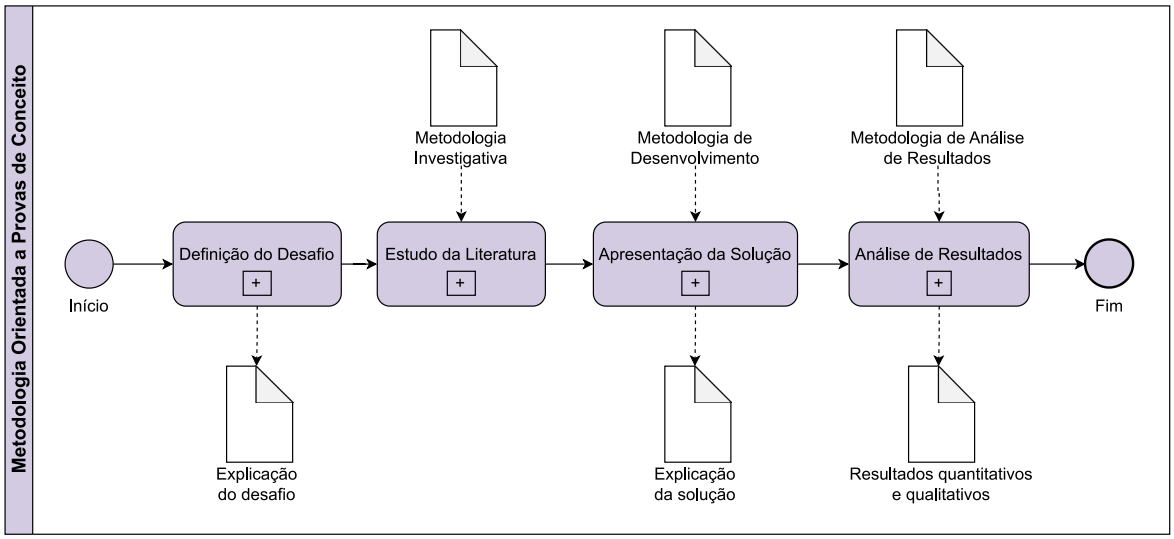
Após a apresentação da solução, faz-se necessária a realização de uma análise profunda e objetiva dos resultados obtidos, através da etapa de Análise de Resultados, podendo esta ser conduzida de maneira qualitativa ou quantitativa. No contexto qualitativo, recorre-se às percepções adquiridas pelo programador e/ou especialista ao longo do tempo. No contexto quantitativo, os insumos métricos advindos de ferramentas de coleta de dados, como o SonarQube, podem ser utilizados. Esta etapa de análise tem o objetivo evidenciar os pontos positivos e negativos percebidos durante o processo (SILVA, 2023).

Na literatura, existem diversas abordagens metodológicas para as provas de conceito, o que torna uma escolha livre para os pesquisadores qual dessas abordagens adotar para o trabalho. Neste trabalho, a abordagem metodológica escolhida está definida em Turilli et al. (2016), que define uma metodologia sequencial para as provas de conceito. Com isso, uma próxima prova de conceito depende do sucesso da prova realizada anteriormente, sendo assim, uma orientação em *Building Blocks*.

De acordo com Turilli et al. (2016), o conceito de *Building Blocks*, assim como o nome sugere, é a realização da prova de conceito por meio de blocos de construção, que são etapas que devem ser executadas de maneira sequencial. Esta abordagem busca reduzir um grande problema em pequenos módulos mais compreensíveis. Ao utilizar este conceito, os autores buscam reduzir o tema proposto, que possui uma certa complexidade, em desafios menores utilizando as provas de conceito.

Ao utilizar as etapas definidas por esta metodologia (Definição do Desafio, Estudo da Literatura, Apresentação da Solução e Análise de Resultados), juntamente com o conceito dos *Building Blocks*, foi possível realizar o estudo exploratório deste trabalho. As provas de conceito foram realizadas de forma sequencial, com ajustes pontuais e justificados caso alguma das provas não fosse concluída com sucesso na íntegra. A Figura 8 ilustra, por meio da modelagem BPMN, as etapas da Metodologia Orientada a Provas de Conceito.

Figura 8 – Fluxograma da Metodologia Orientada a Provas de Conceito



FONTE: Autores

4.4 Metodologia de Desenvolvimento

A metodologia empregada no desenvolvimento prático deste trabalho foi híbrida, fundamentada na combinação dos principais elementos das metodologias ágeis Scrum

(SUTHERLAND, 2014) e Kanban (AHMAD; MARKKULA; OIVO, 2013), considerando a sua relevância e popularidade no atual cenário de mercado e desenvolvimento de *software* (CARVALHO; MELLO, 2012).

O Scrum apresenta diversas características individuais, como reuniões diárias, de planejamento e retrospectivas, além da definição de papéis e responsabilidades dos membros da equipe durante o processo de desenvolvimento de um produto (SUTHERLAND, 2014). No entanto, optou-se por uma abordagem mais sutil e simplificada em relação ao Scrum clássico, priorizando o uso do *Product Backlog* como principal ferramenta, combinando com práticas do Kanban e dispondo as tarefas no Trello.

A construção do *Product Backlog* envolve a especificação do projeto em diversos níveis de granularidades, incluindo Épicos, *Features*, *User Stories*, Tarefas e Critérios de Aceitação, onde tudo o que é definido no *backlog* deve agregar valor ao produto. A partir dessa definição, são iniciadas várias iterações dentro do processo de desenvolvimento, conhecidas como *sprints*. As *sprints* são períodos de tempo curtos e bem definidos pelos times para o desenvolvimento das tarefas do *backlog* (SUTHERLAND, 2014).

Como mencionado anteriormente, o uso do Kanban foi de extrema relevância para o desenvolvimento deste trabalho. O Kanban é uma prática que tem como objetivo visualizar, de forma facilitada, o andamento do processo de desenvolvimento de um produto. Além de facilitar o gerenciamento das tarefas, indica o *status*, o responsável por determinada tarefa e limita a quantidade que cada integrante da equipe pode realizar.

Dessa forma, a metodologia de desenvolvimento prático desse trabalho foi baseada, principalmente em:

- Definição do *Backlog*: Nessa etapa, foi desenvolvido todo o *backlog* de tarefas que foram cumpridas até o final do cronograma definido. Todas as tarefas foram iniciadas com o *status ToDo*, inicialmente seguindo o Kanban (AHMAD; MARKKULA; OIVO, 2013);

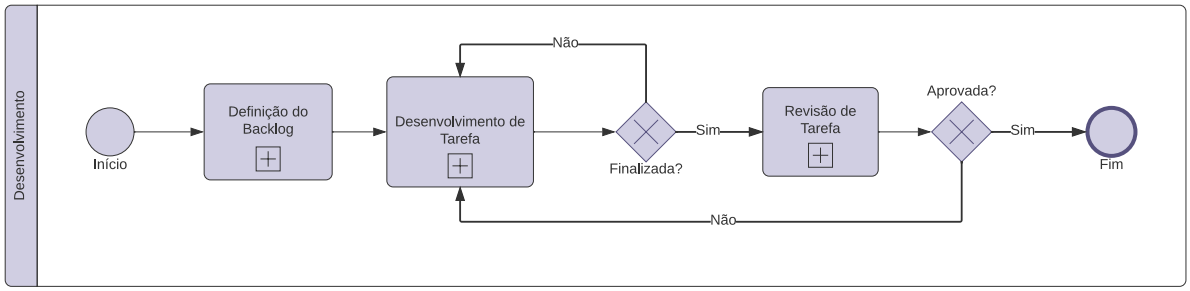
- Desenvolvimento da Tarefa: Nessa etapa, as tarefas que foram selecionadas para serem desenvolvidas dentro da *sprint* foram distribuídas entre os integrantes. O *status* foi alterado para *Doing* no momento em que o respectivo integrante deu início ao desenvolvimento da tarefa. O *status* foi, novamente, alterado para *Done* novamente quando a tarefa foi finalizada com a possibilidade de seguir com o desenvolvimento de uma nova tarefa ou retornar para a revisão de uma tarefa já finalizada, e

- Revisão de Tarefa: Nessa etapa, foi realizado o envio das tarefas para avaliação por parte da equipe do projeto (autores e orientadores). No caso de aprovação da tarefa, o *status* da mesma era alterado para *Reviewed*. Caso fosse necessária alguma

alteração, a tarefa retornava para desenvolvimento, assim como seu *status* que era alterado para *Doing* novamente.

A Figura 9 mostra o fluxo detalhado do processo descrito anteriormente, resumindo a Metodologia de Desenvolvimento.

Figura 9 – Fluxograma de Desenvolvimento



FONTE: Autores

É importante ressaltar que a metodologia de desenvolvimento, descrita nessa seção, foi utilizada juntamente com a Metodologia Orientada a Provas de Conceito, item 4.3.

4.5 Metodologia de Análise de Resultados

A pesquisa de campo é caracterizada por investigações que vão além da pesquisa bibliográfica e/ou documental, envolvendo a coleta de dados junto a pessoas por meio de diferentes tipos de pesquisa, como, por exemplo, *ex-post-facto*, pesquisa-ação e pesquisa participante (GERHARDT; SILVEIRA, 2009). Ainda segundo (GERHARDT; SILVEIRA, 2009), com o propósito de analisar os resultados obtidos durante o desenvolvimento do trabalho, a metodologia adotada será a pesquisa-ação, fundamentada na resolução de questões ou problemas por meio da investigação. Nesse método, os pesquisadores participam de forma colaborativa ou participativa. (THIOLLENT, 1988) explica:

> "A pesquisa ação é um tipo de investigação social com base empírica que é concebida e realizada em estreita associação com uma ação ou com a resolução de um problema coletivo no qual os pesquisadores e os participantes representativos da situação ou do problema estão envolvidos de modo cooperativo ou participativo (p. 14)."

Enquanto (FONSECA, 2002) define por:

> "A pesquisa-ação pressupõe uma participação planejada do pesquisador na situação problemática a ser investigada. O

> processo de pesquisa recorre a uma metodologia sistemática, no sentido de transformar as realidades observadas, a partir da sua compreensão, conhecimento e compromisso para a ação dos elementos envolvidos na pesquisa (p. 34). O objeto da pesquisa-ação é uma situação social situada em conjunto e não um conjunto de variáveis isoladas que se poderiam analisar independentemente do resto. Os dados recolhidos no decurso do trabalho não têm valor significativo em si, interessando enquanto elementos de um processo de mudança social. O investigador abandona o papel de observador em proveito de uma atitude participativa e de uma relação sujeito a sujeito com os outros parceiros. O pesquisador quando participa na ação traz consigo uma série de conhecimentos que serão o substrato para a realização da sua análise reflexiva sobre a realidade e os elementos que a integram. A reflexão sobre a prática implica em modificações no conhecimento do pesquisador (p. 35)."

O objetivo principal desse método de pesquisa é examinar as características de diversos métodos disponíveis, avaliando suas capacidades, potencialidades, limitações ou distorções, além de identificar os pressupostos ou implicações de sua aplicação. Em um nível mais prático, envolve a avaliação de técnicas de pesquisa e a geração ou experimentação de novos métodos relacionados aos modos eficazes de coletar e processar informações. Além disso, busca solucionar diversas categorias de problemas teóricos e práticos na pesquisa, servindo como uma maneira de conceber e organizar o objeto de pesquisa, conforme alinnhado por (THIOLLENT, 1988).

A metodologia de pesquisa-ação possui particularidades para cada pesquisa, dependendo de sua demanda. No desenvolvimento desse trabalho, optou-se por direcionar a metodologia da seguinte maneira:

- Levantamento Investigativo: Nessa etapa, o foco principal foi a obtenção de informações relevantes e significativas visando justificar a elaboração da pesquisa. Dessa forma, foi realizada uma Pesquisa Bibliográfica, já descrita anteriormente em Metodologia Investigativa, no Tópico 4.2, cujos resultados estão documentados nos Capítulos 2 e 3 de Referencial Teórico e de Suporte Tecnológico, respectivamente. Adicionalmente, conduziu-se uma análise cuidadosa de documentos disponíveis para extrair informações pertinentes;

- Planejamento: Nesta etapa, delineou-se claramente os objetivos da pesquisa, indicando o que se esperava alcançar. Além disso, foram elaboradas as questões de

pesquisa que nortearam o presente trabalho, conforme discutido no Capítulo 1. A seleção de métodos e técnicas foi desenvolvida com o objetivo de orientar as atividades ao longo do tempo, sem excluir a possibilidade de adequações conforme a demanda;

- Ação: Com o plano de pesquisa definido, passou-se à implementação efetiva das ações planejadas. Isso incluiu a aplicação dos métodos e técnicas selecionados, bem como a coleta adicional de dados, quando necessário. Durante essa fase, foram registradas observações detalhadas para documentar o progresso e os *insights* obtidos (KRAFTA et al., 2007), e

- Avaliação: Nessa etapa, foi realizada a análise crítica dos resultados obtidos. Isso envolveu examinar os dados coletados, verificando se as hipóteses foram confirmadas ou refutadas, e refletindo criticamente sobre o processo de pesquisa. Com base nessa análise, foram elaboradas conclusões sólidas que resumem os resultados e, quando necessário, identificou-se áreas para melhorias futuras (FURTADO et al., 2008).

4.6 Fluxo de Atividades/Subprocessos

Durante a execução deste estudo, estabeleceram-se metas para otimizar a consecução do resultado desejado na primeira fase, que consistiu na entrega do TCC1. Visando assegurar a conclusão integral desta pesquisa, elaborou-se um fluxograma abrangente que contempla não apenas as atividades da primeira etapa (Figura 10), mas também aquelas planejadas para a segunda etapa do TCC. As Figuras 10 e 11 apresentam os fluxos delineados para a realização dessas etapas.

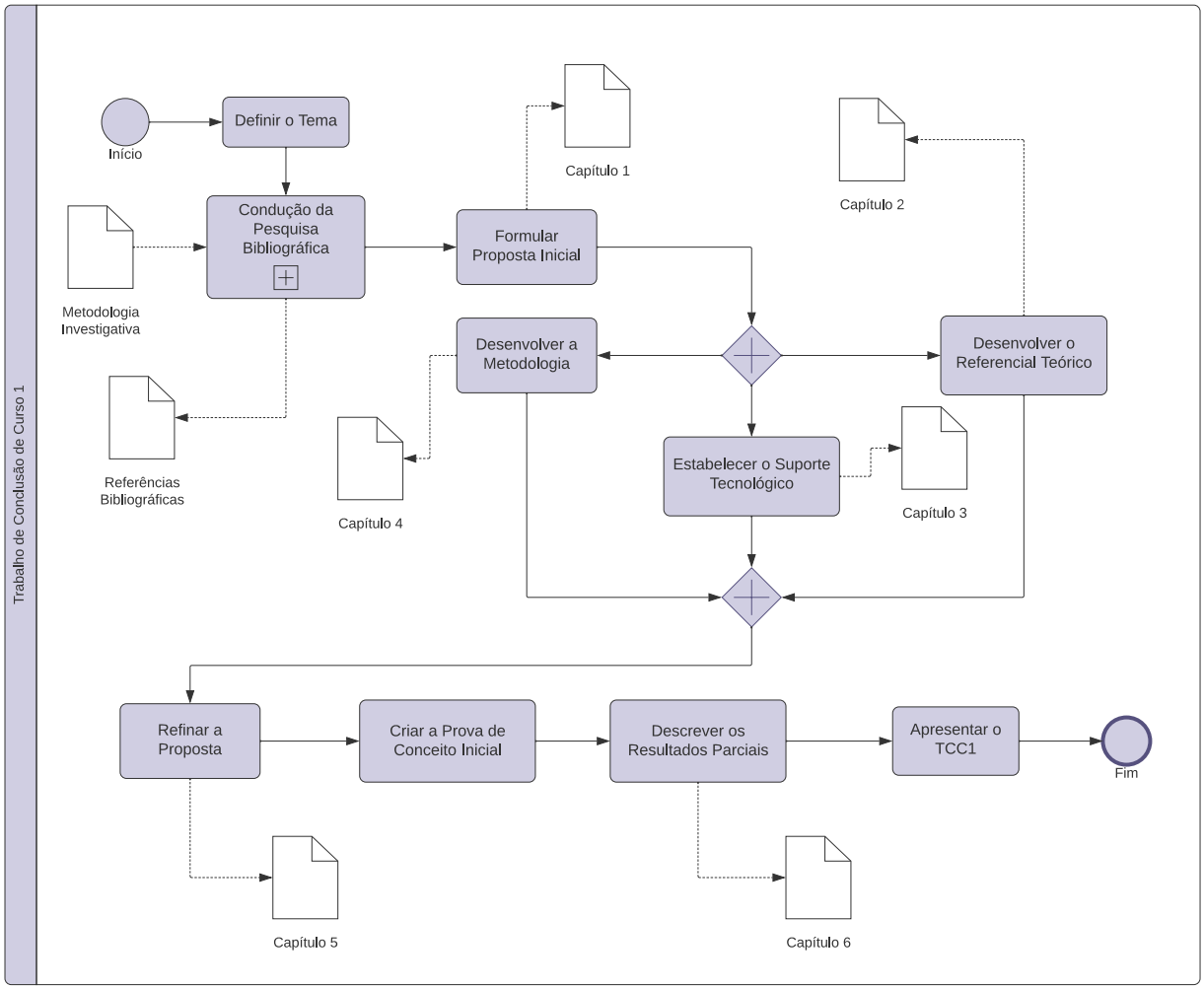
4.6.1 Primeira Etapa do TCC

1. **Definir o Tema:** Essa fase inicial visou discernir e delinear o tema central que guiará toda a pesquisa. A escolha do tema requereu, ao menos, uma compreensão mínima das lacunas no conhecimento existente e a capacidade de formular uma questão de pesquisa clara e significativa. *Status:* Concluída;

2. **Conduzir a Pesquisa Bibliográfica:** Essa fase envolveu a condução de uma pesquisa bibliográfica abrangente, buscando não apenas a coleta de informações, mas também a análise crítica e a síntese de conhecimentos prévios. Este processo exigiu habilidades de discernimento, julgamento acadêmico e a capacidade de situar a pesquisa no contexto mais amplo do campo de estudo. Cumpriu-se os critérios apresentados no item 1, conforme bem destrinchado nos subitens 1.3 e 1.4. *Status:* Concluída;

3. **Formular a Proposta Inicial:** Nessa etapa, reforçou-se que a formulação da proposta inicial não se restringia à articulação dos objetivos da pesquisa, sendo uma tarefa

Figura 10 – Fluxograma de Atividades/Subprocessos do TCC1



FONTE: Autores

de contextualização do trabalho em relação às contribuições já existentes. Demandou-se, aqui, a síntese de ideias e a comunicação eficaz, assegurando que a proposta fosse robusta, inovadora e alinhada aos propósitos do estudo, conforme apresentado também no item 1. *Status:* Concluída;

4. **Desenvolver o Referencial Teórico:** O desenvolvimento do referencial teórico demandou ir além de compilar teorias, requerendo a síntese e a integração de conceitos fundamentais para o tema escolhido, como por exemplo: Reengenharia, Aplicativos Móveis, *Test-Driven Development* (TDD), *Domain-Driven Design* (DDD), Técnicas de Programação, dentre outros. Este processo exigiu separação de literatura relevante para o tema, conjutamente com informações importantes associadas ao tema, conforme apresentado no item 2. *Status:* Concluída;

5. **Estabelecer o Suporte Tecnológico:** Nessa etapa, focou-se na escolha de ferramentas, de modo a viabilizar a pesquisa, alinhando-as estrategicamente aos objetivos do estudo. A descrição dessas ferramentas - para desenvolvimento prático do trabalho -

resultaram no item 3. *Status:* Concluída;

6. **Desenvolver a Metodologia:** Nessa fase, a pesquisa avançou para a elaboração de um plano prático, conforme descrito ao longo do presente capítulo (item 4). O objetivo foi definir os métodos e procedimentos que foram utilizados, assegurando uma metodologia eficiente, focando-se em uma obtenção confiável de resultados. *Status:* Concluída;

7. **Refinar a Proposta:** Nessa fase, concentrou-se os esforços no aprimoramento da proposta inicial com base em *feedbacks* recebidos. O intuito foi tornar o trabalho desenvolvido mais claro, relevante e inovador. *Status:* Concluída;

8. **Criar a Prova de Conceito Inicial:** Essa etapa envolveu a tradução da proposta em algo tangível, como um protótipo ou versão preliminar. O objetivo era validar a viabilidade da abordagem proposta bem como a configuração de ambiente de desenvolvimento, conforme item no Capítulo 5. *Status:* Concluída;

9. **Descrever os Resultados Parciais:** Nessa fase, apresentou-se e analisou-se as descobertas obtidas até o momento (TCC1). A análise crítica para identificar resultados, destacando sua relevância em relação aos objetivos da pesquisa, foi necessária. Essa descrição parcial atuou como um indicador do progresso, fornecendo *insights* valiosos para orientar as próximas etapas (TCC2). *Status:* Concluída, e

10. **Apresentar o TCC1:** Etapa em que foram compartilhados os fundamentos teóricos, metodológicos e os resultados obtidos até o momento de apresentação para a banca examinadora, na primeira etapa do TCC. *Status:* Concluída;

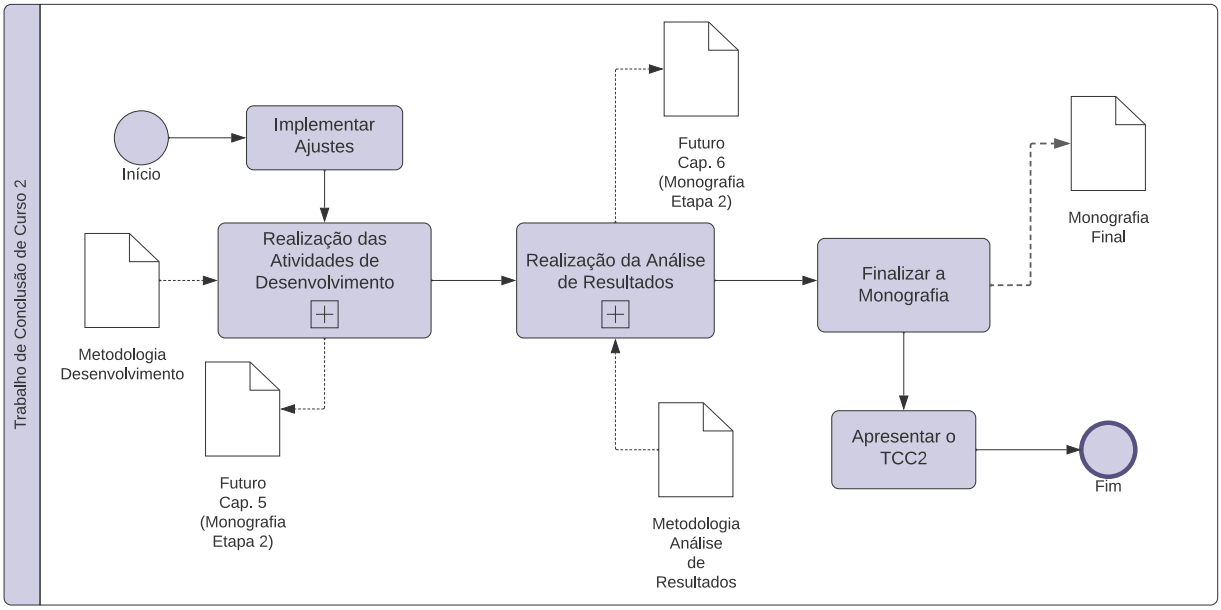
4.6.2 Segunda Etapa do TCC

1. **Implementar Ajustes:** Nessa etapa, procedeu-se à incorporação das modificações sugeridas pela banca avaliadora. Isso implicou na revisão e na adaptação do trabalho de acordo com as orientações recebidas, afim de se aprimorar a qualidade e a consistência do trabalho acadêmico. *Status:* Concluída;

2. **Realização das Atividades de Desenvolvimento:** Essa etapa baseou-se na execução das atividades delineadas no Capítulo 5 - Engenharia Reversa e Reengenharia do Mia Ajuda. A condução do desenvolvimento seguiu a Metodologia Orientada a Provas de Conceito (Tópico 4.3) e a Metodologia de Desenvolvimento (Tópico 4.4) como guias principais. Este processo permitiu a aplicação prática das diretrizes estabelecidas, assegurando a progressão coerente e eficaz do projeto. *Status:* Concluída;

3. **Realização da Análise de Resultados:** A condução dessa atividade seguiu os princípios e diretrizes estabelecidos pela Metodologia de Análise de Resultados, conforme tópico 4.5. O objetivo central foi aplicar uma abordagem sistemática para interpretar e

Figura 11 – Fluxograma de Atividades/Subprocessos do TCC2



FONTE: Autores

compreender os resultados obtidos, proporcionando *insights* fundamentais para a finalização da pesquisa. *Status:* Concluída;

4. **Finalizar a Monografia:** Essa etapa representou o fechamento da monografia, fornecendo uma visão geral do estado final do trabalho. Durante esse processo, destacaram-se os objetivos alcançados e o que pode ser explorado em futuras pesquisas referentes ao tema, indicando-se possíveis direções para investigações subsequentes. *Status:* Concluída, e

5. **Apresentar o TCC2:** Etapa em que serão compartilhados os fundamentos teóricos, metodológicos e os resultados obtidos para a banca examinadora. *Status:* Não Concluída.

4.7 Cronograma de Atividades/Subprocessos

Com base nos fluxos previamente propostos, elaboraram-se os cronogramas correspondentes para a conclusão das etapas inicial (Tabela 3) e subsequente (Tabela 4) deste trabalho.

4.8 Considerações Finais do Capítulo

Este capítulo teve o intuito de descrever a metodologia do trabalho, utilizada na sua confecção teórica e prática. No primeiro momento, a pesquisa foi classificada da seguinte forma: de abordagem híbrida (qualitativa e quantitativa), de natureza aplicada,

Tabela 3 – Cronograma de Atividades/Subprocessos da Primeira Etapa do TCC

Atividades/Subprocessos	**Ago**	**Set**	**Out**	**Nov**	**Dez**
Definir o Tema	X				
Conduzir a Pesquisa Bibliográfica	X				
Formular a Proposta Inicial		X			
Desenvolver o Referencial Teórico			X		
Estabelecer o Suporte Tecnológico			X		
Desenvolver a Metodologia				X	
Refinar a Proposta					X
Criar a Prova de Conceito Inicial					X
Descrever os Resultados Parciais					X
Apresentar o TCC1					X

FONTE: Autores

Tabela 4 – Cronograma de Atividades/Subprocessos da Segunda Etapa do TCC

Atividades/Subprocessos	**Mar**	**Abr**	**Mai**	**Jun**	**Jul**
Implementar Ajustes	X				
Realização das Atividades de Desenvolvimento	X	X	X	X	
Realização da Análise de Resultados				X	X
Finalizar a Monografia					X
Apresentar o TCC2					X

FONTE: Autores

com objetivos exploratórios, e com procedimentos de caráter bibliográficos e orientados a provas de conceito. Depois, o trabalho definiu seus elementos de pesquisa baseados na literatura científica. A partir disso, foram estabelecidas as seguintes metodologias específicas: Metodologia Investigativa, para o desenvolvimento da pesquisa bibliográfica; Metodologia Orientada a Provas de Conceito, para a criação dos desafios práticos do trabalho; Metodologia de Desenvolvimento, que consistiu numa abordagem híbrida, sendo fundamentada na combinação de elementos das metodologias ágeis (Scrum e Kanban); e a Metodologia de Análise de Resultados, para a utilização da metodologia pesquisa-ação que proporcionou insumo para a análise dos resultados obtidos. Por fim, o capítulo expôs fluxos de atividades e cronogramas para as duas etapas do TCC.

5 Engenharia Reversa e Reengenharia do Mia Ajuda

Esse capítulo aborda a proposta principal deste trabalho. Inicialmente, será apresentada uma breve Contextualização 5.1, seguida da descrição do escopo desenvolvido pelos autores 5.2, utilizando reengenharia e técnicas do TDD e do DDD, descritas no Referencial Teórico 2. Utilizou-se ainda a "Metodologia Orientada a Provas de Conceito"conforme item 4.3, demonstrado nos tópicos 5.3, 5.4, 5.5 e 5.6, onde a prova de conceito é descrita com sua respectiva definição do desafio e dos seus requisitos. Por último, encontram-se as Considerações Finais do Capítulo 5.7.

5.1 Contextualização

Conforme discutido anteriormente neste trabalho, o aplicativo MiaAjuda (2020) foi desenvolvido em 2020, durante o início da pandemia de Covid-19, com o objetivo de conectar usuários que necessitam de apoio, oferecendo uma plataforma para que pessoas possam solicitar e oferecer ajuda, seja ela de natureza material ou emocional. Devido à realidade desafiadora da pandemia, a demanda por soluções de cunho social aumentou drasticamente, levando o aplicativo a ter uma urgência no prazo de sua construção. Com isso, o processo de desenvolvimento da aplicação não atendeu os critérios definidos como boas práticas pela comunidade de *software*, deixando uma margem para possíveis evoluções.

O *Design* de *Software* é um processo fundamental para o desenvolvimento de um sistema, por ser o detalhamento de mais baixo nível com um conjunto de princípios, normas e técnicas. Sem ele, o *software* fica suscetível a diversos problemas futuros relacionados à manutenção e à evolução do mesmo (MARTIN, 2019).

A escolha das técnicas de programação utilizadas no desenvolvimento de um *software* é fundamental para garantir qualidade, manutenibilidade e eficiência do código fonte. Existem diversas técnicas de programação dentro da Engenharia de *Software*, com várias particularidades. Diante disso, é necessário uma boa escolha das técnicas, a fim de implementar uma solução eficaz, confiável e de desempenho adequado.

Ante o exposto, a técnica de programação do TDD, que visa uma melhor testabilidade, e o DDD, que define um *design* de *software* que reflete exatamente suas funcionalidades, são abordagens possíveis para a reengenharia de um aplicativo, visando a evolução do mesmo.

5.2 Escopo

Com o objetivo de realizar a reengenharia do aplicativo MiaAjuda (2020), aplicando os conceitos definidos no capítulo de Referencial Teórico 2 do trabalho, optou-se pelo desenvolvimento de uma parte das funcionalidades presentes na aplicação.

O escopo definido não compreende o aplicativo em sua totalidade por ter o objetivo de avaliar a testabilidade facilitada e a adequada modelagem de domínio, não sendo necessária a implementação total do aplicativo. No sistema desenvolvido neste trabalho, os usuários podem solicitar uma ajuda e também oferecer ajuda às solicitações existentes, sendo funcionalidades já implementadas no aplicativo.

Com o desenvolvimento de cada funcionalidade, é relevante avaliar: (i) se o domínio está adequadamente modelado; (ii) se é fácil testá-la, e se a mesma está com uma boa cobertura de testes, e (iii) se é fácil incorporar novas funcionalidades sem comprometer o *design* de *software* estabelecido. A partir dos resultados obtidos e avaliados, concluiu-se em resposta às Questões de Pesquisa e de Desenvolvimento 1.3 estabelecidas para este trabalho.

O sistema desenvolvido é composto por três módulos: Autenticação de Usuário; Solicitação de Ajuda; e Oferecimento de Ajuda. Cada módulo compreende um domínio da aplicação, e foi desenvolvido utilizando a abordagem combinada de TDD e DDD. A partir destes três módulos, criou-se quatro provas de conceito, sendo que cada uma compreendeu uma história de usuário vertical da metodologia *Story Slicing*. Estas provas de conceito serão abordadas a seguir.

5.3 POC 1 - Configuração do Ambiente e Rota de Verificação

Essa seção aborda a configuração do ambiente de desenvolvimento do sistema, além do desenvolvimento de uma rota de verificação. No primeiro momento, é apresentada a Definição do Desafio 5.3.1 atrelada a essa prova de conceito, seguido pelos Requisitos do Desafio 5.3.2, e por fim a Apresentação da Solução 5.3.3.

5.3.1 Definição do Desafio

Tem como objetivo estabelecer todo o ambiente de desenvolvimento, configurando as tecnologias e ferramentas estabelecidas, além de desenvolver, utilizando TDD e DDD, uma primeira rota para verificação da saúde do sistema.

5.3.2 Requisitos do Desafio

Para que a prova de conceito seja considerada finalizada, é necessário cumprir os seguintes requisitos:

- O sistema deve ser acessível através de um *host*;
- O sistema deve ser capaz de gerar insumos a respeito de sua qualidade, e
- O usuário deve ser capaz de verificar a saúde do sistema.

5.3.3 Apresentação da Solução

No primeiro momento dessa solução, tem-se a configuração do ambiente de desenvolvimento, sendo necessário iniciar o serviço *server-side* com o *framework* estabelecido para implementação. Depois, foi necessário inicializar as ferramentas e bibliotecas escolhidas para implementação e análise de código. Com o ambiente configurado, o serviço deve ser capaz de atender requisições HTTP feitas a ele por *clients*.

No segundo momento, tem-se a implementação de uma rota, na qual disponibilizam-se informações sobre a aplicação, sendo as principais: (i) se a mesma está disponível, e (ii) por quanto tempo ela se encontra disponível. Para esta parte da solução, foi necessário utilizar a linguagem de programação e a biblioteca de testes estabelecidas para o desenvolvimento, além de utilizar o TDD e o DDD.

5.3.3.1 Configuração do Ambiente

Para a configuração do ambiente de desenvolvimento, primeiro iniciou-se o projeto utilizando o *framework* Express [1] que tem o Node [2] como base, como foi estabelecido no capítulo de Suporte Tecnológico 3. Depois, configurou-se a ferramenta SonarQube [3], responsável por analisar a qualidade do código fonte e gerar insumos para a análise quantitativa das PoCs. Em seguida, foi feita a configuração do Docker [4] juntamente com o banco de dados MongoDB [5], para a conteinerização do serviço e armazenamento de dados. Com isso, ao iniciar o serviço, tem-se um *host* disponível para requisições HTTP.s

5.3.3.2 Rota de Verificação

Com o objetivo de validar o ambiente criado e criar uma base para a implementação com TDD e DDD, criou-se uma rota para verificação do sistema. Utilizando a técnica do

[1] https://expressjs.com/. Acessado pela última vez em 09/12/2023.
[2] https://nodejs.org. Acessado pela última vez em 09/12/2023.
[3] https://docs.sonarsource.com/sonarqube/latest. Acessado pela última vez em 09/12/2023.
[4] https://www.docker.com. Acessado pela última vez em 09/12/2023.
[5] https://www.mongodb.com. Acessado pela última vez em 26/06/2024.

TDD, primeiramente houve a implementação dos testes unitários para esta funcionalidade. A Figura 12 evidencia um dos testes criados para a funcionalidade.

Figura 12 – Código do Teste Unitário do Serviço da Rota de Verificação

```
const HealthService = require("../../src/modules/health/app/HealthService");

describe("HealthService", () => {
  test("getHealthCheck should return health data", async () => {
    const healthService = new HealthService();
    const healthcheck = await healthService.getHealthCheck();

    expect(healthcheck).toHaveProperty("uptime");
    expect(healthcheck).toHaveProperty("message", "OK");
    expect(healthcheck).toHaveProperty("timestamp");
  });
});
```

FONTE: Autores

Com isso, a primeira fase do TDD, que consiste na criação dos testes com falha, foi concluída. Em seguida implementou-se a rota de verificação, que busca obter sucesso com os testes criados anteriormente. A Figura 13 faz a implementação necessária para o sucesso do teste da Figura 12.

A terceira e última fase do TDD, que consiste na refatoração dos testes e depois da implementação, não se mostrou necessária visto que esta funcionalidade já atendeu completamente os requisitos propostos para esta POC. Para acessar esta rota, basta iniciar o serviço e fazer uma requisição para o *endpoint* criado.

5.3.3.3 Aspectos Positivos

Ao decorrer do desenvolvimento deste desafio, utilizando TDD e DDD, observou-se os seguintes aspectos positivos desta abordagem de desenvolvimento:

- **Organização por domínio**: ao utilizar o DDD, cada módulo é inserido dentro de um domínio do problema, o que ajuda o desenvolvedor a saber onde uma nova *feature* deve ser inserida no projeto. Além disso, a manutenção é facilitada, visto que as correções e melhorias de alguma *feature* do produto já implementada deverão ser feitas no módulo correspondente ao seu domínio, o que facilita para o desenvolvedor encontrar os arquivos a serem alterados, e

Figura 13 – Código da Implementação do Serviço da Rota de Verificação

```
const { format, addSeconds } = require("date-fns");
const HealthCheck = require("../domain/HealthCheck");

class HealthService {
  async getHealthCheck() {
    const now = new Date(0);

    const uptime = format(addSeconds(now, process.uptime()), "HH:mm:ss");
    const message = "OK";
    const timestamp = format(Date.now(), "yyyy-MM-dd HH:mm:ss");

    const healthCheck = new HealthCheck(uptime, message, timestamp);

    return healthCheck;
  }
}

module.exports = HealthService;
```

FONTE: Autores

- **Cobertura de testes**: ao utilizar o TDD, os testes dos módulos são os primeiros a serem implementados, trazendo assim uma grande cobertura de testes antes mesmo da implementação de fato da funcionalidade. Desta forma, o projeto sempre estará mantendo seu padrão de qualidade no quesito dos testes, e colhendo os benefícios desta prática, podendo identificar problemas com mais rapidez e também garantindo que funcionalidades novas não irão prejudicar funcionalidades já implementadas no projeto.

5.3.3.4 Aspectos Negativos

Adicionalmente, foram identificados aspectos negativos desta abordagem de desenvolvimento, sendo eles:

- **Complexidade de implementação**: ao utilizar o TDD e o DDD e implementar uma nova funcionalidade, o desenvolvedor deve se preocupar com a criação de testes e em implementar no domínio correto do produto. Diante disso, o processo de desenvolvimento possui uma maior complexidade, envolvendo mais passos além da própria implementação, trazendo um maior tempo de desenvolvimento, o que pode ser um empecilho para implementações de urgência, e

- **Curva de aprendizado**: por envolver um desenvolvimento orientado ao domínio do produto, existe uma curva de aprendizado para o desenvolvedor em relação à própria técnica do DDD, e também em relação ao domínio de fato. O desenvolvedor deve conhecer o domínio da funcionalidade a ser implementada, para seguir corretamente as diretrizes do DDD, o que pode dificultar a implementação quando o mesmo não possui familiaridade com o domínio do produto.

5.3.3.5 Conclusão

Visto os aspectos positivos e negativos observados, conclui-se que a abordagem de desenvolvimento utilizando a combinação de TDD e DDD pode ser aplicada em contextos nos quais a alta cobertura de testes trará benefícios concretos para a aplicação, como no caso de funcionalidades críticas, que não podem ser modificadas com implementações novas. Além disso, em contextos nos quais o domínio do produto possui certa complexidade, ou então em projetos de longa vida esta abordagem, torna-se interessante por trazer uma grande flexibilidade ao sistema e um maior entendimento de seu domínio. No entanto, esta abordagem torna-se problemática em contextos que exigem urgência nas implementações, e em contextos em que a equipe de desenvolvimento não possui familiaridade com estas técnicas e com o domínio do produto.

5.4 POC 2 - Desenvolvimento do Cadastro e Login de Usuários

Essa seção aborda o desenvolvimento da *feature* relacionada ao cadastro de usuários dentro da plataforma do Mia Ajuda. Inicialmente, é apresentada a Definição do Desafio 5.4.1 atrelada a essa prova de conceito, seguido pelos Requisitos do Desafio 5.4.1.1 e Apresentação da Solução 5.4.2.

5.4.1 Definição do Desafio

Tem como objetivo desenvolver, utilizando o TDD e DDD, o cadastro e *login* de usuários na aplicação Mia Ajuda.

5.4.1.1 Requisitos do Desafio

Para que a prova de conceito seja considerada finalizada, é necessário cumprir os seguintes requisitos:

- O usuário deve ser capaz de efetuar seu cadastro;
- O usuário deve ser capaz de editar suas informações, e

- O usuário deve ser capaz de realizar login utilizando *email* e senha cadastrados previamente.

5.4.2 Apresentação da Solução

Para o desenvolvimento desta prova de conceito, primeiramente, realizou-se a engenharia reversa do sistema de cadastro e *login* de usuários já implementado no projeto Mia Ajuda. Com isso, identificou-se a necessidade da implementação de um novo módulo que engloba o domínio de usuário, que realizará todas as operações de leitura, criação, atualização e remoção necessárias para a implementação dos requisitos definidos.

Após a realização da engenharia reversa, pôde-se realizar a reengenharia do sistema analisado, seguindo uma implementação orientada a TDD e DDD. Primeiro foram criados testes unitários que esperam a implementação de um sistema capaz de cadastrar novos usuários, listar usuários já cadastrados, atualizar usuários e realizar a autenticação de um usuário válido, e depois, a implementação de fato das funcionalidades, satisfazendo assim os testes já criados.

Assim como se espera de uma história de usuário seguindo a metodologia do *Story Slicing* vertical, foi necessário interagir com múltiplas camadas da aplicação para o cumprimento desta prova de conceito. Além de operações no banco de dados, também viu-se necessário interagir com a camada lógica da aplicação, para a implementação das regras de negócio, e também a disponibilização de uma interface para interação de *clients*.

5.4.2.1 Engenharia Reversa

Analisando a implementação de cadastro e *login* de usuários presentes no sistema do Mia Ajuda, definiu-se a necessidade da criação de dois tipos de usuários, sendo eles um usuário capaz de solicitar ajuda na plataforma, e outro capaz de oferecer ajuda aos pedidos existentes. Com isso, pensando no DDD, viu-se necessário a criação de um domínio de usuário, responsável por todas as operações relacionadas aos usuários da aplicação.

O sistema do Mia Ajuda também consta a possibilidade de cadastro de um usuário como pessoa física, fornecendo um CPF, ou então como pessoa jurídica, fornecendo um CNPJ juntamente com algumas informações adicionais. Esta funcionalidade também deve estar presente na reengenharia do sistema.

5.4.2.2 Reengenharia

Para preservar as funcionalidades analisadas na engenharia reversa, mas também sem adicionar complexidades desnecessárias à modelagem do banco de dados, decidiu-se por criar apenas um modelo de usuário, utilizando o e-mail como identificador primário, mas podendo aceitar tanto informações de pessoa física como jurídica. Desta forma, o

banco de dados do sistema possui apenas uma coleção de usuários, facilitando assim as operações de consulta, criação, atualização e remoção de usuários da plataforma.

Outra decisão realizada na etapa de reengenharia foi a de utilizar o mesmo banco de dados para armazenar os dados de usuários e os outros dados do sistema. O sistema original do Mia Ajuda conta com um banco de dados específico para os usuários, e outro para os demais dados da aplicação. A unificação destes dois armazenamentos foi feita pensando num maior controle sobre os dados e na diminuição de custos de hospedagens.

Testes Unitários

Seguindo a técnica de programação do TDD, a primeira etapa da implementação dessa prova de conceito foi a criação de testes unitários. Para isso, primeiramente criou-se o módulo que compreende o domínio de usuário, com a definição do modelo de dados e das funcionalidades necessárias, mas sem a implementação delas, como pode ser visto na Figura 14. O *commit* relacionado com esta definição de funções é:

- *Commit*[6]: Criação da base de arquivos para o domínio de usuários.

Depois, foram criados testes unitários que visam testar todas as funcionalidades definidas para o domínio de usuários. Para esta etapa, o objetivo foi manter a maior cobertura de testes possível, sendo isso um critério importante para a análise quantitativa da solução. No total, foram criados 8 novos arquivos de testes, compreendendo todas as funcionalidades do novo módulo de usuários e também de algumas novas funções criadas no módulo geral. A cobertura total alcançada com os testes falhando foi de aproximadamente 98%, como pode ser observado na Figura 15. O *commit* relacionado com a criação dos testes dessa solução é:

- *Commit*[7]: Criação dos testes unitários para o domínio de usuário.

Implementação

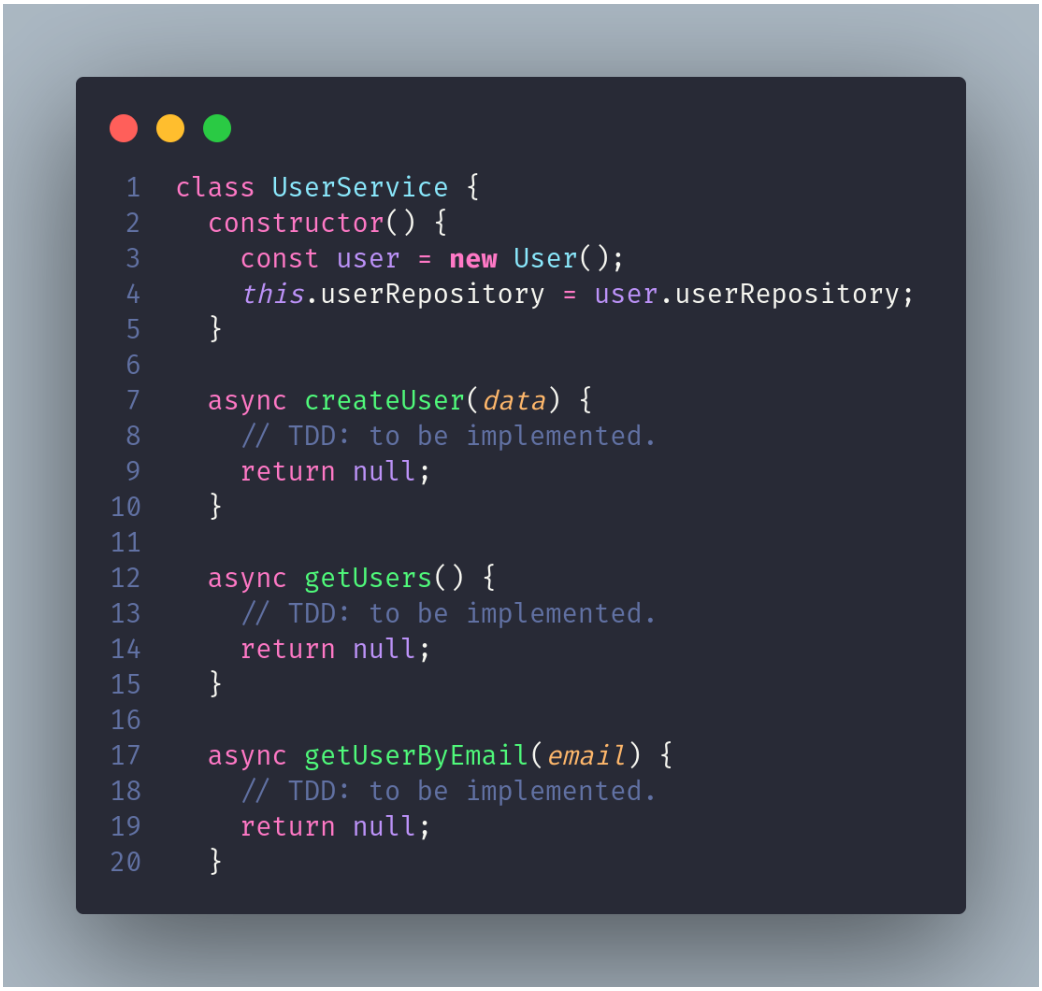
Após a criação do módulo de usuários e a criação dos testes unitários, iniciou-se a implementação das funcionalidades definidas, de modo com que cumpra todos os casos de teste criados. Para a implementação do cadastro de usuários, utilizou-se algumas bibliotecas de validação de dados e de criptografia para a segurança de senhas. Com isso, o sistema é capaz de cadastrar, listar e atualizar usuários por meio de *endpoints* definidos.

Para a implementação do *login* de usuários cadastrados, utilizou-se de uma biblioteca para criação e validação de um *token* de autenticação, que pode ser utilizado por

[6] Disponível em: <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend/commit/b9e6c58285f031615d5b1e8a15c899ce96a4e1cd>. Acessado pela última vez em 26/06/2024.

[7] Disponível em: <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend/commit/cc4decd8d8cfad99c41f87afe3def97daf158206>. Acessado pela última vez em 26/06/2024.

Figura 14 – Exemplo de Arquivo Base para Domínio de Usuários



```
class UserService {
  constructor() {
    const user = new User();
    this.userRepository = user.userRepository;
  }

  async createUser(data) {
    // TDD: to be implemented.
    return null;
  }

  async getUsers() {
    // TDD: to be implemented.
    return null;
  }

  async getUserByEmail(email) {
    // TDD: to be implemented.
    return null;
  }
```

FONTE: Autores

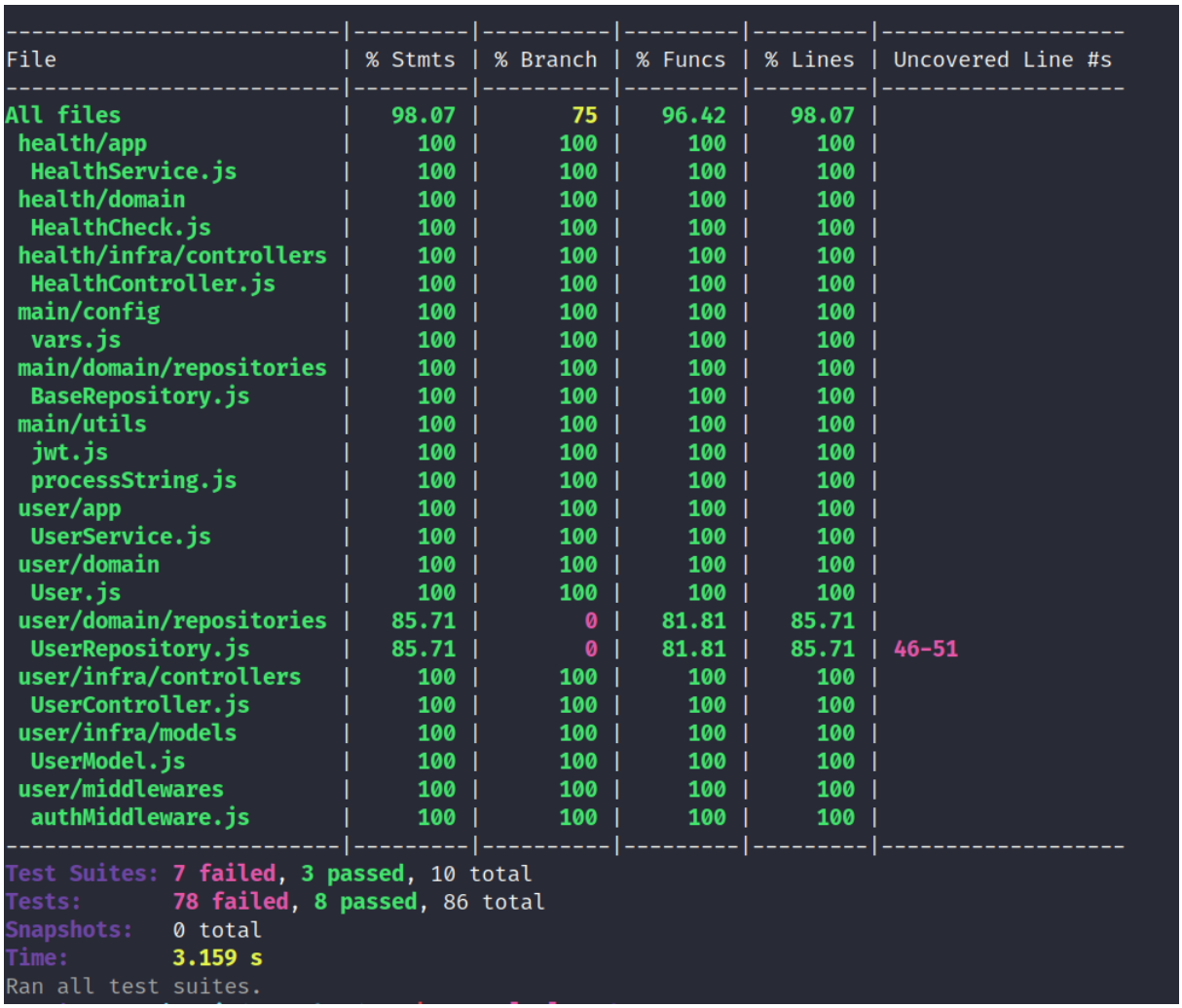
um *client* para acessar as rotas privadas do sistema. Com isso, todas as rotas sensíveis agora estão protegidas, e só podem ser acessadas por meio de um *token* válido criado pela própria API.

Assim como planejado, todos os dados de usuário são salvos no banco de dados MongoDB [8], configurado na prova de conceito anterior, e os *endpoints* definidos foram criados utilizando o *framework* Express [9]. Desta forma, é possível acessar as funcionalidades desta prova de conceito pelo mesmo *host* das outras funcionalidades implementadas por meio de requisições HTTP.s.

Após a implementação completa, todos os casos de testes definidos para esta prova de conceito foram atendidos, e a cobertura total passou a ser de 100%, como pode ser visto na Figura 16. O *commit* relacionado com esta etapa da implementação é:

[8] https://www.mongodb.com. Accesado pela última vez em 26/06/2024
[9] https://expressjs.com/. Acessado pela última vez em 26/06/2024.

Figura 15 – Cobertura de Testes com Falha do Domínio de Usuário



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	98.07	75	96.42	98.07	
health/app	100	100	100	100	
HealthService.js	100	100	100	100	
health/domain	100	100	100	100	
HealthCheck.js	100	100	100	100	
health/infra/controllers	100	100	100	100	
HealthController.js	100	100	100	100	
main/config	100	100	100	100	
vars.js	100	100	100	100	
main/domain/repositories	100	100	100	100	
BaseRepository.js	100	100	100	100	
main/utils	100	100	100	100	
jwt.js	100	100	100	100	
processString.js	100	100	100	100	
user/app	100	100	100	100	
UserService.js	100	100	100	100	
user/domain	100	100	100	100	
User.js	100	100	100	100	
user/domain/repositories	85.71	0	81.81	85.71	
UserRepository.js	85.71	0	81.81	85.71	46-51
user/infra/controllers	100	100	100	100	
UserController.js	100	100	100	100	
user/infra/models	100	100	100	100	
UserModel.js	100	100	100	100	
user/middlewares	100	100	100	100	
authMiddleware.js	100	100	100	100	

Test Suites: 7 failed, 3 passed, 10 total
Tests: 78 failed, 8 passed, 86 total
Snapshots: 0 total
Time: 3.159 s
Ran all test suites.

FONTE: Autores

- *Commit*[10]: Implementação do módulo de usuário.

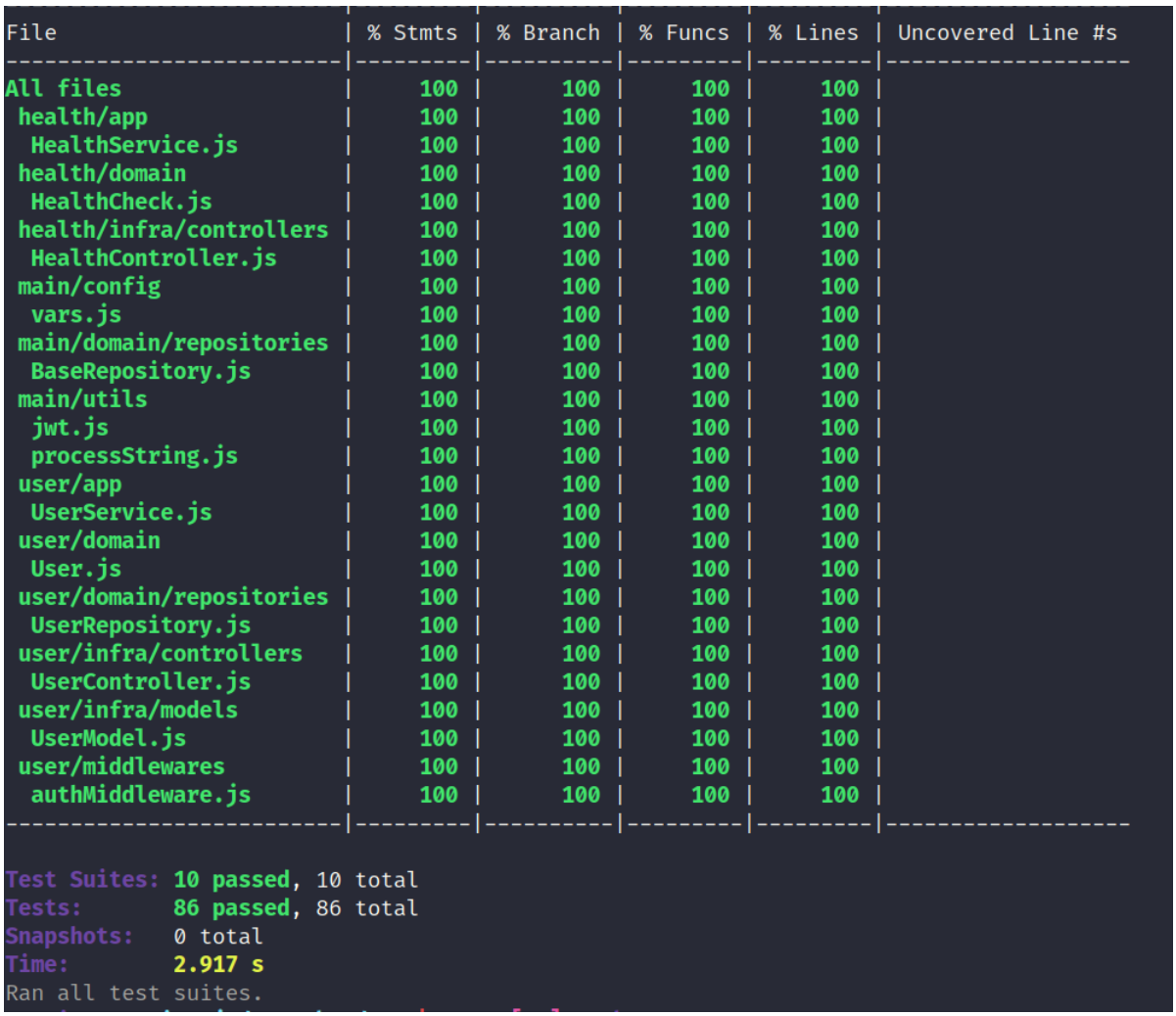
Refatoração

Na Figura 17, é possível ver que após a implementação desta prova de conceito, embora a cobertura de testes esteja em 100%, a ferramenta SonarQube [11] acusou alguns problemas no código, atribuindo assim nota C para esta funcionalidade, fazendo com que o portão de qualidade definido não seja atendido. Com isso, viu-se necessário realizar uma refatoração da implementação feita.

A refatoração consistiu na melhoria da confiabilidade do código, assim como apontado pela ferramenta de análise. Após a solução deste problema, a nota atribuída passou a

[10] Disponível em: <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend/commit/085d7e640e810e490f4ddc65a07a428a49bf79fb>. Acessado pela última vez em 26/06/2024.
[11] https://docs.sonarsource.com/sonarqube/latest. Acessado pela última vez em 26/06/2024.

Figura 16 – Cobertura de Testes com Sucesso do Domínio de Usuário



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
health/app	100	100	100	100	
HealthService.js	100	100	100	100	
health/domain	100	100	100	100	
HealthCheck.js	100	100	100	100	
health/infra/controllers	100	100	100	100	
HealthController.js	100	100	100	100	
main/config	100	100	100	100	
vars.js	100	100	100	100	
main/domain/repositories	100	100	100	100	
BaseRepository.js	100	100	100	100	
main/utils	100	100	100	100	
jwt.js	100	100	100	100	
processString.js	100	100	100	100	
user/app	100	100	100	100	
UserService.js	100	100	100	100	
user/domain	100	100	100	100	
User.js	100	100	100	100	
user/domain/repositories	100	100	100	100	
UserRepository.js	100	100	100	100	
user/infra/controllers	100	100	100	100	
UserController.js	100	100	100	100	
user/infra/models	100	100	100	100	
UserModel.js	100	100	100	100	
user/middlewares	100	100	100	100	
authMiddleware.js	100	100	100	100	

```
Test Suites: 10 passed, 10 total
Tests:       86 passed, 86 total
Snapshots:   0 total
Time:        2.917 s
Ran all test suites.
```

FONTE: Autores

ser A, e o portão de qualidade foi cumprido, como pode ser visto na Figura 18. O *commit* relacionado à esta etapa de refatoração pode ser visto em:

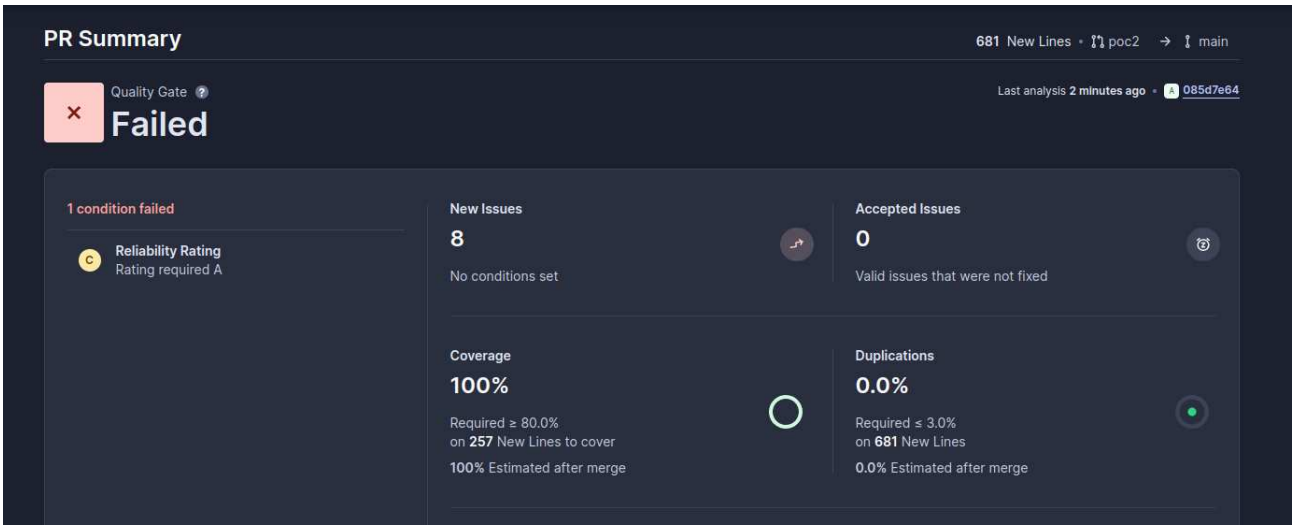
- *Commit*[12]: Refatoração do módulo de usuários.

5.5 POC 3 - Desenvolvimento da Funcionalidade de Pedido de Ajuda

Essa seção aborda o desenvolvimento da *feature* relacionada ao cadastro de pedidos de ajuda. Primeiramente, é apresentada a Definição do Desafio 5.5.1 atrelada a essa prova de conceito, seguido pelos Requisitos do Desafio 5.5.1.1 e Apresentação da Solução 5.5.2.

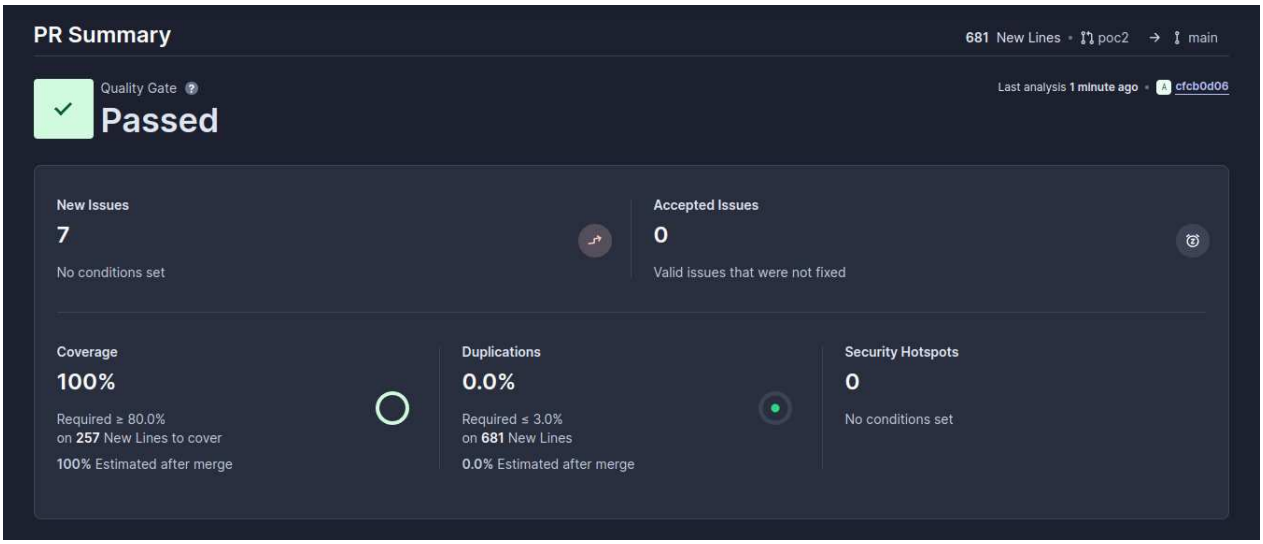
[12] Disponível em: <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend/commit/cfcb0d0657b275f5b12b6c12f0340a4e401a3ed5>. Acessado pela última vez em 26/06/2024.

Figura 17 – Análise 1 do SonarQube sobre a POC 2



FONTE: Autores

Figura 18 – Análise 2 do SonarQube sobre a POC 2



FONTE: Autores

5.5.1 Definição do Desafio

Tem como objetivo desenvolver, utilizando o TDD, o cadastro de pedidos de ajuda por parte de usuários que desejam ser ajudados.

5.5.1.1 Requisitos do Desafio

Para que a prova de conceito seja considerada finalizada, é necessário cumprir os seguintes requisitos:

- O usuário deve ser capaz de visualizar as categorias existentes;

- O usuário deve ser capaz de cadastrar um pedido de ajuda;
- O usuário deve ser capaz de visualizar seus respectivos pedidos de ajuda, e
- O usuário deve ser capaz de desativar um pedido de ajuda que pertence a ele.

5.5.2 Apresentação da Solução

A resolução deste desafio consistiu primeiramente na engenharia reversa da implementação do sistema do Mia Ajuda, para identificar quais funcionalidades já existentes deveriam estar presentes nesta prova de conceito. Após esta análise, identificou-se a necessidade da criação do domínio de ajuda, responsável pelas operações envolvendo os pedidos de ajuda que podem ser criados por qualquer usuário da plataforma, e também o domínio de categoria, responsável pelas categorias que podem ser atribuídas aos pedidos de ajuda.

Com a engenharia reversa feita, a segunda etapa consistiu na realização da reengenharia das funcionalidades identificadas como necessárias para o cumprimento desta POC. Seguindo as metodologias do TDD e DDD, primeiro realizou-se a configuração dos novos domínios a serem implementados, para depois, serem criados os testes unitários que buscam testar todos os requisitos propostos para este desafio.

Por fim, houve a implementação dos métodos e classes necessárias para satisfazer os casos de testes criados previamente, terminando assim, a reengenharia da parte do sistema proposta para esta prova de conceito. Com isso, houve a implementação do sistema de solicitação de ajuda, com *endpoints* disponíveis para uso de *clients*.

5.5.2.1 Engenharia Reversa

Na etapa da engenharia reversa do sistema já implementado do Mia Ajuda, foi feita uma análise da funcionalidade de solicitação de ajuda pela plataforma. Com isso, identificou-se a necessidade de um domínio de ajuda e um domínio de categoria, contendo as funcionalidades de criação, leitura, atualização e remoção de pedidos de ajudas que podem ser feitas por usuários da aplicação, e também das categorias que podem ser atribuídas a estes pedidos.

Além dos *endpoints* necessários para cumprimento desta POC, também foi identificado a necessidade da criação de um arquivo que popula automaticamente o banco de dados com as informações sobre as categorias. Com isso, o *client* não precisa se preocupar com a criação das categorias, apenas com a leitura e utilização delas para a criação dos pedidos de ajuda.

5.5.2.2 Reengenharia

Para a reengenharia da funcionalidade de solicitação de ajuda do aplicativo Mia Ajuda, utilizou-se do TDD e DDD, realizando assim a implementação dos novos domínios necessários. Primeiramente foram feitos os testes unitários para as classes e métodos que foram posteriormente implementadas.

Testes Unitários

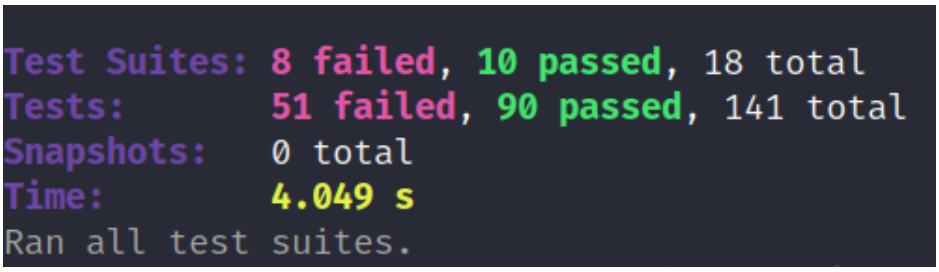
Assim como foi falado anteriormente, a solução para este desafio consistiu na criação de dois novos domínios, necessitando assim da configuração inicial deles. Da mesma forma como foi feita na resolução da POC 2 5.4, a configuração inicial consistiu na criação das classes e funções necessárias para a implementação deste desafio, porém apenas com a declaração das mesmas, sem a implementação de fato. Esta configuração serviu como base para a criação dos testes unitários para os módulos novos. O *commit* relacionado à esta configuração é:

- *Commit*[13]: Criação da base de arquivos para os domínios de categoria e ajuda.

Após isso, houve a criação de 8 novos arquivos de teste, sendo 2 deles para o domínio de categoria, outros 2 para novas funções úteis adicionadas ao módulo geral, e os outros 4 arquivos sendo para testar o domínio de ajuda. A Figura 19 mostra a falha destes casos de teste, sendo esse um cenário esperado pela técnica do TDD, no qual os testes falham no primeiro momento, para serem satisfeitos com a posterior implementação. O *commit* responsável pela criação destes testes foi o de:

- *Commit*[14]: Criação de testes unitários para os domínios de categoria e ajuda.

Figura 19 – Testes com Falha do Domínio de Ajuda



FONTE: Autores

Implementação

[13] Disponível em: <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend/commit/5f10528c446d3b3c7870ac49f8ceb1e70e1da4ea>. Acessado pela última vez em 30/06/2024.

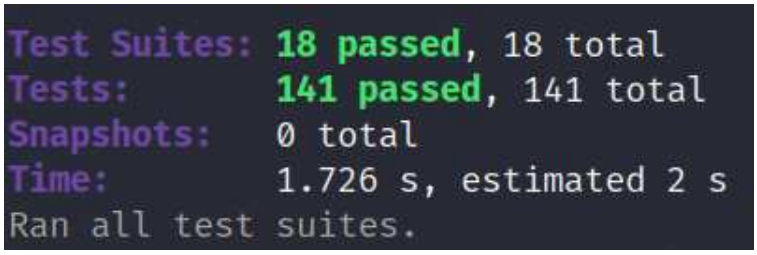
[14] Disponível em: <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend/commit/4c0bc5b8f5032f39afcdc6e40a4fe0035fb0fafc>. Acessado pela última vez em 30/06/2024.

Para o cumprimento dos testes criados na etapa anterior, necessitou-se da implementação das classes e métodos definidos para esta POC. No primeiro momento, houve a implementação do domínio de categoria, que é responsável por disponibilizar as categorias que devem ser adicionadas aos pedidos de ajuda. Para isso, primeiro criou-se um arquivo que popula o banco de dados com as informações predefinidas das categorias. Depois, houve a implementação dos *endpoints* de leitura destas categorias, para disponibilizar uma forma dos *clients* terem acesso às informações necessárias.

Depois, implementou-se o domínio de ajuda, responsável pelas operações que envolvem os pedidos de ajuda. Para isso, os métodos e classes definidas para este domínio foram desenvolvidos de forma a satisfazer seus casos de teste relacionados. Com isso, os *clients* podem acessar os novos *endpoints* da API de forma a adicionar, ler, editar ou excluir pedidos de ajuda da plataforma. Desta forma, todos os casos de teste definidos para este desafio passaram a ter sucesso, como pode ser visto na Figura 20. O *commit* responsável pela implementação desta POC pode ser visto em:

- *Commit*[15]: Implementação dos domínios de categoria e ajuda.

Figura 20 – Testes com Sucesso do Domínio de Ajuda



FONTE: Autores

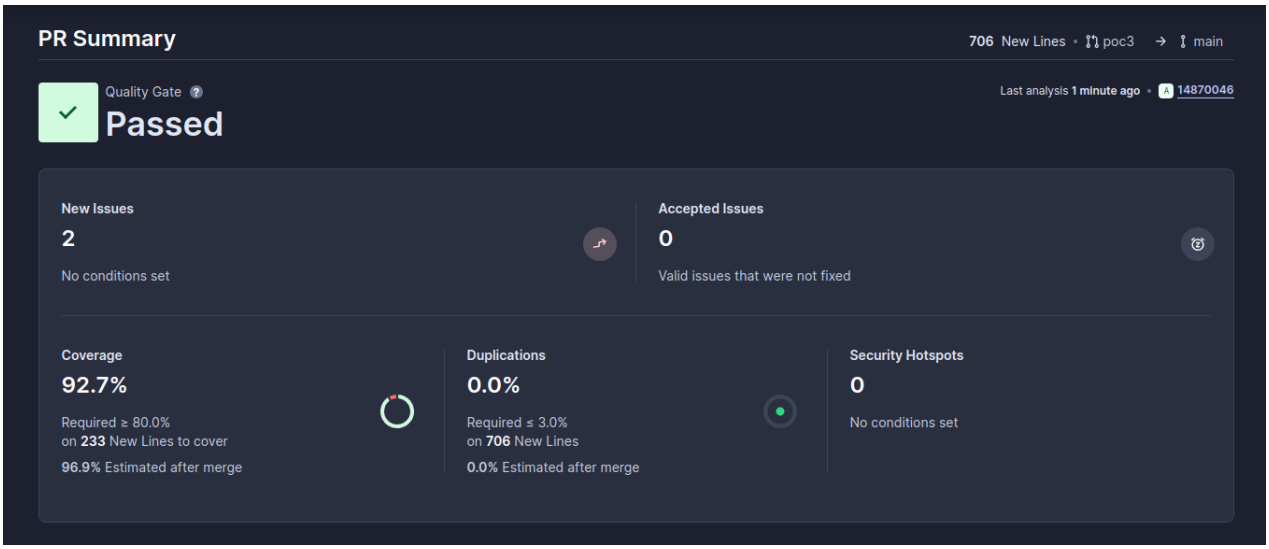
Após a finalização da implementação, não houve a necessidade de uma refatoração do código desenvolvido, visto que a plataforma SonarQube não apontou nenhuma falha, insegurança ou outro aspecto negativo do código que devesse ser refatorado, como pode ser visto na Figura 21. Desta forma, o portão de qualidade definido para o projeto foi satisfeito, concluindo assim a resolução desta POC.

5.6 POC 4 - Desenvolvimento da Funcionalidade de Oferta de Ajuda

Essa seção aborda o desenvolvimento da *feature* relacionada ao cadastro de ofertas de ajuda. Inicialmente, é apresentada a Definição do Desafio 5.6.1 atrelada a essa prova

[15] Disponível em: <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend/commit/14870046af28ac3198eba73f76874829a35e91d1>. Acessado pela última vez em 30/06/2024.

Figura 21 – Análise do SonarQube sobre a POC 3



FONTE: Autores

de conceito, seguido pelos Requisitos do Desafio 5.6.1.1 e Apresentação da Solução 5.6.2.

5.6.1 Definição do Desafio

Tem como objetivo desenvolver, utilizando o TDD, o cadastro de oferta de ajuda por parte de usuários que desejam ajudar outros usuários.

5.6.1.1 Requisitos do Desafio

Para que a prova de conceito seja considerada finalizada, é necessário cumprir os seguintes requisitos:

- O usuário deve ser capaz de visualizar todos os pedidos de ajuda;
- O usuário deve ser capaz de filtrar os pedidos de ajuda por categoria,
- O usuário deve ser capaz de oferecer ajuda para um pedido específico,
- O usuário deve ser capaz de aceitar uma ajuda oferecida, e
- O usuário deve ser capaz de finalizar uma oferta de ajuda.

5.6.2 Apresentação da Solução

Para solucionar esta POC, primeiramente foi feita a Engenharia Reversa do sistema do Mia Ajuda. Com isso, foi possível identificar a necessidade da implementação de um novo domínio para a funcionalidade de ofertas de ajuda, além também da atualização

do domínio já existente da funcionalidade de pedidos de ajuda, para ser possível aceitar ajudas oferecidas.

No segundo momento, foi feita a reengenharia das funcionalidades identificadas. Para isso, houve o desenvolvimento dos métodos já implementados no Mia Ajuda, porém com a abordagem de TDD e DDD. Primeiramente foram criados os testes unitários para os métodos esperados, criando casos de teste que testem todas as camadas do código de forma que satisfaçam os requisitos propostos que foram criados utilizando a metodologia do *storyslicing* vertical.

Por fim, houve a implementação de fato das funcionalidades esperadas pelos casos de teste criados. Desta forma, a reengenharia foi concluída de modo a disponibilizar *endpoints* na API para que usuários possam oferecer ajuda a pedidos de ajuda existentes na plataforma, além de poder aceitar ofertas aos seus pedidos abertos.

5.6.2.1 Engenharia Reversa

A engenharia reversa desta POC consistiu na análise da funcionalidade de oferta de ajuda do aplicativo Mia Ajuda. Com isso, identificou-se a necessidade da criação de um novo domínio para as ofertas de ajuda, para possibilitar a criação, leitura, atualização e remoção de pedidos de ajuda.

Além disso, o sistema do Mia Ajuda também conta com a possibilidade de adicionar possíveis ajudantes aos pedidos de ajuda existentes, além de poder aceitar alguma dessas ofertas. Para esta implementação, também se viu necessário realizar novas implementações no domínio já existente de pedidos de ajuda.

5.6.2.2 Reengenharia

A etapa de reengenharia na solução desta POC consistiu na implementação das funcionalidades identificadas na etapa anterior, utilizando a abordagem do TDD aliado com o DDD. Com isso, foram criados os testes unitários para as funções definidas como necessárias, para depois, ser feita o desenvolvimento das mesmas.

Testes Unitários

Os testes unitários desta POC foram feitos para testar completamente o novo domínio de ofertas de ajuda. Além disso, também foram feitos novos testes para o domínio de pedidos de ajuda, para validar a implementação completa dos requisitos desta prova de conceito.

No primeiro momento, foi feita a configuração inicial do novo domínio, criando classes e arquivos novos apenas com as declarações dos métodos, com uma implementação vazia. Também foram feitos novos métodos vazios para o domínio de pedidos de ajuda,

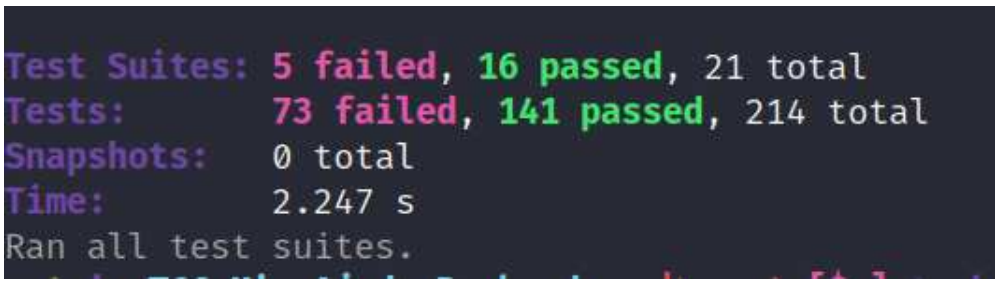
que estão relacionados com a funcionalidade desta POC. O *commit* relacionado com esta etapa de configuração é:

- *Commit*[16]: Criação da base de arquivos para o domínio de oferta.

Com as funções definidas, foi possível criar novos casos de teste para realizar a testagem completa desta POC. Foram criados novos 3 arquivos de teste, para testar o novo domínio criado, e 2 arquivos de teste já existentes, do domínio de pedidos de ajuda foram modificados para adição de mais casos de teste. A Figura 22 mostra estes testes sendo executados com falha dentro da aplicação. O *commit* responsável pela criação destes casos de teste foi:

- *Commit*[17]: Criação de testes unitários para o domínio de oferta.

Figura 22 – Testes Com Falha do Domínio de Oferta



FONTE: Autores

Implementação

A etapa de implementação desta POC aconteceu após a conclusão da criação dos testes unitários. O objetivo desta etapa foi o de satisfazer todos os testes que estavam com falha, seguindo assim a técnica do TDD.

Primeiramente foi desenvolvido o novo domínio de oferta de ajuda, com todas as funcionalidades necessárias para viabilizar a criação, a leitura, a atualização e a remoção de pedidos de ajuda. Com isso, a API passou a aceitar novas requisições para estas finalidades.

Depois, implementou-se as funcionalidades de adicionar possíveis ajudados aos pedidos existentes, e aceitar um ou vários ajudados para este mesmo pedido. Para isso, foi necessário desenvolver mais métodos para o novo domínio de ofertas de ajuda.

Por fim, o mesmo foi feito no domínio de pedidos de ajuda, ou seja, implementou-se a possibilidades de adicionar possíveis ajudantes aos pedidos ajudados, e poder aceitar

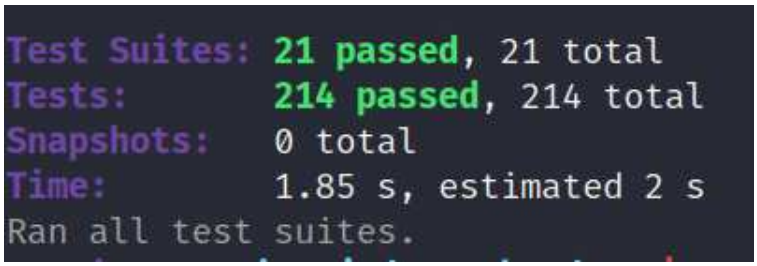
[16] Disponível em: <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend/commit/09a3a173a924efbd5f58b0d216497082727d8776>. Acessado pela última vez em 30/06/2024.

[17] Disponível em: <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend/commit/89442f3aff30c585bd78fba1b046b6e86a1675a3>. Acessado pela última vez em 30/06/2024.

um ajudante desta lista de possíveis ajudantes. Assim, todos os casos de teste criados para esta POC foram satisfeitos, assim como é possível observar na Figura 23. O *commit* relacionado com a implementação das funcionalidades desta prova de conceito é:

- *Commit*[18]: Implementação do domínio de oferta.

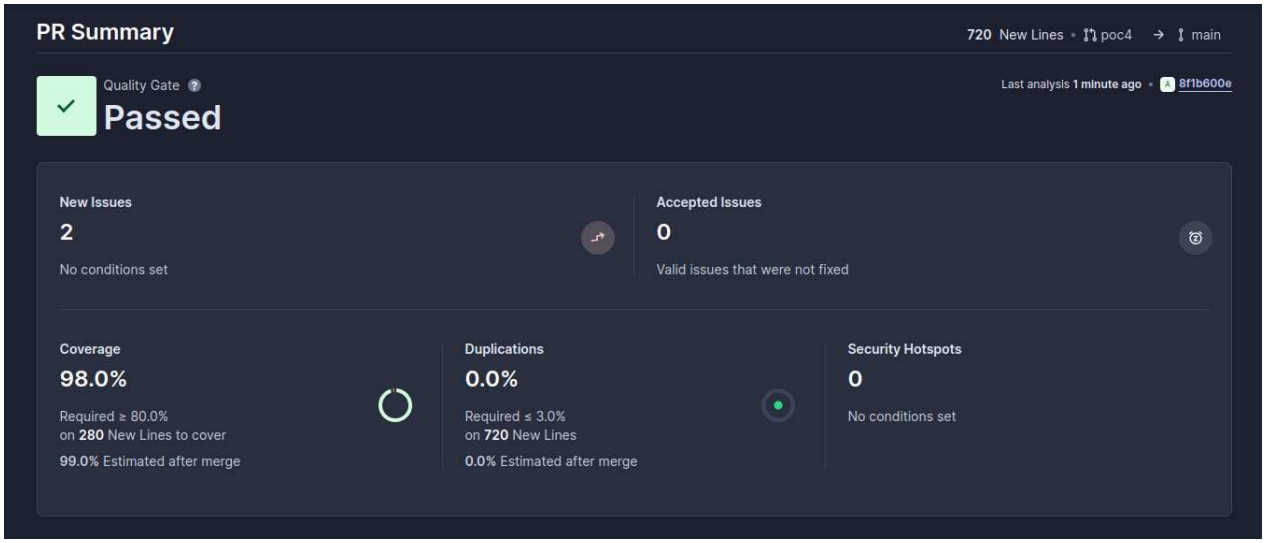
Figura 23 – Testes Com Sucesso do Domínio de Oferta



FONTE: Autores

Após a implementação dos métodos definidos pelos casos de teste, observou-se a avaliação feita pela ferramenta do SonarQube sobre as novas implementações do código fonte, como pode ser visto na Figura 24. Com isso, notou-se que o portão de qualidade configurado para o projeto já havia sido atingido, dispensando assim a necessidade de uma refatoração do código fonte.

Figura 24 – Análise do SonarQube sobre a POC 4



FONTE: Autores

[18] Disponível em: <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend/commit/d3ee67f56ae95aef10ba3055d6a5e4ea2f8dbe41>. Acessado pela última vez em 30/06/2024.

5.7 Considerações Finais do Capítulo

Este capítulo teve o intuito de apresentar o processo de engenharia reversa e reengenharia do Mia Ajuda, orientando-se pela metodologia de Provas de Conceito. Nesse contexto, o aplicativo Mia Ajuda foi reengenheirado utilizando as técnicas do TDD e DDD, descrevendo seu contexto, seguido da descrição do escopo da proposta e da documentação de cada prova de conceito necessária para alcançar o objetivo final do trabalho. Destaca-se que em cada prova de conceito, foram detalhados seus objetivos e seus respectivos requisitos. Além disso, foram apresentados os resultados iniciais obtidos com o desenvolvimento de algumas PoCs.

6 Análise de Resultados

Este capítulo aborda a análise dos resultados obtidos em cada prova de conceito discutida no Capítulo 5. Inicialmente, será abordada a Análise de Resultados das Provas de Conceito 6.1, que detalha as análises quantitativa e qualitativa a respeito a solução de cada prova de conceito. Em seguida,será fornecida uma tabela com os Resultados das Provas de Conceito 9, acompanhada de uma descrição detalhada de cada resultado. Em seguida, serão apresentados os principais Aspectos Observados 6.3 pelos autores na etapa de avaliação do método pesquisa-ação citada no Capítulo 4 na seção Método de Análise de Resultados 4.5. Por fim, serão apresentadas as Considerações Finais do capítulo 4.5.

6.1 Análise de Resultados das Provas de Conceito

Esta seção irá abordar a análise de resultados a respeito de cada prova de conceito abordadas no Capítulo 5. Conforme foi acordado na seção de Classificação da Pesquisa 4.1, que define a abordagem de pesquisa deste trabalho como sendo híbrida, esta análise possui sua avaliação qualitativa e quantitativa. A análise qualitativa foi uma análise criteriosa feita pelos autores, enquanto a análise quantitativa foi realizada por meio da ferramenta SonarQube.

6.1.1 POC 1

Esta subseção aborda a análise qualitativa e quantitativa da POC 1 - Configuração do Ambiente e Rota de Verificação 5.3. A análise foi feita sobre a Apresentação da Solução 5.3.3 desta prova de conceito.

6.1.1.1 Análise Qualitativa

No primeiro momento, será provida uma visão geral sobre o ambiente de desenvolvimento criado, no qual foram configuradas todas as tecnologias necessárias para o desenvolvimento prático deste trabalho. Nesta etapa, criou-se um repositório na plataforma GitHub, para versionamento do código e para desenvolvimento conjunto. Este repositório encontra-se público no seguinte endereço:

- <https://github.com/viniciussaturnino/TCC-Mia-Ajuda-Backend>

Em seguida, será exposta uma visão geral sobre a implementação da primeira funcionalidade do projeto. A implementação desta funcionalidade buscou validar a primeira

parte deste desafio, mostrando a disponibilidade do serviço configurado, e também buscou trazer uma base de desenvolvimento, nos moldes do TDD e DDD para os próximos desafios, que como foi acordado na seção de Metodologia Orientada a Provas de Conceito 8, seguirão o modelo de *Building Blocks*.

Com a configuração de ambiente finalizada, observa-se pela plataforma do GitHub, que consta no projeto todas as tecnologias definidas no capítulo de Suporte Tecnológico 3, e também consta a análise do SonarQube, análise esta que provém insumos para a análise quantitativa. Com isso, esta solução cumpriu todos os requisitos definidos para este desafio na seção de Requisitos do Desafio 3, sendo eles:

- O sistema deve ser acessível através de um *host*;
- O sistema deve ser capaz de gerar insumos a respeito de sua qualidade, e
- O usuário deve ser capaz de verificar a saúde do sistema.

A Figura 25 mostra a estrutura de pastas do projeto, que foi construída em forma de módulos orientados ao domínio, o que evidencia a utilização da abordagem do TDD junto com o DDD no desenvolvimento desta prova de conceito, criando assim um modelo para as demais funcionalidades. Diante do exposto, conclui-se que a solução deste desafio foi satisfatória, por construir uma base de desenvolvimento suficiente para as próximas provas de conceito, e por implementar completamente a definição feita na Definição do Desafio 5.3.1.

6.1.1.2 Análise Quantitativa

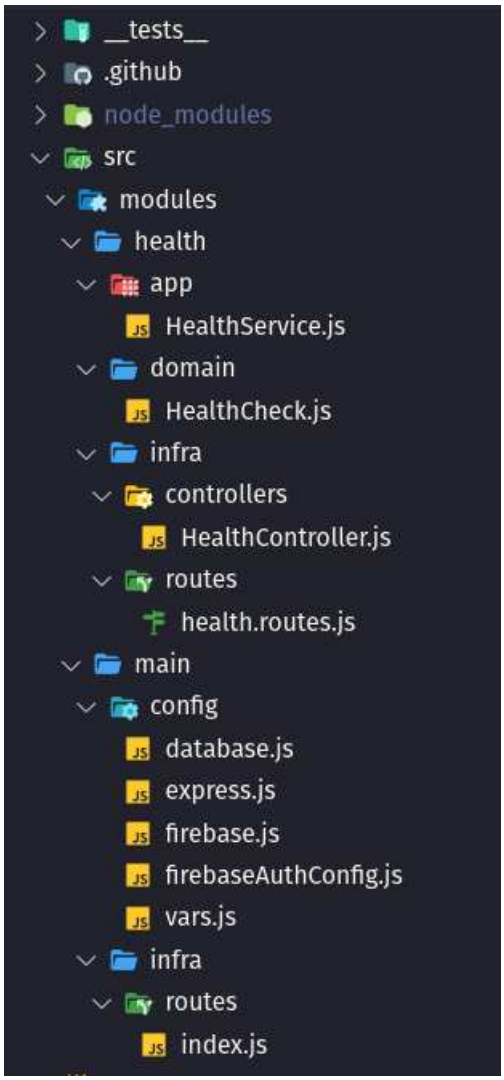
Este desafio incluiu a configuração da ferramenta SonarQube, que conforme abordado no Capítulo de Suporte Tecnológico 3, possibilita que, a cada implementação de código para o projeto hospedado no GitHub, exista uma análise de qualidade do serviço. Diante disso, os insumos coletados por esta ferramenta após o desenvolvimento deste desafio podem ser vistos na Tabela 5.

Tabela 5 – Análise de Qualidade do SonarQube sobre a POC 1

Métrica	**Nota/Porcentagem**
Confiabilidade	A
Manutenibilidade	A
Segurança	A
Cobertura de Testes	100%
Duplicações	0%

FONTE: Autores

Figura 25 – Estrutura de Pastas do Projeto Criado



FONTE: Autores

Observa-se que as métricas de Confiabilidade, Segurança e Manutenibilidade possuem nota A, indicando que não existem *bugs* nem vulnerabilidades no sistema, e a quantidade de *code smells* está menor que 5%. Além disso, observa-se que a Cobertura de Testes se encontra em 100% e a quantidade de Duplicações em 0%, sendo excelentes notas para estes quesitos. Vale ressaltar que, na configuração da Cobertura de Testes do SonarQube, foram incluídos na análise apenas os arquivos de implementação das funcionalidades, sendo não necessário testar os arquivos de configuração. Diante do exposto, conclui-se que esta solução está satisfatória para esta POC.

6.1.2 POC 2

Esta subseção aborda a análise qualitativa e quantitativa da POC 2 - Desenvolvimento do Cadastro e Login de Usuários 5.4. A análise foi feita sobre a Apresentação da

Solução 5.4.2 desta prova de conceito.

6.1.2.1 Análise Qualitativa

Assim como definido, a solução desta prova de conceito seguiu o conceito de *Building Blocks*. Com isso, o desenvolvimento deste desafio contou com o versionamento de código no mesmo repositório público do GitHub definido na prova anterior, assim como também a disponibilização de novos *endpoints* no mesmo *host* definido. Desta forma, a prova de conceito anterior serviu como pré-requisito para o cumprimento desta.

Assim como abordado anteriormente, a solução deste desafio contou com a implementação do domínio de usuários seguindo o DDD e TDD. Com isso, pode-se dizer que esta solução cumpriu todos os requisitos definidos para este desafio, como foi definido na seção de Requisitos do Desafio 5.4.1.1, sendo eles:

- O usuário deve ser capaz de efetuar seu cadastro;
- O usuário deve ser capaz de editar suas informações, e
- O usuário deve ser capaz de realizar login utilizando *email* e senha cadastrados previamente.

A Figura 26 mostra todos os *endpoints* criados para esta prova de conceito, englobando todas as operações necessárias para disponibilizar as funcionalidades de cadastro, *login*, listagem e edição de usuários. Diante do que foi apresentado, conclui-se que a solução deste desafio foi satisfatória em termos qualitativos.

6.1.2.2 Análise Quantitativa

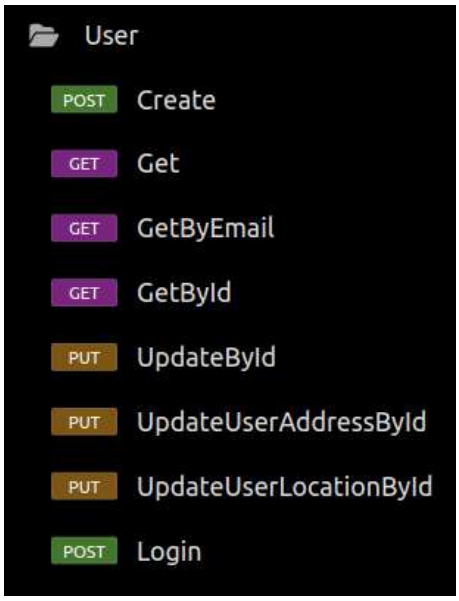
Assim como abordado no Capítulo de Suporte Tecnológico 3, este desafio possui a análise de qualidade da ferramenta SonarQube, que foi configurada na prova de conceito anterior. Com ela, foi possível obter insumos quantitativos do desenvolvimento deste desafio, que podem ser observados na Tabela 6.

Tabela 6 – Análise de Qualidade do SonarQube sobre a POC 1

Métrica	**Nota/Porcentagem**
Confiabilidade	A
Manutenibilidade	A
Segurança	A
Cobertura de Testes	100%
Duplicações	0%

FONTE: Autores

Figura 26 – Rotas Criadas para a POC 2



FONTE: Autores

Observa-se que as métricas de Confiabilidade, Segurança e Manutenibilidade permanecem com a nota A, indicando que o código fonte permanece sem vulnerabilidades e com uma baixa taxa de *code smells*. Além disso, a Cobertura de Testes permanece em 100% e a quantidade de Duplicações em 0%, sendo excelentes notas para estas métricas. Com isso, conclui-se que esta solução está satisfatória para esta POC.

6.1.2.3 Conclusão

Após a Análise de Resultados desta prova de conceito, conclui-se que todos os requisitos foram atendidos tanto em termos qualitativos como quantitativos. É possível observar que as funcionalidades propostas para esta POC foram desenvolvidas corretamente, e as metodologias do DDD e TDD contribuíram com a qualidade do código fonte, como pode ser observado pela alta cobertura de testes que chegou a 100%, além das boas avaliações nas outras métricas de qualidade.

6.1.3 POC 3

Esta subseção aborda a análise qualitativa e quantitativa da POC 3 - Desenvolvimento da Funcionalidade de Pedido de Ajuda 5.5. A análise foi feita sobre a Apresentação da Solução 5.5.2 desta prova de conceito.

6.1.3.1 Análise Qualitativa

A solução desta prova de conceito consistiu na implementação, utilizando TDD e DDD, da funcionalidade de solicitação de ajuda. Com isso, é possível concluir que a

solução cumpriu todos os requisitos definidos na etapa de Requisitos do Desafio, sendo eles:

- O usuário deve ser capaz de visualizar as categorias existentes;
- O usuário deve ser capaz de cadastrar um pedido de ajuda;
- O usuário deve ser capaz de visualizar seus respectivos pedidos de ajuda, e
- O usuário deve ser capaz de desativar um pedido de ajuda que pertence a ele.

A Figura 27 mostra os novos *endpoints* disponíveis na API após a resolução desta POC. Com estes *endpoints*, um usuário consegue listar as categorias disponíveis no sistema, além de também conseguir criar e administrar pedidos de ajuda. Diante do exposto, conclui-se que a solução apresentada para esta POC foi satisfatória em termos qualitativos.

Figura 27 – Rotas Criadas para a POC 3

FONTE: Autores

6.1.3.2 Análise Quantitativa

Assim como nas provas de conceito anteriores, a ferramenta utilizada para a obtenção de insumos quantitativos foi o SonarQube. Com isso, a solução desta POC obteve uma análise da ferramenta, que pode ser vista na Tabela 7.

Tabela 7 – Análise de Qualidade do SonarQube sobre a POC 1

Métrica	**Nota/Porcentagem**
Confiabilidade	A
Manutenibilidade	A
Segurança	A
Cobertura de Testes	92.7%
Duplicações	0%

FONTE: Autores

É possível observar que as métricas de Confiabilidade, Manutenibilidade, Segurança e Duplicações se mantiveram na nota máxima, assegurando a qualidade do código fonte em relação à ausência de vulnerabilidades, falhas e duplicações. A métrica de Cobertura de Testes deixou de ser 100%, porém ainda manteve uma alta nota de 92.7%, satisfazendo assim o portão de qualidade definido. Desta forma, conclui-se que esta solução é satisfatória em termos quantitativos.

6.1.3.3 Conclusão

Após a Análise de Resultados desta POC, é possível observar que a solução apresentada foi satisfatória em termos qualitativos e quantitativos. Além disso, é possível notar que a implementação utilizando TDD e DDD evoluiu o código fonte de forma que a ferramenta SonarQube avaliou a qualidade como satisfatória sem a necessidade de uma posterior refatoração, evidenciado o poder em termos de boas práticas de desenvolvimento que esta abordagem possui.

6.1.4 POC 4

Esta subseção aborda a análise qualitativa e quantitativa da POC 4 - Desenvolvimento da Funcionalidade de Oferta de Ajuda 5.6. A análise foi feita sobre a Apresentação da Solução 5.6.2 desta prova de conceito.

6.1.4.1 Análise Qualitativa

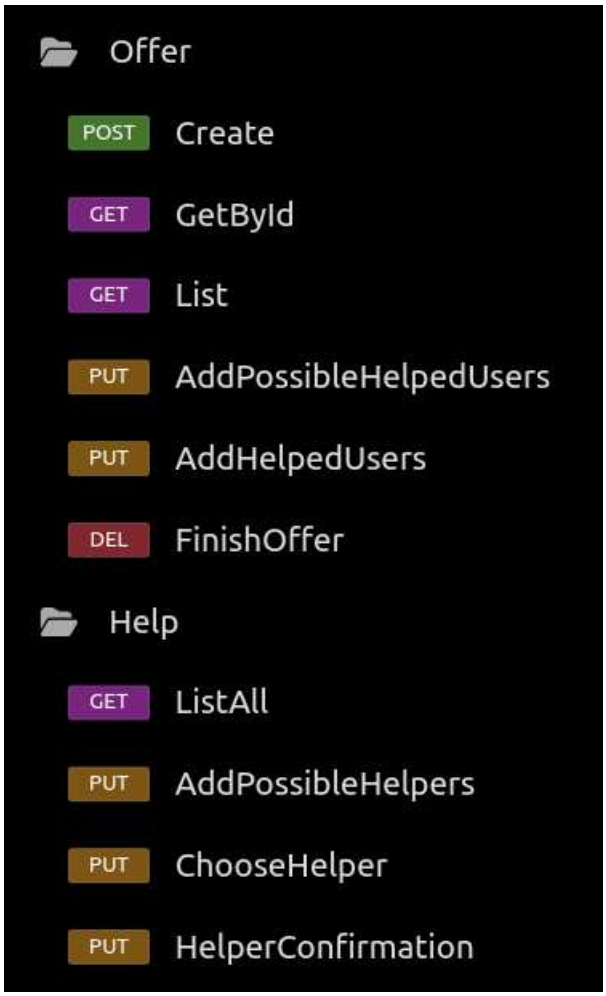
A solução desta POC evidenciou que houve uma implementação, utilizando TDD e DDD, da funcionalidade de oferta de ajuda. Esta implementação cobriu todos os requisitos definidos na etapa de Requisitos do Desafio, sendo eles:

- O usuário deve ser capaz de visualizar todos os pedidos de ajuda;
- O usuário deve ser capaz de filtrar os pedidos de ajuda por categoria,
- O usuário deve ser capaz de oferecer ajuda para um pedido específico,

- O usuário deve ser capaz de aceitar uma ajuda oferecida, e
- O usuário deve ser capaz de finalizar uma oferta de ajuda.

Com isso, a API passou a disponibilizar novos *endpoints*, como pode ser observado na Figura 28, possibilitando a interação dos usuários com o sistema a fim de utilizar a funcionalidade implementada nesta prova de conceito. Observando os novos *endpoints*, nota-se a possibilidade de administrar ofertas de ajuda pela API, além de também poder adicionar possíveis ajudantes ou ajudados, e escolher um ajudante ou um ou mais ajudados. Dito isso, conclui-se que a solução desta POC foi satisfatória em sua análise qualitativa.

Figura 28 – Rotas Criadas para a POC 4



FONTE: Autores

6.1.4.2 Análise Quantitativa

A ferramenta SonarQube, configurada para o projeto, gerou insumos quantitativos para as novas adições no código fonte, feitas com a resolução desta POC. Com isso, obteve-

se a avaliação que pode ser vista na Tabela 8.

Tabela 8 – Análise de Qualidade do SonarQube sobre a POC 1

Métrica	**Nota/Porcentagem**
Confiabilidade	A
Manutenibilidade	A
Segurança	A
Cobertura de Testes	98%
Duplicações	0%

FONTE: Autores

Analisando as notas obtidas pela ferramenta, é possível notar primeiramente que o portão de qualidade foi atingido, sendo isso um requisito mínimo para adição de novas implementações ao código fonte principal. Além disso, pode-se ver que as notas máximas foram mantidas nas métricas de Confiabilidade, Menutenibilidade, Segurança e Duplicações, enquanto a Cobertura de Testes foi avaliada com 98%, sendo uma ótima nota para este quesito. Assim, conclui-se que a solução deste desafio foi satisfatória quantitativamente.

6.1.4.3 Conclusão

Com o resultado das análises qualitativa e quantitativa, conclui-se que esta solução foi satisfatória em ambos os quesitos. Além de realizar a implementação de todos os requisitos definidos para esta POC, a alta qualidade do código também foi mantida de acordo com a avaliação do SonarQube.

6.2 Resultados Obtidos nas Prova de Conceito

Após uma revisão extensiva da literatura, seguida pelo planejamento, desenvolvimento e apresentação das soluções para cada prova de conceito, procedeu-se à análise realizada pelo SonarQube sobre cada solução, conforme detalhado na seção de Método de Análise de Resultados 4.5. Foram definidas algumas métricas considerando aspectos críticos para essa análise, incluindo problemas no código [1], cobertura de código, duplicação de código e complexidade ciclomática. A Tabela 9 resume esses aspectos nas soluções desenvolvidas para cada prova de conceito.

Métrica 1: Quantidade de problemas no código

Métrica 2: Porcentagem de Cobertura de testes

Métrica 3: Quantidade de linhas duplicadas no código

Métrica 4: Complexidade ciclomática

[1] Entende-se por problemas, qualquer tipo de *bug*, falhas de segurança, código não utilizado, entre outros.

Tabela 9 – Resultados da Análise Quantitativa das PoCs

Prova de Conceito	Métrica 1	Métrica 2	Métrica 3	Métrica 4
POC 1	0	100.0%	0	3
POC 2	0	100.0%	0	91
POC 3	0	92.7%	0	152
POC 4	0	98.0%	0	234

FONTE: Autores

O SonarQube é uma ferramenta amplamente utilizada para análise contínua de qualidade de código em uma variedade de linguagens de programação, incluindo JavaScript/TypeScript. Ele oferece uma avaliação abrangente que vai além da detecção de *bugs* e vulnerabilidades, abrangendo também a cobertura de código, a duplicação de código, a complexidade ciclomática, entre outros aspectos. A ferramenta é integrada aos processos de desenvolvimento de *software* para fornecer *feedback* instantâneo aos desenvolvedores, ajudando a melhorar a qualidade do código ao longo do ciclo de vida do projeto. Quando uma aplicação atinge os padrões mínimos estabelecidos pelo SonarQube em todos esses aspectos, isso indica que a aplicação alcançou um alto nível de qualidade de código, contribuindo para a robustez, segurança e manutenibilidade do *software* desenvolvido. Nesse contexto, é importante enfatizar que todas as Provas de Conceito atenderam os requisitos mínimos de qualidade de código definidas pela ferramenta.

6.3 Aspectos Observados

Durante o desenvolvimento da solução de cada prova de conceito, alguns aspectos positivos e negativos foram observados. Esses aspectos são de alta relevância para o trabalho, pois através deles pode-se evidenciar ou não algumas características do uso combinado do TDD e DDD na execução da Arquitetura Limpa e micro front-end, além de identificar cenários que surgiram durante o uso combinado das duas.

Os principais aspectos, positivos e negativos, observados ao longo das provas de conceito foram:

Aspectos positivos:

- Durante o desenvolvimento através da reengenharia utilizando o TDD, foram identificadas situações onde métodos existentes, como por exemplo uma função previamente denominada como *updateUserById*, inicialmente projetada para atualizar usuários pelo ID, na prática utilizavam outros critérios como o email para efetuar a atualização. O uso do TDD permitiu detectar essas inconsistências de forma ante-

cipada, possibilitando correções pontuais e melhorias substanciais na integridade e funcionalidade do sistema;

- por ser uma reengenharia aliada com TDD, o sistema legado deu uma boa base para quais casos de teste poderiam ser criados para a implementação;
- utilizar o TDD e DDD na implementação facilitou a manutenção da qualidade do projeto em termos quantitativos, como pode ser evidenciado pela necessidade de haver apenas uma refatoração durante o desenvolvimento das PoCs para atingir o portão de qualidade definido pela plataforma do SonarQube;
- A utilização de storyslicing vertical na definição dos requisitos das PoCs juntamente com o desenvolvimento utilizando TDD e DDD se mostrou eficaz, pois foi possível separar cada atividade de desenvolvimento, que no contexto deste trabalho se trata das PoCs, por domínio do produto, no qual cada domínio deveria possuir testes do funcionamento de todas as camadas necessárias para seu funcionamento, possuindo assim muita sinergia com o DDD, que possui uma organização por domínio, e com o TDD, que necessita da criaçãeo de testes antes da implementação;
- A implementação de TDD e DDD não apenas melhorou a manutenibilidade do código, facilitando futuras alterações e atualizações, mas também promoveu a modularização do sistema. Isso proporcionou uma estrutura mais organizada e flexível, facilitando a escalabilidade e a evolução contínua do projeto;
- A aplicação de TDD e DDD resultou em um código mais legível e compreensível, o que não só facilitou a interpretação do funcionamento do sistema, mas também aumentou a confiabilidade do código. Isso é crucial para garantir que o *software* desenvolvido seja robusto e livre de erros, promovendo uma experiência positiva para os usuários finais, e
- A estratégia de desenvolvimento baseada em TDD fortaleceu a testabilidade do sistema, garantindo que todas as funcionalidades fossem testadas de maneira abrangente desde o início do desenvolvimento. Isso é essencial para detectar problemas precocemente e assegurar a qualidade do *software* entregue.

Aspectos negativos:

- Em projetos pequenos que não possuem muitos arquivos, o desenvolvimento com DDD pode criar complexidades a mais em termos de arquitetura de pastas do projeto, devido à separação dos arquivos por domínio;
- Devido ao uso de várias ferramentas combinadas, não somente o tempo de desenvolvimento, mas também a complexidade de aplicar essa metodologia de desenvolvimento é maior, e.

- A curva de aprendizado acaba sendo um ponto pertinente, tendo em vista que em uma situação real de desenvolvimento de um produto, todos dentro da equipe precisam dominar bem todas as ferramentas utilizadas.

6.4 Considerações Finais do Capítulo

Este capítulo teve o intuito de apresentar os resultados obtidos por meio da execução da Engenharia Reversa e Reengenharia do Mia Ajuda realizado neste trabalho, oferecendo uma visão detalhada da análise quantitativa e qualitativa conduzida em cada prova de conceito. Foram apresentados os resultados relativos aos problemas de código, duplicação de código, cobertura de testes e complexidade ciclomática de cada prova de conceito. Por fim, foram apresentados os aspectos positivos e negativos observados em cada prova de conceito discutidos em conjunto entre os desenvolvedores (autores), proporcionando uma visão abrangente do progresso ao longo do desenvolvimento.

7 Conclusão

Esse capítulo aborda a conclusão deste trabalho. Inicialmente, será apresentada uma breve contextualização 7.1, seguida do *Status* Atual do Trabalho 7.2, com uma visão geral dos pontos obtidos com o encerramento do trabalho, visando os objetivos e questão de pesquisa, estabelecidos em 1.4 e 1.3 respectivamente. Por fim, serão apresentadas Possíveis Melhorias 7.3 identificadas ao longo do processo de Engenharia Reversa e Reengenharia do Mia Ajuda 5, a fim de guiar futuros desenvolvedores e apresentar oportunidades de aperfeiçoamento do aplicativo.

7.1 Contexto

Conforme discutido no Capítulo 1 - Introdução, a aplicação dos métodos de TDD e DDD combinadas de forma correta, gera um *software* com maior qualidade, confiabilidade, coesão e com um nível de organização elevado considerando o domínio do problema que o *software* visa resolver ((BECK, 2022); (VERNON, 2013)). A reengenharia e a engenharia reversa surgem no contexto onde um aplicativo, já desenvolvido, pudesse ser incorporado. Uma vez que sua evolução é uma tarefa bastante improvável (PEREIRA; SOUZA, 2023) devido ao alto acoplamento, baixa coesão e a dificuldade de manter uma alta cobertura de testes no sistema.

Assim, foram estabelecidas Questões de Pesquisa e de Desenvolvimento com o objetivo de conduzir uma investigação aprofundada e prática. Para atingir esses objetivos, foi necessário realizar uma extensa pesquisa na literatura especializada, abordando conceitos de Engenharia Reversa, Reengenharia, TDD, DDD e Técnicas de Programação para Aplicativos Móveis que se encontram na fase de Manutenção Evolutiva do ciclo de vida de um software. Com base nessa pesquisa, identificou-se a necessidade de aplicar Engenharia Reversa ao aplicativo Mia Ajuda, seguida de uma Reengenharia orientada pelos princípios de TDD e DDD.

7.2 Status Atual do Trabalho

Ainda no Capítulo 1, foram definidos o objetivo geral do trabalho. Por consequência, com o objetivo de cumprir esse objetivo, estabeleceu-se, ainda, os seguintes objetivos específicos:

- Estudo sobre Engenharia Reversa de Aplicativos Móveis que se encontram na etapa de Manutenção Evolutiva. Status: Concluído no Capítulo 2 - Referencial Teórico;

- Estudo sobre Reengenharia de Aplicativos Móveis que se encontram na etapa de Manutenção Evolutiva. Status: Concluído no Capítulo 2 - Referencial Teórico;
- Levantamento sobre TDD. Status: Concluído no Capítulo 6 - Análise de Resultados;
- Levantamento sobre DDD. Status: Concluído no Capítulo 6 - Análise de Resultados;
- Levantamento sobre Técnicas de Programação. Status: Concluído no Capítulo 6 - Análise de Resultados;
- Documentação das principais recomendações da Engenharia de *Software* acordadas nos estudos e levantamentos realizados. Status: Concluído no Capítulo 6 - Análise de Resultados;
- Aplicação das principais recomendações no Aplicativo Mia Ajuda. Status: Concluído no desenvolvimento das PoCs descritas no Capitulo 5, e
- Exposição dos resultados obtidos. Status: Concluído no Capítulo 6 - Análise de Resultados.

Dessa forma, com todos os objetivos específicos concluídos, tem-se que o objetivo geral também foi concluído, sendo ele o de realizar a reengenharia de um aplicativo móvel existente, visando testabilidade facilitada eadequada modelagem de domínio, sendo esse processo apoiado em práticas que possam ser usadas em aplicativos móveis de cunho similar.

O propósito de um objetivo geral é responder a uma questão específica de pesquisa. No caso deste trabalho, a seguinte questão foi respondida: Quais são as principais recomendações da Engenharia de *Software* no que diz respeito aos processos de Engenharia Reversa e Reengenharia de Aplicativos Móveis existentes, tomando como base TDD, DDD e Técnicas de Programação?

Com isso, é possível concluir que essa questão foi respondida por meio da execução da Engenharia Reversa e Reengenharia do Mia Ajuda, que ajudou a validar a pertinência das técnicas investigadas. Com base na Metodologia de Análise de Resultados 4.5 concluise que há viabilidade e benefícios em aplicar metodologias modernas na reengenharia de aplicativos móveis.

7.3 Possíveis Melhorias

O trabalho realizado até aqui permite uma base sólida para futuras investigações e desenvolvimentos, abrindo caminho para a exploração de novas metodologias, ferramentas e técnicas que possam aprimorar ainda mais a reengenharia de *software*.

Após alcançar os objetivos definidos neste trabalho, é possível continuar os estudos sobre Reengenharia e Engenharia Reversa utilizando TDD e DDD, explorando desde diferentes métodos até novas ferramentas. Devido ao escopo mais restrito deste trabalho, algumas oportunidades para trabalhos futuros podem ser exploradas. São elas:

- Aplicação de diferentes arquiteturas (Arquitetura Hexagonal, por exemplo) no desenvolvimento, verificando se os aspectos gerais apresentados neste trabalho se mantém;
- Aplicar outras técnicas do DDD como *Event Storming* ou *Domain Storytelling*, para aprimorar a compreensão do domínio e a modelagem do *software*;
- Adotar a integração do BDD (*Behavior Driven Development*) com TDD e DDD para um ciclo de desenvolvimento aprimorado, mais coeso e focado nas necessidades do usuário;
- Adotar a análise de performance e escalabilidade, para identificar possíveis gargalos e garantir maior eficiência do aplicativo, tais como abordagens que executem testes de carga e verifiquem o desempenho do sistema em diferentes condições, e
- Adotar diferentes tecnologias, *frameworks* e ferramentas de desenvolvimento com o objetivo de identificar diferenças e/ou dificuldades na abordagem e elaboração do trabalho.

Referências

AHMAD, M. O.; MARKKULA, J.; OIVO, M. Kanban in software development: A systematic literature review. In: IEEE. *2013 39th Euromicro conference on software engineering and advanced applications*. [S.l.], 2013. p. 9–16. Citado na página 53.

AL., S. et. A evolução da tecnologia: Vivendo uma nova era. XI EPCC - Encontro Internacional de Produção Científica, 2019. Citado na página 23.

BECK, K. *Test driven development: By example*. [S.l.]: Addison-Wesley Professional, 2022. Citado 6 vezes nas páginas 23, 24, 25, 32, 50 e 93.

BIRCHALL, C. *Re-engineering legacy software*. [S.l.]: Simon and Schuster, 2016. Citado na página 37.

CAGNIN, M. I. *PARFAIT: uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks*. Tese (Doutorado) — Universidade de São Paulo, 2005. Citado 4 vezes nas páginas 24, 36, 37 e 39.

CARVALHO, B. V. d.; MELLO, C. H. P. Aplicação do método ágil scrum no desenvolvimento de produtos de software em uma pequena empresa de base tecnológica. *Gestão & Produção*, SciELO Brasil, v. 19, p. 557–573, 2012. Citado na página 53.

CHIKOFSKY, E. J.; CROSS, J. H. Reverse engineering and design recovery: A taxonomy. *IEEE software*, IEEE, v. 7, n. 1, p. 13–17, 1990. Citado 3 vezes nas páginas 24, 36 e 50.

COUTINHO, G. L. A era dos smartphones: Um estudo exploratório sobre o uso dos smartphones no brasil. 2014. Citado na página 33.

DEV. *DEV*. 2023. Disponível em: <https://dev.to/womakerscode/o-que-e-tdd-4b5f>. Citado 2 vezes nas páginas 31 e 32.

DICIO, D. online de português. *Barreira, facilitador e motivação. Disponível em:< https://www. dicio. com. br>. Acesso em*, v. 21, 2020. Citado na página 37.

EILAM, E. *Reversing: secrets of reverse engineering*. [S.l.]: John Wiley & Sons, 2011. Citado na página 35.

FGV. *FGV*. 2023. Disponível em: <https://portal.fgv.br/noticias/uso-ti-brasil-pais-tem-mais-dois-dispositivos-digitais-habitante-revela-pesquisa>. Citado 2 vezes nas páginas 33 e 34.

FONSECA, J. J. S. da. *Apostila de metodologia da pesquisa científica*. [S.l.]: João José Saraiva da Fonseca, 2002. Citado na página 54.

FOWLER, M. *Refactoring*. [S.l.]: Addison-Wesley Professional, 2018. Citado na página 50.

FREITAS, B. C. d. Flutter e react native: uma análise comparativa entre dois frameworks de desenvolvimento mobile multiplataforma. Universidade Federal do Rio de Janeiro, 2022. Citado na página 34.

FURTADO, A. T. et al. Avaliação de resultados e impactos da pesquisa e desenvolvimento: avanços e desafios metodológicos a partir de estudo de caso. *Gestão & Produção*, SciELO Brasil, v. 15, p. 381–392, 2008. Citado na página 56.

GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de pesquisa*. [S.l.]: Plageder, 2009. Citado 3 vezes nas páginas 47, 48 e 54.

GIL, A. C. et al. *Como elaborar projetos de pesquisa*. [S.l.]: Atlas São Paulo, 2002. v. 4. Citado na página 48.

GITHUB. *GitHub*. 2023. Disponível em: <https://github.com/>. Citado na página 26.

GUDWIN, R. R. *Componentes, Frameworks e Design Patterns*. [S.l.]: UNICAMP, 2010. Citado na página 24.

KINSTA. *Kinsta*. 2023. Disponível em: <https://kinsta.com/pt/blog/aplicativos-node-js/>. Citado na página 41.

KRAFTA, L. et al. O método da pesquisa-ação: um estudo em uma empresa de coleta e análise de dados. *Revista Quanti & Quali*, 2007. Citado na página 56.

LARMAN, C. *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. [S.l.]: Pearson Education India, 2012. Citado na página 25.

MARTIN, R. C. Design principles and design patterns. *Object Mentor*, v. 1, n. 34, p. 597, 2000. Citado 2 vezes nas páginas 25 e 50.

MARTIN, R. C. *Clean Code-Refactoring, Patterns, Testen und Techniken für sauberen Code: Deutsche Ausgabe*. [S.l.]: MITP-Verlags GmbH & Co. KG, 2013. Citado na página 25.

MARTIN, R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series)*. [S.l.]: Prentice Hall, 2019. Citado 3 vezes nas páginas 29, 30 e 61.

MATTOS, K. M.; DOLL, L. M.; ALMEIDA, J. Domain-driven design e test-driven development. 2010. Citado na página 24.

MELO, D. N. A. d. Ux. br: um modelo de maturidade de experiência de usuário para aplicações governamentais. 2023. Citado na página 34.

MIAAJUDA. *Mia Ajuda*. 2020. Disponível em: <https://miaajuda.netlify.app/>. Citado 5 vezes nas páginas 23, 25, 39, 61 e 62.

MONGODB. *Why Use MongoDB and When to Use It?* 2023. Disponível em: <https://www.mongodb.com/pt-br/why-use-mongodb>. Citado na página 43.

PARADIGM, V. *User Story Splitting - Vertical Slice vs Horizontal Slice*. 2024. Disponível em: <https://www.visual-paradigm.com/scrum/user-story-splitting-vertical-slice-vs-horizontal-slice/>. Citado 2 vezes nas páginas 32 e 33.

PEREIRA, G. D. S.; SOUZA, D. G. d. Arquitetura de software: um estudo orientado ao desenvolvimento de aplicativos móveis híbridos. 2023. Citado 2 vezes nas páginas 24 e 93.

PRASANNA, K. et al. Poc design: a methodology for proof-of-concept (poc) development on internet of things connected dynamic environments. *Security and Communication Networks*, Hindawi Limited, v. 2021, p. 1–12, 2021. Citado 2 vezes nas páginas 48 e 50.

ROSENBERG, L. H.; HYATT, L. E. Software re-engineering. *Software Assurance Technology Center*, Citeseer, p. 2–3, 1996. Citado 2 vezes nas páginas 37 e 38.

SANTOS, I. S. et al. Definição e aplicação de um processo de testes ágeis: um relato de experiência. In: SBC. *Anais do XIV Simpósio Brasileiro de Qualidade de Software*. [S.l.], 2015. p. 228–235. Citado na página 34.

SILVA, P. V. d. Microfrontends e arquitetura limpa: Um estudo exploratório orientado a provas de conceito. 2023. Citado 2 vezes nas páginas 50 e 51.

SNEED, H. M. Planning the reengineering of legacy systems. *IEEE software*, IEEE, v. 12, n. 1, p. 24–34, 1995. Citado na página 37.

SONARQUBE. *SonarQube*. 2023. Disponível em: <https://docs.sonarqube.org/latest>. Citado na página 44.

SUTHERLAND, J. *SCRUM: A arte de fazer o dobro de trabalho na metade do tempo*. [S.l.]: Leya, 2014. Citado 2 vezes nas páginas 52 e 53.

THIOLLENT, M. *Metodologia da Pesquisa-aç ao*. [S.l.]: Cortez, Sao Paulo, 1988. Citado 2 vezes nas páginas 54 e 55.

TURILLI, M. et al. Designing workflow systems using building blocks. *arXiv preprint arXiv:1609.03484*, 2016. Citado na página 52.

VERNON, V. *Implementing domain-driven design*. [S.l.]: Addison-Wesley, 2013. Citado 4 vezes nas páginas 24, 25, 50 e 93.

VERNON, V. *Domain-driven design distilled*. [S.l.]: Addison-Wesley Professional, 2016. Citado na página 23.