UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
BACHARELADO EM ENGENHARIA DE SOFTWARE

Pedro Henrique Barros Mendonça

Estudo de caso: Construção de ferramenta de apoio à melhoria da qualidade de software utilizando modelo de inteligência artificial (IA) generativa na automatização de casos de teste de sistema e inspeções de usabilidade

Manaus
2026

Pedro Henrique Barros Mendonça

Estudo de caso: Construção de ferramenta de apoio à melhoria da qualidade de software utilizando modelo de inteligência artificial (IA) generativa na automatização de casos de teste de sistema e inspeções de usabilidade

Monografia de Graduação apresentada ao Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para a obtenção do grau de bacharel em Engenharia de Software.

Orientador(a) Prof. Msc. Felipe Sonntag Manzoni

Manaus
2026

Pedro Henrique Barros Mendonça

Estudo de caso: Construção de ferramenta de apoio à melhoria da qualidade de software utilizando modelo de inteligência artificial (IA) generativa na automatização de casos de teste de sistema e inspeções de usabilidade

Monografia de Graduação apresentada ao Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para a obtenção do grau de bacharel em Engenharia de Software.

Este trabalho foi defendido e aprovado pela banca em 08/07/2026.

BANCA EXAMINADORA

Prof. Msc. Felipe Sonntag Manzoni - UFAM

Orientador(a)

Prof. Dr. Leonardo Carneiro Marques - UFAM

Membro da Banca Examinadora

Dedico este trabalho à minha família: à minha mãe, Patrícia Cristina de Oliveira Barros; ao meu pai, Valdenir da Silva Mendonça; e ao meu irmão, Ethevaldo Alves de Mendonça Neto. Obrigado por despertarem o melhor que há em mim, especialmente, nos momentos difíceis.

AGRADECIMENTOS

Gostaria de expressar minha gratidão à Universidade Federal do Amazonas (UFAM) por proporcionar um ambiente acolhedor para os que buscam conhecimento e pela oportunidade de realizar o curso de Bacharelado em Engenharia de Software e usufruir de todos os programas pensados para beneficiar os estudantes.

Ao Instituto de Computação (ICOMP), na figura de todo o corpo docente, destaco meu respeito e admiração pelo trabalho de excelência realizado sempre e pelas diversas iniciativas tomadas a fim de nos tornar profissionais da mais alta qualidade e por sempre inovar em suas políticas.

Sou grato também à Coordenação do curso de Engenharia de Software, na figura da Prof. Dra. Ana Carolina Oran Rocha, pelo empenho em trazer continuamente diversas melhorias para o programa e por todo o esforço em garantir a qualidade dos profissionais formados no curso e o bem-estar dos alunos durante a jornada.

Ao meu orientador, Prof. Msc. Felipe Sonntag Manzoni, que gentilmente aceitou me guiar na construção deste trabalho e com muita paciência realizou todas as orientações necessárias para que essa etapa fosse proveitosa, um muito obrigado!

Por fim, agradeço aos meus queridos amigos feitos nesse caminho, em especial, a David Yan dos Santos Prado, Lucas Eduardo Souza de Moura, Nicolas Araujo Sampaio e Thiago Vinicius Costa Guimarães, pelo companheirismo desde o início.

“Nada de desgosto, nem de desânimo; se acabas de fracassar, recomeça.”

Marco Aurélio, *Meditações*

RESUMO

A aplicação de técnicas de teste de sistema e de inspeção de usabilidade é reconhecidamente cara para os times de desenvolvimento de software, pois demanda um grande investimento de tempo para ser planejada e executada, o que reduz sua cobertura ao longo do ciclo de vida de um projeto. Essa carência decorre, sobretudo, da falta de ferramentas que automatizem essas verificações e da complexidade das soluções existentes para profissionais pouco versados em programação. Diante desse cenário, o presente trabalho tem como objetivo desenvolver uma ferramenta de apoio à melhoria da qualidade de software que utilize modelos de inteligência artificial (IA) generativa na automatização de casos de teste de sistema e de inspeções de usabilidade. A proposta organiza as informações de um projeto em personas e requisitos funcionais com critérios de aceitação, permite a gravação do fluxo correto de uma funcionalidade e possibilita sua reexecução autônoma, gerando relatórios em formato de chat sobre a conformidade do sistema aos critérios de aceite, além de sugestões de melhoria de interface fundamentadas em heurísticas de usabilidade. O trabalho apresenta ainda um levantamento de ferramentas e arquiteturas de teste de sistema e de inspeção de usabilidade já existentes, comparando-as com a solução proposta.

Palavras-chave: Qualidade de Software; Teste de Sistema; Inspeção de Usabilidade; Inteligência Artificial Generativa; Automação de Testes.

ABSTRACT

Applying system testing and usability inspection techniques is notoriously expensive for software development teams, since it demands a large investment of time to be planned and executed, which reduces its coverage throughout a project's life cycle. This shortage stems mainly from the lack of tools that automate these verification tasks and from the complexity of existing solutions for professionals with little programming background. Given this scenario, the present work aims to develop a tool to support software quality improvement that uses generative artificial intelligence (AI) models to automate system test cases and usability inspections. The proposal organizes project information into personas and functional requirements with acceptance criteria, allows recording the correct flow of a feature and enables its autonomous re-execution, generating chat-formatted reports on the system's compliance with the acceptance criteria, in addition to interface improvement suggestions grounded in usability heuristics. The work also presents a survey of existing system testing and usability inspection tools and architectures, comparing them with the proposed solution.

Keywords: Software Quality; System Testing; Usability Inspection; Generative Artificial Intelligence; Test Automation.

Lista de figuras

Figura 1 – Arquitetura em camadas da aplicação. 39
Figura 2 – Organização das páginas da aplicação: barra lateral de navegação e Dashboard. 40
Figura 3 – Fluxo de funcionalidades da aplicação. 41
Figura 4 – Página de Projetos: criação, edição, exclusão e seleção do projeto ativo. 42
Figura 5 – Modelo entidade-relacionamento do banco de dados. 43
Figura 6 – Recorte do quadro de *brainstorming* – domínio de projetos. 47
Figura 7 – Recorte do quadro de *brainstorming* – inspeção de usabilidade e requisitos funcionais. 48
Figura 8 – Recorte do quadro de *brainstorming* – agente de teste de sistema e critério de aceite. 49
Figura 9 – Recorte do quadro de *brainstorming* – agente de inspeção de usabilidade. 49
Figura 10 – Recorte do quadro de *brainstorming* – execução do teste de sistema. 50
Figura 11 – Domínio da qualidade de software e seus subdomínios. 51
Figura 12 – Subdomínio de Projetos (suporte) e seu único contexto delimitado. . 52
Figura 13 – Subdomínio de Teste de Sistema (principal) e seu único contexto delimitado. 53
Figura 14 – Subdomínio de Inspeção de Usabilidade (principal) e seu único contexto delimitado. 53
Figura 15 – Pacotes de trabalho cadastrados na plataforma OpenProject. 57
Figura 16 – Página de Inspeção de Usabilidade com o resultado de uma execução real do agente. 61
Figura 17 – Página de Teste de Sistema com o histórico de execuções reais do agente. 62
Figura 18 – Prompt do sistema do agente de conformidade de Requisitos Funcionais, com exemplo de formato válido e inválido. 63
Figura 19 – Página de Personas com uma persona real cadastrada no sistema. 64

Figura 20 – Página de Requisitos Funcionais com um requisito e critério de aceite reais. 65

Lista de tabelas

Tabela 1 – Critérios utilizados na comparação entre as ferramentas. 37
Tabela 2 – Tabela comparativa entre as ferramentas e o presente trabalho. . . 38
Tabela 3 – Domínios identificados e seus propósitos. 51
Tabela 4 – Glossário da linguagem onipresente do problema. 54
Tabela 5 – Amostra da base de conhecimento de heurísticas de usabilidade, agrupada por heurística de Nielsen. 60

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
AI-TMM	*AI Trust Maturity Model* (Modelo de Maturidade de Confiança em IA)
API	*Application Programming Interface* (Interface de Programação de Aplicações)
BDD	*Behavior-Driven Development* (Desenvolvimento Orientado a Comportamento)
BPMN	*Business Process Model and Notation* (Modelo e Notação de Processos de Negócio)
CI/CD	*Continuous Integration / Continuous Delivery* (Integração Contínua / Entrega Contínua)
DDD	*Domain-Driven Design* (Design Orientado a Domínio)
IA	Inteligência Artificial
IHC	Interação Humano-Computador
LLM	*Large Language Model* (Modelo de Linguagem de Grande Escala)
MVC	*Model-View-Controller* (Modelo-Visão-Controlador)
RAG	*Retrieval-Augmented Generation* (Geração Aumentada por Recuperação)
RPA	*Robotic Process Automation* (Automação Robótica de Processos)
SEO	*Search Engine Optimization* (Otimização para Motores de Busca)
UML	*Unified Modeling Language* (Linguagem de Modelagem Unificada)
UX	*User Experience* (Experiência do Usuário)

Sumário

1 INTRODUÇÃO 15
1.1 CONTEXTUALIZAÇÃO 15
1.2 OBJETIVO GERAL 17
1.3 OBJETIVOS ESPECÍFICOS 17
1.4 ORGANIZAÇÃO DA MONOGRAFIA 17

2 REFERENCIAL TEÓRICO 19
2.1 CONCEITOS RELEVANTES 19
2.2 TRABALHOS E FERRAMENTAS RELACIONADAS 25
2.2.1 Ferramentas de teste de sistema 25
2.2.1.1 Selenium 25
2.2.1.2 Cypress 26
2.2.1.3 Playwright 26
2.2.1.4 Appium 26
2.2.1.5 Robot Framework 28
2.2.1.6 Katalon 28
2.2.1.7 Testim 29
2.2.1.8 mabl 30
2.2.1.9 Functionize 30
2.2.1.10 TestCafe 31
2.2.2 Arquitetura das ferramentas de teste 31
2.2.3 Ferramentas de inspeção de usabilidade 33
2.2.3.1 Google Lighthouse 33
2.2.3.2 Hotjar 34
2.2.3.3 Microsoft Clarity 34
2.2.3.4 UXCam 35
2.2.3.5 Crazy Egg 35
2.2.3.6 Applitools 36
2.2.3.7 Maze 36
2.2.3.8 Optimal Workshop 37

2.2.4 Tabela comparativa entre as ferramentas e o presente trabalho . . . 37

3 MODELO DO PROJETO . . . 39
3.1 ARQUITETURA . . . 39
3.2 FLUXO DE FUNCIONALIDADES . . . 41
3.3 MODELAGEM DO BANCO DE DADOS . . . 42
3.4 TECNOLOGIAS . . . 44
3.4.1 Python . . . 44
3.4.2 Flask . . . 44
3.4.3 SQLite . . . 44
3.4.4 Playwright . . . 44
3.4.5 API do ChatGPT . . . 45

4 METODOLOGIA DE TRABALHO . . . 46
4.1 PARADIGMA DA PESQUISA . . . 46
4.2 ELICITAÇÃO E ANÁLISE DE REQUISITOS . . . 46
4.2.1 Pesquisa bibliográfica . . . 46
4.2.2 Brainstorming . . . 46
4.2.3 Domain-Driven Design . . . 50
4.2.4 Linguagem do problema . . . 52
4.2.5 Descrição e gerenciamento dos requisitos . . . 56
4.2.6 Fluxo de trabalho . . . 57

5 RESULTADOS . . . 59
5.1 BASE DE CONHECIMENTO DE HEURÍSTICAS DE USABILIDADE . . . 59
5.2 BASE DE CONHECIMENTO DE INTERFACES DE REFERÊNCIA . . . 60
5.3 AGENTE INTELIGENTE DE INSPEÇÃO DE USABILIDADE . . . 61
5.4 AGENTE INTELIGENTE DE TESTE DE SISTEMA . . . 62
5.5 AGENTES DE CONFORMIDADE E O FRAMEWORK AI-TMM . . . 63
5.6 DEMAIS PÁGINAS DA APLICAÇÃO . . . 64

6 CONSIDERAÇÕES FINAIS . . . 66
6.1 TRABALHOS FUTUROS . . . 67
6.1.1 Justificativa para a não realização do estudo comparativo . . . 67

6.1.2 A ausência do estudo comparativo e a completude dos objetivos do trabalho 68

REFERÊNCIAS 69

1 INTRODUÇÃO

"A qualidade de software refere-se ao grau em que um sistema atende às necessidades dos usuários e às expectativas implícitas" [1].

Ao aplicar fundamentos, heurísticas, métricas e técnicas de qualidade de software já presentes na literatura científica no desenvolvimento de novos algoritmos, é possível resolver problemas de forma eficiente, além de promover o processo de melhoria contínua de serviços úteis a um determinado grupo de indivíduos de maneira sustentável (possível de manter a longo prazo).

A não aplicação desses fundamentos e técnicas gera uma série de prejuízos financeiros (ocasionados pela perda de usuários que utilizam o software) que costumam ser causados por problemas como: lentidão, dificuldade em efetuar manutenção de código, funcionalidades com falhas, interfaces pouco intuitivas, infraestrutura sensível, falta de documentação, dentre outros.

1.1 CONTEXTUALIZAÇÃO

Ao construir um sistema web, é necessário aplicar várias técnicas diferentes para garantir a sua qualidade. Na literatura de qualidade de software, há duas técnicas em específico que são descritas como "caras" aos times de desenvolvimento, pois levam muito tempo para serem planejadas e executadas (principalmente), mas possuem grande valor agregado, sendo elas: Teste de Sistema (verifica se ao realizar uma série de passos é possível chegar a um resultado esperado, percorrendo todas as camadas pertinentes de uma aplicação) e Inspeção de Usabilidade (avalia se interfaces possuem elementos que tornam a experiência do usuário melhor através de heurísticas).

Esse tempo elevado possui origem na carência de ferramentas que automatizam as operações de verificação de software incluídas nessas técnicas, fazendo com que sejam aplicadas de forma manual. Além disso, as ferramentas existentes são complexas para profissionais da área de qualidade de software pouco versados em programação (exigem alteração na codificação para alterar o fluxo da automação).

Contudo, apesar do alto valor agregado que essas técnicas carregam, elas costumam ser utilizadas em momentos e partes pontuais do projeto, devido ao alto

investimento de tempo necessário para executá-las, o que implica a redução da cobertura de testes de sistema e o aumento da construção de interfaces que não passam por nenhum processo de inspeção de usabilidade.

Nesse contexto, o presente trabalho busca desenvolver uma plataforma que automatize, com auxílio de modelos de inteligência artificial generativa, o processo de execução de testes de sistema e inspeção de usabilidade.

Dentro da aplicação, as informações serão organizadas em projetos, com o intuito de criar “personas” e requisitos funcionais dispostos de forma ordenada, contendo descrições com seus critérios de aceitação, além da opção de gravar a execução dessa funcionalidade para o fluxo desejado (atendendo aos critérios de aceite contidos na descrição).

Após a gravação, o testador poderá solicitar que a aplicação execute novamente — de forma autônoma — os passos do teste de sistema, levando a ferramenta a gerar um relatório em formato de mensagem de “chat” informando se o sistema verificado atende aos critérios de aceite (passou ou não passou no teste), permitindo também que um retorno em forma de comentário seja efetuado sobre a qualidade da execução do teste e sobre qualquer outra modificação pertinente que altere o fluxo de execução. Todas as vezes que o teste for executado, ele terá como entrada de dados o nome da funcionalidade, a descrição com os critérios de aceite, a gravação com o fluxo correto da operação e o histórico de mensagens do chat. Esses dados permitem que o agente inteligente responsável pelo teste de sistema e pela interação com o modelo de inteligência artificial selecionado injete no “payload” da requisição novas informações a fim de melhorar as respostas fornecidas.

A plataforma também permitirá realizar inspeções de usabilidade tomando como base a gravação do fluxo de execução, as personas criadas, uma base de conhecimento de heurísticas de usabilidade e as execuções do teste de sistema que falharam, gerando como saída um relatório, no mesmo formato de chat, sugerindo melhorias na interface. O testador poderá ainda inserir comentários para ajudar o agente inteligente responsável pela inspeção de usabilidade a fazer melhores solicitações de análise ao modelo selecionado.

Deste modo, a criação dessa plataforma tornará possível realizar inspeções de usabilidade e testes de sistema com mais agilidade, bem como modificá-los quando necessário. A implementação dessas rotinas tem como efeito a melhoria do processo

de garantia da qualidade de software ao submeter sistemas web a quantidades maiores de inspeções de usabilidade e testes de sistema.

1.2 OBJETIVO GERAL

O objetivo geral do trabalho é desenvolver uma ferramenta de apoio à melhoria da qualidade de software utilizando modelos de inteligência artificial (IA) generativa na automatização de casos de teste de sistema e inspeções de usabilidade.

1.3 OBJETIVOS ESPECÍFICOS

- Construir uma base de conhecimento com heurísticas de usabilidade utilizadas para inspecionar aplicações;
- Construir uma base de conhecimento com interfaces de sistemas que atendam a heurísticas de usabilidade;
- Desenvolver agente inteligente integrado com IA que avalie interfaces de acordo com heurísticas selecionadas;
- Desenvolver agente inteligente integrado com IA que replique os passos previamente definidos a fim de realizar o teste de sistema para verificar se é possível chegar a um determinado estado.

1.4 ORGANIZAÇÃO DA MONOGRAFIA

Além deste capítulo introdutório, esta monografia está organizada em mais cinco capítulos, descritos a seguir.

O Capítulo 2 apresenta o referencial teórico do trabalho, incluindo os conceitos relevantes de qualidade de software, usabilidade e inteligência artificial generativa, além de um levantamento de ferramentas e trabalhos relacionados de teste de sistema e de inspeção de usabilidade, com a arquitetura empregada em cada uma delas e uma tabela comparativa entre essas soluções e a proposta deste trabalho.

O Capítulo 3 apresenta o modelo do projeto da aplicação desenvolvida, incluindo sua arquitetura em camadas, o fluxo de funcionalidades, a modelagem do banco de dados e as tecnologias selecionadas para a sua construção.

O Capítulo 4 descreve a metodologia de trabalho, apresentando o paradigma de pesquisa adotado e o processo de elicitação e análise de requisitos utilizado para chegar à solução proposta frente ao problema apresentado neste capítulo.

O Capítulo 5 apresenta os resultados obtidos com o desenvolvimento e a avaliação da ferramenta proposta.

Por fim, o Capítulo 6 traz as considerações finais do trabalho, retomando os objetivos propostos, sintetizando as contribuições alcançadas e apontando sugestões de trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo serão abordados os principais conceitos que norteiam os aspectos de qualidade de software e as ferramentas conhecidas na área de qualidade de software. A compreensão desses conceitos é de suma importância para entendimento da abordagem escolhida para desenvolvimento da aplicação.

2.1 CONCEITOS RELEVANTES

Essa seção aborda conceitos norteadores e de importante compreensão para entendimento das tomadas de decisão acerca de metodologias, teorias, modelos e boas práticas de engenharia de software adotadas neste trabalho:

- **Processo de Software**: Um processo é um conjunto de atividades, ações e tarefas realizadas na criação de algum artefato. Sommerville afirma que embora existam muitos processos diferentes, todos eles incluem de alguma forma as atividades de: Especificação (definição das funcionalidades e restrições do software), Desenvolvimento (construção do software atendendo às especificações), Validação (avaliação por parte do cliente para julgar se o software está adequado ao uso) e Evolução (o software deve evoluir para atender às necessidades mutáveis dos clientes) [1].

- **Modelo Iterativo e Incremental**: É um modelo de processo que se baseia na ideia de realizar as entregas de um projeto de software em módulos utilizáveis para o usuário. É chamado de iterativo por entregar algo funcional ao cliente e aprimorar o produto a cada rodada, e de incremental por adicionar novas funcionalidades a cada ciclo [2].

- **Modelo Prototipação**: O paradigma da prototipação consiste em, através de comunicação com os clientes, elaborar um “projeto rápido”, um protótipo do sistema que permita abrir uma discussão sobre requisitos que precisam ser modificados ou melhor abordados, refiná-los e realizar uma entrega [3]. Esse modelo é aplicado para capturar requisitos ocultos ou não explicitados durante a comunicação

dos engenheiros de software com os clientes e pode ser aplicado como uma técnica dentro dos outros modelos de processo, sob condições avaliadas pelo engenheiro de software.

- **Metodologias Ágeis**: É um conjunto de modelos de processo que se propõem a ser adaptáveis às mudanças de requisitos que ocorrem durante o desenvolvimento de um software, de forma incremental; eles se baseiam na ideia de entregar software em ciclos e obter *feedback* dos clientes o mais rápido possível para que as equipes, que são auto-organizáveis, possam reagir às necessidades não atendidas e discutir os próximos passos [4]. Essa capacidade de adaptação constante é uma das características centrais da engenharia de software praticada atualmente [5].

- **Scrum**: O Scrum é uma abordagem ágil para o desenvolvimento de produtos e serviços inovativos baseada em equipes auto-organizáveis, entregas incrementais denominadas sprints, seguindo um conjunto de trabalho a ser realizado definido em uma lista de tarefas priorizadas e estimadas chamada backlog, comunicação com o cliente, feedback e reações rápidas [6].

- **Qualidade de Software**: Qualidade de software pode ser definida como uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam. Dessa forma, refere-se a um grau de atendimento aos requisitos especificados e às necessidades de utilização dos usuários [3].

- **Teste de Software**: É a atividade do processo de software que verifica e valida a adequação ao uso através de uma série de técnicas [3].

- **Teste de Sistema**: São testes que simulam o uso de um sistema por um usuário real. Testes de sistema também são chamados de testes ponta-a-ponta (*end-to-end*) ou então testes de interfaces. São os testes mais caros, que demandam maior esforço para implementação e que executam em mais tempo.

- **Paralelismo**: É a execução de mais de uma tarefa, de forma simultânea, através de distribuição de carga entre os núcleos (*cores*) do processador. Quanto mais núcleos, mais as tarefas podem ser distribuídas [7].

- **Desenvolvimento Orientado a Comportamento (*Behavior-Driven Development* – BDD)**: É um paradigma que visa descrever comportamentos (regras de negócio principalmente) de forma que todas as partes envolvidas sejam capazes de entender, estabelecendo uma linguagem única e estimulando o desenvolvimento colaborativo do software e eliminando a ambiguidade que costuma causar desencontros de conceitos e expectativas durante as entregas [8]. Essa técnica também tem sido aplicada como apoio ao levantamento de requisitos de software [9].

- **Design Orientado a Domínio (*Domain-Driven Design* – DDD)**: O DDD é um conjunto de ferramentas que auxiliam na concepção e implementação de softwares que oferecem alto valor, tanto estratégica quanto taticamente. Essas ferramentas ajudam as equipes de desenvolvimento a fazer escolhas melhores de design de software de forma competitiva e a tomar boas decisões de integração para as empresas [10].

- **Padrões de Projeto**: Padrões de projeto são descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto num contexto particular [11].

- **Interação Humano-Computador**: A interação humano-computador (IHC) é um campo multidisciplinar que estuda como as pessoas interagem com os computadores e sistemas digitais, visando melhorar a experiência do usuário e a eficácia das tecnologias [12].

- **Usabilidade**: Segundo a norma ISO/IEC 9126, que se refere à qualidade de software, a usabilidade é um conjunto de atributos de software relacionados ao esforço necessário para seu uso e para o julgamento individual por determinados usuários [13]. Esse conceito também é discutido sob a ótica de dispositivos móveis, com heurísticas e diretrizes específicas para o design de suas interfaces [14].

- **Inspeção de Usabilidade**: É um método que visa encontrar problemas de usabilidade nas interfaces do software submetido através da análise de especialistas em usabilidade ou profissionais de desenvolvimento de software. Esse método normalmente costuma utilizar heurísticas e possui um baixo custo em relação a

outras técnicas existentes, mas ainda assim demanda um grande esforço para as equipes de desenvolvimento [15].

- **Heurísticas de Nielsen**: As heurísticas de Nielsen são regras utilizadas para identificar erros de design que comprometam a usabilidade de um sistema. Acredita-se que de três a cinco avaliadores especialistas sejam capazes de identificar de 75% a 80% dos problemas de usabilidade de interfaces computacionais [16]. As heurísticas estão divididas nas seguintes categorias:

    1. **Visibilidade do estado do sistema**: o design deve sempre manter os usuários informados sobre o que está acontecendo, por meio de *feedback* apropriado dentro de um prazo razoável;
    2. **Correspondência entre o sistema e o mundo real**: o design deve falar a linguagem dos usuários, usando palavras, frases e conceitos familiares ao usuário, em vez de jargões internos, e seguindo as convenções do mundo real, apresentando as informações em uma ordem natural e lógica;
    3. **Controle e liberdade do usuário**: os usuários frequentemente realizam ações por engano. Eles precisam de uma "saída de emergência" claramente sinalizada para interromper a ação indesejada sem ter que passar por um processo demorado;
    4. **Consistência e padrões**: os usuários não devem ter dúvidas se palavras, situações ou ações diferentes significam a mesma coisa. O sistema deve seguir as convenções da plataforma e do setor;
    5. **Prevenção de erros**: boas mensagens de erro são importantes, mas os melhores projetos previnem cuidadosamente que os problemas ocorram em primeiro lugar, eliminando as condições propensas a erros ou apresentando aos usuários uma opção de confirmação antes que eles executem a ação;
    6. **Reconhecimento em vez de recordação**: a carga de memória do usuário deve ser minimizada, tornando os elementos, ações e opções visíveis. O usuário não deve precisar memorizar informações de uma parte da interface para outra;

7. **Flexibilidade e eficiência de uso**: atalhos, ocultos para usuários iniciantes, podem acelerar a interação para usuários experientes, permitindo que o design atenda tanto a usuários inexperientes quanto experientes;
8. **Design estético e minimalista**: as interfaces não devem conter informações irrelevantes ou raramente necessárias, já que cada unidade extra de informação compete com as unidades de informação relevantes e diminui sua visibilidade relativa;
9. **Ajudar os usuários a reconhecer, diagnosticar e recuperar-se de erros**: as mensagens de erro devem ser expressas em linguagem clara (sem códigos de erro), indicando precisamente o problema e sugerindo uma solução construtiva;
10. **Ajuda e documentação**: o ideal é que o sistema não precise de explicações adicionais; no entanto, pode ser necessário fornecer documentação para ajudar os usuários a entender como concluir suas tarefas.

- **Versionamento de Código**: Um Sistema de Controle de Versão é um software que gerencia e rastreia as alterações em arquivos ao longo do tempo. Ele permite que se reverta um arquivo ou um projeto inteiro para um estado anterior, compare mudanças, identifique quem modificou o código e quando, entre outras funcionalidades [17].

- **Linguagem de Modelagem Unificada (*Unified Modeling Language* – UML)**: A UML é uma notação gráfica para modelagem de software que surgiu a partir de um esforço para unificar as notações gráficas que surgiram no final da década de 80. A linguagem define um conjunto de diagramas para documentar e ajudar no design de sistemas de software, particularmente sistemas orientados a objetos [3].

- **Modelo e Notação de Processos de Negócio (*Business Process Model and Notation* – BPMN)**: *Business Process Management* (BPM), ou Gestão de Processos de Negócio, é uma abordagem aplicada para gerenciar os negócios da empresa de modo integrado. Para dar suporte às iniciativas de BPM, foram criadas várias linguagens para modelagem, dentre elas a BPMN, que é capaz de descrever processos e seus agentes [18].

- **Integração Contínua/Entrega Contínua (*Continuous Integration/Continuous Delivery* – CI/CD)**: A CI permite que os programadores compartilhem as suas contribuições constantemente, eliminando a necessidade de esperar que o seu trabalho seja finalizado, enquanto a CD permite a entrega contínua de software, automatizando processos de implantação [19].

- **Agentes Inteligentes Artificiais**: Um agente autônomo é um sistema situado dentro e participante de um ambiente e que sente (através de sensores) e age sobre ele (por meio de atuadores), ao longo do tempo, no exercício da sua própria agenda (metas e tarefas), afetando assim o que ele sentirá no futuro (planejamento e ação) [20].

- **AI Trust Maturity Model (AI-TMM)**: É um modelo criado para ajudar organizações a avaliar e desenvolver confiança em sistemas de Inteligência Artificial, funcionando como um *framework* de governança e maturidade que permite analisar o nível de preparação de uma organização para desenvolver, implantar e operar IA de forma responsável, segura e confiável [21]. Ele tem como objetivo promover nos sistemas:

  1. **Governança**: estabelece políticas e responsabilidades para o uso de IA bem definidas;
  2. **Transparência e explicabilidade**: avalia se os sistemas de IA podem ser compreendidos;
  3. **Justiça e mitigação de viés**: verifica se o sistema evita discriminação;
  4. **Segurança e robustez**: avalia se o sistema é resiliente a falhas e ataques;
  5. **Privacidade e proteção de dados**: verifica se os dados utilizados respeitam a privacidade.

- **Inteligência Artificial Generativa**: A inteligência artificial generativa constitui um paradigma dentro do campo mais amplo do aprendizado de máquina, caracterizado pela capacidade de criar conteúdo novo e contextualmente relevante a partir de padrões aprendidos em dados de treinamento. A sua relevância pode ser observada em *chats* que ganharam popularidade, como o ChatGPT, da empresa OpenAI, que disponibiliza uma série de *Large Language Models* (LLMs)

– modelos de inteligência artificial capazes de compilar um número grande de informações e apresentar respostas mais concisas ao cliente [22]. Modelos dessa natureza também vêm sendo aplicados na classificação automática de requisitos de software [23].

- **Engenharia de Prompt**: A engenharia de *prompt* se ocupa da preparação de estratégias de códigos ou estímulos que conduzem os modelos de Inteligência Artificial (IA), gerando respostas características, concisas e proeminentes, de maneira a utilizar os recursos que sustentam o modelo de forma eficiente [24].
- **Geração Aumentada por Recuperação (*Retrieval-Augmented Generation* – RAG)**: A abordagem RAG combina recuperação de informações e geração de texto, permitindo que os modelos tenham acesso a bases de conhecimento e utilizem esses dados em combinação com *prompts* estruturados para fornecer melhores respostas aos usuários [25].

2.2 TRABALHOS E FERRAMENTAS RELACIONADAS

Esta seção aborda trabalhos e ferramentas que cumprem alguns dos objetivos propostos no presente trabalho e apresenta um comparativo entre elas.

2.2.1 Ferramentas de teste de sistema

2.2.1.1 Selenium

Selenium é um projeto de código livre, criado por Jason Huggins em 2004 enquanto trabalhava na consultoria ThoughtWorks em Chicago [26]. Ele abrange um conjunto de ferramentas e bibliotecas que emulam a interação do usuário com o navegador e permitem a automação de operações em navegadores web [27], suportando uma série de linguagens como Python, Java, Javascript, C#, Ruby, dentre outras [28].

O projeto é dividido em alguns módulos:

- **Selenium WebDriver**: fornece APIs para interagir com navegadores;
- **Selenium IDE**: ambiente de desenvolvimento integrado para *scripts* que permite a gravação e reprodução da interação dos usuários com o navegador;

- **Selenium Grid**: ferramenta que permite executar os testes em diferentes navegadores e máquinas utilizando paralelismo para otimizar o processo.

2.2.1.2 Cypress

Cypress é uma ferramenta criada por Brian Mann por volta de 2014-2015, com o objetivo de solucionar as limitações de automação de testes baseados em Selenium [29]. Ela simplifica o processo de desenvolvimento, execução e acompanhamento de testes de software em ambientes web através de uma API feita em Javascript, destacando-se pela sua curva de aprendizado baixa, execução de testes no mesmo ambiente do navegador utilizado pelo desenvolvedor e visualização detalhada de depuração [30].

2.2.1.3 Playwright

Playwright é uma ferramenta para automação de testes de software, introduzida pela Microsoft, que possui API para Javascript, C#, Python e Java e compatibilidade com uma série de navegadores [31]. Essa biblioteca possui recursos únicos, destacados a seguir [32]:

- Captura de execução de testes, em que o usuário realiza operações na página web e essas ações são transformadas em linhas de código que irão configurar os casos de teste;
- Revisão assistida de testes, em que, em caso de falha, é possível obter informações como o vídeo de execução, logs, estado da página, dentre outras;
- Teste *mobile*, onde os casos de teste são executados em ambiente *mobile* simulado.

2.2.1.4 Appium

É um projeto *open source*, mantido pela OpenJS Foundation e criado pela empresa Sauce Labs em 2012, que se propõe a facilitar o desenvolvimento de automações que interagem com a UI de aplicações em diversas plataformas, como: *mobile* (iOS,

Android, Tizen), *browser* (Chrome, Firefox, Safari), *desktop* (macOS, Windows), TV (Roku, tvOS, Android TV, Samsung) e outros [33]. A arquitetura do Appium permite o desenvolvimento de testes em diferentes linguagens (Java, Python, Ruby, JS e outras), já que o teste é recebido pelo cliente WebDriver (padrão criado pelo Selenium), que envia os comandos do teste para um servidor Appium responsável por transpilar o teste escrito na linguagem escolhida para comandos na linguagem da plataforma alvo, chamar os *drivers* e executar os testes nessa plataforma [34]. Esse projeto se destaca pelas seguintes características [35]:

- Interação com interface gráfica: interage com elementos da UI simulando o usuário real;
- Localização de elementos na interface: permite capturar elementos utilizando diferentes abordagens;
- Execução multiplataforma: realiza testes em diferentes plataformas;
- Versatilidade com linguagens de programação: o *script* de teste pode ser construído utilizando diferentes linguagens;
- Controle de estado de aplicativos: no caso de aplicativos *mobile*, é possível controlar o seu ciclo de vida;
- Interação com recursos de *hardware*: consegue utilizar recursos como localização, GPS, notificações, bateria e outros;
- Captura de evidências: consegue capturar *prints* de telas, registrar *logs* e coletar informações de *hardware* da execução do teste;
- Integração com ferramentas de entrega e integração contínuas: integração com *scripts* de validação e implantação de aplicações;
- Paralelismo para otimização de *performance*: utiliza paralelismo para executar vários testes utilizando os recursos do *hardware* de forma eficiente.

2.2.1.5 Robot Framework

Robot Framework é um *framework open source* para automação robótica de processos (RPA), mantido pela Robot Framework Foundation, desenvolvido pelos engenheiros da Nokia em 2005 e tornado *open source* em 2008 [36]. Ele se diferencia por seu paradigma peculiar, desenvolvimento de teste orientado a palavras-chave (criado pelo pesquisador finlandês Pekka Klärck enquanto fazia sua tese de mestrado), no qual é possível descrever testes em formato de linguagem natural, ou seja, em alto nível [37]. A ferramenta foi desenvolvida em Python, mas permite a criação de *scripts* em outras linguagens, como .NET e Java [38]. Ele possui uma série de recursos úteis:

- Linguagem tabular: os testes são escritos em tabelas e em linguagem natural utilizando palavras-chave para descrever as ações a serem realizadas;
- Palavras-chave: são unidades de execução que habilitam ações dentro do *script* de teste e facilitam a criação modular dos casos;
- Integração com desenvolvimento orientado a comportamento (BDD): habilita a criação de casos de teste utilizando linguagem Gherkin;
- Extensibilidade: é possível estender o comportamento do *framework* através da criação de módulos personalizados e utilização de bibliotecas;
- Multiplataforma: é compatível com Windows, Linux e macOS;
- Captura de evidências: registra *log* e emite relatórios de execução.

2.2.1.6 Katalon

Katalon é uma plataforma de automação de testes com licença proprietária, criada em 2016 pela empresa Katalon, para atender testadores e desenvolvedores auxiliando no desenvolvimento de testes em plataformas web, *mobile* e *desktop* através de *scripts* construídos utilizando a linguagem Groovy e estrutura amparada pelo Selenium e Appium [39]. Ele possui funcionalidades como:

- Criação de testes sem codificação: permite que sejam desenvolvidos testes de forma visual e *low-code*;

- Integração com outras plataformas: permite a integração com ferramentas de desenvolvimento e CI/CD, como Jenkins, Git e Jira;
- Utilização de inteligência artificial: aplica IA na criação de casos de teste;
- Captura de execução: permite a gravação de execução dos testes;
- Testes baseados em palavras-chave: utiliza linguagem natural para estruturar os *scripts*;
- Paralelismo: utiliza paralelismo para otimizar a execução dos casos de teste.

2.2.1.7 Testim

Testim é uma ferramenta colaborativa, criada em 2014 e mantida atualmente pela empresa Tricentis, que utiliza IA na criação, execução e manutenção dos testes para otimização de processos [40]. Seus principais recursos são:

- Criação de casos sem codificação: é possível criar testes através de linguagem natural e transformar em passos na linguagem Javascript;
- Multiplataforma: executa testes em ambientes web e *mobile*;
- Paralelismo: utiliza paralelismo para otimizar as execuções de teste;
- Colaboração: a ferramenta permite manter os grupos de casos em equipe;
- Integração com ferramentas: integra-se com processos de integração/implantação contínua (CI/CD);
- Análise avançada: através da IA, a plataforma oferece um relatório detalhado sobre aspectos de execução;
- Adaptabilidade inteligente: a plataforma, por meio de IA, consegue editar os casos de teste automaticamente quando algo muda na interface da aplicação do usuário.

2.2.1.8 mabl

mabl é uma ferramenta de testes baseada em IA, criada em 2017 por Dan Belcher e Izzy Azeri, antigo engenheiro do Google [41], com a ideia de facilitar a manutenção do ciclo de vida de testes através de *low-code* [42]. Ela também é apontada como uma alternativa moderna a ferramentas tradicionais como o Selenium [43], sendo avaliada positivamente por seus usuários [44]. Essa ferramenta possui as seguintes funcionalidades:

- Elaboração visual de casos de teste: permite criar casos de teste de forma *low-code*;
- Aprendizado de interface: a aplicação aprende a compreender o contexto do software analisado;
- Autocorreção automática: a plataforma refaz os testes quando o software analisado já não segue o fluxo desenhado inicialmente;
- Gravação de teste: permite construir testes a partir de gravações do navegador;
- Integração com ferramentas externas: permite a conexão com plataformas de integração e entrega contínuas de software;
- Geração de relatórios: emite relatórios de execução dos casos de teste.

2.2.1.9 Functionize

Functionize é uma plataforma de automação de testes, criada em 2014 e baseada em inteligência artificial, voltada para automatizar testes funcionais e de sistema principalmente em aplicações web, com foco em reduzir a necessidade de programação e manutenção manual [45]. A ferramenta é bem avaliada por seus usuários em plataformas especializadas de análise de software corporativo [46]. Possui os seguintes recursos:

- Gravação automática de fluxos de teste: permite elaborar os casos de teste por meio de gravações;

- Criação de testes usando linguagem natural: utiliza linguagem natural para construir os testes;
- *Auto-healing* de testes: refatora o caso de teste quando há mudanças na interface da aplicação;
- Integração com *pipelines* CI/CD: permite fazer ligações com ferramentas externas.

2.2.1.10 TestCafe

É uma ferramenta *open source* desenvolvida para automatizar testes de software que interagem com a UI de aplicações. Criada pela empresa DevExpress em 2016, a ferramenta tem como diferencial a simplificação de comandos que seriam significativamente maiores em outras ferramentas, como o Selenium. Com essa ferramenta, é possível criar *scripts* em TypeScript ou Javascript e executar os testes sem *webdriver* [47]. Suas funcionalidades incluem:

- Execução sem *webdriver*: os comandos são injetados dentro do navegador via *proxy*;
- Execução *Headless*: funciona sem abrir o navegador visualmente;
- *Multibrowser*: é compatível com uma variedade de navegadores;
- Paralelismo: executa os *scripts* com mais eficiência ao utilizar melhor o *hardware* do computador;
- Permite gravar a execução dos testes: após a execução dos testes, é possível coletar os registros.

2.2.2 Arquitetura das ferramentas de teste

As ferramentas abordadas trazem diferentes perspectivas sobre como interagir com o navegador para executar os casos de teste. Desse modo, é necessário entender as diferenças entre as arquiteturas das soluções para além das funcionalidades. Nesta seção, serão abordadas as principais arquiteturas identificadas:

1. **Arquitetura WebDriver**: o navegador é controlado externamente através de um *Driver* intermediário, no qual o *script* é enviado para o cliente WebDriver, um servidor HTTP que processa a informação, chama o *Driver* do navegador alvo e executa os comandos. Essa arquitetura traz os seguintes benefícios e problemas:

   Benefícios:

   - Compatibilidade entre navegadores;
   - Padronização na indústria;
   - Facilita a execução de testes remotos.

   Problemas:

   - Número elevado de requisições para executar os comandos no navegador, já que cada instrução é uma requisição;
   - Depende de *drivers* específicos para cada navegador;
   - Dificuldade em capturar determinados eventos internos do navegador.

2. **Arquitetura Proxy com Injeção de Script**: um *proxy* é colocado entre o navegador e a aplicação web, modificando as páginas e inserindo *scripts* diretamente dentro do HTML e da árvore de elementos da página (DOM). Essa arquitetura traz os seguintes benefícios e problemas:

   Benefícios:

   - Não depende de *driver*, pois o *script* funciona em qualquer navegador que permita a injeção de Javascript;
   - Facilidade em inspecionar e manipular a árvore de elementos da página.

   Problemas:

   - *Scripts* injetados podem causar efeitos colaterais indesejados;
   - Pode haver incompatibilidade com navegadores modernos que possuem proteção contra injeção de *scripts*.

3. **Arquitetura In-Browser**: o *script* de teste é carregado dentro do navegador e executa os testes no mesmo contexto dele, podendo interagir diretamente com a DOM, eventos e API. Essa arquitetura traz os seguintes benefícios e problemas:

Benefícios:

- Captura eventos de forma mais precisa;
- Maior velocidade, pois não passa por intermediários;
- Menor dependência de *drivers*.

Problemas:

- Limitado a navegadores compatíveis com o *script*;
- Menor padronização entre os navegadores, pois a abstração que a camada intermediária fazia já não está mais presente;
- Dificuldade em fazer testes em múltiplos navegadores ao mesmo tempo.

4. **Arquitetura Browser Protocol**: o navegador expõe uma API de baixo nível, denominada *Browser Protocol*, utilizada pelo *framework* de teste para interação com a página web sem *drivers* ou injeção de *scripts*, na qual os comandos são enviados pelo *framework* e a API retorna o resultado da execução. Essa arquitetura traz os seguintes benefícios e problemas:

   Benefícios:

   - Controle preciso do navegador, já que consome uma API interna;
   - Mais fluidez, já que não passa por intermediários nem injeta *script*.

   Problemas:

   - Menor portabilidade, já que cada navegador possui sua própria API;
   - Atualizações nos navegadores podem exigir alteração dos casos de teste.

2.2.3 Ferramentas de inspeção de usabilidade

2.2.3.1 Google Lighthouse

O Google Lighthouse é uma ferramenta criada com NodeJS pela Google para fazer análise de desempenho de páginas web, focada em usabilidade e monitoramento de recursos de *hardware* para buscar pontos de otimização nos *sites* [48]. Entre os principais recursos, destacam-se:

- Análise de otimização de mecanismo de busca (SEO): aponta melhorias que podem ser feitas para ajudar a página a ser encontrada mais rapidamente pelo navegador e aparecer nos resultados principais;
- Acessibilidade: monitora cobertura de traduções e elementos que auxiliam pessoas com deficiência a entender recursos das páginas web;
- Desempenho: observa utilização e tamanho de recursos web;
- Análise de boas práticas: mostra que técnicas podem ser aplicadas para melhorar a renderização de imagens, eliminar vulnerabilidades e prevenir erros do navegador.

2.2.3.2 Hotjar

O Hotjar é uma ferramenta de análise comportamental e *feedback* do usuário, criada por David Darmanin, que opera através de injeção de código Javascript simples nas páginas alvo e proporciona ao mantenedor da página análises utilizadas para melhorar a experiência dos usuários [49]. Dentre as suas funcionalidades, destacam-se:

- Análise de *clique* de usuários;
- Análise de movimento de barra de rolagem;
- *Replay* de sessões individuais de uso da página.

2.2.3.3 Microsoft Clarity

O Microsoft Clarity é uma ferramenta de análise comportamental utilizada para descobrir pontos de atrito na utilização dos *sites* pelos usuários [50]. Entre as suas funcionalidades, estão:

- Acesso gratuito à plataforma da ferramenta;
- Mapa de calor para ver onde os usuários clicam e rolam;
- Gravações de sessões de uso individual nas interfaces;

- Apontamento de problemas de UX com base em heurísticas.

2.2.3.4 UXCam

A UXCam é uma plataforma de análise de produto com um assistente de IA integrado, voltada para aplicativos *mobile* e *sites* [51]. Como funcionalidades, destacam-se:

- Gravações de sessão para entender o uso dos clientes;
- Mapa de calor para identificar áreas de maior interação com o usuário;
- Análise de funil e retenção para identificar em qual etapa os usuários deixam de realizar uma determinada operação;
- Detecção automática de erros para reportar problemas de desempenho;
- Agente com IA que analisa anomalias do sistema.

2.2.3.5 Crazy Egg

O Crazy Egg é uma ferramenta de análise e otimização de *sites* que ajuda a entender como os usuários interagem com uma página, mostrando onde clicam, até onde rolam a tela e quais partes recebem mais atenção [52]. Com essas informações, é possível identificar problemas no site e propor melhorias que tornem a navegação mais fácil e eficiente [53]. Entre as suas funcionalidades:

- Mapas de calor, que mostram as áreas mais acessadas ou clicadas da página;
- Gravações de sessão, que permitem acompanhar o comportamento dos visitantes durante a navegação;
- Testes A/B, usados para comparar versões diferentes de uma página;
- Relatórios de rolagem, que indicam até que ponto os usuários descem em uma página.

2.2.3.6 Applitools

A Applitools é uma plataforma voltada para testes automatizados de software que usa Inteligência Artificial Visual para analisar interfaces de *sites*, aplicativos e documentos, identificando diferenças visuais, falhas de *layout* e problemas que podem afetar a experiência do usuário [54]. Entre os recursos oferecidos:

- Testes visuais automatizados e testes funcionais;
- Testes de API;
- Testes de acessibilidade e validação em diferentes navegadores e dispositivos;
- Detecção automática de problemas visuais;
- Criação e execução de testes com recursos de IA.

2.2.3.7 Maze

A ferramenta Maze permite validar o comportamento real dos usuários em um *site* ativo, gerando informações como mapas de calor, caminhos percorridos e gravações de sessão [55]. Entre as suas funcionalidades:

- Teste em *site* real: permite testar o comportamento dos usuários em um ambiente verdadeiro, não apenas em protótipos;
- Definição de caminhos de sucesso: possibilita indicar qual seria o caminho ideal que o usuário deveria seguir para concluir uma tarefa;
- Análise de caminhos percorridos: mostra se os participantes seguiram o caminho esperado ou se usaram rotas alternativas;
- Mapas de calor e gravações de sessão: ajudam a visualizar e acompanhar a navegação dos participantes;
- Testes de funil, pesquisas e recrutamento de participantes: ajudam a validar etapas importantes e coletar percepções dos usuários.

2.2.3.8 Optimal Workshop

A Optimal Workshop é uma plataforma voltada para pesquisa de UX, testes de *design* e coleta de *insights* sobre a experiência dos usuários em produtos digitais [56]. Entre os recursos fornecidos:

- *Card sorting*: ajuda a entender como os usuários agrupam informações e categorias;
- *Tree testing*: testa se a estrutura de navegação de um sistema é fácil de entender;
- *First-click testing*: analisa onde o usuário clica primeiro ao tentar realizar uma tarefa;
- *Prototype testing* e *live site testing*: permitem testar protótipos e sites reais, sem precisar mexer no código;
- *Surveys* e *interviews*: permitem coletar opiniões e organizar entrevistas com usuários;
- Análise com IA: usa recursos de inteligência artificial para identificar padrões, temas e *insights* nos dados coletados.

2.2.4 Tabela comparativa entre as ferramentas e o presente trabalho

A Tabela 2 resume as principais diferenças entre as ferramentas relacionadas e a proposta apresentada neste trabalho, considerando os critérios descritos na Tabela 1.

Tabela 1 – Critérios utilizados na comparação entre as ferramentas.

Critério	**Descrição**
C1	Permite gravar um fluxo de execução
C2	Permite criar um teste de sistema e replicá-lo posteriormente
C3	O teste de sistema é capaz de identificar/adaptar-se à alteração da interface sem envolvimento do desenvolvedor
C4	Realiza inspeções de usabilidade em interfaces web
C5	Permite inserção de personas para direcionar ajustes em interfaces web
C6	Utiliza IA em alguma operação de suporte

Fonte: Elaborado pelo autor.

Tabela 2 – Tabela comparativa entre as ferramentas e o presente trabalho.

Ferramenta	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**
Selenium	Sim	Sim	Não	Não	Não	Não
Cypress	Não	Sim	Não	Não	Não	Não
Playwright	Sim	Sim	Não	Não	Não	Não
Appium	Não	Sim	Não	Não	Não	Não
Robot Framework	Não	Sim	Não	Não	Não	Não
Katalon	Sim	Sim	Parcial	Não	Não	Sim
Testim	Parcial	Sim	Sim	Não	Não	Sim
mabl	Sim	Sim	Sim	Não	Não	Sim
Functionize	Sim	Sim	Sim	Não	Não	Sim
TestCafe	Parcial	Sim	Não	Não	Não	Não
Google Lighthouse	Não	Não	Não	Sim	Não	Não
Hotjar	Sim	Não	Não	Parcial	Não	Não
Microsoft Clarity	Sim	Não	Não	Parcial	Não	Parcial
UXCam	Sim	Não	Não	Parcial	Não	Sim
Crazy Egg	Sim	Não	Não	Parcial	Não	Não
Applitools	Parcial	Sim	Sim	Parcial	Não	Sim
Maze	Sim	Parcial	Não	Sim	Não	Não
Optimal Workshop	Parcial	Não	Não	Sim	Não	Sim
Presente Trabalho	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**

Fonte: Elaborado pelo autor.

3 MODELO DO PROJETO

O modelo do projeto foi construído a partir do Domain-Driven Design, detalhado na Seção 4.2, que resultou em três domínios: Projetos (suporte), Teste de Sistema (*core*) e Inspeção de Usabilidade (*core*). Este capítulo descreve a arquitetura de software resultante, o fluxo de funcionalidades da aplicação, a modelagem do banco de dados e as tecnologias empregadas na implementação.

3.1 ARQUITETURA

A aplicação foi implementada como um monólito modular organizado em camadas, seguindo os limites dos contextos delimitados definidos pelo Domain-Driven Design, em vez de uma arquitetura de microsserviços — opção que reduz a complexidade operacional (implantação, comunicação entre serviços e consistência de dados) sem abrir mão da separação de responsabilidades entre domínios [57]. A Figura 1 ilustra as camadas da aplicação e suas dependências.

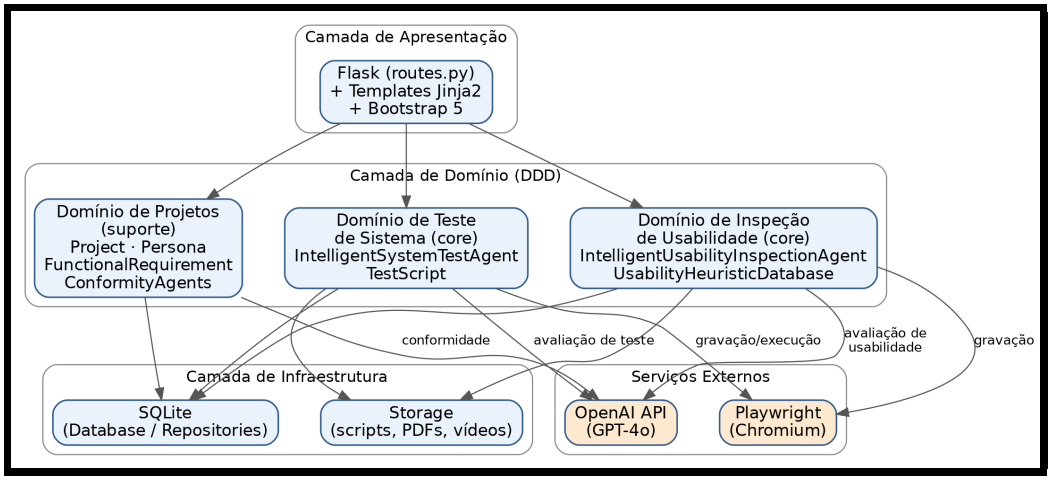

Figura 1 – Arquitetura em camadas da aplicação.

Fonte: Elaborado pelo autor.

A **camada de apresentação** é responsável por toda a interação com o usuário: o Flask expõe as rotas HTTP (`routes.py`) e renderiza os templates Jinja2, estilizados com Bootstrap 5, organizados em uma barra lateral de navegação fixa e um conteúdo

principal por página (Dashboard, Projetos, Personas, Requisitos, Inspeção, Teste de Sistema e Configurações), conforme ilustrado na Figura 2.

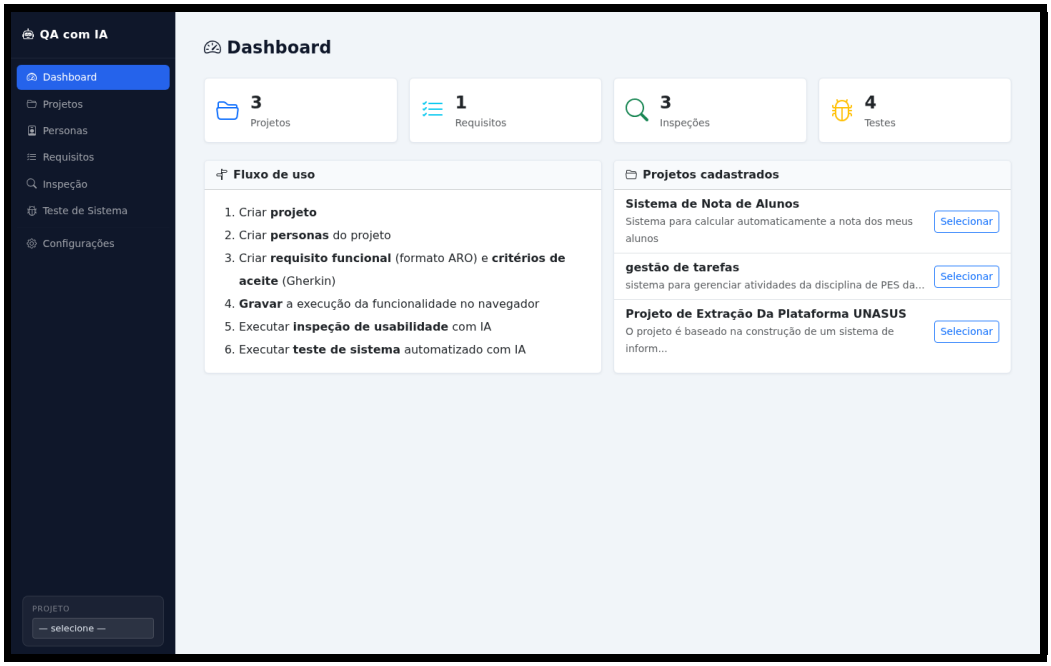

Figura 2 – Organização das páginas da aplicação: barra lateral de navegação e Dashboard.

Fonte: Elaborado pelo autor.

A **camada de domínio** concentra as regras de negócio dentro de cada contexto delimitado (`Model/Projects`, `Model/SystemTest` e `Model/UsabilityInspection`), contendo entidades (*Project*, *Persona*, *FunctionalRequirement*, *TestScript*, *Functionality*), serviços de aplicação (por exemplo, *CreateProjectService*, *ExecuteSystemTestService*, *ExecuteUsabilityInspectionService*) e dois tipos de agentes inteligentes: os **agentes de conformidade**, que validam campos de entidades antes da persistência segundo políticas próprias de cada entidade, e os **agentes inteligentes de domínio** (*IntelligentSystemTestAgent* e *IntelligentUsabilityInspectionAgent*), que se integram com o modelo GPT-4o da OpenAI para avaliar execuções gravadas. Todos os agentes seguem o framework AI-TMM (Seção 4.2), com políticas próprias de *system prompt* configuráveis pela interface de Configurações.

A **camada de infraestrutura** implementa a persistência via SQLite, acessada por meio de repositórios especializados por entidade (`Database/repositories`), e o armazenamento de artefatos gerados durante os testes e inspeções (*scripts* Playwright,

PDFs de gravação e vídeos *webm*) em diretórios próprios por requisito, dentro de `Storage/`.

Por fim, a aplicação depende de dois **serviços externos**: a API da OpenAI, utilizada por todos os agentes inteligentes, e o Playwright, utilizado tanto para gravar a execução de funcionalidades (via `playwright codegen`) quanto para reexecutar os *scripts* gerados de forma automatizada.

3.2 FLUXO DE FUNCIONALIDADES

O uso da aplicação segue um fluxo sequencial de seis etapas, ilustrado na Figura 3, no qual cada etapa depende dos artefatos criados na etapa anterior.

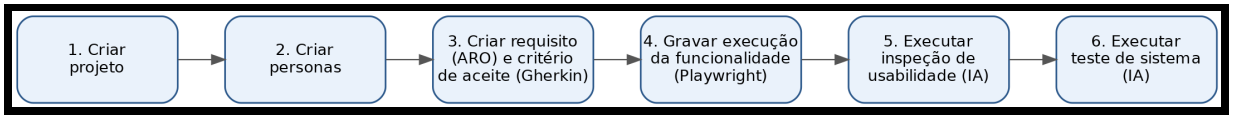

Figura 3 – Fluxo de funcionalidades da aplicação.

Fonte: Elaborado pelo autor.

Primeiro, o usuário cria um **projeto** (Figura 4), validado por um agente de conformidade antes de ser persistido. Em seguida, cadastra as **personas** do projeto, cada uma com seis campos (oportunidades, atributos-chave, descrição, necessidades e desafios, além do nome). Na terceira etapa, cria **requisitos funcionais** no formato ARO associados a uma ou mais personas, e, para cada requisito, um ou mais **critérios de aceite** no formato Gherkin (*Given/When/Then*) — todos validados por agentes de conformidade próprios.

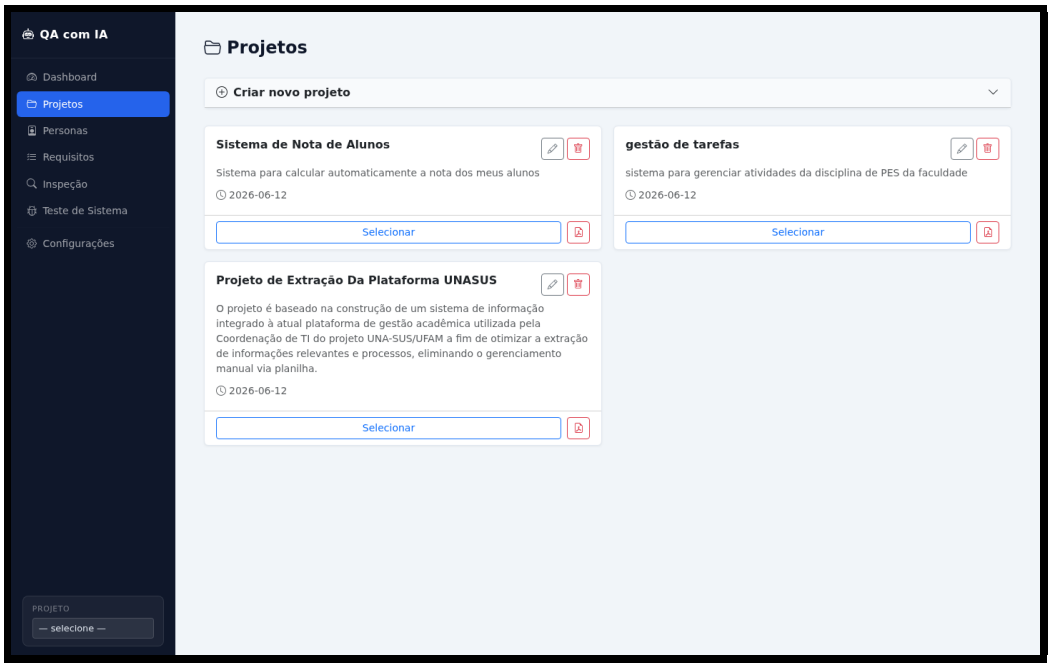

Figura 4 – Página de Projetos: criação, edição, exclusão e seleção do projeto ativo.

Fonte: Elaborado pelo autor.

Com o requisito ativo selecionado, a quarta etapa consiste em **gravar a execução** da funcionalidade correspondente no navegador: o `playwright codegen` captura cliques, preenchimentos e navegações e gera um *script* Python preciso, que é então reproduzido de forma *headless* com gravação de vídeo para produzir um PDF da execução. A partir desses artefatos, o usuário pode executar a **inspeção de usabilidade** (quinta etapa), na qual o agente inteligente analisa o PDF da gravação com base nas dez heurísticas de Nielsen e em uma base de conhecimento interna, ou executar o **teste de sistema** (sexta etapa), no qual o *script* gravado é reexecutado e avaliado por IA multimodal em conjunto com os critérios de aceite — reescrevendo automaticamente o *script* quando detecta que a falha decorre de uma mudança na interface, e não de uma quebra real de comportamento.

3.3 MODELAGEM DO BANCO DE DADOS

Todos os artefatos do sistema são persistidos em um banco **SQLite** único (`Storage/tcc.db`), acessado por meio de repositórios especializados por entidade. A Figura 5 apresenta o modelo entidade-relacionamento das nove tabelas do banco.

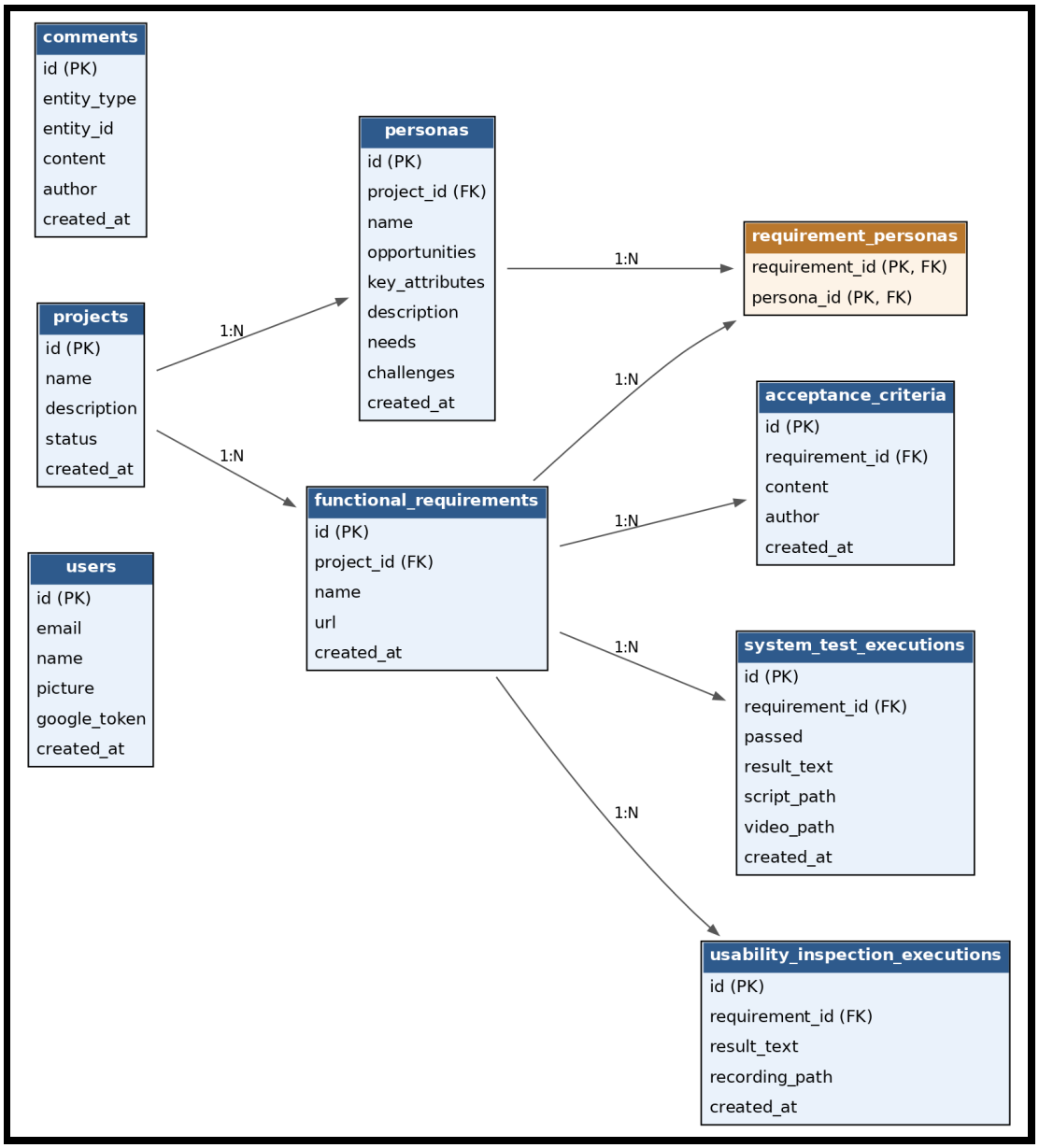

Figura 5 – Modelo entidade-relacionamento do banco de dados.

Fonte: Elaborado pelo autor.

A tabela `projects` é o agregado raiz do domínio de suporte e se relaciona 1:N com `personas` e `functional_requirements`. Cada requisito funcional se relaciona 1:N com `acceptance_criteria`, `system_test_executions` e `usability_inspection_executions`, e N:N com `personas` por meio da tabela associativa `requirement_personas`. A tabela `comments` armazena, de forma genérica (por `entity_type` e `entity_id`), os comentários do *tester* sobre execuções de teste e inspeções. A tabela `users`, destinada à autenticação via Google, ainda não possui relacionamento com as demais tabelas na versão atual do sistema. Exclusões de projetos

e de requisitos são feitas em cascata, removendo também os registros dependentes.

3.4 TECNOLOGIAS

3.4.1 Python

Python é a linguagem principal do projeto (versão 3.10), escolhida pela legibilidade, pelo suporte nativo do Playwright e pela ampla disponibilidade de bibliotecas para integração com APIs de inteligência artificial [58]. Todo o domínio da aplicação (entidades, serviços e agentes) é implementado em Python puro, sem uso de *frameworks* de ORM.

3.4.2 Flask

Flask é o *microframework* web utilizado para a camada de apresentação da aplicação [59]. Sua natureza minimalista permitiu expor as rotas HTTP de cada página (`routes.py`) e renderizar os templates Jinja2 sem a sobrecarga de convenções de *frameworks* mais completos, mantendo a lógica de negócio isolada na camada de domínio.

3.4.3 SQLite

SQLite é o sistema gerenciador de banco de dados utilizado para persistência local de todos os artefatos do sistema [60]. Por ser um banco embarcado (armazenado em um único arquivo, sem necessidade de um servidor separado), simplifica a instalação e a execução da aplicação, sendo adequado ao escopo de um projeto acadêmico de demonstração.

3.4.4 Playwright

Playwright é a biblioteca de automação de navegador utilizada tanto para gravar quanto para reexecutar as funcionalidades testadas [61]. O recurso `codegen` grava as interações do usuário em um *script* Python, que é posteriormente reproduzido de forma *headless* pelos agentes inteligentes de teste de sistema e de inspeção de usabilidade,

com gravação de vídeo da execução.

3.4.5 API do ChatGPT

A API da OpenAI, por meio do modelo multimodal GPT-4o, é o motor de inteligência artificial de todos os agentes do sistema [62]. Ela é utilizada tanto para tarefas de linguagem estruturada (validação de conformidade dos campos das entidades, em formato JSON) quanto para avaliação multimodal, na qual o PDF da gravação de uma execução é enviado diretamente ao modelo para julgar se os critérios de aceite foram atendidos ou se há problemas de usabilidade, conforme as heurísticas de Nielsen.

4 METODOLOGIA DE TRABALHO

4.1 PARADIGMA DA PESQUISA

O presente trabalho segue o paradigma de pesquisa aplicada, do tipo desenvolvimento de solução, apoiado na modelagem do problema por meio do *Domain-Driven Design* (Seção 4.2) e na consequente construção de um artefato de software que materializa essa modelagem. A avaliação da solução proposta, apresentada no Capítulo 5, tem natureza técnica: verifica, a partir de execuções reais sobre um projeto cadastrado na ferramenta, se os agentes inteligentes desenvolvidos executam corretamente as tarefas de teste de sistema e de inspeção de usabilidade para as quais foram projetados. Um estudo de caso comparativo, com times de desenvolvimento reais, que mensure quantitativamente a redução de tempo proporcionada pela ferramenta em relação à execução manual dessas técnicas, não foi conduzido no escopo desta monografia, conforme discutido na Seção 6.1.

4.2 ELICITAÇÃO E ANÁLISE DE REQUISITOS

4.2.1 Pesquisa bibliográfica

Para o presente trabalho, foi realizada uma pesquisa a respeito de abordagens modernas de engenharia de software para modelar cenários, e, como principal técnica, foi utilizado o *Domain-Driven Design*, pela sua característica de analisar o espaço do problema e da solução com foco em desenvolver soluções criativas e de alto valor agregado para o cliente [10]. Além disso, foram analisadas 18 ferramentas que atuam na área de teste de sistema e inspeção de usabilidade, buscando gerar um diferencial na solução proposta no presente trabalho. Para a criação da base heurística, foram coletados e agrupados sites considerados em conformidade com as heurísticas de Nielsen.

4.2.2 Brainstorming

Para elicitar os requisitos necessários para construir uma solução adequada ao problema, foi feito um *brainstorm* por meio da plataforma Excalidraw [63], a fim de

descrever como seria uma plataforma que atendesse aos problemas abordados. A Figura 6 apresenta o recorte do quadro relacionado ao domínio de projetos, personas e requisitos funcionais.

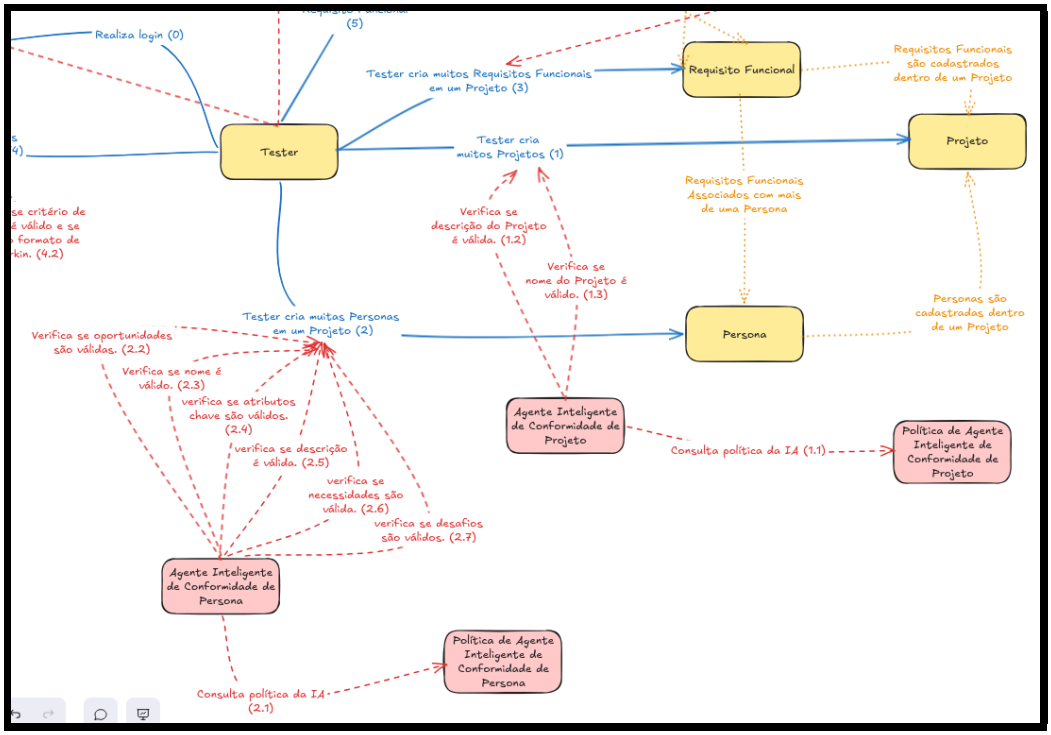

Figura 6 – Recorte do quadro de *brainstorming* – domínio de projetos.

Fonte: Elaborado pelo autor.

A Figura 7 apresenta o recorte relacionado à inspeção de usabilidade e à gravação de execução de requisitos funcionais.

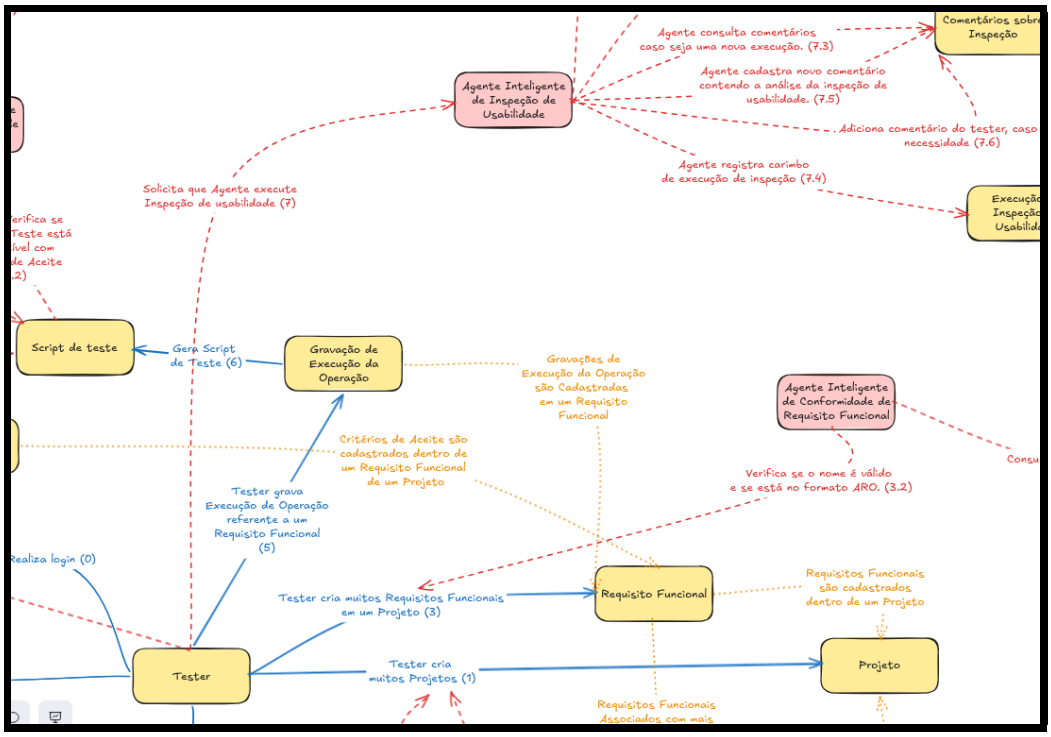

Figura 7 – Recorte do quadro de *brainstorming* – inspeção de usabilidade e requisitos funcionais.

Fonte: Elaborado pelo autor.

A Figura 8 detalha o fluxo do agente inteligente de teste de sistema e do critério de aceite.

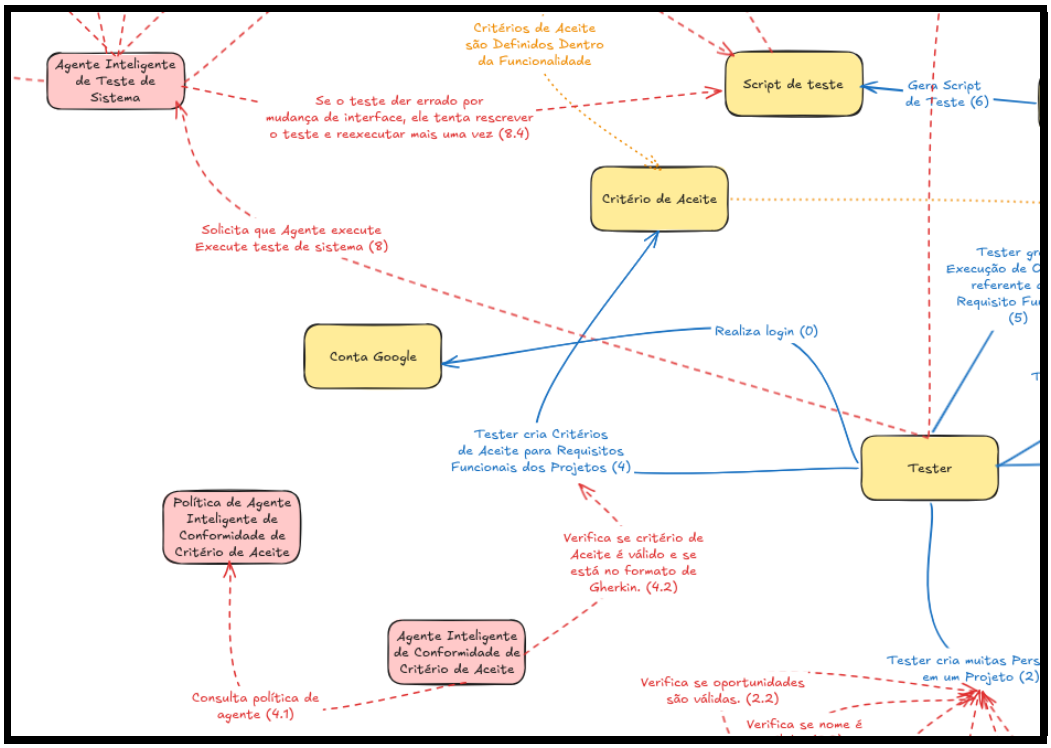

Figura 8 – Recorte do quadro de *brainstorming* – agente de teste de sistema e critério de aceite.

Fonte: Elaborado pelo autor.

A Figura 9 detalha o fluxo do agente inteligente de inspeção de usabilidade.

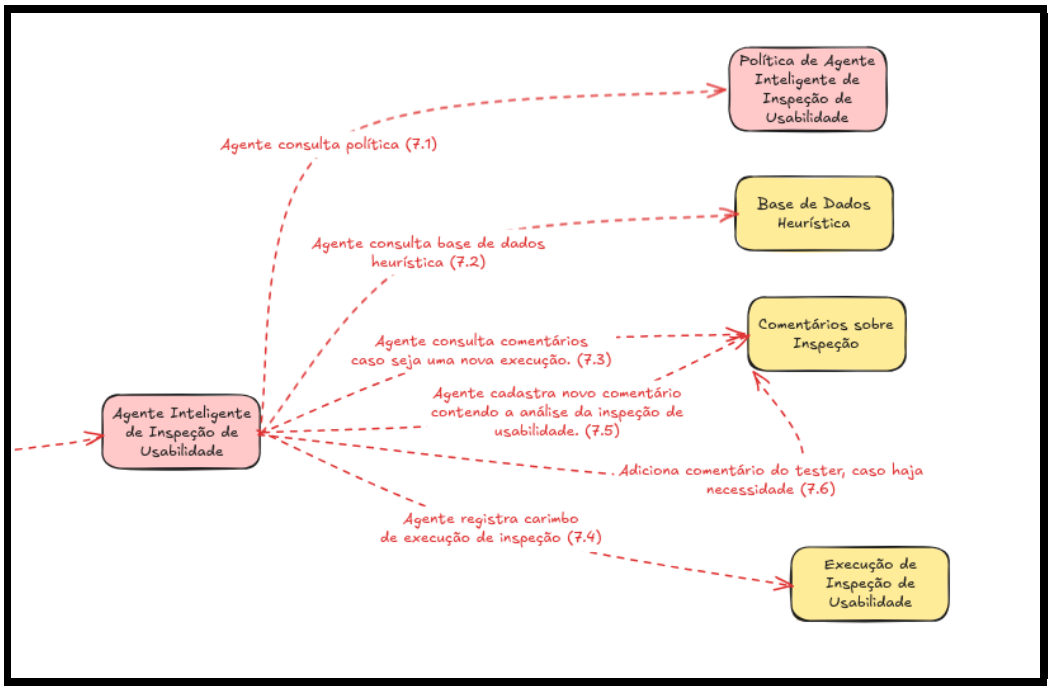

Figura 9 – Recorte do quadro de *brainstorming* – agente de inspeção de usabilidade.

Fonte: Elaborado pelo autor.

Por fim, a Figura 10 detalha o fluxo de execução do teste de sistema propriamente dito.

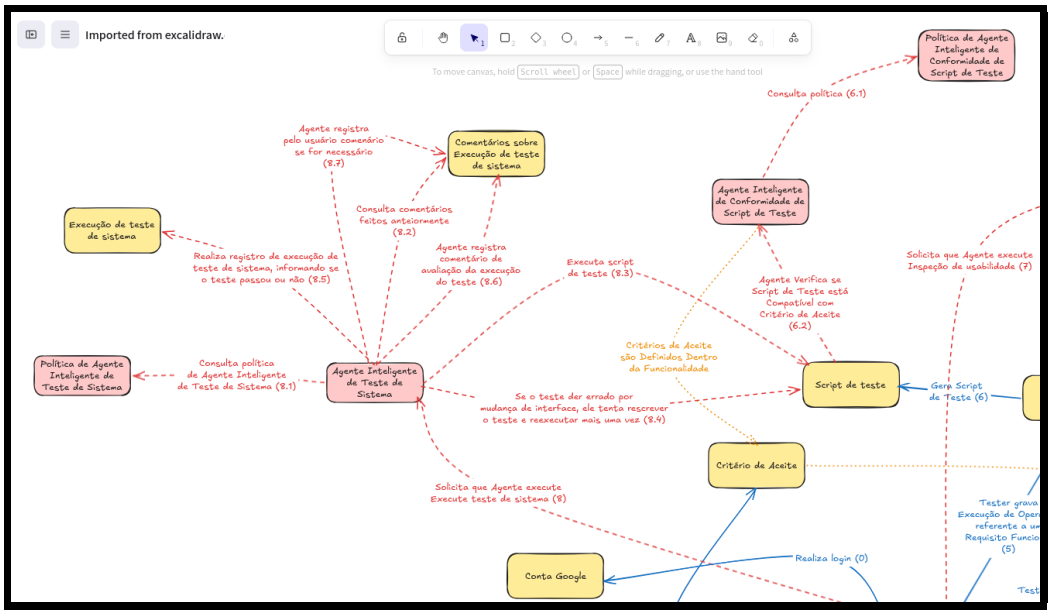

Figura 10 – Recorte do quadro de *brainstorming* – execução do teste de sistema.

Fonte: Elaborado pelo autor.

Em seguida, a ideia foi destrinchada dentro das áreas de problema e solução utilizando *Domain-Driven Design* para explicitar essas divisões e construir uma linguagem do problema.

4.2.3 Domain-Driven Design

O *Domain-Driven Design* (DDD) é uma abordagem de engenharia de software que enfatiza a modelagem das aplicações visando “atacar” o “coração do problema”, ou seja, os problemas que, caso solucionados, agregam muito valor para o usuário [10]. Essa abordagem separa a modelagem da aplicação em duas dimensões: a dimensão estratégica, que foca na solução no nível do negócio, por meio de “domínios”, e a dimensão tática, que foca em como a solução será construída, através de contextos delimitados. Após a análise bibliográfica das soluções existentes no mercado, através do DDD, o cenário atual foi modelado, resultando em três domínios: Projetos, Teste de Sistema e Inspeção de Usabilidade, ilustrados na Figura 11.

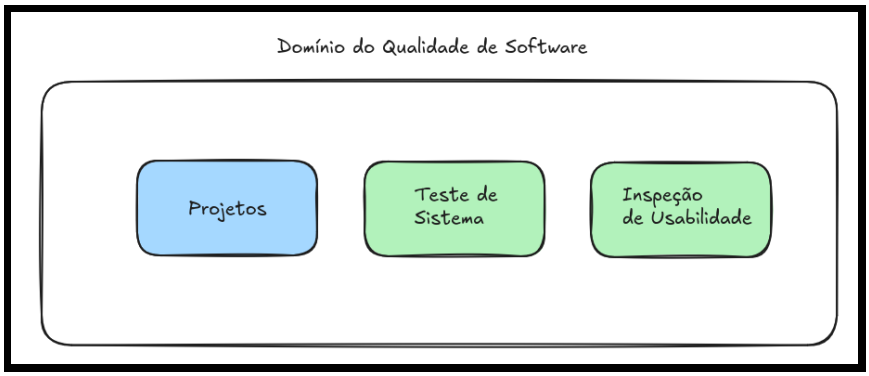

Figura 11 – Domínio da qualidade de software e seus subdomínios.

Fonte: Elaborado pelo autor.

O propósito e a classificação de cada domínio são descritos na Tabela 3.

Tabela 3 – Domínios identificados e seus propósitos.

Domínio	**Propósito**	**Tipo de domínio**
Domínio de Projetos	Permite que um usuário gerencie projetos, criando personas, requisitos funcionais e critérios de aceite	Suporte – já existem muitas soluções que atendem aos problemas desse domínio no mercado, sendo a solução algo adaptado ao contexto do problema, mas que não traz nenhum diferencial, servindo de apoio para outros domínios
Domínio de Teste de Sistema	Permite analisar registros de execução de requisitos funcionais e inferir, através de critérios de aceite, se foi possível chegar a um determinado resultado, configurando um teste de sistema	Principal – existem poucas soluções no mercado que atendem aos problemas desse domínio, o que o torna especial, já que precisa de uma solução feita sob encomenda para um cenário específico
Domínio de Inspeção de Usabilidade	Permite analisar registros de execução de requisitos e analisar, através de heurísticas de usabilidade, possíveis elementos que contribuem ou comprometem a usabilidade do usuário	Principal – existem poucas soluções no mercado que atendem aos problemas desse domínio, o que o torna especial, já que precisa de uma solução feita sob encomenda para um cenário específico

Fonte: Elaborado pelo autor.

Após a definição dos domínios, foi feita uma expansão da análise do problema

para encontrar os subdomínios (subáreas do problema) e definir os contextos delimitados (áreas da solução, ou seja, os elementos que farão parte da solução do problema); nesse contexto, cada subdomínio tem um contexto delimitado de mesmo nome. A Figura 12 apresenta o contexto delimitado do subdomínio de Projetos.

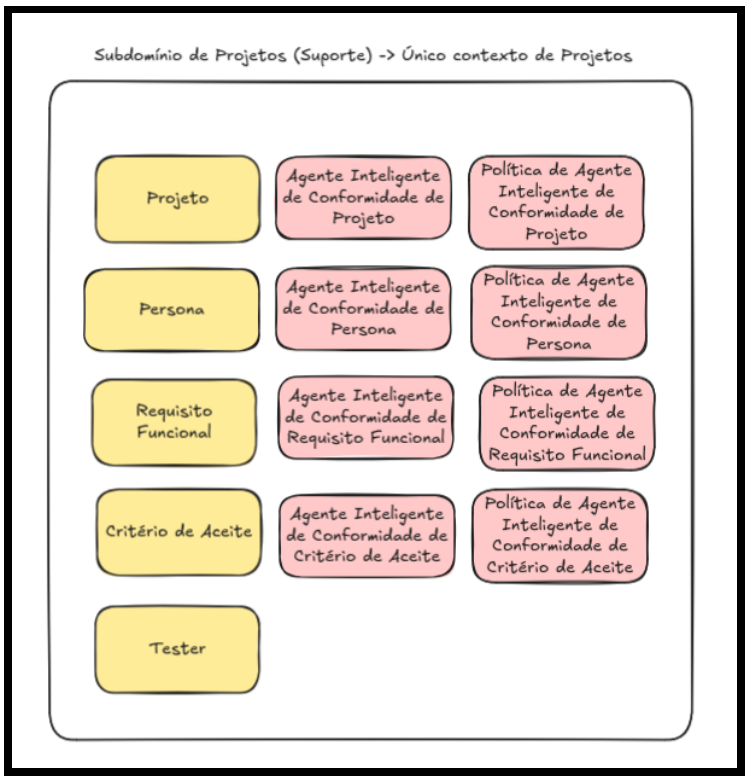

Figura 12 – Subdomínio de Projetos (suporte) e seu único contexto delimitado.

Fonte: Elaborado pelo autor.

4.2.4 Linguagem do problema

De forma complementar, as Figuras 13 e 14 apresentam os contextos delimitados dos subdomínios de Teste de Sistema e de Inspeção de Usabilidade, respectivamente.

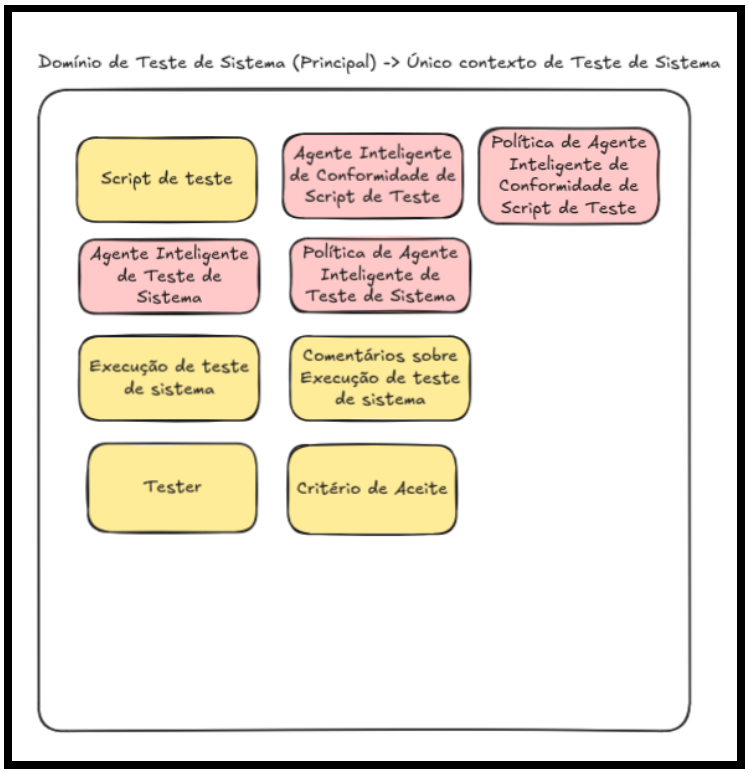

Figura 13 – Subdomínio de Teste de Sistema (principal) e seu único contexto delimitado.

Fonte: Elaborado pelo autor.

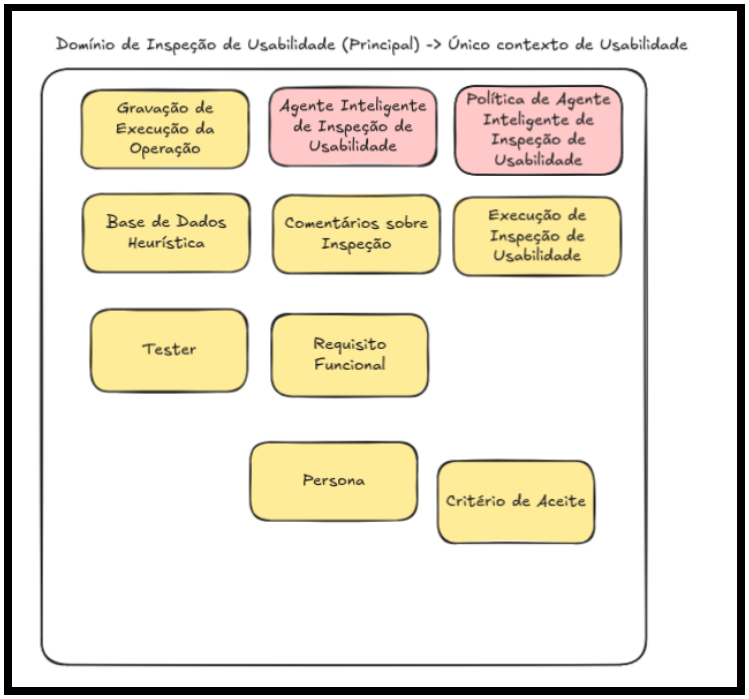

Figura 14 – Subdomínio de Inspeção de Usabilidade (principal) e seu único contexto delimitado.

Fonte: Elaborado pelo autor.

O DDD recomenda que seja construída uma linguagem única do problema para guiar a construção do software, pois uma mesma entidade pode ter diferentes significados em diferentes contextos [10]. Nesse sentido, foi construído o glossário apresentado na Tabela 4 para guiar o entendimento do problema e da solução.

Tabela 4 – Glossário da linguagem onipresente do problema.

Termo	**Definição**
Tester	Profissional da área de software que executa testes para aferir a qualidade do sistema sobre aspectos previamente definidos.
Login	Ato de adentrar em um sistema utilizando alguma credencial.
Projeto	Representação lógica de um conjunto de atividades, no caso do sistema, contendo nome, descrição e status.
AI Trust Maturity Model	Modelo criado para ajudar organizações a avaliar e desenvolver confiança em sistemas de Inteligência Artificial.
Política de Agente de Conformidade de Projeto	Restrições definidas para o estado de um projeto.
Agente Inteligente de Conformidade de Projeto	Entidade que verifica se o Projeto está em conformidade com a Política de Agente de Conformidade de Projeto.
Persona	Representação lógica de um perfil de uma pessoa, no caso do sistema, que pertence a um projeto e contém oportunidades, nome, atributos-chave, descrição, necessidades e desafios.
ARO	Formato utilizado na Engenharia de Software para documentar requisitos. Segue a estrutura *Actor + Requirement + Object*, ou seja, Ator + Requisito + Objeto. Exemplo: o Tester executa Script de Teste.

Continua na próxima página

Termo	**Definição**
Requisito Funcional	Representação lógica de uma funcionalidade oferecida para o usuário em um sistema, pertencente a um projeto e descrita em formato ARO.
Critério de Aceite	Verificação a ser feita para considerar um requisito funcional como concluído, no contexto do sistema, em formato de linguagem Gherkin.
Política de Agente Inteligente de Conformidade de Critério de Aceite	Restrições definidas para o estado de um Critério de Aceite.
Agente Inteligente de Conformidade de Critério de Aceite	Entidade que verifica se o Critério de Aceite está em conformidade com a Política de Agente de Conformidade de Critério de Aceite.
Execução de Operação	Ato de executar uma funcionalidade no sistema testado.
Gravação de Execução da Operação	Ato de gravar a execução de uma funcionalidade no sistema testado.
Script de Teste	Script que, ao ser executado, realiza passos predeterminados, simulando a interação de um usuário com a interface de um sistema.
Política de Agente Inteligente de Conformidade de Script de Teste	Restrições definidas para o estado de um Script de Teste.
Agente Inteligente de Conformidade de Script de Teste	Entidade que verifica se o Script de Teste está em conformidade com a Política de Agente de Conformidade de Script de Teste.
Inspeção de Usabilidade	Método que visa encontrar problemas de usabilidade nas interfaces do software submetido, por meio da análise de especialistas em usabilidade ou profissionais de desenvolvimento de software.

Continua na próxima página

Termo	**Definição**
Política de Agente Inteligente de Inspeção de Usabilidade	Restrições definidas para o estado de uma Inspeção de Usabilidade.
Agente Inteligente de Inspeção de Usabilidade	Entidade que verifica se a Inspeção de Usabilidade está em conformidade com a Política de Agente Inteligente de Inspeção de Usabilidade.
Base de Dados Heurística	Base de dados contendo interfaces que são adequadas às heurísticas de usabilidade.
Comentários Sobre Inspeção	Comentários inseridos pelo Tester para ajustar as respostas do Agente Inteligente de Inspeção de Usabilidade.
Teste de Sistema	Testes que simulam o uso de um sistema por um usuário real. Também são chamados de testes ponta-a-ponta, *end-to-end*, ou testes de interfaces.
Política de Agente Inteligente de Teste de Sistema	Restrições definidas para o estado de um Teste de Sistema.
Execução de Teste de Sistema	Execução do Script de Teste que executa os passos registrados de um perfil de usuário.
Comentários Sobre Execução de Teste de Sistema	Comentários inseridos pelo Tester para ajustar as respostas do Agente Inteligente de Teste de Sistema.

4.2.5 Descrição e gerenciamento dos requisitos

Foi utilizado o formato de épicos e funcionalidades (*features*) para descrever os requisitos do projeto na plataforma OpenProject [64], conforme ilustrado na Figura 15.

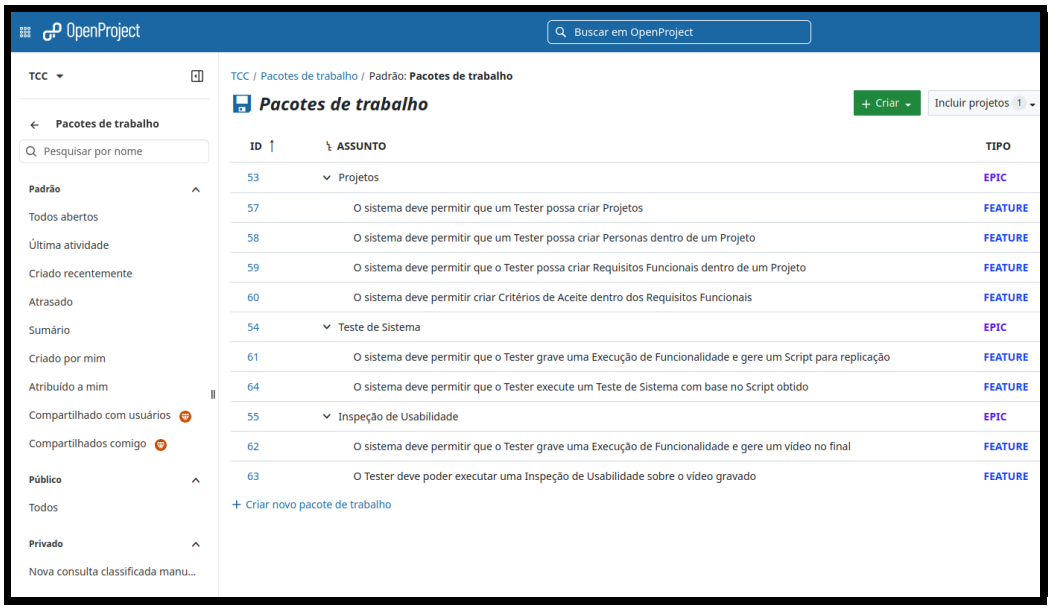

Figura 15 – Pacotes de trabalho cadastrados na plataforma OpenProject.

Fonte: Elaborado pelo autor.

4.2.6 Fluxo de trabalho

Para programar a solução, foi utilizado o Visual Studio Code [65]; para o controle de versionamento, o Git [66]; e, para a hospedagem, o GitHub [67]. O Visual Studio Code é um editor de código da Microsoft amplamente usado para desenvolvimento de aplicações; o Git é uma ferramenta de versionamento *open source* utilizada para armazenar versões de arquivos; e o GitHub é a plataforma de repositórios mais popular do mundo, permitindo que desenvolvedores compartilhem e armazenem arquivos.

O processo de desenvolvimento consiste, basicamente, em trabalhar de forma iterativa-incremental através de uma adaptação do Scrum, coletando as tarefas do OpenProject e as implementando. O fluxo de desenvolvimento funciona da seguinte maneira:

1. Criar uma *branch* nova com um prefixo que indique a sua intenção;
2. Realizar testes e executar *pipelines* implementadas;
3. Abrir uma *Pull Request* no GitHub para a *branch main*;
4. Aguardar aprovação ou aprovar a *Pull Request*;

5. Baixar a nova versão.

5 RESULTADOS

Este capítulo apresenta os resultados obtidos com o desenvolvimento da ferramenta descrita nos Capítulos 3 e 4, relacionando-os ao objetivo geral e a cada um dos objetivos específicos definidos na Seção 1.3 deste trabalho: a construção de uma base de conhecimento de heurísticas de usabilidade, a construção de uma base de conhecimento de interfaces de referência, o desenvolvimento de um agente inteligente de inspeção de usabilidade e o desenvolvimento de um agente inteligente de teste de sistema.

5.1 BASE DE CONHECIMENTO DE HEURÍSTICAS DE USABILIDADE

Em atendimento ao primeiro objetivo específico, foi construída uma base de conhecimento contendo as dez heurísticas de Nielsen [16] (detalhadas na Seção 2.1), utilizada pelo agente inteligente de inspeção de usabilidade como critério de avaliação das interfaces submetidas. A Tabela 5 apresenta uma amostra dessa base.

Tabela 5 – Amostra da base de conhecimento de heurísticas de usabilidade, agrupada por heurística de Nielsen.

Heurística de Nielsen	**Sistema de referência**	**URL**
Visibilidade do estado do sistema	Google Search	google.com
	Gmail	mail.google.com
	Amazon	amazon.com
Correspondência entre o sistema e o mundo real	YouTube	youtube.com
	Spotify	spotify.com
	Netflix	netflix.com
Consistência e padrões	Slack	slack.com
	Discord	discord.com
	Microsoft Teams	teams.microsoft.com
Prevenção de erros	GitHub	github.com
	GitLab	gitlab.com
	Jira	atlassian.com/software/jira
Reconhecimento em vez de recordação	Trello	trello.com
	Notion	notion.so
	Airtable	airtable.com
Flexibilidade e eficiência de uso	Figma	figma.com
	Canva	canva.com
	CapCut	capcut.com
Ajuda e documentação	Shopify	shopify.com
	Stripe	stripe.com
	PayPal	paypal.com

Fonte: Elaborado pelo autor, a partir da base de conhecimento heurística do sistema.

Para cada uma das 10 heurísticas, a base contém um total de 50 sistemas de referência, conforme detalhado na Seção 5.2.

5.2 BASE DE CONHECIMENTO DE INTERFACES DE REFERÊNCIA

Em atendimento ao segundo objetivo específico, cada uma das dez heurísticas foi associada a um conjunto de 50 interfaces de referência reais e amplamente utilizadas (como Google Search, Gmail, Amazon, YouTube, Netflix, GitHub, Notion e Figma, entre outras), totalizando 500 entradas na base de conhecimento. Essas referências são utilizadas pelo agente inteligente de inspeção de usabilidade como parâmetro de comparação ao analisar se uma interface submetida atende ou viola cada heurística, conforme ilustrado nos resultados apresentados na Seção 5.3.

5.3 AGENTE INTELIGENTE DE INSPEÇÃO DE USABILIDADE

Em atendimento ao terceiro objetivo específico, foi desenvolvido o agente inteligente de inspeção de usabilidade, responsável por analisar o PDF gerado a partir da gravação de uma funcionalidade e apontar violações das heurísticas de Nielsen, com base na base de conhecimento apresentada na Seção 2.1. A Figura 16 apresenta o resultado de uma inspeção real, executada sobre a tela de login de um dos projetos cadastrados na ferramenta (Seção 3.2).

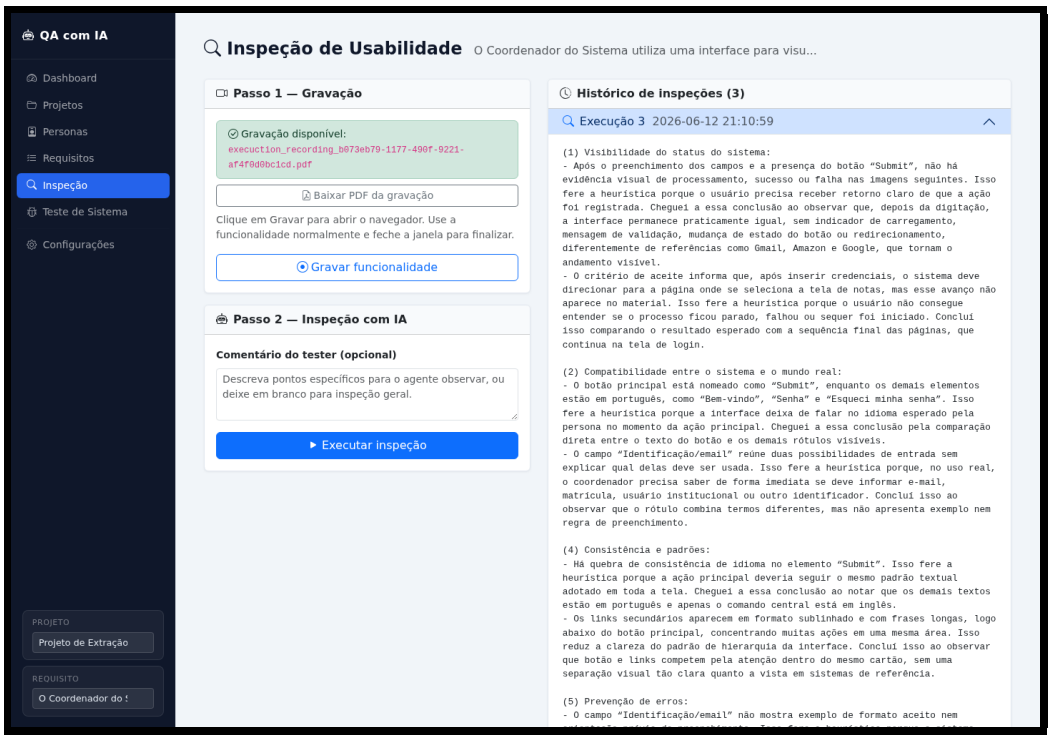

Figura 16 – Página de Inspeção de Usabilidade com o resultado de uma execução real do agente.

Fonte: Elaborado pelo autor.

Nessa execução, o agente identificou, entre outras violações, a ausência de retorno visual após o envio do formulário de login (ferindo a heurística de visibilidade do estado do sistema) e a mistura do idioma do botão principal ("Submit", em inglês) com o restante da interface, em português (ferindo a heurística de correspondência entre o sistema e o mundo real e a de consistência e padrões), sempre justificando a conclusão com base na comparação direta entre a interface analisada e as referências da base de conhecimento. Esse comportamento demonstra que o agente é capaz de aplicar heurísticas de usabilidade de forma automatizada e fundamentada, cumprindo

o objetivo específico proposto.

5.4 AGENTE INTELIGENTE DE TESTE DE SISTEMA

Em atendimento ao quarto objetivo específico, foi desenvolvido o agente inteligente de teste de sistema, responsável por reexecutar o *script* Playwright gerado durante a gravação de uma funcionalidade e avaliar, por meio do modelo multimodal GPT-4o, se o critério de aceite associado foi cumprido. A Figura 17 apresenta o histórico de execuções reais desse agente para o mesmo requisito funcional utilizado na Seção 5.3.

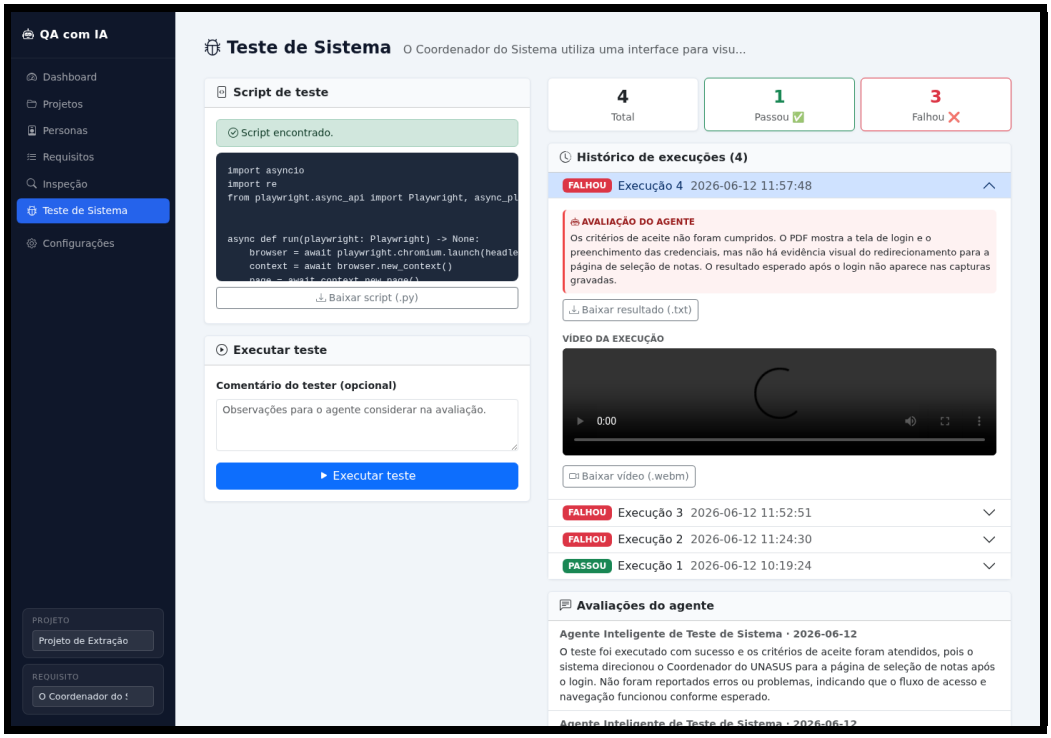

Figura 17 – Página de Teste de Sistema com o histórico de execuções reais do agente.

Fonte: Elaborado pelo autor.

No histórico apresentado, de quatro execuções realizadas sobre o mesmo requisito, uma foi avaliada pelo agente como aprovada e três como reprovadas, sempre acompanhadas de uma justificativa textual sobre o que foi ou não observado nas capturas de tela da execução (por exemplo, a ausência de evidência visual do redirecionamento esperado após o preenchimento das credenciais). Esse resultado demonstra que o agente não se limita a verificar se o *script* foi executado sem erros, mas efeti-

vamente avalia, a partir da gravação em vídeo e das capturas geradas, se o estado esperado pelo critério de aceite foi de fato alcançado, cumprindo o objetivo específico proposto.

5.5 AGENTES DE CONFORMIDADE E O FRAMEWORK AI-TMM

Além dos dois agentes inteligentes de domínio apresentados nas Seções 5.3 e 5.4, a ferramenta conta com cinco agentes de conformidade (Projetos, Personas, Requisitos Funcionais, Critérios de Aceite e Scripts de Teste), responsáveis por validar cada entidade antes de sua persistência no banco de dados, seguindo o framework AI-TMM (Seção 2.1). A Figura 18 apresenta o *prompt* do sistema real utilizado pelo agente de conformidade de Requisitos Funcionais, exibido na página de Configurações da aplicação.

Figura 18 – Prompt do sistema do agente de conformidade de Requisitos Funcionais, com exemplo de formato válido e inválido.

Fonte: Elaborado pelo autor.

Esse *prompt* ilustra a aplicação prática de três dos pilares do AI-TMM: a **transparência e explicabilidade**, ao exigir que o agente sempre retorne um campo `feedback` explicando o motivo da validação ou rejeição, em vez de apenas um valor booleano; a **justiça e mitigação de viés**, ao fornecer um exemplo válido (“O Tester executa o Script de Teste”, no formato Ator + Requisito + Objeto) e um exemplo inválido (“fazer login”, sem ator nem objeto claros) explicitamente no *prompt*, de modo que o critério de avaliação seja objetivo e reprodutível, em vez de depender de um julgamento implícito

do modelo; e a **segurança e robustez**, ao restringir o agente a validar exclusivamente o formato ARO, sem opinar sobre o conteúdo de negócio do requisito. Os agentes de conformidade de Projetos e de Personas seguem o mesmo padrão de resposta em JSON, acrescentando instruções específicas para os pilares de **privacidade e proteção de dados** e de justiça, como a exigência de "linguagem formal, objetiva e sem discriminação" e a rejeição de conteúdo inapropriado ou ilegal.

5.6 DEMAIS PÁGINAS DA APLICAÇÃO

Complementando as páginas de Dashboard e Projetos apresentadas no Capítulo 3, esta seção apresenta as demais páginas da aplicação utilizadas no fluxo de trabalho descrito na Seção 3.2. A Figura 19 apresenta a página de Personas, com os seis campos validados pelo agente de conformidade correspondente.

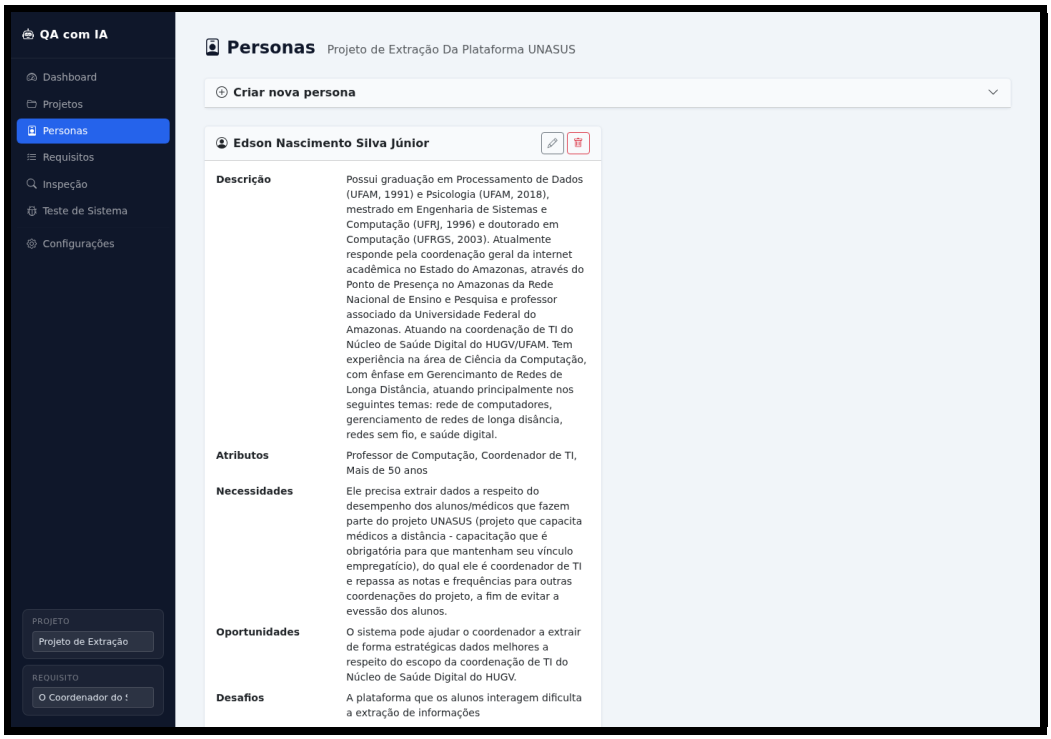

Figura 19 – Página de Personas com uma persona real cadastrada no sistema.

Fonte: Elaborado pelo autor.

A Figura 20 apresenta a página de Requisitos Funcionais, incluindo o requisito no formato ARO e o critério de aceite no formato Gherkin utilizados como base para os resultados apresentados nas Seções 5.3 e 5.4.

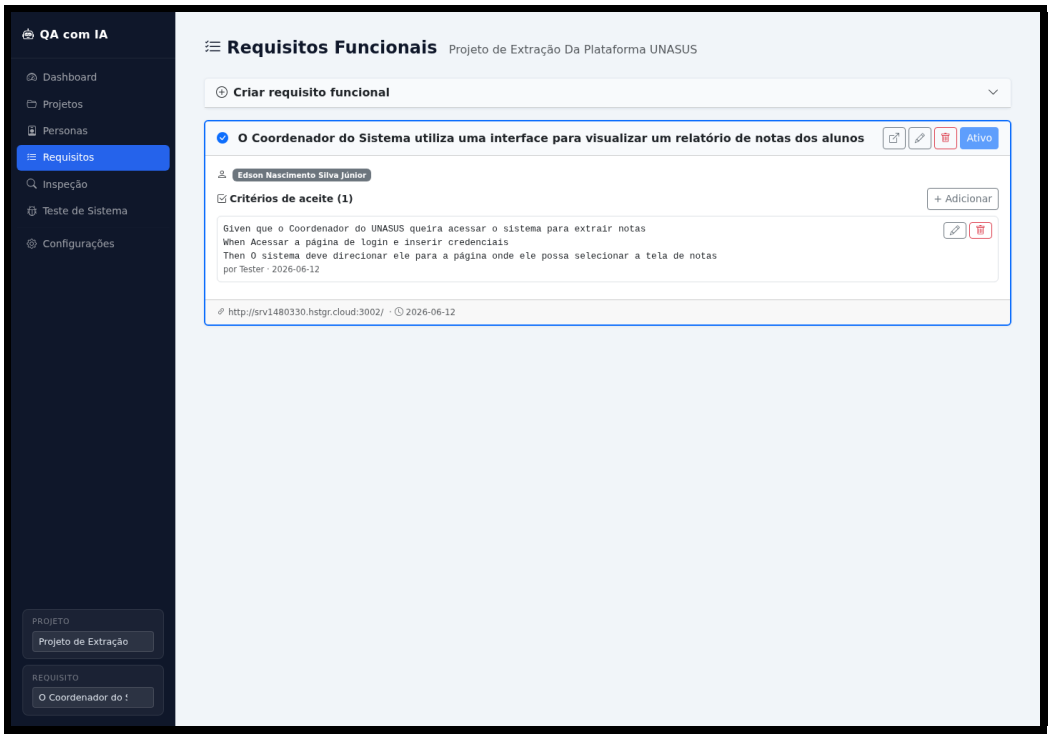

Figura 20 – Página de Requisitos Funcionais com um requisito e critério de aceite reais.

Fonte: Elaborado pelo autor.

Em conjunto, os resultados apresentados neste capítulo demonstram que a ferramenta desenvolvida cumpre os quatro objetivos específicos propostos e, consequentemente, o objetivo geral do trabalho.

6 CONSIDERAÇÕES FINAIS

Diante da carência de ferramentas acessíveis que automatizem o teste de sistema e a inspeção de usabilidade, este trabalho teve como objetivo desenvolver uma ferramenta de apoio à melhoria da qualidade de software que utiliza modelos de inteligência artificial generativa para automatizar essas duas técnicas, reduzindo o tempo necessário para sua aplicação ao longo do ciclo de vida de um projeto.

A ferramenta desenvolvida organiza as informações de um projeto em personas e requisitos funcionais com critérios de aceite, permite a gravação do fluxo correto de uma funcionalidade e possibilita sua reexecução autônoma, tanto para a geração de relatórios de teste de sistema quanto para inspeções de usabilidade fundamentadas nas heurísticas de Nielsen. Para alcançar a completude dos objetivos traçados, foi necessário realizar uma modelagem do problema por meio do Domain-Driven Design (Seção 4.2), que resultou nos três domínios que orientaram a arquitetura da aplicação (Capítulo 3): Projetos, Teste de Sistema e Inspeção de Usabilidade.

A adoção de uma arquitetura em camadas organizada segundo esses domínios, aliada às tecnologias selecionadas — Python, Flask, SQLite, Playwright e a API da OpenAI —, permitiu implementar a ferramenta como um monólito modular de baixa complexidade operacional, adequado ao escopo de um projeto acadêmico, mas sem abrir mão da separação de responsabilidades entre as áreas do problema. Os resultados apresentados no Capítulo 5, obtidos a partir de execuções reais dos agentes inteligentes de inspeção de usabilidade e de teste de sistema, evidenciam que ambos são capazes de fundamentar suas avaliações na base de conhecimento heurística construída para este trabalho, cumprindo os quatro objetivos específicos propostos.

Dessa forma, considerando os resultados obtidos, conclui-se que este trabalho contribui para a melhoria do processo de garantia de qualidade de software ao viabilizar a aplicação de testes de sistema e inspeções de usabilidade com maior agilidade e menor exigência de conhecimento técnico em programação por parte do profissional que as executa.

Cabe ressaltar que, embora a pergunta de pesquisa que motivou este trabalho envolva, em sua essência, avaliar se a automatização proposta reduz o tempo de aplicação do teste de sistema e da inspeção de usabilidade em relação à execução

manual dessas técnicas, o objetivo geral e os quatro objetivos específicos definidos na Seção 1.3 para este Trabalho de Conclusão de Curso foram integralmente cobertos pelo trabalho desenvolvido, conforme demonstrado no Capítulo 5. A resposta completa à pergunta de pesquisa, no entanto, depende de uma etapa adicional de validação empírica — o estudo comparativo discutido na Seção 6.1 —, cuja condução exige estudos e desenvolvimentos próprios que extrapolam o escopo delimitado para esta monografia, sem que essa dependência comprometa a completude dos objetivos específicos efetivamente alcançados.

6.1 TRABALHOS FUTUROS

Como sugestões de trabalhos futuros de evolução direta da ferramenta, destacam-se: a ampliação da base de conhecimento heurística com interfaces adicionais e heurísticas específicas de dispositivos móveis; e a evolução do modelo do projeto apresentado na Seção 4.2 para incorporar autenticação de múltiplos usuários por meio do domínio de autenticação já mapeado no código-fonte da aplicação.

Além dessas frentes, evidencia-se a necessidade de um estudo comparativo do uso da ferramenta desenvolvida com times de desenvolvimento reais, a fim de mensurar de forma quantitativa a diminuição do tempo gasto pelos usuários na execução de testes de sistema e de inspeções de usabilidade, em relação à execução manual dessas mesmas técnicas.

6.1.1 Justificativa para a não realização do estudo comparativo

A condução desse estudo comparativo não foi realizada dentro do escopo desta monografia em razão da restrição de tempo inerente ao Trabalho de Conclusão de Curso. Um estudo dessa natureza demandaria o recrutamento de times de desenvolvimento reais, a definição de um protocolo experimental (ou de estudo de caso) com grupos de controle, a aplicação da ferramenta e das técnicas manuais equivalentes em projetos comparáveis e a coleta de dados ao longo de um período suficiente para produzir resultados estatisticamente relevantes sobre o ganho de tempo — etapas que, somadas, excedem o prazo disponível para a conclusão deste trabalho.

6.1.2 A ausência do estudo comparativo e a completude dos objetivos do trabalho

Ainda que essa avaliação seja necessária para responder de forma completa à pergunta de pesquisa que motivou este trabalho, sua ausência não exime a completude dos objetivos específicos definidos na Seção 1.3, tampouco do objetivo geral deles decorrente. Isso porque tais objetivos — a construção das bases de conhecimento de heurísticas e de interfaces de referência e o desenvolvimento dos agentes inteligentes de inspeção de usabilidade e de teste de sistema — dizem respeito à construção e ao funcionamento corretos da ferramenta, e foram integralmente cumpridos e demonstrados no Capítulo 5 a partir de execuções reais. O estudo comparativo constitui, portanto, uma etapa de validação complementar do valor agregado pela ferramenta, e não um pré-requisito para a completude dos objetivos específicos e geral já alcançados por este trabalho.

REFERÊNCIAS

[1] SOMMERVILLE, I. **Engenharia de Software**. 10. ed. São Paulo: Pearson, 2018. 15, 19

[2] Wrike. **What is iterative and incremental development?** 2026. Wrike Project Management Guide. Disponível em: <https://www.wrike.com/project-management-guide/faq/what-is-iterative-incremental-development/>. Acesso em: 5 mar. 2026. 19

[3] PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software: uma abordagem profissional**. 9. ed. Porto Alegre: AMGH, 2021. 19, 20, 23

[4] WAZLAWICK, R. S. **Engenharia de software: conceitos e práticas**. Rio de Janeiro: Elsevier, 2013. 20

[5] TAVARES, M. T. V. **Engenharia de software moderna: princípios e práticas para desenvolvimento de software com produtividade**. Belo Horizonte: Independente, 2020. 20

[6] RUBIN, K. S. **Scrum Essencial: um guia prático para o mais popular processo ágil**. Rio de Janeiro: Alta Books, 2017. Tradução: Roberto Rezende. 20

[7] TEDESCO, K. **Concorrência, paralelismo, processos, threads, programação síncrona e assíncrona**. 2019. Blog da TreinaWeb. Disponível em: <https://www.treinaweb.com.br/blog/concorrencia-paralelismo-processos-threads-programacao-sincrona-e-assincrona>. Acesso em: 5 mar. 2026. 20

[8] ALMEIDA, D. M. de. **Desenvolvimento orientado por comportamento (BDD)**. 2011. DevMedia. Disponível em: <https://www.devmedia.com.br/desenvolvimento-orientado-por-comportamento-bdd/21127>. Acesso em: 5 mar. 2026. 21

[9] CEZERINO, A.; NASCIMENTO, F. Utilização da técnica de desenvolvimento orientado por comportamento (bdd) no levantamento de requisitos. **RICA – Revista de Iniciação Científica da Ânima**, 2016. Disponível em: <https://portaldeperiodicos.animaeducacao.com.br/index.php/rica/article/view/17874>. Acesso em: 5 mar. 2026. 21

[10] VERNON, V. **Domain-driven design destilado**. Rio de Janeiro: Alta Books, 2024. 21, 46, 50, 54

[11] GAMMA, E. et al. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000. 21

[12] BENYON, D. **Interação humano-computador**. 2. ed. São Paulo: Pearson Prentice Hall, 2011. 21

[13] SILVA, A. M. da et al. Lógica nebulosa aplicada na análise de usabilidade de software. **Revista de Informática Aplicada**, v. 11, n. 1, 2015. Disponível em: <https://www.researchgate.net/profile/

Alisson-Silva-3/publication/264405051_Logica_Nebulosa_Aplicada_na_Analise_de_Usabilidade_de_Software/links/5f0a12b645851550509d0aa2/Logica-Nebulosa-Aplicada-na-Analise-de-Usabilidade-de-Software.pdf>. Acesso em: 5 mar. 2026. 21

[14] NETO, O. J. M. **Usabilidade da interface de dispositivos móveis: heurísticas e diretrizes para o design**. Dissertação (Dissertação de Mestrado em Ciências de Computação e Matemática Computacional) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2013. Disponível em: <https://www.teses.usp.br/teses/disponiveis/55/55134/tde-07012014-110754/publico/dissertacaoOlibario.pdf>. Acesso em: 5 mar. 2026. 21

[15] SANTOS, F. H. O. dos. **Apoio ao processo de inspeção de usabilidade para aplicações de software**. 101 p. Dissertação (Dissertação de Mestrado em Informática) — Universidade Federal do Amazonas, Manaus, 2011. Disponível em: <http://tede.ufam.edu.br/handle/tede/2895>. Acesso em: 5 mar. 2026. 22

[16] NIELSEN, J. **10 usability heuristics for user interface design**. 2020. Nielsen Norman Group. Disponível em: <https://www.nngroup.com/articles/ten-usability-heuristics/>. Acesso em: 5 mar. 2026. 22, 59

[17] CASTRO, L. S. N. de; SANTOS, P. C. dos. Impactos do versionamento de código no trabalho colaborativo em equipes de desenvolvimento. In: IFSULDEMINAS. **Jornada Científica e Tecnológica e Simpósio de Pós-Graduação do IFSULDEMINAS (JOSIF)**. 2025. Disponível em: <https://josif.ifsuldeminas.edu.br/ojs/index.php/anais/article/download/2739/2506>. Acesso em: 5 mar. 2026. 23

[18] RIBEIRO, T. de O. et al. Benefícios do bpmn na modelagem dos processos: um estudo exploratório. In: **Simpósio de Engenharia de Produção (SIMPEP)**. [s.n.], 2015. Disponível em: <https://www.researchgate.net/profile/Tiago-Ribeiro-23/publication/318792227_BENEFICIOS_DO_BPMN_NA_MODELAGEM_DOS_PROCESSOS_UM_ESTUDO_EXPLORATORIO/links/597f2a1daca272d5681875d3/BENEFICIOS-DO-BPMN-NA-MODELAGEM-DOS-PROCESSOS-UM-ESTUDO-EXPLORATORIO.pdf>. Acesso em: 5 mar. 2026. 23

[19] MATA, J. T. P. **Aprimorando a entrega e disponibilidade de um sistema de informação com CI/CD e estratégias de implantação**. Dissertação (Trabalho de Conclusão de Curso em Sistemas de Informação) — Universidade Federal do Rio Grande do Norte, Centro de Ensino Superior do Seridó, Caicó, 2024. Disponível em: <https://repositorio.ufrn.br/server/api/core/bitstreams/8b5188f2-6cce-40e9-b5bd-08985995ec0b/content>. Acesso em: 5 mar. 2026. 24

[20] NAKAMITI, E. K. **Agentes inteligentes artificiais**. 91 p. Dissertação (Dissertação de Mestrado em Comunicação e Semiótica) — Pontifícia Universidade Católica de São Paulo, São Paulo, 2009. Disponível em: <https://repositorio.pucsp.br/bitstream/handle/5240/1/Eduardo%20Kiochi%20Nakamiti.pdf>. Acesso em: 5 mar. 2026. 24

[21] ISLAM, M. R.; SINGH, S.; KHAN, S. Ai trust framework and maturity model (ai-tmm): A conceptual framework for trustworthy artificial intelligence. **Entropy**, Basel, v. 25,

n. 10, 2023. Art. 1429. Disponível em: <https://www.mdpi.com/1099-4300/25/10/1429>. Acesso em: 5 mar. 2026. 24

[22] PINHEIRO, J. H. S. Inteligência artificial generativa na transformação dos processos de desenvolvimento de software: oportunidades, desafios e impactos na produtividade. **Aurum Revista Multidisciplinar**, Curitiba, v. 1, n. 5, p. 201–219, 2025. Disponível em: <https://doi.org/10.63330/armv1n5-016>. Acesso em: 5 mar. 2026. 25

[23] WEICHBROTH, P.; SŁOWIK, A.; WĄTRÓBSKI, J. Using large language models for software requirements classification. In: **FEDCSIS – Conference on Computer Science and Intelligence Systems**. [s.n.], 2024. Annals of Computer Science and Information Systems, v. 40. Disponível em: <https://annals-csis.org/Volume_40/drp/pdf/3747.pdf>. Acesso em: 18 fev. 2026. 25

[24] SILVA, W. J. L. da; FROGERI, R. F. **Engenharia de prompt: uma análise das "alucinações" em inteligências artificiais generativas**. Varginha, 2024. Disponível em: <http://repositorio.unis.edu.br/handle/prefix/2727>. Acesso em: 5 mar. 2026. 25

[25] SCHMIDT, D. et al. Inteligência artificial aplicada à gestão do conhecimento: construção de chatbots com llms e rag. **Revista SODEBRAS: Soluções Para o Desenvolvimento do País**, v. 20, set./dez. 2025. Disponível em: <https://revista.sodebras.com.br/index.php/revista/article/view/144>. Acesso em: 6 mar. 2026. 25

[26] Bright Data. **What is Selenium?** 2026. Disponível em: <https://brightdata.com.br/faqs/selenium/what-is-selenium>. Acesso em: 5 mar. 2026. 25

[27] Selenium. **Selenium automates browsers**. 2026. Disponível em: <https://www.selenium.dev/>. Acesso em: 5 mar. 2026. 25

[28] PINHEIRO, I. **Selenium: o que é? Sabemos realmente para que ele serve?** 2022. Medium. Disponível em: <https://medium.com/@isabelle.pinheiro/selenium-o-que-%C3%A9-sabemos-realmente-para-que-ele-serve-6c105968bb34>. Acesso em: 5 mar. 2026. 25

[29] SANTOS, E. M. dos. **Cypress: passo a passo para começar a usar**. 2021. SoftDesign. Disponível em: <https://www.softdesign.com.br/blog/cypress_passo_a_passo_para_comecar_a_usar/>. Acesso em: 5 mar. 2026. 26

[30] Cypress. **Cypress: testing framework for JavaScript**. 2026. Disponível em: <https://www.cypress.io>. Acesso em: 5 mar. 2026. 26

[31] Microsoft. **Playwright: fast and reliable end-to-end testing for modern web apps**. 2026. Disponível em: <https://playwright.dev/>. Acesso em: 5 mar. 2026. 26

[32] CABRAL, R. de Brito de O. **Playwright básico: introdução à automação de testes web**. 2023. Medium. Disponível em: <https://medium.com/@rodrigo.oliveiracabral/playwright-b%C3%A1sico-introdu%C3%A7%C3%A3o-%C3%A0-automa%C3%A7%C3%A3o-de-testes-web-e90428b176d2>. Acesso em: 5 mar. 2026. 26

[33] Appium. **Appium documentation**. 2026. Disponível em: <https://appium.io/docs/en/latest/>. Acesso em: 5 mar. 2026. 27

[34] WANG, J.; WU, J. Research on mobile application automation testing technology based on appium. In: **2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)**. Jishou, China: [s.n.], 2019. p. 247–250. Doi: 10.1109/ICVRIS.2019.00068. Disponível em: <https://ieeexplore.ieee.org/document/8920799>. Acesso em: 18 fev. 2026. 27

[35] Assert Quality Assurance. **Appium: automação para apps mobile**. 2026. Medium. Disponível em: <https://medium.com/assertqualityassurance/appium-automa%C3%A7%C3%A3o-para-apps-mobile-a256db64a27d>. Acesso em: 18 fev. 2026. 27

[36] LAIHONEN, P. **Robot Framework: past, present and future**. 2025. Eficode Blog. Disponível em: <https://www.eficode.com/blog/en/blog/robot-framework>. Acesso em: 5 mar. 2026. 28

[37] Objective. **Robot Framework: o que é e como funciona**. 2026. Objective Insights. Disponível em: <https://www.objective.com.br/insights/robot-framework/>. Acesso em: 5 mar. 2026. 28

[38] Robot Framework. **Robot Framework**. 2026. Disponível em: <https://robotframework.org/>. Acesso em: 5 mar. 2026. 28

[39] Profissão Cloud. **O que é Katalon: plataforma de automação de testes**. 2026. Disponível em: <https://profissaocloud.com.br/glossario/o-que-e-katalon-plataforma-de-automacao-de-testes/>. Acesso em: 5 mar. 2026. 28

[40] FERRIS, M. **What is Testim automation tool, and what are its alternatives?** 2023. Qameta Blog. Disponível em: <https://qameta.io/blog/testim-automation-tool/>. Acesso em: 5 mar. 2026. 29

[41] KLUBNIKIN, A. **What is Mabl & what makes it a game-changer in test automation?** 2025. ITRex Blog. Disponível em: <https://itrexgroup.com/blog/what-is-mabl/>. Acesso em: 5 mar. 2026. 30

[42] mabl. **AI-powered test automation platform**. 2026. Disponível em: <https://www.mabl.com/>. Acesso em: 5 mar. 2026. 30

[43] Mirante Tecnologia. **Mabl: uma alternativa ao Selenium**. 2026. Disponível em: <https://www.mirante.net.br/solucoes/plataforma-para-testes-ai/mabl-e-uma-alternativa-ao-selenium/>. Acesso em: 5 mar. 2026. 30

[44] Capterra. **mabl: preço, avaliações e classificação**. 2026. Disponível em: <https://www.capterra.com.br/software/175029/mabl>. Acesso em: 5 mar. 2026. 30

[45] Functionize. **Functionize: AI-powered test automation platform**. 2026. Disponível em: <https://www.functionize.com/>. Acesso em: 5 mar. 2026. 30

[46] Gartner. **Functionize reviews and ratings**. 2026. Gartner Peer Insights. Disponível em: <https://www.gartner.com/reviews/product/functionize>. Acesso em: 5 mar. 2026. 30

[47] TestCafe. **TestCafe: cross-browser end-to-end testing framework**. 2026. Disponível em: <https://testcafe.io/>. Acesso em: 5 mar. 2026. 31

[48] Redação KingHost. **Google Lighthouse: tire suas dúvidas sobre essa ferramenta**. 2026. KingHost Blog. Disponível em: <https://king.host/blog/tecnologia/google-lighthouse/>. Acesso em: 16 jun. 2026. 33

[49] HostGator Brasil. **Hotjar: o que é e como ele pode turbinar seu site?** 2025. Blog HostGator. Disponível em: <https://www.hostgator.com.br/blog/hotjar-o-que-e/>. Acesso em: 16 jun. 2026. 34

[50] Microsoft. **Microsoft Clarity**. 2026. Microsoft Learn. Disponível em: <https://learn.microsoft.com/pt-br/startups/benefits/technical-benefits/microsoft-clarity>. Acesso em: 16 jun. 2026. 34

[51] UXCam. **Seja o time de produto que sempre tem a resposta**. 2026. Disponível em: <https://uxcam.com/br/>. Acesso em: 16 jun. 2026. 35

[52] Arimetrics. **What is Crazy Egg**. 2019. Disponível em: <https://www.arimetrics.com/en/digital-glossary/crazy-egg>. Acesso em: 16 jun. 2026. 35

[53] Crazy Egg, Inc. **Crazy Egg Website – Optimization: heatmaps, recordings, surveys & A/B testing**. 2026. Disponível em: <https://www.crazyegg.com/>. Acesso em: 16 jun. 2026. 35

[54] Applitools. **Applitools: AI-Automated Compliance Testing**. 2026. Disponível em: <https://applitools.com/>. Acesso em: 16 jun. 2026. 36

[55] Maze. **Website Testing**. 2026. Disponível em: <https://maze.co/features/live-website-testing/>. Acesso em: 16 jun. 2026. 36

[56] Optimal Workshop. **Optimal Workshop: UX Research & Design Testing Platform**. 2026. Disponível em: <https://www.optimalworkshop.com/>. Acesso em: 16 jun. 2026. 37

[57] JÚNIOR, O. A. **Arquitetura de micro serviços: uma comparação com sistemas monolíticos**. Dissertação (Trabalho de Conclusão de Curso em Sistemas de Informação) — Universidade Federal da Paraíba, João Pessoa, 2017. Disponível em: <https://repositorio.ufpb.br/jspui/bitstream/123456789/3235/1/OAJ14062017.pdf>. Acesso em: 6 mar. 2026. 39

[58] Python Software Foundation. **Python: a linguagem de programação Python**. 2026. Disponível em: <https://www.python.org/>. Acesso em: 2 jul. 2026. 44

[59] Pallets Projects. **Flask: web development, one drop at a time**. 2026. Disponível em: <https://flask.palletsprojects.com/>. Acesso em: 2 jul. 2026. 44

[60] SQLite Consortium. **SQLite: small. fast. reliable. Choose any three**. 2026. Disponível em: <https://www.sqlite.org/>. Acesso em: 2 jul. 2026. 44

[61] Microsoft. **Playwright: fast and reliable end-to-end testing for modern web apps**. 2026. Disponível em: <https://playwright.dev/>. Acesso em: 2 jul. 2026. 44

[62] OpenAI. **OpenAI Platform: API de modelos GPT**. 2026. Disponível em: <https://platform.openai.com/>. Acesso em: 2 jul. 2026. 45

[63] Excalidraw. **Excalidraw: collaborative whiteboarding tool**. 2026. Disponível em: <https://excalidraw.com/>. Acesso em: 2 jul. 2026. 46

[64] OpenProject. **OpenProject: open source project management software**. 2026. Disponível em: <https://www.openproject.org/>. Acesso em: 2 jul. 2026. 56

[65] Microsoft. **Visual Studio Code: code editing, redefined**. 2026. Disponível em: <https://code.visualstudio.com/>. Acesso em: 2 jul. 2026. 57

[66] Git. **Git**. 2026. Disponível em: <https://git-scm.com/>. Acesso em: 2 jul. 2026. 57

[67] GitHub. **GitHub: let's build from here**. 2026. Disponível em: <https://github.com/>. Acesso em: 2 jul. 2026. 57