UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Gabriele Monteiro de Medeiros

TESTES AUTOMATIZADOS DE ACESSIBILIDADE COM LLMS: UMA ANÁLISE DE ESTRATÉGIAS DE PROMPT EM PORTAIS GOVERNAMENTAIS

Recife

2026

Gabriele Monteiro de Medeiros

TESTES AUTOMATIZADOS DE ACESSIBILIDADE COM LLMS: UMA ANÁLISE DE ESTRATÉGIAS DE PROMPT EM PORTAIS GOVERNAMENTAIS

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Cleyton Vanut C. de Magalhães

Recife

2026

“A persistência é o caminho do êxito”
(Charles Chaplin)

Resumo

A utilização de Modelos de Linguagem de Grande Escala (LLMs) tem aumentado as possibilidades de aplicação da Inteligência Artificial em diferentes atividades da Engenharia de Software, incluindo a geração automatizada de artefatos de teste. Nesse contexto, este trabalho apresenta um estudo comparativo sobre a influência da Engenharia de Prompt na geração de scripts de teste automatizados de acessibilidade para plataformas web do Governo de Pernambuco. A pesquisa teve como objetivo avaliar a capacidade do modelo Gemini 3 Flash (Tier Pago) em elaborar scripts automatizados referentes a cenários reais de teste, considerando funcionalidades de acessibilidade presentes no Portal da Transparência de Pernambuco e no Portal do Governo de Pernambuco. Inicialmente, foram elaborados scripts manuais de referência utilizando Cypress, JavaScript, Cucumber, Gherkin e o padrão Page Object Model. Em seguida, foram desenvolvidas e refinadas estratégias de Engenharia de Prompt para a geração automática dos scripts pela LLM. Os artefatos produzidos foram analisados com base em critérios relacionados à legibilidade, facilidade de manutenção, quantidade de ajustes manuais necessários, utilização de seletores corretos, validação das funcionalidades de acessibilidade e tempo de execução. Os resultados demonstraram que o refinamento dos prompts contribuiu para a melhoria da qualidade dos scripts gerados, especialmente em aspectos relacionados à organização do código, utilização de seletores mais adequados e aderência aos fluxos de teste definidos. Entretanto, foi possível observar que os scripts gerados demandaram intervenções humanas para correções de lógica, ajustes de configuração e adequação de validações específicas do contexto analisado. A Engenharia de Prompt possui potencial para apoiar atividades de automação de testes, reduzindo parte do esforço de implementação, mas não elimina a necessidade da atuação de profissionais de QA na validação e adaptação dos artefatos produzidos.

Palavras-chave: Engenharia de Prompt; Testes Automatizados; Acessibilidade Web; Large Language Models.

Abstract

The use of Large Language Models (LLMs) has expanded the possibilities for applying Artificial Intelligence to various Software Engineering activities, including the automated generation of testing artifacts. In this context, this work presents a comparative study on the influence of Prompt Engineering in the generation of automated accessibility test scripts for web platforms of the Government of Pernambuco. The objective of this research was to evaluate the capability of the Gemini 3 Flash (Paid Tier) model to generate automated scripts for real testing scenarios involving accessibility features available on the Pernambuco Transparency Portal and the Pernambuco Government Portal. Initially, manual reference scripts were developed using Cypress, JavaScript, Cucumber, Gherkin, and the Page Object Model pattern. Subsequently, Prompt Engineering strategies were designed and refined to enable the automatic generation of scripts by the LLM. The generated artifacts were analyzed based on criteria related to readability, maintainability, the number of required manual adjustments, the use of correct selectors, validation of accessibility features, and execution time. The results demonstrated that prompt refinement contributed to improving the quality of the generated scripts, particularly regarding code organization, the use of more appropriate selectors, and adherence to the defined testing workflows. However, it was also observed that the generated scripts required human intervention for logic corrections, configuration adjustments, and the adaptation of validations to the specific context under analysis. Prompt Engineering has the potential to support test automation activities by reducing part of the implementation effort; however, it does not eliminate the need for QA professionals to validate and adapt the generated artifacts.

Keywords: Prompt Engineering; Automated Testing; Web Accessibility; Large Language Models.

Lista de ilustrações

Figura 1 – Pirâmide de Teste. 17
Figura 2 – Arquivo portal_da_transparencia_acessibilidade.feature 21
Figura 3 – Arquivo comuns.js 21
Figura 4 – Metodologia do trabalho 36
Figura 5 – Interface de acessibilidade do Portal da Transparência de Pernambuco 39
Figura 6 – Interface de acessibilidade do Portal do Governo de Pernambuco . 40
Figura 7 – Etapa 4: Detalhes da metodologia utilizada na Execução dos Scripts Gerados pela LLM 48

Lista de tabelas

Tabela 1 – Mapeamento comparativo entre esta pesquisa e trabalhos relacionados. 34
Tabela 2 – Principais componentes do projeto de automação do Portal da Transparência de Pernambuco 44
Tabela 3 – Principais componentes do projeto de automação do Portal do Governo de Pernambuco 46
Tabela 4 – Trechos do primeiro prompt elaborado para o Portal de transparência de Pernambuco 47
Tabela 5 – Trechos do segundo prompt elaborado para o Portal de transparência de Pernambuco 49
Tabela 6 – Trechos do terceiro prompt elaborado para o Portal do Governo de Pernambuco 51
Tabela 7 – Estrutura da resposta gerada pela LLM a partir do Prompt 1 54
Tabela 8 – Estrutura da resposta gerada pela LLM a partir do Prompt 2 57
Tabela 9 – Estrutura da resposta gerada pela LLM a partir do Prompt 3 59
Tabela 10 – Comparação entre os scripts do Portal da Transparência de Pernambuco 62
Tabela 11 – Comparação entre os scripts do Portal do Governo de Pernambuco 63

Sumário

Lista de ilustrações **5**

1 INTRODUÇÃO **9**
1.1 Justificativa **11**
1.2 Objetivos **12**
1.2.1 Objetivo Geral 12
1.2.2 Objetivo Específico 12

2 REFERENCIAL TEÓRICO **13**
2.1 Fundamentos de Teste de Software **13**
2.1.1 Papel da garantia de qualidade 13
2.1.2 Princípios de teste 14
2.1.3 Classificação dos Testes de Software 15
2.1.4 Níveis de Teste 16
2.1.5 Pirâmide de Teste 17
2.1.6 Testes End-to-End 18
2.2 Tecnologias e Estrutura da Automação de Testes **19**
2.2.1 Uso do Cypress 19
2.2.2 Visual Studio Code e Extensões 19
2.2.3 BDD, Gherkin e Cucumber 20
2.2.4 Page Object Model 22
2.2.5 Estrutura dos Scripts Automatizados 22
2.3 Fundamentos de Inteligência Artificial e Engenharia de Prompt . . **23**
2.3.1 Inteligência Artificial e Aplicações 23
2.3.2 Google Gemini 24
2.3.3 Engenharia de Prompt 24
2.3.3.1 Técnicas de Engenharia de Prompt 25
2.3.4 Large Language Models aplicada a Testes 26
2.4 Acessibilidade Web em Plataformas Governamentais **27**

3 TRABALHOS RELACIONADOS **29**
3.1 Um estudo sobre a aplicação de LLMs no teste de software **29**
3.2 A Case Study on Test Case Construction with Large Language Models: Unveiling Practical Insights and Challenges **30**
3.3 Turning Manual Tasks into Actions: Assessing the Effectiveness of Gemini-generated Selenium Tests **31**

3.4 Posicionamento desta pesquisa em relação aos trabalhos relacionados 33

4 METODOLOGIA 36
4.1 Etapa 1: Preparação do Ambiente 37
4.1.1 Definição de Stack 37
4.1.2 Definição dos Critérios de Avaliação 37
4.1.3 Definição dos Portais Governamentais para Validação de Acessibilidade . . 38
4.1.4 Acessibilidade nos Portais Governamentais Testados 38
4.2 Etapa 2: Construção dos Artefatos de Referência 40
4.2.1 Definição dos Cenários e Fluxos de Teste dos Portais Governamentais . . . 40
4.2.2 Elaboração Manual dos Scrips Automatizados para os Portais Governamentais 42
4.3 Etapa 3: Elaboração dos Prompts 47
4.4 Etapa 4: Execução dos Scripts Gerados pela LLM 48
4.5 Etapa 5: Avaliação dos Resultados 51

5 RESULTADOS 53
5.1 Iteração 1 — Portal da Transparência de Pernambuco 53
5.1.1 Artefatos Gerados pela LLM 53
5.1.2 Resultados da Execução dos Scripts 55
5.2 Iteração 2 — Portal da Transparência de Pernambuco 56
5.2.1 Artefatos Gerados pela LLM 56
5.2.2 Resultados da Execução dos Scripts 58
5.3 Iteração 3 — Portal do Governo de Pernambuco 59
5.3.1 Artefatos Gerados pela LLM 59
5.3.2 Resultados da Execução dos Scripts 60
5.4 Comparação entre os Scripts de Automação Elaborados Manualmente e os Scripts de Automação Gerados por LLM 62
5.4.1 Portal da Transparência de Pernambuco 62
5.4.2 Portal do Governo de Pernambuco 63
5.5 Discussão 64

6 AMEAÇAS À VALIDADE 66
6.1 Validade Interna 66
6.2 Validade Externa 67

7 CONSIDERAÇÕES FINAIS 68
7.1 Trabalhos Futuros 68

8 USO DE INTELIGÊNCIA ARTIFICIAL NO TRABALHO 70

REFERÊNCIAS 71

1 Introdução

Nos últimos anos, a crescente complexidade dos sistemas de software tem intensificado as exigências em torno da garantia de qualidade de desempenho, segurança, usabilidade e confiabilidade dos produtos e serviços desenvolvidos (DIAS, 2023). Nesse contexto, segundo Muller (2020), os testes de software assumem um papel fundamental de validar se o sistema atende não só às especificações técnicas, mas também às expectativas dos usuários e das partes interessadas no ambiente do produto. Diante desse cenário, muitos processos ainda dependem de testes manuais por serem considerados, inicialmente, menos custosos e efetivos. Apesar disso, a execução e a repetição manual dos testes de verificação rápida (sanity tests), de fumaça (smoke tests) e testes de regressão (regression tests) gera um esforço operacional capaz de comprometer a agilidade do ciclo de desenvolvimento e elevar os custos operacionais a longo prazo (BERNARDO; KON, 2008).

Diante disso, depender unicamente da abordagem manual é inviável no cenário de desenvolvimento atual. É neste contexto que a automação de testes surge para garantir agilidade, reprodutibilidade e maior cobertura na validação de sistemas, contribuindo para entregas mais confiáveis (SOUSA, 2023). Nesse cenário, ferramentas como o Cypress, Selenium e Playwright se destacam pela abordagem moderna, integração nativa com aplicações web mais atuais e facilidade de uso.

Nesse contexto, a acessibilidade digital se consolidou como requisito de qualidade essencial, sendo garantidas por leis que obrigam a remoção das barreiras em plataformas governamentais. Garantir que portais públicos possam ser usados por todas as pessoas com diferentes limitações é fundamental para o exercício da cidadania e para a transparência pública (CARVALHO; CAGNIN; PAIVA, 2025). No entanto, testar essas páginas de forma contínua traz desafios práticos para as equipes de qualidade. Isso ocorre porque verificar funções de acessibilidade, como o ajuste de contraste, o tamanho das letras e o comportamento dos elementos na tela, exige estratégias eficientes, que combinem a velocidade dos testes automatizados com a atenção necessária aos critérios de inclusão.

Paralelamente a isso, é possível observar a incorporação de Inteligência Artificial (IA) em diversas áreas do mercado como a área de Medicina com a análise de grandes volumes de dados clínicos (BARUFFALDI et al., 2020), na área Jurídica com revisão de de documentos legais (CABROL; ÁVALOS, 2021), e na área de engenharia de software com a aplicação de Modelos de Linguagem de Grande Porte (LLMs) para artefatos de teste de software (JUNIOR et al., 2023). De acordo com Ignácio, Oliveira

e Francez (2024) os estudos em IA estão evoluindo das teorias conceituais para soluções práticas, se tornando uma área de grande demanda no mercado. Ainda assim, embora a inteligência artificial tenha avançado consideravelmente, há um longo caminho a ser percorrido para explorar todo o potencial, incluindo o contexto da garantia de qualidade.

Nesse sentido, a utilização de Inteligência Artificial aplicada à automação de testes pode contribuir para melhorar a velocidade e a precisão na construção de scripts automatizados, auxiliando profissionais em atividades repetitivas e apoiando o processo de validação de sistemas (ANDRADE et al., 2025). A literatura recente sobre a utilização de Inteligência Artificial aplicada à automação de testes tem investigado a conversão direta de descrições de tarefas manuais e trechos de código HTML em scripts Web executáveis funcionais (PEIXOTO et al., 2025), o mapeamento da eficácia de LLMs na geração de casos de teste unitários em múltiplos projetos (LI et al., 2025) e a investigação de como o uso dessas tecnologias impacta os subfatores de qualidade da documentação técnica, demonstrando avanços na clareza e na consistência de casos de teste gerados artificialmente a partir de especificações em linguagem natural (JUNIOR et al., 2023). No entanto, a qualidade dos resultados produzidos por esses modelos depende diretamente da forma como os prompts são estruturados e contextualizados. Estudos evidenciam desafios práticos em aberto na literatura, como a ocorrência de alucinações na seleção de localizadores de elementos de interface em páginas web (PEIXOTO et al., 2025).

Diante disso, este trabalho propõe um estudo sobre a aplicação de engenharia de prompt no teste de software, considerando a construção e o refinamento de prompts para geração de scripts de testes end-to-end com a ferramenta Cypress. A pesquisa toma como referência inicial a escrita manual de scripts automatizados, utilizada para compreender os requisitos, fluxos esperados e boas práticas de automação, servindo como base para análise dos artefatos gerados por Inteligência Artificial. A partir disso, serão avaliados aspectos relacionados à produtividade, clareza, manutenibilidade e eficiência dos scripts produzidos pelas LLMs, além da necessidade de intervenções humanas durante o processo de execução, correção e refinamento dos testes gerados.

A proposta busca compreender como diferentes estratégias de engenharia de prompt podem influenciar a qualidade dos scripts automatizados gerados por LLMs, contribuindo para uma análise mais realista sobre o uso de Inteligência Artificial no contexto da Qualidade de Software, sem comprometer o controle e a criticidade exigidos na atividade de testes.

1.1 Justificativa

A escolha deste tema se justifica pela relevância atual do debate em torno da utilização de Inteligência Artificial na Engenharia de Software e mais especificamente nos processos de automação de testes. O debate na literatura recente concentra-se na capacidade real dos Modelos de Linguagem de Grande Escala (LLMs) em gerar artefatos para a área de teste de software. O trabalho propõe analisar como diferentes estratégias de construção de prompts, como Few-Shot Prompting, Expert Prompting e a combinação entre essas técnicas, podem influenciar a geração de scripts de teste automatizados em aplicações web. Essa análise fundamenta-se na necessidade de compreender a eficiência dos artefatos produzidos por inteligência artificial, avaliando o limite entre a automação autônoma e o esforço de refinamento manual.

Com o crescimento da utilização de Modelos de Linguagem de Grande Escala (LLMs) no desenvolvimento de software, torna-se necessário compreender como a construção dos prompts pode impactar diretamente a qualidade, clareza e precisão dos artefatos gerados. Nesse contexto, este trabalho busca investigar de que forma diferentes estratégias de elaboração de prompts influenciam a geração de scripts automatizados, considerando não apenas os resultados produzidos, mas também a necessidade de intervenções humanas para correções, ajustes e refinamentos dos testes gerados.

Além disso, há uma preocupação recorrente no setor sobre a ideia equivocada de que a IA pode substituir os profissionais de garantia de qualidade (QA). Este trabalho busca desconstruir essa visão, demonstrando que a IA pode atuar como uma ferramenta complementar, ampliando a produtividade e reduzindo tarefas repetitivas, sem eliminar o valor analítico, crítico e estratégico do testador humano, com o intuito de obter ambientes de teste de software mais equilibrados, com testes manuais e automatizados coexistindo da forma mais eficaz.

Assim, o estudo contribui para uma análise mais realista e técnica sobre o papel da engenharia de prompt na automação assistida por Inteligência Artificial, oferecendo reflexões úteis tanto para profissionais quanto para acadêmicos da área. A pesquisa avança integrando essa discussão ao domínio da acessibilidade digital, fornecendo um panorama prático sobre a viabilidade de utilizar LLMs para automatizar a validação de interfaces inclusivas de forma célere e sistemática.

Além disso, a escolha do Portal da Transparência de Pernambuco e do Portal do Governo de Pernambuco como ambientes experimentais se justifica pela relevância social dessas plataformas e pela presença de recursos de acessibilidade voltados à inclusão digital. Por serem canais oficiais de acesso à informação pública, é fundamental que garantam condições adequadas de uso para diferentes perfis de usuários.

O foco na validação dessas funcionalidades permite explorar cenários reais de interação em aplicações web governamentais, contribuindo para a avaliação da utilização de engenharia de prompt na geração de scripts automatizados.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo central deste trabalho é analisar a efetividade da engenharia de prompt na geração de scripts automatizados de testes end-to-end por meio de Modelos de Linguagem de Grande Escala (LLMs), avaliando os artefatos produzidos em relação a scripts elaborados manualmente para a validação de funcionalidades de acessibilidade no Portal da Transparência de Pernambuco e no Portal do Governo de Pernambuco.

1.2.2 Objetivo Específico

- Investigar, por meio de revisão da literatura, os fundamentos da automação de testes, da engenharia de prompt e da aplicação de Modelos de Linguagem de Grande Escala (LLMs) na geração de artefatos de software.
- Definir uma stack tecnológica baseada em Cypress, JavaScript, Cucumber, Gherkin e no padrão Page Object Model (POM) para o desenvolvimento dos testes automatizados.
- Elaborar manualmente scripts automatizados de testes end-to-end para funcionalidades de acessibilidade dos portais governamentais selecionados, utilizando-os como referência de comparação.
- Construir e refinar prompts utilizando técnicas de engenharia de prompt para a geração de scripts automatizados por meio da LLM Gemini 3 Flash (Tier Pago).
- Executar e analisar os scripts gerados pela LLM nos ambientes selecionados, identificando ajustes necessários, limitações e dificuldades encontradas durante sua utilização.
- Comparar os scripts elaborados manualmente e os scripts gerados pela LLM com base em critérios de legibilidade, facilidade de manutenção, quantidade de ajustes manuais, tempo de execução, utilização de seletores corretos e capacidade de validação das funcionalidades de acessibilidade.
- Avaliar a influência do refinamento dos prompts na qualidade dos scripts automatizados produzidos pela LLM.

2 Referencial Teórico

2.1 Fundamentos de Teste de Software

Os testes de software desempenham um papel essencial para assegurar que os sistemas atendam tanto às especificações técnicas quanto às necessidades dos usuários e demais partes interessadas. Segundo o Certified Tester Foundation Level Syllabus (INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD, 2025), testar software vai além da simples execução de programas para encontrar falhas: envolve um conjunto estruturado de atividades que devem estar alinhadas ao ciclo de vida de desenvolvimento (SDLC).

Nesse processo, destacam-se conceitos fundamentais como os tipos de teste, o papel da garantia da qualidade, que visa prevenir defeitos e melhorar processos, e princípios de teste que orientam as melhores práticas na área. Juntos, esses elementos ajudam a reduzir riscos, aumentar a confiabilidade e promover a entrega de produtos mais robustos e aderentes às expectativas do mercado.

2.1.1 Papel da garantia de qualidade

Ainda de acordo com o ISTQB® Certified Tester Foundation Level Syllabus 4.0 (INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD, 2025), o ciclo de vida de desenvolvimento de software (SDLC) corresponde a uma visão de alto nível que organiza e relaciona, de forma lógica e cronológica, as fases do processo de desenvolvimento. Entre os modelos mais conhecidos estão os sequenciais, como o cascata e o modelo em V; os iterativos, como o espiral e a prototipagem; e os incrementais, como o Processo Unificado. Além disso, metodologias ágeis como Scrum, Kanban, TDD (Test-Driven Development) e BDD (Behavior-Driven Development) detalham práticas que fortalecem a integração entre desenvolvimento e testes ao longo do ciclo.

No contexto do controle e da garantia da qualidade, os testes de software desempenham papel essencial ao permitir a identificação de defeitos durante o SDLC, contribuindo diretamente para a redução de falhas na produção e para a tomada de decisão sobre a evolução do projeto. Além disso, os testes representam uma forma de trazer a perspectiva do usuário ao desenvolvimento, especialmente quando a participação direta de usuários finais é inviável.

É importante diferenciar garantia da qualidade (QA) e controle da qualidade (QC). O QA é uma abordagem preventiva, orientada ao processo, que busca assegurar

que procedimentos adequados sejam seguidos para que o produto final atenda ao nível de qualidade esperado. Já o QC é uma abordagem corretiva e orientada ao produto, que utiliza técnicas como testes, simulações e prototipagem para detectar defeitos e corrigi-los antes da entrega. Os testes, nesse cenário, são considerados uma das principais ferramentas de QC.

Outro ponto essencial destacado pelo ISTQB (INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD, 2025) é o entendimento sobre erros, defeitos e falhas: erros humanos podem introduzir defeitos nos artefatos de software, que, se não identificados, podem causar falhas quando o sistema for executado. Além disso, causas externas, como fatores ambientais, também podem provocar falhas mesmo em sistemas aparentemente corretos. Para reduzir a recorrência desses problemas, é recomendada a análise de causa-raiz, que busca compreender os motivos fundamentais que levaram ao erro ou defeito, favorecendo ações preventivas em ciclos futuros.

Assim, a garantia da qualidade não atua apenas para prevenir defeitos técnicos, mas também como parte fundamental da gestão de riscos, da melhoria contínua dos processos e da entrega de valor aos usuários finais.

2.1.2 Princípios de teste

Segundo o ISTQB® Certified Tester Foundation Level Syllabus 4.0 (INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD, 2025), ao longo dos anos foram consolidados sete princípios fundamentais que orientam a prática de testes de software e ajudam profissionais a planejar e executar testes de forma mais eficiente.

O primeiro princípio afirma que os testes demonstram a presença, e não a ausência de defeitos. Embora os testes reduzam a probabilidade de defeitos não detectados, é impossível garantir que um sistema esteja completamente livre de erros (BUXTON, 1970).

O segundo princípio destaca que testes exaustivos são inviáveis, exceto em casos extremamente simples (MANNA, 1978). Em vez disso, recomenda-se priorizar casos de teste e aplicar técnicas baseadas em risco para concentrar os esforços onde são mais necessários.

O terceiro princípio aponta que testes antecipados economizam tempo e recursos. Detectar e corrigir defeitos ainda nas fases iniciais do ciclo de vida reduz custos e evita a propagação de erros para etapas posteriores (BOEHM, 1981).

O quarto princípio observa que os defeitos tendem a se concentrar em partes específicas do sistema, conhecido como fenômeno do agrupamento de defeitos ou Princípio de Pareto (ENDERS, 1975). Esse conhecimento é útil para definir prioridades de testes.

O quinto princípio alerta que os testes perdem eficácia quando repetidos sem mudanças (BEIZER, 1990). Para continuar identificando novos problemas, pode ser necessário atualizar ou criar novos testes. Ainda assim, testes repetitivos podem ter valor em contextos como testes de regressão automatizados.

O sexto princípio ressalta que os testes dependem do contexto: não existe uma abordagem única que sirva para todos os projetos. A natureza do sistema, os riscos envolvidos e o ambiente influenciam diretamente a estratégia de testes (KANER, 2011).

Por fim, o sétimo princípio trata da falácia da ausência de defeitos: mesmo que todos os requisitos sejam testados e todos os defeitos identificados sejam corrigidos, isso não garante que o sistema atenderá às necessidades do usuário ou aos objetivos de negócio (BOEHM, 1981). Por isso, além da verificação técnica, a validação com foco nas expectativas dos usuários é indispensável.

Esses princípios fornecem uma base conceitual sólida para apoiar a tomada de decisões na elaboração, execução e manutenção de testes de software ao longo de todo o ciclo de vida.

2.1.3 Classificação dos Testes de Software

De acordo com o ISTQB® Certified Tester Foundation Level Syllabus 4.0 (INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD, 2025), existem diversos tipos de testes que podem ser aplicados a projetos de software, cada um com objetivos específicos. Entre eles, destacam-se quatro tipos principais: testes funcionais, testes não funcionais, testes caixa-preta e testes caixa-branca.

Os testes funcionais são direcionados para verificar o que o sistema deve fazer, avaliando as funções especificadas que um componente ou sistema precisa executar. Seu principal objetivo é confirmar a integridade, a correção e a adequação funcional da solução desenvolvida, avaliando o funcionamento do software com base na perspectiva do usuário final ou requisitos técnicos estabelecidos.

Já o teste não funcional avalia como o sistema se comporta, ou seja, verifica atributos de qualidade que não estão ligados diretamente à funcionalidade. Segundo a classificação da norma ISO/IEC 25010, esses atributos incluem: eficiência de performance, compatibilidade, usabilidade, confiabilidade, segurança, capacidade de manutenção e portabilidade. Ainda conforme o INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD (2025), é recomendável que alguns testes não funcionais sejam planejados desde o início do ciclo de vida do projeto, pois a identificação tardia de falhas relacionadas a esses aspectos pode comprometer significativamente a entrega do produto. Além disso, testes não funcionais podem exigir ambientes específicos, como laboratórios de usabilidade.

O teste caixa-preta baseia-se nas especificações externas ao sistema, ou seja, deriva casos de teste a partir da documentação funcional e requisitos, buscando verificar se o comportamento observado corresponde ao esperado. Já o teste caixa-branca tem como foco a estrutura interna do sistema, como código-fonte, fluxos de dados e arquitetura, e seu objetivo principal é garantir uma cobertura adequada dessas estruturas pelo conjunto de testes.

É importante ressaltar que esses quatro tipos de teste podem ser aplicados em diferentes níveis do ciclo de vida do software (como testes de unidade, integração, sistema e aceitação), embora o foco de cada um possa variar conforme o estágio e os objetivos do projeto. A escolha entre técnicas e tipos de teste deve levar em conta as características do sistema em desenvolvimento e as restrições do projeto.

2.1.4 Níveis de Teste

Ainda segundo o Syllabus (INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD, 2025), os níveis de teste são cinco:

- Teste de Unidade (ou unitário) é realizado nas fases iniciais, com o objetivo de verificar partes isoladas do sistema para identificar erros rapidamente e facilitar a correção.
- Teste de Integração de Componentes ocorre quando diferentes módulos começam a interagir, sendo utilizado para verificar se a comunicação entre eles acontece corretamente. É essencial para evitar falhas na integração entre partes do sistema.
- Teste de Sistema é executado quando o sistema já está completo, com o objetivo de validar seu funcionamento como um todo, geralmente incluindo teste de ponta a ponta (ou end-to-end). Permite verificar se os requisitos funcionais e não funcionais são atendidos em cenários próximos ao uso real.
- Teste de Integração de Sistema é aplicado quando há necessidade de validar a comunicação com sistemas externos. Esse nível é importante para garantir o correto funcionamento das integrações em ambientes mais próximos do operacional.
- Teste de Aceite ocorre nas etapas finais, com o objetivo de validar se o sistema atende às necessidades do negócio. Ele é essencial para garantir que o software esteja pronto para uso sob a perspectiva do usuário ou do cliente.

Além dos níveis de teste, existem abordagens específicas que definem quando e com qual objetivo os testes são executados, como os testes de regressão e de ve-

rificação rápida. O teste de regressão tem como objetivo verificar se alterações no sistema não introduziram falhas em funcionalidades já existentes, podendo ser aplicado em diferentes níveis, sendo mais comum em testes automatizados. Já o teste de verificação rápida é uma verificação rápida realizada após pequenas alterações, com o intuito de validar se as principais funcionalidades continuam operando conforme o esperado. Em geral, o teste de verificação rápida é aplicado em níveis mais altos, como testes de sistema, por focar em fluxos essenciais do sistema.

2.1.5 Pirâmide de Teste

O Syllabus 4.0 (INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD, 2025) deixa claro que a Pirâmide de Teste é um modelo que organiza os diferentes tipos de teste de acordo com o nível de granularidade e complexidade, auxiliando na definição de estratégias de automação e na distribuição do esforço de testes. Nesse modelo, cada camada representa um grupo de testes com características específicas. À medida que se sobe na pirâmide, os testes tendem a abranger partes maiores do sistema, tornando-se menos isolados e com maior tempo de execução, como é possível observar na figura abaixo baseado no modelo original da pirâmide de testes (COHN, 2009).

Figura 1 – Pirâmide de Teste.

Fonte: Adaptado de Cohn (2009).

Na base da pirâmide estão os testes mais simples, rápidos e isolados, que verificam pequenas partes da funcionalidade. Por esse motivo, geralmente são aplicados em maior quantidade para garantir uma cobertura adequada. Na camada intermediária, estão os testes que possuem granularidade média, cobrindo mais de uma unidade

de código, mas ainda com certo nível de controle e isolamento. Já a camada superior representa testes mais amplos e complexos, como os testes de ponta a ponta (end-to-end), que validam fluxos completos do sistema. Estes testes costumam ser mais lentos e, por cobrirem uma parte maior da aplicação, são utilizados em menor quantidade.

A quantidade de camadas e sua nomenclatura podem variar conforme o modelo adotado. Uma das propostas mais conhecidas organiza a pirâmide em três níveis: testes de unidade, testes de integração e testes de ponta a ponta, sendo cada nível voltado para objetivos específicos dentro do processo de validação do software.

2.1.6 Testes End-to-End

Os testes end-to-end (E2E), também conhecidos como testes de ponta a ponta ou testes de sistema, têm como objetivo avaliar o funcionamento completo de uma aplicação a partir da integração de seus componentes. Esse tipo de teste busca validar se o sistema, como um todo, atende aos requisitos definidos, exercitando fluxos reais de uso e verificando se os elementos integrados executam corretamente as funções esperadas, como foi citado anteriormente na seção sobre Níveis de Teste.

Além da validação funcional de cenários completos, os testes E2E também podem abranger aspectos não funcionais, como usabilidade e comportamento em um ambiente representativo do uso real. Por esse motivo, são geralmente aplicados em estágios mais avançados do desenvolvimento e podem ser conduzidos por equipes independentes de teste. Dessa forma, esses testes fornecem uma visão geral do sistema, apoiando a verificação de conformidade com requisitos e auxiliando na tomada de decisão sobre sua qualidade e prontidão para uso.

Na estrutura da pirâmide de testes, os testes end-to-end estão posicionados no topo, o que implica maior custo de execução, maior tempo de processamento e maior esforço de manutenção quando comparados aos testes das camadas inferiores. No entanto, essa abordagem apresenta vantagens importantes, como a validação de fluxos completos do sistema, a identificação de falhas que só se manifestam na integração entre múltiplos componentes e a verificação do comportamento da aplicação em condições próximas ao uso real. Dessa forma, apesar de mais custosos, os testes E2E são fundamentais para garantir uma visão mais abrangente da qualidade do sistema.

Dessa forma, este trabalho possui uma abordagem de testes end-to-end (E2E) com o objetivo de validar o comportamento do Portal da Transparência de Pernambuco sob a perspectiva do usuário final.

Como o foco do estudo está na geração automatizada de scripts de teste a partir de engenharia de prompt, a utilização de testes E2E permite avaliar de forma mais realista a capacidade dos Modelos de Linguagem de Grande Escala (LLMs) em com-

preender, estruturar e implementar cenários que representem interações completas com a aplicação.

2.2 Tecnologias e Estrutura da Automação de Testes

2.2.1 Uso do Cypress

O Cypress é um framework open source desenvolvido inicialmente por Brian Mann, lançado oficialmente em 2017, voltado para automação de testes end-to-end em aplicações web (MACHADO, 2017; TAKY, 2021). Construído em Node.js e distribuído como módulo npm, utiliza JavaScript para escrita de scripts de teste e traz recursos herdados do jQuery que facilitam a manipulação da árvore DOM, eventos e requisições Ajax (MWAURA, 2021).

Diferente de outras ferramentas que executam comandos remotamente, o Cypress opera diretamente no navegador, permitindo observar e até alterar o comportamento do navegador em tempo real, o que contribui para testes mais rápidos e precisos (KHETARPAL, 2021). Além disso, a ferramenta oferece recursos como paralelização de testes, geração de relatórios via dashboard, suporte a múltiplos navegadores baseados em Chromium e possibilidade de integração com plugins e serviços pagos, como detecção de flaky tests e integração com ferramentas de gerenciamento de projetos.

O Cypress destaca-se por sua interface intuitiva, facilidade de configuração e possibilidade de realizar diferentes tipos de testes (unitários, de integração e end-to-end), o que o torna especialmente útil para desenvolvedores e profissionais de garantia de qualidade que atuam em aplicações web modernas.

2.2.2 Visual Studio Code e Extensões

De acordo com Barreiros et al. (BARREIROS et al., 2025), o Visual Studio Code (VS Code) é um editor de código-fonte multiplataforma desenvolvido pela Microsoft, amplamente reconhecido pela sua leveza, flexibilidade e grande variedade de extensões disponíveis na comunidade. Lançado em 2015, o VS Code rapidamente conquistou popularidade entre desenvolvedores de diferentes linguagens, como Python, JavaScript e C/C++, destacando-se também pelos recursos de depuração integrada, suporte a controle de versão com Git e funcionalidades avançadas de edição.

Um dos diferenciais mais relevantes do VS Code está em seu ecossistema de extensões, que possibilita personalizar o ambiente de desenvolvimento conforme as necessidades específicas de cada projeto (Microsoft, 2025). No contexto do presente trabalho, a escolha do VS Code contribuiu para otimizar a organização dos arquivos do projeto, integrar facilmente o repositório Git, automatizar fluxos de testes e facilitar

o desenvolvimento em Python, ampliando a agilidade e produtividade ao longo das etapas de implementação.

Além disso, o VS Code dispõe de diversas extensões que auxiliam no desenvolvimento e na manutenção do código. Dentre as extensões utilizadas neste trabalho, destacam-se: Bracket Pair Colorization Toggler, que melhora a visualização e organização de estruturas aninhadas; Cucumber (Gherkin) Full Support, que oferece suporte à escrita de cenários de teste no padrão Behavior-Driven Development (BDD); Cypress Snippets e ES6 Mocha Snippets, que facilitam a escrita de testes automatizados ao disponibilizar trechos de código reutilizáveis; Fold Plus, que auxilia na organização e navegação em arquivos extensos; e Material Icon Theme que melhora a identificação visual dos arquivos e pastas no ambiente de desenvolvimento.

O VS Code em conjunto com as extensões, contribuíram para tornar o processo de desenvolvimento dos testes automatizados mais estruturado, eficiente e alinhado com as boas práticas de engenharia de software.

2.2.3 BDD, Gherkin e Cucumber

Segundo Sousa et al. (SOUSA et al., 2025), o Behavior-Driven Development (BDD) é uma metodologia ágil de desenvolvimento orientada ao comportamento do sistema, que garante que os desenvolvedores, testadores e partes não técnicas interessadas usem a mesma linguagem. Essa abordagem tem contribuído para a melhoria da qualidade de software, uma vez que ele é eficaz na redução de discrepâncias entre requisitos do cliente e a implementação técnica, favorecendo a melhoria da comunicação entre as equipes, o compartilhamento de conhecimento e a construção de visão integrada do sistema, além de funcionar como uma forma de documentação viva e atualizada ao longo do desenvolvimento.

De acordo com Mughal et al. (MUGHAL, 2024), a utilização de cenários reutilizáveis em frameworks de teste contribui com a modularidade e a eficiência dos testes, especialmente em ambiente de desenvolvimento ágil. Nesse contexto, diferente das abordagens tradicionais, que podem se concentrar em detalhes técnicos, o BDD prioriza a validação do comportamento esperado do sistema pela perspectiva do usuário.

O Gherkin, por sua vez, é uma linguagem que desempenha papel central na implementação da metodologia BDD, que adota uma sintaxe simples e estruturada baseada em texto natural. Por meio de palavras-chave padronizadas, o Gherkin permite a descrição de cenários de teste de forma compreensível para humanos e passível de automação (SOUSA et al., 2025). No contexto de testes de software, Elias (ELIAS, 2023) cita que o Gherkin tem a função de padronizar a forma de descrever especificações de cenários, baseado na regra de negócio, estruturado em forma de “passo a

passo” que especifica cada interação do usuário com o sistema.

A estrutura dos cenários em Gherkin é composta por palavras-chave que representam etapas do comportamento esperado: Dado (Given), que define as pré-condições; Quando (When), que representa a ação executada; Então (Then), que valida o resultado esperado; e E (And), utilizado para complementar etapas do fluxo. Esses cenários são organizados em arquivos com extensão .feature, que descrevem funcionalidades e seus respectivos comportamentos por meio de cenários estruturados por meio de outras palavras-chave: Funcionalidade (Feature) e Cenário (Scenario).

Figura 2 – Arquivo portal_da_transparencia_acessibilidade.feature

```
# language: pt

Funcionalidade: Validação do menu de acessibilidade no Portal da Transparência PE

  Cenário: Validação do menu de acessibilidade no Portal da Transparência PE
    Dado que o usuário acessa o portal de transparência
    Quando ele abre o menu de acessibilidade
    E clica em AUMENTAR TEXTO
    E clica em ESCALA DE CINZA
    Quando o usuário clica em DESPESAS
    E o usuário clica em RECEITAS
    E o usuário clica em RECURSOS HUMANOS
    Então a interface deve estar em escala de cinza e acessível ao usuário
    E o usuário clica em LICITACOES E CONTRATOS
    E o usuário clica em RESPONSABILIDADE FISCAL
    E o usuário clica em GESTAO ESTADUAL
    E o usuário clica em PARTICIPACAO CIDADA
    Então o tamanho da fonte deve ser aumentado
    Quando ele abre o menu de acessibilidade
    E clica em DIMINUIR TEXTO
    E clica em CONTRASTE NEGATIVO
    Quando o usuário clica em GESTAO ESTADUAL
    E o usuário clica em RESPONSABILIDADE FISCAL
    E o usuário clica em LICITACOES E CONTRATOS
    Então a interface deve estar em contraste negativo e acessível ao usuário
    E o usuário clica em RECURSOS HUMANOS
    E o usuário clica em RECEITAS
    E o usuário clica em DESPESAS
    Então o tamanho da fonte deve ser diminuido
```

Fonte: Elaborada pela autora (2026).

Após a definição dos cenários, é necessário programar os steps, responsáveis por executar as ações descritas, como cliques e validações, por exemplo. Essa programação é realizada na linguagem de programação adotada no projeto, no caso deste trabalho, o JavaScript integrado ao framework de testes Cypress.

Figura 3 – Arquivo comuns.js

```
const { Given: Dado, When: Quando, Then: Entao } = require("@badeball/cypress-cucumber-preprocessor");

//Acessa o portal da transparência
Dado('que o usuário acessa o portal de transparência', () => {
  cy.visit('https://transparencia.pe.gov.br/');
});
```

Fonte: Elaborada pela autora (2026).

Para viabilizar a integração entre especificação e execução, utiliza-se o Cucumber, uma ferramenta de apoio à automação de testes que oferece suporte ao BDD. Essa ferramenta interpreta os cenários descritos em linguagem natural, por meio do

Gherkin e estabelece a correspondência entre as etapas descritas e as implementações em código que executam as ações no sistema (Cucumber, 2024). Dessa forma, atua como um elo entre a definição dos requisitos e a validação automatizada, permitindo maior rastreabilidade, legibilidade e alinhamento entre as equipes envolvidas no processo de desenvolvimento.

2.2.4 Page Object Model

Enquanto o BDD, com o uso do Gherkin e do Cucumber, facilita a organização dos cenários de teste a partir do comportamento esperado do sistema, o Page Object Model (POM) atua na estruturação do código responsável pela execução desses testes. Segundo Faustino (FAUSTINO, 2025), o POM é um padrão de projeto que separa a lógica de teste e a estrutura da interface da aplicação, mantendo a organização.

Essa abordagem busca diminuir a dependência dos testes automatizados em relação às mudanças de interface gráfica, algo comum em aplicações web. Dessa forma, o objetivo de utilizar Page Objects é organizar a inspeção dos elementos na tela, de modo que os métodos que realizam ações nestes elementos sejam simplificados, de acordo com Souza (SOUSA, 2025).

O POM é amplamente utilizado no desenvolvimento de testes automatizados, uma vez que proporciona uma organização eficiente e modular do código, o que contribui para a redução do esforço de manutenção e maior reutilização dos componentes de teste. Em cenários onde diversos testes interagem com as mesmas páginas ou elementos, essa abordagem evita duplicação de código e facilita a atualização de testes quando existem alterações na interface (FAUSTINO, 2025).

No contexto deste trabalho, o POM foi utilizado para estruturar os scripts automatizados desenvolvidos com Cypress, permitindo maior clareza, padronização e escalabilidade, essenciais para a comparação entre diferentes estratégias de testes.

2.2.5 Estrutura dos Scripts Automatizados

A estrutura dos scripts automatizados desenvolvidos neste trabalho foi organizada com o objetivo de facilitar a manutenção, reutilização e escalabilidade dos testes. Para isso, foi adotada uma arquitetura baseada na separação entre cenários de teste, implementação dos steps e centralização dos elementos da interface.

Os cenários foram descritos em arquivos com extensão `.feature`, escritos em linguagem Gherkin, contendo os fluxos de interação e os comportamentos esperados da aplicação. Esses arquivos representam a camada de especificação dos testes e permitem uma leitura mais próxima da linguagem natural.

As implementações dos steps foram desenvolvidas em arquivos JavaScript integrados ao Cypress e ao Cucumber, sendo responsáveis por executar as ações descritas nos cenários, como cliques, navegação entre páginas e validações da interface.

Além disso, foi utilizado o padrão Page Object Model (POM) para organizar os elementos da aplicação e os métodos de interação. Dessa forma, os seletores e funções relacionadas às páginas foram centralizados em arquivos específicos, reduzindo a duplicação de código e facilitando futuras manutenções.

A estrutura do projeto também foi organizada em diretórios separados para features, step definitions, pages e arquivos de configuração, permitindo maior padronização e clareza durante o desenvolvimento dos testes automatizados.

2.3 Fundamentos de Inteligência Artificial e Engenharia de Prompt

2.3.1 Inteligência Artificial e Aplicações

Definir o que é Inteligência Artificial (IA) não é uma tarefa simples, pois ao longo das últimas décadas diferentes autores propuseram visões distintas sobre seu significado e seus objetivos. De acordo com Haugeland (1985), a IA pode ser entendida como a construção de sistemas que “pensam como seres humanos”, ou seja, máquinas capazes de processar informações de forma semelhante à mente humana. Já Kurzweil (1990) descreve a IA como a criação de sistemas que “atuam como seres humanos”, desempenhando funções que exigem inteligência quando realizadas por pessoas.

Por outro lado, existem linhas de pensamento que buscam um ideal de racionalidade, medindo o sucesso do sistema não pela semelhança com o comportamento humano, mas pela capacidade de tomar decisões corretas com base nas informações disponíveis. Nesse sentido, Chamiak e McDermott (1985) defendem o desenvolvimento de sistemas que “pensam racionalmente”, utilizando modelos computacionais para simular as faculdades mentais.

Conforme observado por Russell e Norvig (2004), algumas linhas de pensamento concentram-se no processo de raciocínio e avaliam o sucesso pela semelhança ao desempenho humano, enquanto outras priorizam o comportamento apresentado pelo sistema e medem pela capacidade de alcançar um ideal abstrato de racionalidade.

A integração dessas abordagens permitiu que a IA avançasse não apenas em áreas teóricas, mas também em aplicações práticas que hoje impactam setores como saúde, indústria, educação e, no contexto deste trabalho, a automação de testes de software.

2.3.2 Google Gemini

De acordo com a Google (Google, 2025), o Gemini é a interface de um modelo de linguagem de grande escala (LLM) multimodal, ou seja, capaz de processar e gerar conteúdos em diferentes formatos, como texto, áudio e imagens. Lançado em 2023 com o nome Bard, ele vem sendo utilizado para apoiar atividades como depuração de problemas de código, aprendizagem de conceitos difíceis e auxílio na escrita de textos e e-mails.

Segundo Tagawa et al. (TAGAWA; AGUIAR; VILLANI, 2025), o Gemini é um dos modelos generativos mais avançados da atualidade e se diferencia pela capacidade de combinar linguagem natural e visão computacional para tarefas complexas, como análise de imagens e apoio à diagnósticos, demonstrando a versatilidade em diferentes ambiente profissionais.

No contexto de teste de software, é possível citar o artigo *Turning Manual Tasks into Actions: Assessing the Effectiveness of Gemini-generated Selenium Tests*, publicado na 2ª conferência Internacional ACM/IEEE sobre Software com Inteligência Artificial (AIware). O estudo analisa o uso de LLMs na geração de scripts de testes com Selenium, a partir de descrições em linguagem natural.

Os resultados indicaram que o Gemini apresentou bom desempenho na geração de scripts automatizados de teste, com sucesso em tarefas simples, como cliques e preenchimentos de formulários. No entanto, o modelo demonstrou dificuldades em cenários complexos e dinâmicos, o que pode impactar na precisão dos testes gerados.

De modo geral, o trabalho concluiu que os LLMs possuem potencial para reduzir o esforço manual da automação de testes, desde que tenha uma validação humana e aprimoramento das estratégias de engenharia de prompt para garantir uma maior precisão e confiabilidade nos resultados (PEIXOTO et al., 2025).

Diante desse contexto, o Gemini foi escolhido como modelo de linguagem base para a execução deste trabalho, considerando sua relevância atual, capacidade multimodal e aplicabilidade no apoio à geração de scripts de testes automatizados.

2.3.3 Engenharia de Prompt

Os modelos de linguagem transformam entradas de texto simples em saídas complexas, e esse processo pode ser facilitado por instruções estruturadas para orientar o comportamento das LLMs. Essa prática deu origem à engenharia de prompt que é um conjunto de técnicas e métodos para projetar, escrever e otimizar instruções para Modelos de Linguagem Generativos, de modo que as respostas do modelo sejam precisas, concretas replicáveis e factualmente corretas (NASCIMENTO, 2024).

Nascimento (NASCIMENTO, 2024) ainda cita o uso de diferentes estratégias para a elaboração de prompts, como prompts diretos, baseados em exemplos, com corrente de pensamento e abordagens. além disso, os prompts podem ser classificados por estrutura, finalidade e nível de complexidade, variando de instruções simples a solicitações mais elaboradas que orientam o raciocínio do modelo.

A Engenharia de Prompt também é caracterizada por seu caráter iterativo, exigindo melhorias contínuas para alcançar melhores resultados. Técnicas como contextualização, definição de persona, especificação de formato de saída e uso de exemplos contribuem para respostas mais eficazes. Por outro lado, prompts mal elaborados podem gerar respostas imprecisas, ambíguas ou tendenciosas, sendo necessária a verificação das informações produzidas.

Desse modo, Nascimento (NASCIMENTO, 2025) destaca a clareza, objetividade e especificidade como fatores essenciais na construção de prompts melhores. Essa prática permite explorar o potencial dos modelos de linguagem, ampliando ainda mais a aplicação em diferentes áreas.

2.3.3.1 Técnicas de Engenharia de Prompt

De acordo com Wang et al. 2025, a exploração das capacidades de aprendizado e raciocínio das LLMs depende dos prompts utilizados e das informações fornecidas para esses modelos. Nesse contexto, diversas estratégias surgiram com o objetivo de otimizar o desempenho desses modelos. Essas estratégias são denominadas como Técnicas de Engenharia de Prompt.

Entre as técnicas mais utilizadas estão:

- Zero-Shot Prompting: consiste em fornecer uma instrução básica sem exemplos adicionais para a LLM. É a estratégia mais simples utilizada nas LLMs, mas alguns modelos mais antigos tendem a apresentar resultados menos satisfatórios quando utilizam exclusivamente esse tipo de prompt;
- Few-Shot Prompting: também conhecido como aprendizado em contexto (in-context learning), essa técnica consiste em fornecer um conjunto de exemplos com as instruções da tarefa para o modelo de linguagem. É a estratégia que auxilia o modelo a compreender melhor o objerivo e o contexto na atividade, o que permite que os resultados sejam mais satisfatórios, principalmente quando relacionados à geração de código;
- Chain-of-Thought (CoT) Prompting: estratégia que orienta a LLM a produzir soluções intermediárias ou explicitar o processo de raciocínio antes de chegar ao

resultado. Essa técnica é mais utilizada para resolução de problemas que exigem raciocínio estruturado;

- Expert Prompting: estratégia de atribuir ao modelo um papel de especialista. Segundo Wang et al. 2025, essa técnica tem demonstrado aumentar a eficácia das LLMs em diversas atividades relacionadas ao desenvolvimento de software;
- Critique Prompting: técnica mais utilizada em tarefas relacionadas a código. Nessa estratégia é solicitado que o modelo identifique possíveis erros na resposta inicial, critique a solução produzida e realiza as correções necessárias.

A partir dessas técnicas, surgiram abordagens mais sofisticadas que combinam as estratégias simultaneamente. O objetivo dessas combinações é melhorar ainda mais os resultados das LMMs nas tarefas de Engenharia de Software, explorando mecanismos de raciocínio, refinamento e contextualização.

No contexto deste trabalho, esses conceitos fundamentam a elaboração e o refinamento dos prompts utilizados para a geração de scripts automatizados de acessibilidade, permitindo analisar como a combinação de estratégias de Engenharia de Prompt influenciam a qualidade dos artefatos produzidos pela LLM.

2.3.4 Large Language Models aplicada a Testes

Um Modelo de Linguagem de Grande Escala (Large Language Model), é um modelo de inteligência artificial que passa por treinamento com linguagem natural, o que permite que esses modelos desenvolvam a habilidade de capturar relações semânticas e contextuais complexas em grandes volumes, de acordo com Oliveira (OLIVEIRA, 2025).

Oliveira (OLIVEIRA, 2025) ainda afirma que em vez de comandos rígidos, os desenvolvedores descrevem suas intenções em texto e o modelo responde com uma solução provável. Essa capacidade permite aplicações inovadoras em vários cenários diferentes, com processamento de linguagem natural e produção textual automatizada.

A aplicação da LLM no contexto de teste de software tem se consolidado como um campo promissor de pesquisa e desenvolvimento dentro da engenharia de software. De modo geral, a IA busca construir sistemas capazes de executar tarefas que, tradicionalmente, exigiriam algum nível de inteligência humana. No âmbito dos testes de software, essa tecnologia abre espaço para ganhos significativos em eficiência, precisão e cobertura dos processos de teste (MAIA, 2023).

De acordo com Maia et al. (MAIA, 2023), a IA tem sido aplicada de maneira cada vez mais relevante em diferentes etapas do ciclo de teste, como na automação da geração de casos de teste, na detecção de defeitos e na priorização de cenários.

Entre as aplicações recentes, destacam-se estudos exploratórios que utilizam modelos de linguagem, como o ChatGPT, para gerar testes unitários automaticamente, demonstrando o potencial dessas ferramentas para reduzir esforço manual e tempo de desenvolvimento.

No entanto, apesar dos avanços, ainda existem lacunas na literatura quando o foco se volta para a geração automatizada de testes de aceitação, que exigem maior aderência ao comportamento esperado do sistema sob a perspectiva do usuário final. Como apontam Maia et al. (MAIA, 2023), ainda são necessários estudos que avaliem se o uso de IA nessa etapa consegue manter a qualidade, clareza e cobertura que são características essenciais para garantir que o software atenda aos requisitos definidos pelo cliente.

Assim, a integração de técnicas de LLM nos processos de teste representa não apenas uma inovação tecnológica, mas também um caminho para repensar o papel do testador, que passa a assumir uma função mais estratégica, utilizando essas ferramentas como apoio na geração e análise de artefatos de teste. Essa perspectiva está alinhada ao objetivo desta pesquisa, que investiga o uso de LLMs, analisando suas potencialidades e limitações em cenários reais de teste.

2.4 Acessibilidade Web em Plataformas Governamentais

A acessibilidade digital é a capacidade de plataformas, aplicativos e sistemas serem utilizados por todas as pessoas, independentemente de limitações físicas, sensoriais ou cognitivas. A Lei Brasileira de Inclusão (Lei nº 13.146/2015) estabelece que a acessibilidade em plataformas digitais governamentais é obrigatória, garantindo a inclusão e o acesso universal à informação.

O trabalho de Nunes et al. (2025) evidencia que serviços públicos digitais apresentam barreiras de acessibilidade, mesmo quando são prioritários, indicando a necessidade de avaliações contínuas. Dessa forma, a análise de um portal de transparência estadual permite investigar, em um cenário real, como essas barreiras podem impactar o acesso à informação por diferentes perfis de usuários.

O estudo de Carvalho, Cagnin e Paiva (2025) destaca a importância dos sistemas de governo aberto e transparência pública como instrumentos essenciais para o fortalecimento da cidadania e do controle social. Segundo os autores, essas plataformas precisam disponibilizar informações de maneira acessível e compreensível para serem efetivas, o que reforça a necessidade de garantir não apenas a disponibilidade dos dados, mas também sua usabilidade por toda a população.

Além disso, o portal selecionado oferece um ambiente propício para a aplicação

de técnicas de automação de testes, uma vez que possui diversos fluxos de interação, como consultas com filtros, navegação entre diferentes categorias de dados e visualização de informações detalhadas. Esses elementos possibilitam a construção de cenários de teste que avaliam simultaneamente aspectos funcionais e requisitos de acessibilidade.

Dessa forma, o Portal da Transparência do Estado de Pernambuco foi escolhido como a plataforma central para os testes automatizados deste trabalho por se tratar de um sistema público que precisa garantir que os cidadãos com diferentes limitações consigam acessar informações de forma autônoma.

3 Trabalhos Relacionados

Este capítulo apresenta alguns trabalhos relacionados à aplicação de Modelos de Linguagem de Grande Escala (LLMs) no contexto da Engenharia de Software, com ênfase em atividades de teste de software e engenharia de prompt. O objetivo é analisar pesquisas recentes que investigam o uso dessas tecnologias na geração de casos de teste, automação de testes e apoio às atividades de garantia da qualidade de software.

A seleção dos estudos considerou trabalhos que abordam a utilização de LLMs em cenários práticos de teste de software, bem como pesquisas que discutem estratégias de engenharia de prompt voltadas à obtenção de resultados mais precisos e consistentes. Os trabalhos analisados apresentam diferentes abordagens para utilização desses modelos, incluindo geração automatizada de casos de teste, construção de scripts de automação e definição de padrões para interação com sistemas baseados em inteligência artificial.

A análise desses estudos permite identificar contribuições, limitações e oportunidades de pesquisa ainda existentes na área, fornecendo embasamento teórico para o desenvolvimento deste trabalho e justificando a investigação sobre a aplicação da engenharia de prompt em atividades de teste de software.

3.1 Um estudo sobre a aplicação de LLMs no teste de software

Oliveira (2025) apresenta uma investigação sobre o uso de Modelos de Linguagem de Grande Escala em atividades de teste de software, com foco na geração de artefatos de teste por meio de técnicas de engenharia de prompt. O objetivo do trabalho foi elaborar e validar uma abordagem baseada em prompts reutilizáveis capazes de auxiliar atividades relacionadas a testes funcionais, testes estruturais e testes automatizados.

A autora elaborou um conjunto de diretrizes para criação de prompts destinados à geração de casos de teste organizado em três categorias: testes funcionais, testes estruturais e testes automatizados. Para cada atividade, foram definidos prompts genéricos contendo uma persona específica, instruções detalhadas sobre a tarefa a ser executada e um modelo padronizado de saída contendo informações como identificador do teste, objetivo, entradas, saídas esperadas e ambiente de teste.

Tendo foco maior na geração de script automatizado, o guia contempla a geração de testes automatizados por meio de prompts destinados à criação de scripts

executáveis a partir do código-fonte ou de especificações previamente definidas. As diretrizes orientam a utilização de frameworks específicos, a criação de assertivas adequadas e a produção de testes reprodutíveis e compatíveis com o ambiente de desenvolvimento utilizado.

A estrutura dos prompts foi desenvolvida com base nos trabalhos analisados na revisão da literatura e fundamentada em padrões reconhecidos de engenharia de prompt, especialmente o uso de personas e templates estruturados para orientar o comportamento dos modelos. Segundo a autora, essa padronização busca aumentar a consistência dos resultados gerados pelos LLMs e facilitar sua aplicação em diferentes contextos de teste de software.

A pesquisa adotou uma metodologia experimental quantitativa para avaliar a eficácia dos artefatos gerados. Foram utilizadas métricas de cobertura de código obtidas por meio da ferramenta JaCoCo e métricas de detecção de falhas obtidas através de testes de mutação utilizando a ferramenta PIT. A análise permitiu verificar o desempenho dos testes produzidos pelos LLMs em diferentes cenários e critérios de teste.

Os resultados indicaram que os LLMs possuem potencial significativo para auxiliar atividades de garantia da qualidade de software, especialmente na criação de casos de teste e scripts automatizados. Contudo, o estudo também aponta a necessidade de aperfeiçoamento das estratégias de prompt para aumentar a robustez dos testes gerados e melhorar a capacidade de detecção de defeitos.

A relevância deste trabalho para a presente pesquisa está na proposição de um conjunto sistematizado de diretrizes para construção de prompts aplicados ao teste de software. As estratégias de estruturação de prompts apresentadas por Oliveira (2025) serviram como base para a elaboração dos prompts utilizados neste estudo, especialmente no uso de personas, definição explícita da tarefa e de contexto. Dessa forma, o trabalho fornece um referencial prático para a aplicação da engenharia de prompt na geração de artefatos de teste assistida por inteligência artificial.

3.2 A Case Study on Test Case Construction with Large Language Models: Unveiling Practical Insights and Challenges

O trabalho de Lima Junior et al (2023) investiga a aplicação de Modelos de Linguagem de Grande Escala (LLMs) na construção de casos de teste para software. O estudo foi conduzido por meio de um estudo de caso utilizando uma aplicação real denominada Da.tes, uma plataforma web voltada à conexão entre startups, investidores e empresas. O objetivo da pesquisa foi avaliar a viabilidade prática da utilização de LLMs na geração de casos de teste e identificar benefícios e limitações dessa abordagem.

Para a geração dos casos de teste, os autores desenvolveram uma estratégia baseada em engenharia de prompt. O processo foi dividido em três etapas: geração de requisitos, definição das condições de teste e construção dos casos de teste. Inicialmente, uma descrição estruturada da aplicação era fornecida ao modelo, contendo informações como nome do software, objetivo, plataforma e descrição da funcionalidade analisada. Em seguida, o modelo gerava requisitos e condições de teste que serviam de base para a construção dos casos de teste completos.

A avaliação foi realizada por engenheiros de qualidade de software que compararam casos de teste produzidos por inteligência artificial com casos desenvolvidos manualmente por uma equipe de QA. Os resultados demonstraram que os casos de teste produzidos pelo GPT-3.5 apresentaram qualidade semelhante à dos casos elaborados por especialistas humanos, alcançando avaliações ligeiramente superiores em alguns critérios analisados.

Como principal contribuição, o estudo demonstra que LLMs podem atuar como ferramentas de apoio na elaboração de casos de teste, reduzindo o esforço manual nas atividades. Entretanto, os autores destacam que a qualidade dos resultados está diretamente relacionada à forma como os prompts são estruturados e ao nível de contexto fornecido ao modelo. A estratégia adotada, baseada na decomposição da tarefa em etapas sucessivas evidenciou que a organização das instruções influencia significativamente a qualidade dos artefatos produzidos.

Essa constatação é particularmente relevante para a presente pesquisa, uma vez que reforça a importância da engenharia de prompt na obtenção de resultados consistentes com LLMs. Além disso, a utilização de descrições estruturadas para fornecer contexto ao modelo serviu como referência para a elaboração dos prompts empregados neste trabalho, contribuindo para a definição de estratégias de contextualização e especificação das tarefas relacionadas à geração de artefatos de teste assistida por LLM.

3.3 Turning Manual Tasks into Actions: Assessing the Effectiveness of Gemini-generated Selenium Tests

Entre os trabalhos apresentados na trilha de Testes de Software e Garantia da Qualidade Baseados em LLMs da AIWARE 2025, o trabalho de Peixoto et al. (2025) investiga a capacidade de Modelos de Linguagem de Grande Escala (LLMs) em gerar scripts de automação de testes utilizando Selenium a partir de descrições textuais de tarefas manuais e trechos de código HTML.

O estudo tem como objetivo avaliar a efetividade do modelo Gemini 2.5 Flash

na conversão de instruções em linguagem natural em scripts automatizados capazes de reproduzir ações realizadas por usuários em interfaces web, contribuindo para a redução do esforço necessário na criação manual de testes automatizados.

A metodologia adotada consistiu na definição de oito categorias de tarefas comuns em aplicações web: busca de itens, aplicação de filtros, navegação por links, interação com carrosséis, manipulação de modais, preenchimento de formulários, adição de itens a coleções e cliques em botões. Para cada categoria foram criadas 25 instâncias distintas, totalizando 200 pares de ação e página web. Cada tarefa foi descrita em linguagem natural e combinada com um trecho de HTML extraído de páginas reais, servindo como entrada para o Gemini 2.5 Flash. O modelo recebeu instruções para gerar exclusivamente código Python utilizando Selenium, sem comentários ou explicações adicionais.

Os scripts gerados foram executados e avaliados manualmente quanto à sua executabilidade e correção funcional. Os resultados demonstraram que 87,5% dos scripts executaram corretamente na primeira tentativa, enquanto outros 10,5% tornaram-se executáveis após pequenas correções, resultando em uma taxa de executabilidade próxima de 98%. Em relação ao comportamento funcional, 51,5% dos scripts realizaram corretamente a tarefa proposta sem intervenção humana, percentual que aumentou para 80,5% após ajustes mínimos realizados pelos avaliadores. Quando desconsiderados casos bloqueados por mecanismos de proteção dos próprios sites, a taxa de sucesso funcional alcançou 99,4% após correções.

A análise dos resultados revelou que tarefas envolvendo componentes simples e estáticos, como caixas de busca, formulários e botões, apresentaram melhores taxas de sucesso. Por outro lado, elementos dinâmicos, como modais e carrosséis, mostraram-se mais desafiadores para o modelo, exigindo maior número de correções. Os autores identificaram ainda problemas relacionados à seleção incorreta de elementos, uso inadequado de configurações do Selenium e ocorrências ocasionais de alucinação, nas quais o modelo gerava seletores ou elementos inexistentes na estrutura HTML fornecida. Apesar dessas limitações, a maioria dos erros exigiu apenas pequenas modificações, indicando que os scripts produzidos pelo modelo podem servir como uma base inicial para atividades de automação de testes.

A relevância deste estudo para a presente pesquisa está na demonstração prática de como modelos generativos podem ser utilizados para produzir artefatos de automação de testes a partir de instruções em linguagem natural. Além disso, o trabalho também evidencia que a qualidade dos scripts gerados depende diretamente da clareza das instruções e do contexto fornecido ao modelo, reforçando a importância da engenharia de prompt como elemento fundamental para obtenção de resultados consistentes. Essa constatação está alinhada ao objetivo deste trabalho, que busca

investigar a aplicação de estratégias de engenharia de prompt na geração de artefatos de teste assistida por inteligência artificial.

3.4 Posicionamento desta pesquisa em relação aos trabalhos relacionados

A análise dos trabalhos relacionados permitiu identificar estratégias, desafios e oportunidades associadas à utilização de Modelos de Linguagem de Grande Escala em atividades de teste de software. Os resultados observados nessas pesquisas serviram como base para a definição da metodologia adotada neste trabalho.

O trabalho ”Um estudo sobre a aplicação de LLMs no teste de software”(OLIVEIRA, 2025) contribuiu principalmente para a estruturação dos prompts utilizados na pesquisa. A utilização de personas, a definição explícita do papel do modelo, a descrição detalhada da tarefa e a especificação do formato esperado para as respostas serviram como referência para a elaboração dos prompts empregados na geração dos scripts automatizados. Além disso, o trabalho reforçou a importância da padronização das instruções como mecanismo para aumentar a consistência dos resultados produzidos pelos LLMs.

O estudo ”A Case Study on Test Case Construction with Large Language Models: Unveiling Practical Insights and Challenges”(JUNIOR et al., 2023) evidenciou a relevância da contextualização das solicitações realizadas aos modelos de linguagem. A estratégia de decomposição das tarefas em etapas sucessivas e o fornecimento de informações detalhadas sobre o sistema influenciaram a construção dos prompts desta pesquisa, motivando a inclusão de descrições completas dos fluxos de teste, tecnologias utilizadas e critérios de validação esperados.

Por fim, o estudo ”Turning Manual Tasks into Actions: Assessing the Effectiveness of Gemini-generated Selenium Tests”(PEIXOTO et al., 2025) exerceu influência direta sobre a definição do experimento realizado. A proposta de avaliar scripts gerados automaticamente a partir de descrições em linguagem natural inspirou a comparação entre scripts produzidos manualmente e scripts gerados por LLMs. Além disso, os problemas relatados pelos autores, como seleção incorreta de elementos, necessidade de ajustes manuais e ocorrência de alucinações, motivaram a definição dos critérios de avaliação adotados nesta pesquisa, incluindo análise de seletores corretos, quantidade de ajustes necessários, facilidade de manutenção, legibilidade e validação funcional dos cenários de acessibilidade.

A tabela 1 abaixo apresenta uma comparação das pesquisas relatadas, destacando os seus respectivos objetivos, técnicas aplicadas e principais contribuições.

Tabela 1 – Mapeamento comparativo entre esta pesquisa e trabalhos relacionados.

Trabalho	**Objetivo**	**Técnicas Usadas**	**Contribuição**
Um estudo sobre a aplicação de LLMs no teste de software (OLIVEIRA, 2025)	Elaborar e validar uma abordagem baseada em prompts reutilizáveis para apoiar testes funcionais, estruturais e automatizados.	Engenharia de prompt estruturada com uso de personas, instruções detalhadas e templates de saída.	Proposição de um conjunto sistematizado de diretrizes e modelos padronizados de prompts para testes.
A Case Study on Test Case Construction with Large Language Models: Unveiling Practical Insights and Challenges (JUNIOR et al., 2023)	Avaliar a viabilidade prática da utilização de LLMs na geração de casos de teste em uma plataforma web real (*Da.tes*).	Decomposição da tarefa em etapas sucessivas (requisitos, condições de teste e construção de casos).	Demonstração de que a organização e o particionamento sequencial das instruções influenciam a qualidade dos artefatos.
Turning Manual Tasks into Actions: Assessing the Effectiveness of Gemini-generated Selenium Tests (PEIXOTO et al., 2025)	Avaliar a efetividade do modelo Gemini em converter descrições de tarefas manuais e HTML em scripts Selenium funcionais.	Engenharia de prompt combinando descrições textuais em linguagem natural e trechos de código HTML.	Mapeamento de taxas de executabilidade em diferentes componentes de interface e identificação de alucinações de seletores.
Esta pesquisa	Investigar a influência de estratégias de engenharia de prompt (Few-Shot Prompting e Expert Prompting) na geração de scripts de automação de testes end-to-end de acessibilidade web.	Contextos de acessibilidade e refinamento interativo; análise comparativa entre a escrita manual e a gerada por LLMs.	Avaliação da eficácia das LLMs na automação de critérios de acessibilidade (escala de cinza, contraste, aumento e diminuição de fonte) em plataformas governamentais de Pernambuco, avaliando manutenibilidade e executabilidade.

Fonte: Elaborada pela autora (2026).

Em resumo, embora os trabalhos relacionados utilizem técnicas consolidadas de engenharia de prompt para otimizar a geração de artefatos de teste, nenhum dos

estudos mapeados aborda os desafios específicos da acessibilidade digital. Enquanto Oliveira (2025) foca na estruturação genérica de diretrizes reutilizáveis, Lima Junior et al. (2023) na geração sequencial de cenários funcionais e Peixoto et al. (2025) na conversão de tarefas manuais de interface usando Selenium, esta pesquisa preenche uma lacuna ao direcionar o foco para a acessibilidade nas plataformas governamentais. O grande diferencial deste projeto reside na união da validação automatizada de recursos visuais críticos (como alto contraste, escala de cinza, aumento e diminuição de fontes) em portais governamentais reais de Pernambuco, avaliando o impacto das estratégias de combinação de Few-Shot e Expert Prompting junto ao esforço de refinamento no framework Cypress. Portanto, enquanto as pesquisas citadas nesse capítulo se concentram em avaliar a capacidade das LLMs gerar artefatos no cenário de testes de software, este trabalho avalia o potencial e o custo de manutenção dos scripts automatizados gerados pela LLM, voltada à garantia da cidadania digital em plataformas governamentais.

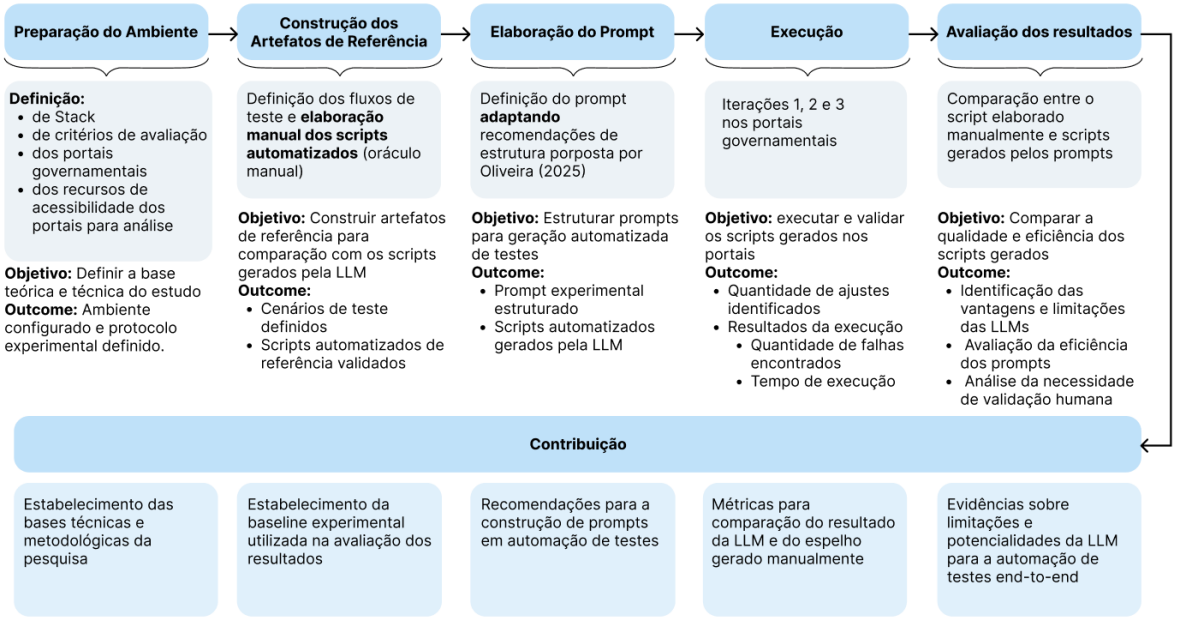
4 Metodologia

Esta pesquisa caracteriza-se como exploratória e aplicada, com abordagem qualitativa e comparativa, utilizando o estudo de caso como método de investigação. O estudo de caso foi adotado por, segundo Yin (2017), possibilitar a investigação de um fenômeno contemporâneo em um contexto real, permitindo analisar a utilização de Modelos de Linguagem de Grande Escala (LLMs) e técnicas de engenharia de prompt na geração automatizada de scripts de testes de acessibilidade em plataformas governamentais reais.

Como objeto de estudo, foram selecionados o Portal da Transparência de Pernambuco e o Portal do Governo de Pernambuco, ambos escolhidos por disponibilizarem funcionalidades de acessibilidade voltadas à adaptação da interface para diferentes perfis de usuários. A pesquisa buscou analisar como modelos de linguagem de grande escala (LLMs) podem auxiliar na construção de scripts automatizados para validação dessas funcionalidades.

A metodologia foi organizada em cinco etapas principais, conforme apresentado na Figura 4.

Figura 4 – Metodologia do trabalho



Fonte: Elaborada pela autora (2026).

4.1 Etapa 1: Preparação do Ambiente

4.1.1 Definição de Stack

Na primeira etapa, foi realizada a definição da stack tecnológica utilizada no desenvolvimento da pesquisa. Para a implementação dos testes automatizados, foi adotado o framework Cypress, escolhido por oferecer recursos voltados à automação de testes end-to-end em aplicações web.

A linguagem JavaScript foi utilizada para o desenvolvimento das ações, validações e demais componentes dos scripts automatizados. Para a definição dos cenários de teste, foi adotada a abordagem Behavior-Driven Development (BDD), utilizando a linguagem Gherkin para a descrição dos fluxos em linguagem natural e o Cucumber para integrar os cenários escritos em arquivos ”feature”à implementação dos testes automatizados.

Além disso, foi empregado o padrão de projeto Page Object Model (POM), com o objetivo de organizar os elementos da interface e os métodos de interação em estruturas reutilizáveis, favorecendo a manutenção e a legibilidade dos scripts

Por fim, o modelo Gemini 3 Flash (Tier Pago) foi utilizado como Largue Language Model (LLM) para a geração dos scripts automatizados analisados ao longo do estudo. Os experimentos foram executados entre os meses de maio e junho de 2026, utilizando a interface web oficial do Gemini. Durante a realização dos testes, foram mantidas as configurações padrão do modelo, sem ajuste explícito dos parâmetros de temperatura ou top-p.

A partir da definição da stack tecnológica, foram estabelecidos os critérios que seriam utilizados para avaliar os scripts produzidos manualmente e aqueles gerados pela LLM.

4.1.2 Definição dos Critérios de Avaliação

Os critérios de avaliação, posteriormente utilizados na análise comparativa entre os scripts automatizados elaborados manualmente e os scripts gerados pela LLM, tiveram como objetivo verificar a quanlidade e a aderência dos artefatos produzidos em relação ao comportamento das plataformas analisadas, além da capacidade da ferramenta em gerar códigos automatizados.

Os aspectos de legibilidade e facilidade de manutenção foram utilizados para verificar se os scripts apresentavam uma organização clara, nomenclatura compreensível e estrutura compatível com as boas práticas de automação. Esses critérios permitiram avaliar o nível de compreensão e manutenção dos artefatos produzidos.

Também foram analisados a quantidade de ajustes manuais necessários e a utilização de seletores corretos. Esses critérios tiveram como objetivo identificar o grau de intervenção humana exigido após a geração dos scripts, considerando correções em seletores, adequações de lógica, ajustes de configuração e outras modificações necessárias para deixar os testes executáveis.

Por fim, foram avaliados o tempo de execução dos testes e a validação das funcionalidades de acessibilidade. Para isso, verificou-se se os scripts eram capazes de validar corretamente os comportamentos relacionados ao aumento e diminuição de fonte, aplicação de escala de cinza e de contraste negativo.

Após a definição dos critérios de avaliação, foram selecionadas as plataformas governamentais utilizadas como ambiente experimental da pesquisa.

4.1.3 Definição dos Portais Governamentais para Validação de Acessibilidade

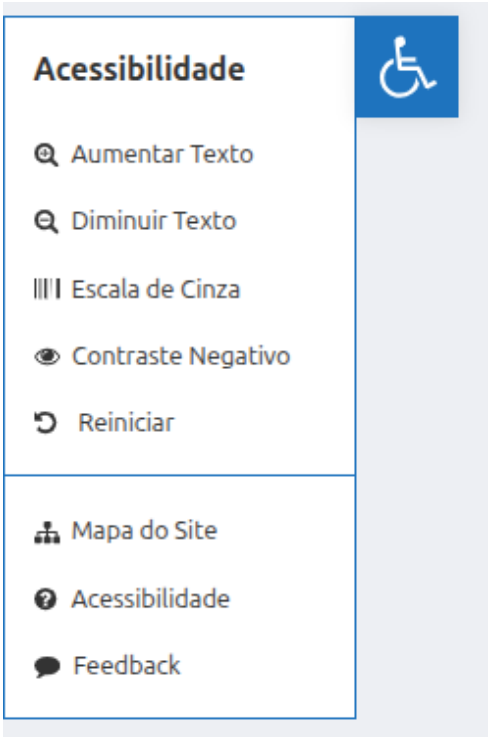
Com o objetivo de avaliar a capacidade das LLMs na geração de scripts automatizados para funcionalidades de acessibilidade, foram selecionadas duas plataformas governamentais do Estado de Pernambuco: o Portal da Transparência de Pernambuco e o Portal do Governo de Pernambuco. A escolha dessas plataformas ocorreu devido à disponibilidade de recursos de acessibilidade voltados à personalização da interface, permitindo a elaboração de cenários de teste relacionados à persistência e validação dessas funcionalidades durante a navegação.

Além disso, por serem canais de acesso a informações, é importante que esses portais garantam a acessibilidade e a inclusão digital de diferentes perfis de usuários.

4.1.4 Acessibilidade nos Portais Governamentais Testados

O Portal da Transparência de Pernambuco possui uma interface que atende às principais recomendações de acessibilidade indicadas pelas Diretrizes de Acessibilidade de Conteúdo Web (W3C), visando garantir maior inclusão no acesso às informações públicas.

Figura 5 – Interface de acessibilidade do Portal da Transparência de Pernambuco



Fonte: Elaborada pela autora (2026).

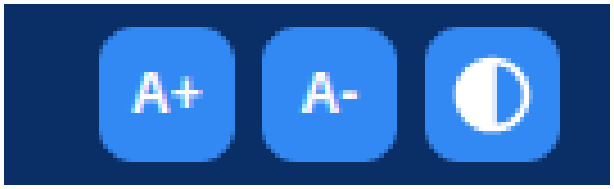
A plataforma dispõe de recursos que permitem a personalização da experiência do usuário, como botões para aumento e diminuição da fonte, aplicação de escala de cinza e ativação de contraste negativo, garantindo o ajuste da visibilidade dos textos e conteúdo do Portal conforme as necessidades. Além disso, a plataforma oferece funcionalidades como mapa do site e navegação por meio de teclas de atalho, permitindo a interação sem o auxílio de mouse.

Outro recurso importante é a integração com a ferramenta VLibras, que traduz conteúdos digitais do Português para a Língua Brasileira de Sinais (Libras), tornando a plataforma acessível também para pessoas com deficiência auditiva.

O foco deste trabalho será limitado às funcionalidades de acessibilidade relacionadas ao ajuste de tamanho de fonte, escala de cinza e contraste negativo do Portal da Transparência de Pernambuco.

Já o Portal do Governo de Pernambuco também possui uma interface voltada para a acessibilidade dos usuários, como é possível observar na imagem acima, porém com uma quantidade mais limitada de recursos quando comparado ao Portal da Transparência de Pernambuco. A plataforma disponibiliza funcionalidades de aumento e diminuição de fonte, além da ativação de contraste negativo, permitindo que os usuários ajustem a visualização do conteúdo conforme suas necessidades de leitura e visibilidade.

Figura 6 – Interface de acessibilidade do Portal do Governo de Pernambuco



Fonte: Elaborada pela autora (2026).

No entanto, o Portal não possui outros recursos adicionais de acessibilidade, como escala de cinza, mapa do site acessível, teclas de atalho ou integração com ferramentas de tradução em Libras, tornando sua estrutura de acessibilidade mais simplificada em relação ao Portal da Transparência de Pernambuco.

As diferenças entre as funcionalidades disponíveis nos dois portais motivaram a elaboração de fluxos de teste específicos para cada plataforma.

4.2 Etapa 2: Construção dos Artefatos de Referência

Na segunda etapa, foram definidos os cenários de teste relacionados às funcionalidades de acessibilidade dos portais selecionados. Os scripts manuais foram desenvolvidos com o objetivo de servir como referência para a comparação com os artefatos gerados pelas LLMs, sendo estruturados com Cypress, Cucumber, Gherkin e Page Object Model, seguindo boas práticas de organização, reutilização de código e manutenção.

Durante a elaboração dos scripts, os cenários passaram por adaptações decorrentes de limitações técnicas identificadas durante a implementação e execução dos testes automatizados.

4.2.1 Definição dos Cenários e Fluxos de Teste dos Portais Governamentais

Para a definição dos cenários, inicialmente, foram definidos quatro cenários de teste que combinavam a validação das funcionalidades de acessibilidade com fluxos de negócio existentes no Portal da Transparência de Pernambuco. A proposta consistia em validar recursos como aumento e diminuição de fonte, escala de cinza, contraste negativo, reinicialização da interface e mapa do site durante a execução de consultas reais disponibilizadas pelo portal.

O primeiro cenário previa a ativação dos recursos de aumento e diminuição de fonte, seguida da navegação pelo módulo de despesas públicas. Além da validação das alterações visuais, o fluxo incluía a consulta das despesas gerais e a verificação da regra de negócio em que o valor da despesa fixada atualizada deveria ser superior ao valor da despesa empenhada.

O segundo cenário combinava a utilização da funcionalidade de escala de cinza com consultas relacionadas às receitas estaduais, incluindo a aplicação de filtros e validação dos resultados retornados pela plataforma.

O terceiro cenário envolvia a ativação do contraste negativo, aumento de fonte e posterior reinicialização das configurações de acessibilidade. Em seguida, seriam executadas consultas relacionadas às licitações e contratos.

Por fim, o quarto cenário utilizava o mapa do site como ponto de navegação para acessar funcionalidades relacionadas à Gestão Estadual e Responsabilidade Fiscal, realizando consultas e validações dos dados apresentados.

Porém, durante a implementação desses cenários, foi identificado um desafio relacionado à utilização de iframes pelo Portal da Transparência de Pernambuco. Parte significativa das consultas e relatórios disponibilizados pela plataforma é carregada por meio de componentes externos incorporados ao portal através de iframes, muitos deles provenientes de serviços de Business Intelligence hospedados em domínios distintos da aplicação principal.

Essa característica gerou limitações para a automação dos testes utilizando Cypress, uma vez que a ferramenta possui restrições para interação com elementos localizados em iframes de origem diferente da aplicação principal devido às políticas de segurança entre domínios (Cypress Documentation, 2026). Como consequência, não foi possível realizar a validação direta de diversos elementos previstos nos cenários originalmente definidos.

As tentativas de interação com esses componentes resultaram em falhas de execução, inviabilizando a automação completa dos fluxos de negócio inicialmente planejados. Dessa forma, foi necessário redefinir os cenários de teste para concentrar a validação exclusivamente nas funcionalidades de acessibilidade presentes na interface principal do portal.

Após a identificação das limitações relacionadas aos iframes, os cenários foram modificados para validar a persistência das configurações de acessibilidade durante a navegação entre as páginas do portal.

O fluxo de validação para o Portal da Transparência de Pernambuco passou a contemplar as funcionalidades de aumento e diminuição de fonte, escala de cinza e contraste negativo, seguida da navegação pelas seções de despesa, receitas, rescursos humanos, licitações e contratos, responsabilidade fiscal, gestão estadual e participação cidadã do menu de barra superior do sistema. O objetivo foi verificar se as alterações permaneciam mesmo após a troca de páginas, reproduzindo um comportamento real de utilização da plataforma.

Essa reformulação permitiu a construção de cenários totalmente automatizá-

veis, mantendo o foco da pesquisa na avaliação das funcionalidades de acessibilidade e na comparação entre os scripts desenvolvidos manualmente e aqueles gerados pelas LLMs.

Já para o Portal do Governo de Pernambuco foi adotada uma estratégia semelhante à utilizada na versão final dos cenários do Portal da Transparência, já considerando as limitações identificadas durante a etapa anterior.

O fluxo do teste foi estruturado para validar as funcionalidades de aumento e diminuição de fonte e contraste negativo durante a navegação entre diferentes páginas acessadas pelo menu principal.

Os cenários contemplaram a ativação dos recursos de acessibilidade, a navegação pelas seções de serviços, notícias e governo, além da verificação da persistência das alterações visuais ao longo da navegação. Como mencionando anteriormente, diferentemente do Portal da Transparência, essa plataforma não disponibiliza da funcionalidade escala de cinza, o que resultou em um conjunto mais reduzido de cenários de teste.

Os cenários definidos nessa etapa constituíram uma base para a elaboração manual dos scripts automatizados.

4.2.2 Elaboração Manual dos Scrips Automatizados para os Portais Governamentais

Conforme apresentado na Tabela 2, o projeto de automação desenvolvido para o Portal da Transparência de Pernambuco foi estruturado visando promover maior organização, reutilização de código e facilidade de manutenção.

O arquivo ”portal_da_transparencia_acessibilidade.feature”foi utilizado para descrever o cenário de teste em linguagem Gherkin, representando o comportamento esperado das funcionalidades de acessibilidade durante a navegação entre os módulos do portal. Nesse cenário, foram contempladas ações relacionadas ao aumento e diminuição do tamanho da fonte, aplicação de escala de cinza, ativação de contraste negativo e validação da persistência dessas configurações ao longo da navegação.

As interações e validações dos recursos de acessibilidade foram centralizadas no arquivo ”acessibilidade.page.js”. Nesse componente foram implementados métodos responsáveis pela abertura do menu de acessibilidade, acionamento dos recursos disponíveis e validações das alterações visuais da interface. Para a funcionalidade de aumento e diminuição da fonte, foi adotada uma estratégia baseada na comparação da altura do elemento ”body”antes e após a execução da ação, utilizando o método ”getBoundingClientRect()”. Já para a validação da escala de cinza e do contraste ne-

gativo, foram realizadas verificações sobre propriedades CSS e sobre a visibilidade de elementos gráficos e textuais da aplicação, garantindo que os conteúdos permanecessem acessíveis ao usuário após a aplicação dos filtros visuais.

A navegação entre os módulos do portal foi implementada no arquivo ”consultas.page.js”, no qual foram implementados os seletores e métodos responsáveis pelo acesso às páginas Despesas, Receitas, Recursos Humanos, Licitações e Contratos, Responsabilidade Fiscal, Gestão Estadual e Participação Cidadã. Essa separação permitiu isolar as regras de navegação das validações de acessibilidade, favorecendo a modularização da solução.

Os arquivos ”acessibilidade.js”e ”consultas.js”foram responsáveis pela implementação dos Step Definitions, estabelecendo a ligação entre os passos definidos em Gherkin e os métodos implementados nos respectivos Page Objects. Dessa forma, os cenários descritos na linguagem de negócio puderam ser executados automaticamente pelo Cypress, mantendo uma estrutura alinhada às boas práticas de automação orientada por comportamento.

Durante a execução do cenário relacionado à validação do contraste negativo, foi identificado um comportamento que impactou a estabilidade da automação. Mesmo a funcionalidade de contraste negativo sendo aplicada corretamente à interface, alguns elementos da página não ficaram visíveis, atrapalhando a experiência do usuário.

Como consequência, as asserções de visibilidade implementadas no Cypress retornavam erro, uma vez que os elementos continuavam presentes no DOM, porém ocultos ao usuário. Nesse contexto, o script cumpriu seu papel ao identificar uma inconsistência na experiência de navegação com o contraste negativo habilitado, evidenciando que a funcionalidade não preservava integralmente a acessibilidade da interface durante a navegação.

Tabela 2 – Principais componentes do projeto de automação do Portal da Transparência de Pernambuco

Componente	**Responsabilidade**	**Trecho do Script**

Estrutura de Arquivos	Organização dos arquivos do projeto	`cypress/`
		`  e2e/`
		`  _acessibilidade.feature`
		`  support/`
		`  pages/`
		`  acessibilidade.page.js`
		`  consultas.page.js`
		`  step_definitions/`
		`  acessibilidade.js`
		`  consultas.js`

Arquivo portal_da_transparencia_acessibilidade.feature	Definição dos cenários de teste em Gherkin.	`Cenário: Validação do menu de acessibilidade`
		`  Quando clica em AUMENTAR TEXTO`
		`  Então o tamanho da fonte deve ser aumentado`

Arquivo acessibilidade.page.js	Implementação das interações e validações relacionadas aos recursos de acessibilidade.	`clicaAumentarTexto()`
		`  ...`
		`  cy.get('.pojo...').click()`
		`}`

Arquivo consultas.page.js	Implementação da navegação pelos módulos do portal.	`clicaDespesas()`
		`  cy.get('#menu-item-4548 ...').click()`
		`}`

Arquivo acessibilidade.js	Associação entre os passos Gherkin e os métodos do Page Object.	`Quando('clica em AUMENTAR TEXTO')`
		`  acessibilidadePage.`
		`  clicaAumentarTexto();`
		`})`

Arquivo consultas.js	Implementação dos passos de navegação da aplicação.	`Quando('o usuário clica em DESPESAS')`
		`  consultasPage.clicaDespesas();`
		`})`

Fonte: Elaborada pela autora (2026).

Dessa forma, o cenário evidenciou a capacidade dos testes automatizados de atuar não apenas como validação funcional, mas também como apoio à identificação de defeitos relacionados à usabilidade e acessibilidade da aplicação.

O projeto de automação desenvolvido para o Portal do Governo de Pernambuco,

conforme apresentado na Tabela 3, também foi estruturado com base na combinação entre BDD, Gherkin, Cypress e o padrão Page Object Model, mantendo a mesma organização arquitetural adotada no Portal da Transparência de Pernambuco para facilitar a comparação entre os resultados obtidos.

O arquivo ”gov_pe_acessibilidade.feature”foi utilizado para descrever os cenários de validação das funcionalidades de acessibilidade presentes na plataforma. Os testes contemplaram ações de aumento e diminuição de fonte, ativação e desativação do contraste negativo e navegação pelos menus Serviços, Notícias, Governo e Início, verificando a persistência das preferências de acessibilidade ao longo da utilização do portal.

No arquivo ”acessibilidade.page.js”foram implementados os métodos responsáveis pelas interações com os controles de acessibilidade e pelas validações das alterações visuais da interface. Diferentemente da estratégia utilizada no Portal da Transparência, a validação do aumento e da redução da fonte foi realizada por meio da captura e comparação direta da propriedade CSS ”font-size”dos elementos da página, uma vez que o comportamento do portal era o aumento da fonte e não do ”body”da plataforma. Para isso, foi utilizado o método ”window.getComputedStyle()”, permitindo a obtenção dos valores numéricos da fonte antes e após a aplicação das alterações de acessibilidade.

As funcionalidades de navegação foram implementadas no arquivo ”consultas.page.js”, responsável por implementar os seletores e métodos de acesso às diferentes páginas do portal. Essa abordagem permitiu centralizar as interações com os elementos da interface e reduzir o impacto de possíveis alterações estruturais da aplicação.

Por fim, os arquivos de Step Definitions foram utilizados para associar os passos descritos em linguagem Gherkin às implementações dos respectivos Page Objects. Essa separação contribuiu para a legibilidade dos cenários, além de favorecer a manutenção dos scripts e a reutilização das funcionalidades implementadas ao longo do projeto.

Tabela 3 – Principais componentes do projeto de automação do Portal do Governo de Pernambuco

Componente	**Responsabilidade**	**Trecho do Script**

Estrutura de Arquivos	Organização dos arquivos do projeto	`cypress/`
		`  e2e/`
		`  gov_pe_acessibilidade.feature`
		`  support/`
		`  pages/`
		`  acessibilidade.page.js`
		`  consultas.page.js`
		`  step_definitions/`
		`  acessibilidade.js`
		`  consultas.js`

Arquivo gov_pe_ acessibilidade.feature	Definição dos cenários de teste em Gherkin	`Cenário: Validação do menu de acessibilidade no Gov PE`
		`Quando clica em AUMENTAR TEXTO`
		`E clica em CONTRASTE NEGATIVO`

Arquivo acessibilidade.page.js	Interações e validações dos recursos de acessibilidade	`clicaAumentarTexto() {`
		`  ...`
		`  cy.get('[aria-label=...]')`
		`.click();`
		`}`

Arquivo consultas.page.js	Navegação pelos menus do Portal do Governo	`clicaServicos() {`
		`  cy.get('a.css-883erp[href...]')`
		`  .click();`
		`}`

Arquivo Step Definitions de Acessibilidade	Implementação das ações e validações automatizadas	`Quando('clica em AUMENTAR TEXTO')`
		`  acessibilidadePage.`
		`  clicaAumentarTexto();`
		`})`

Arquivo Step Definitions de Consultas	Implementação dos passos de navegação	`Quando('o usuário clica em SERVIÇOS')`
		`  consultasPage.`
		`  clicaServicos();`
		`}`

Fonte: Elaborada pela autora (2026).

Durante a execução dos cenários elaborados para o Portal do Governo de Pernambuco não foram identificadas falhas relacionadas à navegação ou à aplicação dos recursos de acessibilidade avaliados. As funcionalidades de aumento e diminuição de fonte, bem como a ativação e desativação do contraste negativo, permaneceram consistentes ao longo da navegação entre as páginas selecionadas. Dessa forma, todos os passos previstos nos cenários foram executados com sucesso, permitindo a valida-

ção completa dos comportamentos esperados pela automação.

Os scripts produzidos nessa etapa constituíram os artefatos de referência utilizados posteriormente na análise comparativa com os scripts gerados pelas LLMs.

4.3 Etapa 3: Elaboração dos Prompts

Na terceira etapa, foi desenvolvido o prompt experimental utilizado na interação com o Gemini. A construção do primeiro prompt adaptou as recomendações de Oliveira (2025) para engenharia de prompt, incluindo papel da LLM, tarefa que a LLM deve executar, definição de contexto, descrição detalhada do fluxo de teste, como é possível observar nos trechos de prompt na tabela abaixo.

Tabela 4 – Trechos do primeiro prompt elaborado para o Portal de transparência de Pernambuco

Componente do Prompt	**Descrição**	**Trecho de Exemplo**
Papel do modelo	Define o comportamento esperado da LLM	“Você é um especialista em QA e automação de testes.”
Objetivo	Define a tarefa que a LLM deve executar	“Gere um script de teste automatizado que valide o menu acessibilidade do Portal da Transparência de Pernambuco com base no fluxo abaixo”
Contexto	Apresenta informações sobre o sistema e tecnologia utilizada	“Sistema: Portal da Transparência de Pernambuco; Framework: Cypress; Linguagem: JavaScript; Padrão: Page Object, BDD, Gherkin; Tipo: Teste E2E”
Fluxo de teste	Descreve passo a passo das ações e validações	“1. O usuário acessa o portal da transparência; 2. Acessa o menu de acessibilidade; 3. Clica em...”

Fonte: Elaborada pela autora (2026).

O prompt é uma combinação das técnicas de Zero-Shot Prompting e Expert Prompting da engenharia de prompt, onde cada componente do prompt possui uma função específica, mas nenhum exemplo de mapeamento foi dado. O papel do modelo foi de definir o comportamento da LLM, direcionando a resposta para o contexto

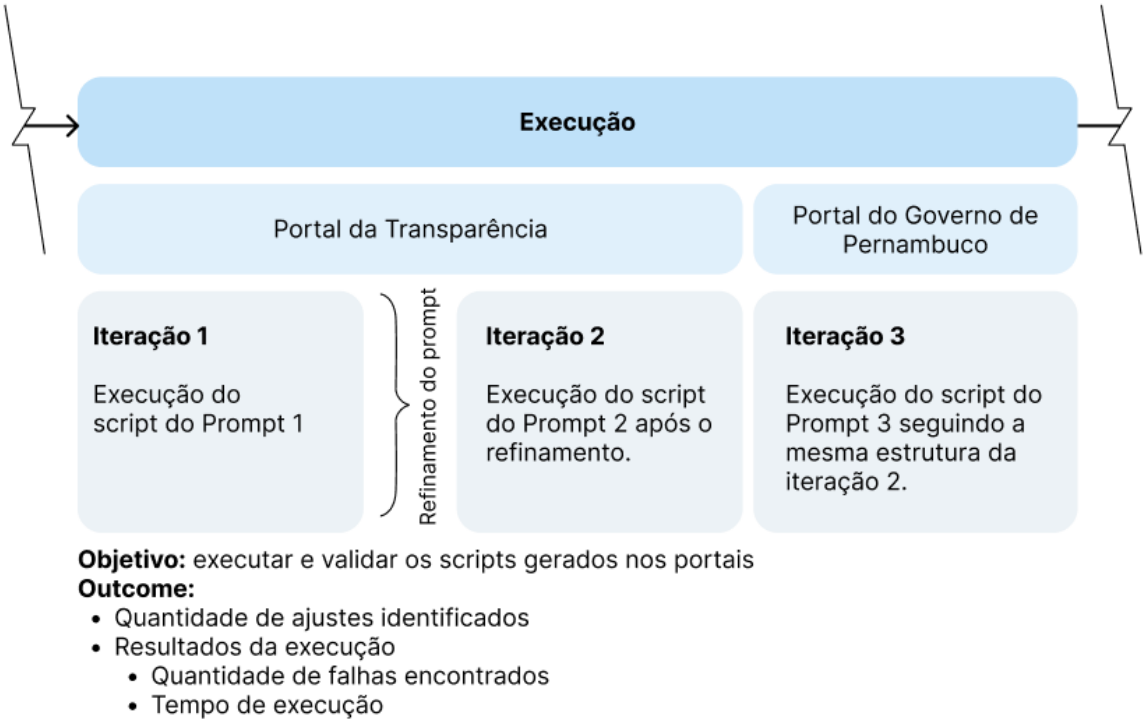
de garantia de qualidade e automação de testes. O objetivo descreve a tarefa que deveria ser executada pelo modelo, especificando a geração de scripts automatizados para validação das funcionalidades de acessibilidade do Portal da Transparência de Pernambuco.

O contexto foi utilizado para fornecer informações relacionadas ao sistema testado, às tecnologias empregadas e ao padrão de desenvolvimento esperado, incluindo o uso do Cypress, JavaScript, BDD, Gherkin e Page Object Model. Já o fluxo de teste foi responsável por descrever passo a passo as ações e validações esperadas durante a execução dos cenários automatizados.

Essa divisão em componentes buscou tornar o prompt mais claro, permitindo reduzir ambiguidades e melhorar a qualidade dos scripts gerados pela LLM.

4.4 Etapa 4: Execução dos Scripts Gerados pela LLM

Figura 7 – Etapa 4: Detalhes da metodologia utilizada na Execução dos Scripts Gerados pela LLM



Fonte: Elaborada pela autora (2026).

Na quarta etapa, que está representada na imagem acima, o script gerado pela LLM para o Portal da Transparência, foi executado no ambiente de testes utilizando os cenários definidos anteriormente. Essa etapa foi organizada em iterações, permitindo refinar os prompts e ajustar os scripts conforme os problemas identificados durante a execução.

No Portal da Transparência de Pernambuco, inicialmente foi executado o script gerado a partir do primeiro prompt. Após a identificação de muitas falhas relacionadas aos seletores, foi realizado um refinamento do prompt, seguido de uma nova geração e execução dos scripts para o Portal de Transparência de Pernambuco, como é detalhado na tabela abaixo.

Tabela 5 – Trechos do segundo prompt elaborado para o Portal de transparência de Pernambuco

Componente do Prompt	**Descrição**	**Trecho de Exemplo**
Papel do modelo	Define o comportamento esperado da LLM	“Você é um especialista em QA e automação de testes.”
Objetivo	Define a tarefa que a LLM deve executar	“Gere um script de teste automatizado que valide o menu acessibilidade do Portal da Transparência de Pernambuco com base no fluxo abaixo”
Contexto	Apresenta informações sobre o sistema e tecnologia utilizada	“Sistema: Portal da Transparência de Pernambuco; Framework: Cypress; Linguagem: JavaScript; Padrão: Page Object, BDD, Gherkin; Tipo: Teste E2E”
Fluxo de teste	Descreve passo a passo das ações e validações	“1. O usuário acessa o portal da transparência; 2. Acessa o menu de acessibilidade; 3. Clica em...”
Elementos da interface	Fornece seletores e referências da UI	“<a data-action=’resize-plus’>”
Restrições	Limita o comportamento da LLM para evitar erros	“Não inventar seletores”
Boas práticas	Define padrões esperados de código	“Usar boas práticas, validações e organização”

Fonte: Elaborada pela autora (2026).

Com relação à primeira iteração, foram adicionados mais e componentes na estrutura do prompt 2. O componente “Elementos da interface” foi incluído para fornecer seletores e referências específicas da interface do sistema com o objetivo de diminuir problemas relacionados à identificação incorreta de elementos pela LLM.

O componente “Restrições” foi utilizado para limitar determinados comporta-

mentos da LLM durante a geração do código, com o objetivo de aumentar a precisão dos scripts gerados automaticamente. Já o componente “Boas práticas” teve como finalidade orientar a LLM quanto à organização do código, incentivando aproximar os scripts gerados das práticas recomendadas para automação de testes com Cypress, BDD e Page Object Model.

Posteriormente, essa mesma estrutura de prompt foi adaptada e aplicada ao Portal do Governo de Pernambuco, permitindo analisar o comportamento da LLM em um segundo ambiente governamental com recursos de acessibilidade diferentes. Após as adaptações realizadas, o prompt pôde se classificado como uma combinação das técnicas Few-Shot Prompting e Expert Prompting, uma vez que foram dados seletores de mapeamento para o modelo copiar. Os trechos do prompt 3 estão presentes na Tabela 6. Considerando o objetivo de gerar scripts automatizados de acessibilidade, optou-se pela utilização das técnicas Few-Shot Prompting e Expert Prompting, por apresentarem maior aderência à geração de código estruturado e à contextualização de cenários de teste.

Tabela 6 – Trechos do terceiro prompt elaborado para o Portal do Governo de Pernambuco

Componente do Prompt	**Descrição**	**Trecho de Exemplo**
Papel do modelo	Define o comportamento esperado da LLM	“Você é um especialista em QA e automação de testes.”
Objetivo	Define a tarefa que a LLM deve executar	“Gere um script de teste automatizado que valide o menu acessibilidade do Portal da Transparência de Pernambuco com base no fluxo abaixo”
Contexto	Apresenta informações sobre o sistema e tecnologia utilizada	“Sistema: Portal do Governo de Pernambuco; Framework: Cypress; Linguagem: JavaScript; Padrão: Page Object, BDD, Gherkin; Tipo: Teste E2E”
Fluxo de teste	Descreve passo a passo das ações e validações	“1. Acessa o portal do Governo de Pernambuco; 2. Clica em AUMENTAR TEXTO; 3. Clica em...”
Elementos da interface	Fornece seletores e referências da UI	“<a href=”/app/catalog/carta-de-servicos”class=”css-883erp»Serviços”
Restrições	Limita o comportamento da LLM para evitar erros	“Não inventar seletores”
Boas práticas	Define padrões esperados de código	“Usar boas práticas, validações e organização”

Fonte: Elaborada pela autora (2026).

4.5 Etapa 5: Avaliação dos Resultados

Na quinta etapa, foi realizada a análise comparativa entre os scripts elaborados manualmente e os scripts gerados pela LLM. A avaliação considerou critérios previamente definidos durante a preparação do ambiente, permitindo verificar o nível de aderência das soluções geradas automaticamente em relação à estrutura esperada.

A comparação envolveu aspectos como legibilidade, necessidade de ajustes manuais, tempo de execução e esforço de manutenção.

Além disso, também foi realizada a comparação entre os prompts utilizados nas iterações 1 e 2, buscando identificar como o refinamento das instruções impactou a qualidade dos scripts gerados.

Os resultados obtidos foram organizados em tabelas comparativos, registros de execução e análises descritivas, permitindo identificar vantagens, limitações e contribuições da utilização de engenharia de prompt na automação de testes end-to-end voltados à acessibilidade em plataformas governamentais.

5 Resultados

Este capítulo apresenta os resultados obtidos a partir da execução dos prompts elaborados durante a pesquisa, bem como a análise dos scripts gerados pelo Gemini 3 Flash. Inicialmente são apresentados os resultados individuais de cada iteração, seguidos por uma comparação entre os artefatos gerados e os scripts elaborados manualmente.

5.1 Iteração 1 — Portal da Transparência de Pernambuco

5.1.1 Artefatos Gerados pela LLM

O primeiro prompt foi executado utilizando o modelo Gemini 3 Flash (Tier Pago) no dia 03 de maio de 2026, em uma nova sessão, com o objetivo de evitar interferências de interações anteriores e garantir maior controle experimental durante a geração dos scripts automatizados.

A resposta dada pela LLM demonstrou aderência às instruções do primeiro prompt apresentado na Tabela 4, apresentando uma estrutura com o padrão Behavior-Driven Development (BDD), contendo arquivos feature, step definitions e page objects, além da utilização de JavaScript integrado ao framework Cypress, Page Object Model (POM) e Gherkin, como é possível observar na tabela abaixo:

Tabela 7 – Estrutura da resposta gerada pela LLM a partir do Prompt 1

Componente Gerado	**Descrição**	**Trecho da Resposta**

Estrutura de Arquivos	Organização dos arquivos do projeto em BDD e POM	`cypress/`
		`  e2e/`
		`  accessibility.feature`
		`  support/`
		`  step_definitions/`
		`  accessibility_steps.js`
		`  page_objects/`
		`  TransparencyPage.js`

Especificação BDD	Definição dos fluxos de teste em linguagem natural	“Funcionalidade: Validação do Menu de Acessibilidade”

Page Object Model	Implementação dos seletores e métodos da interface	`class TransparencyPage {`
		`  get btnAccessibility()`
		`  return`
		`  cy.get('#barra-acessibilidade');`

Step Definitions	Implementação das ações e validações automatizadas	`` `When("acessa o menu de acessibilidade", () => {` `  TransparencyPage...'' ``
Notas de Implementação de Especialista:	Recomendações técnicas e boas práticas de automação	“Identificadores (Selectors): No código acima, utilizei seletores genéricos”

Fonte: Elaborada pela autora (2026).

Conforme apresentado na Tabela 7, a LLM trouxe uma proposta de organização dos arquivos do projeto em diretórios específicos, separando os cenários Gherkin, os arquivos de Step Definitions e os componentes do padrão Page Object Model.

No componente “Especificação BDD”, a LLM gerou o arquivo “accessibility.feature”, contendo a descrição dos cenários de acessibilidade em linguagem Gherkin. Os cenários incluíram ações relacionadas ao aumento e diminuição de fonte, aplicação de escala de cinza, ativação de contraste negativo e navegação entre páginas do Portal da Transparência de Pernambuco, seguindo o fluxo de teste descrito no Prompt 1.

Além disso, nos componentes “Page Object Model” e “Step Definitions”, a LLM também gerou implementações contendo seletores, métodos de interação e validações automatizadas. Os scripts incluíram funções para abertura do portal, interação com os botões de acessibilidade e validações relacionadas às alterações visuais da interface.

Por fim, a resposta também apresentou um componente denominado “Notas de Implementação de Especialista”, no qual a LLM forneceu recomendações adicionais

relacionadas ao uso de seletores, estratégias de espera e possíveis melhorias para validação de acessibilidade, indicando preocupações com boas práticas de automação e manutenção dos testes.

Apesar da estrutura gerada tenha demonstrado aderência às tecnologias e ao fluxo solicitado, foi necessário verificar se os artefatos produzidos eram efetivamente executáveis no ambiente de testes.

5.1.2 Resultados da Execução dos Scripts

A execução dos scripts gerados a partir do primeiro prompt mostrou que, embora o Gemini 3 Flash (Tier Pago) tenha elaborado uma estrutura inicial coerente e alinhada com as boas práticas de automação, a solução não se mostrou plenamente funcional sem a realização de ajustes manuais.

Foi necessária as configurações adicionais no ambiente de teste com a instalação e configuração do Cucumber para integração com o Cypress, etapa que não foi explicitamente detalhada na resposta da LLM. Além disso, foi preciso ajustar o arquivo de configuração cypress.config.js para garantir a correta abertura da aplicação durante a execução dos testes.

Para a interface da aplicação, foi necessário ajustar as dimensões de visualização para a garantir a visibilidade dos elementos da barra superior, importantes para a execução dos cenários de teste. Também foi preciso adicionar uma função de tratamento de erro no arquivo e2e.js para suprimir a exceção “$ is not a function”, a qual não impactava diretamente o fluxo testado, mas causava falhas na execução automatizada.

Também foi possível observar que os seletores utilizados pela LLM não correspondiam aos elementos reais da aplicação. Dessa forma, foi necessário substituir diversos seletores no arquivo TransparencyPage.js, incluindo os seletores relacionados ao menu de acessibilidade e às funcionalidades de aumento e diminuição de fonte, escala de cinza e contraste negativo, adequando às classes utilizadas pela interface.

Foram realizados ajustes nos cenários em Gherkin, principalmente relacionados à padronização textual dos nomes das seções do sistema, respeitando o padrão correto exigido pela aplicação. Também foram necessárias modificações nos steps definitions, como a adequação de padrões de parâmetros e a implementação de passos que não haviam sido previamente definidos pela LLM.

Com relação às validações, foram identificadas inconsistências na verificação do aumento e da redução do tamanho da fonte, o que demandou ajustes nas funções responsáveis por essas validações. Além disso, foi necessário implementar uma nova função específica para validar corretamente a redução da fonte.

Além disso, a validação do contraste negativo ocorreu de forma indevida, uma vez que a abordagem inicial não considerava a visibilidade dos elementos na interface. Para contornar esse comportamento, foi necessário adicionar uma lógica de verificação que inclui validações adicionais sobre propriedades visuais dos elementos (como fill e opacity em elementos SVG) e a confirmação da visibilidade de conteúdos textuais críticos na interface.

Esses resultados indicam que, embora a LLM tenha sido capaz de gerar uma estrutura inicial consistente, a solução exigiu diversas intervenções humanas, implicando tempo adicional para a identificação das inconsistências, compreensão dos problemas e implementação das correções adequadas. Dessa forma, os problemas identificados durante a execução serviram de base para o refinamento do prompt utilizado na segunda iteração

5.2 Iteração 2 — Portal da Transparência de Pernambuco

5.2.1 Artefatos Gerados pela LLM

O segundo prompt, com o devido refinamento representado na Tabela 5, também foi utilizado no modelo Gemini 3 Flash (Tier Pago), no dia 06 de maio de 2026 e em uma nova sessão, e, diferentemente da primeira iteração, este prompt foi elaborado com a intenção de reduzir a criação de seletores genéricos.

A resposta da LLM apresentou melhorias em relação à primeira iteração, principalmente na definição dos seletores, como pode ser observado na nabela abaixo.

Tabela 8 – Estrutura da resposta gerada pela LLM a partir do Prompt 2

Componente Gerado	**Descrição**	**Trecho da Resposta**
Arquivo de Funcionalidade	Definição dos fluxos de teste em linguagem natural	“Funcionalidade: Acessibilidade no Portal da Transparência de Pernambuco”
Page Object Model	Implementação dos seletores e métodos da interface	`class TransparenciaPage { elements = { btnAcessibilidade: () => cy.get('.pojo-a11y-toolbar-toggle a'),`
Step Definitions	Implementação das ações e validações automatizadas	`` `When("acessa o menu de acessibilidade", () => { TransparencyPage...' ``
Notas Técnicas do Especialista	Recomendações técnicas e boas práticas de automação	“Seletores Dinâmicos: O menuItem(texto) utiliza o texto visível...”

Fonte: Elaborada pela autora (2026).

Dessa vez a LLM não trouxe a estrutura de arquivos que estava presente na resposta do prompt anterior. Entretanto, assim como no Prompt 1, a solução gerada utilizou Cypress integrado ao padrão Page Object Model (POM), Gherkin e estrutura baseada em Behavior-Driven Development (BDD).

Entre os principais artefatos gerados, a LLM produziu um arquivo ”feature”contendo os cenários de acessibilidade em linguagem Gherkin, com ações relacionadas ao aumento e diminuição de fonte, aplicação de escala de cinza, ativação de contraste negativo e persistência dos recursos de acessibilidade durante a navegação entre os menus do portal, como foi solicitado no prompt.

Além disso, a resposta apresentou uma implementação mais detalhada para o arquivo ”TransparenciaPage.js”, contendo seletores específicos baseados em atributos reais da interface, como ”data-action”e classes relacionadas ao plugin de acessibilidade utilizado pelo portal, como ”.pojo-a11y-toolbar-toggle a”. Também foram implementados métodos responsáveis pela abertura do menu de acessibilidade, interação com os botões da interface e validações relacionadas ao estado visual da aplicação.

Nos arquivos de ”Step Definitions”, a LLM gerou definições responsáveis pela execução das ações descritas no cenário Gherkin, incluindo navegação dinâmica pelos menus da aplicação e validações relacionadas à persistência das funcionalidades de acessibilidade.

Além da geração dos scripts automatizados, a resposta também apresentou um

conjunto de “Notas Técnicas do Especialista”, contendo recomendações relacionadas ao uso de seletores dinâmicos por meio da função ”menuItem(texto)”, persistência das configurações de acessibilidade e utilização de parâmetros, com o objetivo de evitar problemas relacionados ao comportamento dos menus suspensos da aplicação.

5.2.2 Resultados da Execução dos Scripts

A execução dos scripts gerados a partir do segundo prompt demonstrou uma melhoria em relação à organização e aderência às instruções fornecidas, especialmente no que se refere à estrutura do código e à definição das validações. Ainda assim, a solução gerada pelo Gemini 3 flash (Tier Pago) não se mostrou completamente funcional sem a realização de ajustes manuais, evidenciando limitações semelhantes às observadas no primeiro prompt.

Assim como no prompt anterior, foram necessárias configurações adicionais no ambiente de testes. A instalação e integração do Cucumber com o Cypress precisaram ser realizadas manualmente, uma vez que essa etapa também não foi explicitamente detalhada pela LLM. Além disso, ajustes no arquivo cypress.config.js foram necessários para garantir o correto carregamento da aplicação durante a execução dos testes.

Também foi necessário ajustar as dimensões de visualização para assegurar a exibição da barra superior da aplicação, elemento essencial para a execução dos cenários propostos. Assim como o resultado do primeiro prompt, também foi incluído um tratamento de erro no arquivo e2e.js para evitar falhas decorrentes da exceção “$ is not a function”, que compromete a execução.

No que se refere às validações, foram identificadas inconsistências na verificação do aumento de fonte. A função inicialmente gerada pela LLM considerava uma classe genérica para validação, sendo necessário ajustá-la para refletir corretamente a classe aplicada pela interface no estado de aumento de fonte, garantindo assim maior precisão na validação do comportamento esperado.

A validação do contraste negativo apresentou as mesmas limitações de visualização, sendo necessário adicionar novamente a função dos scripts criados manualmente, que incorpora validações relacionadas às propriedades visuais dos elementos e a confirmação da visibilidade de conteúdos textuais relevantes na página.

De forma geral, os resultados obtidos com o segundo prompt indicam que, apesar do maior nível de detalhamento das instruções ter contribuído para uma resposta mais alinhada aos requisitos, ainda houve necessidade de intervenções humanas para ajustes de configuração, refinamento das validações e adequação ao comportamento real da aplicação. Isso reforça a importância da validação humana no uso de LLMs para geração de testes automatizados, mesmo em cenários com prompts mais estru-

turados.

5.3 Iteração 3 — Portal do Governo de Pernambuco

5.3.1 Artefatos Gerados pela LLM

A mesma estrutura de prompt refinada foi executada utilizando o modelo Gemini 3 Flash (Tier Pago) também em uma nova sessão, no dia 21 de Maio de 2026, com o objetivo de avaliar o comportamento do prompt e da LLM em um segundo ambiente governamental com funcionalidades de acessibilidade diferentes das presentes no Portal da Transparência de Pernambuco.

A resposta gerada pela LLM demonstrou aderência à estrutura refinada do prompt, apresentando novamente uma organização baseada em Behavior-Driven Development (BDD), utilizando arquivos ”feature”, ”step definitions”e ”page objects”, integrados ao framework Cypress e ao padrão Page Object Model (POM).

Tabela 9 – Estrutura da resposta gerada pela LLM a partir do Prompt 3

Componente Gerado	**Descrição**	**Trecho da Resposta**
Cenário em Gherkin	Definição dos fluxos de teste em linguagem natural	“Funcionalidade: Validação do Menu de Acessibilidade”
Page Object	Implementação dos seletores e métodos da interface	`class AcessibilidadePage { \\Seletores fornecidos na especificação get btnAumentarTexto() { return cy.get('button[aria-label="Aumentar fonte A+"]'); } get btnDiminuirTexto() { return cy.get('button[aria-label="Diminuir fonte A-"]'); ... }`
Step Definitions / Código Cypress	Implementação das ações e validações automatizadas	`` `Given('que eu acesso o portal do Governo de Pernambuco', () => { acessibilidadePage.visitarPortal; \\Armazena o tamanho da fonte padrão do sistema antes das alterações TransparencyPage...'' ``
Boas Práticas Aplicadas neste Script	Recomendações técnicas e boas práticas de automação	“Conversão Numérica Estrita: No método guardarEValidarFonteAumentada, usamos...”

Fonte: Elaborada pela autora (2026).

Diferentemente da Iteração 1, a LLM utilizou os seletores reais fornecidos no prompt, especialmente atributos ”aria-label”e caminhos ”href”, reduzindo a utilização de seletores genéricos. Além disso, a resposta apresentou validações voltadas às propriedades CSS da interface, utilizando comandos como ”have.css”e verificações relacionadas ao tamanho da fonte e contraste visual.

No componente referente à especificação BDD, O Gemini gerou cenários relacionados à persistência das configurações de acessibilidade durante a navegação entre páginas do Portal do Governo de Pernambuco, contemplando ações de aumento e diminuição de fonte, ativação e remoção de contraste negativo e validações relacionadas à navegação pelos menus da plataforma.

No componente “Page Object”, foram implementados métodos responsáveis pelas interações com os botões de acessibilidade e pelos menus principais do portal, além da criação de funções para captura dinâmica do tamanho atual da fonte da aplicação. A resposta também incluiu validações baseadas em propriedades CSS aplicadas ao elemento ”body”, buscando verificar alterações visuais relacionadas ao contraste e tamanho da fonte.

Já nos “Step Definitions”, a LLM gerou a implementação das ações descritas em Gherkin, incluindo interações com menus, manipulação das funcionalidades de acessibilidade e validações automatizadas utilizando a comparação do tamanho da fonte antes e depois das interações do usuário.

Por fim, a resposta apresentou novamente um conjunto de “Boas Práticas Aplicadas”, no qual a LLM descreveu recomendações relacionadas à conversão numérica de valores CSS, reutilização de seletores reais da interface e persistência do estado das funcionalidades de acessibilidade durante a navegação entre páginas. Entre essas recomendações, destacou-se a utilização exclusiva de seletores presentes na estrutura HTML fornecida no prompt, evitando a criação de elementos inexistentes e aproximando os scripts do comportamento real da aplicação.

Após a análise dos artefatos produzidos, o script foi executado para verificar a aderência ao comportamento do portal.

5.3.2 Resultados da Execução dos Scripts

O prompt voltado para o Portal do Governo de Pernambuco foi executado em uma nova sessão, com o objetivo de evitar que interações anteriores influenciassem a resposta da LLM e, assim, garantir maior controle experimental e imparcialidade nos resultados obtidos.

A execução dos scripts gerados demonstrou que a LLM foi capaz de produzir uma estrutura inicial compatível com o fluxo solicitado, incluindo cenários em Gherkin,

implementação de steps e validações relacionadas aos recursos de acessibilidade da plataforma. Entretanto, assim como observado nos testes anteriores, a solução gerada não se mostrou completamente funcional sem a realização de intervenções manuais.

Novamente, foi necessário instalar e configurar manualmente a integração entre Cucumber e Cypress, uma vez que essa etapa não foi explicitamente detalhada pela LLM. Também foi necessário ajustar o arquivo *cypress.config.js* para garantir que a aplicação fosse carregada corretamente durante a execução automatizada dos testes.

Além disso, necessário adicionar um tratamento no arquivo e2e.js para impedir que a exceção “$ is not a function”, não relacionada diretamente ao fluxo validado, interrompesse a execução dos testes.

Mesmo com a indicação no prompt de que a aplicação possuía elementos repetidos na interface, a LLM gerou comandos que ocasionaram o erro *cy.click() can only be called on a single element*, indicando que múltiplos elementos estavam sendo retornados pelos seletores utilizados. Para corrigir esse comportamento, foi necessário refinar os identificadores da barra superior, adicionando a classe *a.css-883erp* aos seletores utilizados nos scripts automatizados.

Também foram identificadas inconsistências nas validações de contraste negativo. No step responsável pela verificação da acessibilidade da interface em modo de contraste negativo, a LLM gerou a instrução *.or('exist')*, resultando no erro *or is not a function*. Dessa forma, a validação precisou ser ajustada para uma verificação compatível com o comportamento suportado pelo Cypress.

Outra limitação observada esteve relacionada à validação do aumento de fonte. A estratégia inicialmente gerada pela LLM utilizava o tamanho da fonte aplicado ao elemento *body*, abordagem que havia funcionado corretamente no Portal da Transparência de Pernambuco. No entanto, no Portal do Governo de Pernambuco o elemento *body* não sofria alteração de tamanho durante a utilização do recurso de acessibilidade. Assim, foi necessário modificar a lógica da função responsável pela captura do tamanho da fonte, passando a utilizar elementos de texto da própria página para realizar a validação.

Além disso, a lógica originalmente criada para validar a redução do tamanho da fonte não correspondia ao comportamento real da plataforma. Em razão disso, foi necessário ajustar a implementação do step responsável por verificar o retorno ao tamanho de fonte esperado após as interações de aumento e redução de texto, adequando a comparação ao comportamento efetivamente apresentado pela interface.

De forma geral, os resultados obtidos demonstram que, embora a LLM tenha sido capaz de gerar uma estrutura inicial coerente com os requisitos fornecidos, ainda houve necessidade de intervenções humanas para ajustes de configuração, refina-

mento de seletores, correção de validações e adaptação ao comportamento real da aplicação. Esses resultados reforçam que, mesmo com prompts mais estruturados e detalhados, a participação humana continua sendo essencial para garantir a confiabilidade e o funcionamento adequado dos testes automatizados gerados por modelos de linguagem.

5.4 Comparação entre os Scripts de Automação Elaborados Manualmente e os Scripts de Automação Gerados por LLM

A comparação entre os scripts manuais (oráculos manuais) e os scripts gerados pela LLM permitiu avaliar o impacto da engenharia de prompt na qualidade dos artefatos produzidos, assim como identificar limitações da geração automática de testes em diferentes contextos de acessibilidade.

5.4.1 Portal da Transparência de Pernambuco

Tabela 10 – Comparação entre os scripts do Portal da Transparência de Pernambuco

Critério	**Oráculo Manual**	**Iteração 1**	**Iteração 2**
Boa legibilidade	Sim	Sim	Sim
Facilidade de manutenção	Sim	Sim	Sim
Quantidade de ajustes manuais	-	19	5
Tempo de execução do cenário, após os ajustes (segundos)	27	28	26
Seletores corretos	Sim	Não	Sim
Validação de aumento de fonte	Sim	Não	Sim
Validação de diminuição de fonte	Sim	Não	Sim
Validação de escala de cinza	Sim	Sim	Sim
Validação de contraste negativo	Sim	Não	Não

Fonte: Elaborada pela autora (2026).

No Portal da Transparência de Pernambuco, observou-se uma evolução significativa entre as Iterações 1 e 2. Em ambos os casos, presentes na Tabela 10, os scripts gerados apresentaram boa legibilidade e facilidade de manutenção, características também observadas no script oráculo manual. Entretanto, a primeira iteração demandou 19 ajustes manuais para que pudesse ser executada adequadamente, en-

quanto a segunda iteração necessitou de apenas 5 ajustes, representando uma redução expressiva no esforço de correção. Essa melhoria evidencia o impacto positivo do refinamento do prompt na qualidade dos artefatos produzidos pela LLM.

Com relação aos critérios funcionais, a primeira iteração apresentou falhas na utilização dos seletores, na validação de aumento e diminuição de fonte e na validação do contraste negativo. Apenas a validação da escala de cinza apresentou comportamento equivalente ao script oráculo manual. Já na segunda iteração, a LLM passou a utilizar seletores corretos e implementou adequadamente as validações relacionadas ao aumento e à diminuição da fonte, mantendo o funcionamento da escala de cinza. Contudo, a validação do contraste negativo continuou apresentando inconsistências quando comparada ao comportamento real da aplicação. Em relação ao tempo de execução, após os ajustes realizados, o script oráculo manual foi executado em 27 segundos, enquanto a primeira iteração levou 28 segundos e a segunda 26 segundos.

Esses resultados indicam que a segunda iteração conseguiu alcançar desempenho semelhante ao do script manual, além de apresentar maior aderência aos critérios estabelecidos.

5.4.2 Portal do Governo de Pernambuco

Tabela 11 – Comparação entre os scripts do Portal do Governo de Pernambuco

Critério	**Óráculo Manual**	**Iteração 3**
Boa legibilidade	Sim	Sim
Facilidade de manutenção	Sim	Sim
Quantidade de ajustes manuais	-	7
Tempo de execução do cenário, após os ajustes (segundos)	14	12
Seletores corretos	Sim	Não
Validação de aumento de fonte	Sim	Não
Validação de diminuição de fonte	Sim	Não
Validação de contraste negativo	Sim	Sim

Fonte: Elaborada pela autora (2026).

No Portal do Governo de Pernambuco, a comparação entre o script oráculo manual e a terceira iteração demonstrou um comportamento distinto, como é possível observar na Tabela 11. Apesar de o script gerado pela LLM apresentar boa legibilidade

e facilidade de manutenção, foram necessários 7 ajustes manuais para adequação ao comportamento da aplicação. Além disso, a LLM não conseguiu implementar corretamente os seletores utilizados pelo portal, nem as validações relacionadas ao aumento e à diminuição de fonte.

Diferentemente do Portal da Transparência de Pernambuco, o Portal do Governo de Pernambuco não possui a funcionalidade de escala de cinza, impossibilitando a comparação desse critério. Ainda assim, observou-se que a principal dificuldade da LLM permaneceu relacionada aos seletores, especialmente aqueles que estavam duplicados, incluindo os do Menu de Navegação da prataforma.

Quanto ao tempo de execução, o script oráculo manual foi executado em 14 segundos, enquanto a terceira iteração executou em 12 segundos. Embora o tempo tenha sido ligeiramente inferior, esse resultado não representa necessariamente maior qualidade do script, uma vez que diversas validações precisaram ser corrigidas manualmente para refletir o comportamento esperado da aplicação.

De forma geral, os resultados demonstram que a utilização de engenharia de prompt contribuiu para melhorar a qualidade dos scripts gerados pela LLM ao longo das iterações, especialmente no Portal da Transparência de Pernambuco. A redução da quantidade de ajustes manuais e a correção de problemas relacionados aos seletores evidenciam a importância do refinamento das instruções fornecidas ao modelo. Entretanto, as dificuldades observadas nas validações de acessibilidade indicam que a geração automática ainda depende de validação humana para garantir aderência ao comportamento real das aplicações testadas.

5.5 Discussão

Os dados apresentados nas tabelas comparativas entre os scripts dos Portais mostram que o Gemini 3 Flash tem uma capacidade intermediária nos cenários avaliados. Ao analisar o trabalho de Oliveira (2025), cujo objetivo foi propor diretrizes e prompts reutilizáveis, os resultados deste TCC confirmam que o modelo do Google é muito bom para seguir arquiteturas estruturadas. O código gerado respeitou com precisão a divisão de arquivos em Gherkin, Step Definitions e Page Object Model. Além disso, Oliveira (2025) e esta pesquisa revelam limitações das LLMs.

Esse comportamento também apoia a conclusão de Lima Junior et al. (2023), que demonstrou que o particionamento e a organização das instruções influenciam diretamente a qualidade dos resultados da LLM. No Portal da Transparência, a evolução da Iteração 1 para a Iteração 2 (onde o prompt foi refinado) diminuiu drasticamente os erros de 19 para apenas 5 ajustes manuais. Contudo, este estudo vai além do trabalho de Lima Junior et al. (2023) ao mostrar que, mesmo entregando os seletores e contex-

tos reais na instrução, como foi feito na Iteração 3 no Portal do Governo, o modelo ainda erra por não entender a fundo o comportamento visual da interface. Isso gerou problemas práticos de automação nesta pesquisa, como a repetição de elementos que causou o erro de múltiplos cliques e os comandos de lógica inválidos.

Em relação às contribuições sobre executabilidade, este trabalho conversa com o estudo de Peixoto et al. (2025), que mapeou o sucesso do Gemini em transformar tarefas manuais e HTML em scripts. Embora os códigos gerados nesta pesquisa tenham sido executadas após as correções manuais, os resultados acendem um alerta importante: avaliar o sucesso da IA apenas pelo fato de o script ser executado até o fim esconde falhas de validação.

Para o dia a dia da engenharia de software, fica claro que a combinação de técnicas como Few-Shot Prompting e Expert Prompting ajuda a estruturar os testes, mas não substitui o profissional de QA, transformando seu papel de quem escreve o código para quem revisa o que a IA fez. Confiar totalmente nos scripts gerados por inteligência artificial em plataformas governamentais cria uma falsa sensação de que o site obedece às leis da LBI e do eMAG. Portanto, o ganho de tempo e a manutenibilidade ainda dependem de uma revisão humana para garantir que critérios como escala de cinza, contraste e alteração de fonte funcionem de verdade para o cidadão.

6 Ameaças à validade

Os resultados obtidos nesta pesquisa estão sujeitos a limitações que podem influenciar sua interpretação. Para reduzir esses impactos, foram adotadas estratégias de controle experimental durante a execução dos testes e análise dos resultados. As principais ameaças à validade são apresentadas a seguir.

6.1 Validade Interna

A validade interna está relacionada à confiança de que os resultados observados foram realmente causados pelos fatores investigados e não por variáveis externas.

Uma possível ameaça refere-se à própria natureza probabilística das LLMs. Mesmo utilizando o mesmo prompt, modelos generativos podem produzir respostas diferentes em execuções distintas. Para reduzir esse efeito, cada prompt foi executado em uma nova sessão do Gemini 3 Flash (Tier Pago), minimizando a influência de interações anteriores e garantindo maior consistência na geração dos scripts. Ainda assim, foram observadas algumas variações entre as respostas produzidas durante as iterações. Como exemplo, a primeira iteração apresentou uma proposta para a estruturação de arquivos, enquanto as outras iterações não incluiram esse detalhamento, mesmo mantendo a estrutura inicial do prompt.

Outra ameaça está associada à necessidade de ajustes manuais realizados após a geração dos scripts. Em todos os cenários analisados foram necessárias intervenções para correção de seletores, adequação das validações, configuração do ambiente de testes e tratamento de erros específicos das aplicações avaliadas. Embora essas modificações tenham sido registradas e documentadas ao longo do estudo, existe a possibilidade de que decisões tomadas durante os ajustes tenham influenciado os resultados finais da comparação.

Além disso, os portais avaliados são aplicações reais e sujeitas a alterações em sua interface, estrutura HTML e comportamento durante o período da pesquisa. Mudanças realizadas pelos mantenedores dos sistemas poderiam impactar a execução dos testes automatizados e exigir adaptações adicionais não relacionadas diretamente à qualidade dos scripts gerados pela LLM.

Por fim, a avaliação dos critérios de legibilidade, facilidade de manutenção e necessidade de ajustes manuais possui um componente qualitativo dependente da análise da autora. Apesar da utilização de critérios previamente definidos, parte dessa avaliação está sujeita à interpretação pessoal.

6.2 Validade Externa

A validade externa está relacionada à capacidade de generalização dos resultados para outros contextos.

Este estudo foi conduzido utilizando exclusivamente o modelo Gemini 3 Flash (Tier Pago), o que limita a extrapolação dos resultados para outras LLMs, como GPT, Claude ou modelos open source. Diferentes modelos podem apresentar comportamentos distintos na geração de scripts automatizados, mesmo quando submetidos aos mesmos prompts.

Outra limitação estão relacionadas ao conjunto de aplicações avaliadas. Os experimentos foram realizados apenas em duas plataformas governamentais do Estado de Pernambuco, ambas voltadas à disponibilização de informações públicas e possuindo recursos específicos de acessibilidade. Dessa forma, os resultados podem não refletir o comportamento das LLMs em aplicações de outros domínios, como comércio eletrônico, sistemas bancários ou aplicações móveis.

Além disso, os cenários se concetram em funcionalidades específicas de acessibilidade, como aumento e diminuição de fonte, escala de cinza e contraste negativo. Outros tipos de funcionalidades, requisitos de acessibilidade ou fluxos de negócio mais complexos podem apresentar desafios diferentes para a geração automática de scripts, como é o caso do problema com Iframe citado na Subseção 4.2.1.

Por fim, a pesquisa utilizou uma stack tecnológica específica composta por Cypress, JavaScript, Cucumber, Gherkin e o padrão Page Object Model. Dessa forma, os resultados observados podem variar quando aplicados a outros frameworks de automação, linguagens de programação ou arquiteturas de teste.

Visando minimizar essas limitações e garantir a transparência, auditabilidade e reprodutibilidade desta pesquisa, todo o material técnico foi disponibilizado publicamente em um repositório no GitHub[1]. Nessa plataforma, encontram-se a implementação do código na íntegra, os scripts de automação gerados e de referência, permitindo que o estudo seja reproduzido e expandido para outros cenários e modelos.

[1] Disponível em: <https://github.com/gabdemedeiros/TCC_BSI>

7 Considerações Finais

Este trabalho teve como objetivo analisar a efetividade da Engenharia de Prompt na geração de scripts de testes automatizados de acessibilidade utilizando o modelo Gemini 3 Flash (Tier Pago), considerando funcionalidades presentes no Portal da Transparência de Pernambuco e no Portal do Governo de Pernambuco. Para isso, foram elaborados scripts de referência desenvolvidos manualmente e, posteriormente, comparados com scripts gerados pela LLM a partir do refinamento de prompts.

Os resultados obtidos demonstraram que a Engenharia de Prompt exerceu influência direta na qualidade dos artefatos produzidos pela LLM. À medida que os prompts foram refinados e passaram a fornecer mais contexto sobre o sistema e os seletores, foi possível observar uma evolução no resultado dos scripts gerados pela LLM.

Entretanto, a pesquisa também demonstrou que a geração de scripts pelas LLMs ainda apresenta limitações. Em diferentes momentos foram identificados seletores incorretos, validações incompatíveis com o comportamento real das aplicações, ausência de informações relevantes para execução dos testes e necessidade de ajustes manuais para adequação dos scripts ao contexto dos portais analisados. Além disso, características específicas das aplicações, como restrições relacionadas ao uso de iframes, exigiram adaptações nos cenários inicialmente planejados, evidenciando que o conhecimento técnico do profissional de QA continua sendo essencial durante o processo de desenvolvimento dos testes.

Como principal contribuição, este estudo apresenta uma análise prática da aplicação da Engenharia de Prompt na geração de testes automatizados de acessibilidade em plataformas governamentais, fornecendo evidências sobre os benefícios e limitações da utilização de LLMs nesse contexto. Os resultados indicam que essas ferramentas podem atuar como apoio às atividades de automação, reduzindo parte do esforço inicial de implementação e acelerando a construção de scripts, mas não substituem a necessidade de validação humana.

7.1 Trabalhos Futuros

A partir das contribuições geradas por esta pesquisa e também levando em consideração as limitações operacionais identificadas ao longo das iterações, para os trabalhos futuros, sugere-se o aprofundamento do estudo para outras LLMs, como GPT e Claude, visando comparar o impacto de diferentes modelos na geração de scripts de testes automatizados.

Além disso, recomenda-se o aprofundamento da pesquisa no mapeamento e na superação de barreiras dos portais governamentais. Esse esforço deve focar na geração de prompts robustos o suficiente para lidar com restrições dos iframes, para possibilitar a validação automatizada dos componentes de acessibilidade digital dos Portais Governamentais.

Outra proposta é a investigação de diferentes técnicas de Engenharia de Prompt, buscando identificar estratégias de contextualização que se mostrem mais eficientes para reduzir a ocorrência de alucinações e mitigar a necessidade de ajustes manuais por parte do analista, elevando a aderência dos scripts gerados aos requisitos de teste pré-estabelecidos.

8 Uso de Inteligência Artificial no Trabalho

Este trabalho utilizou ferramentas de inteligência artificial generativa como recurso de apoio à pesquisa e à escrita acadêmica. A ferramenta Gemini foi utilizada para as seguintes finalidades:

- Geração dos scripts automatizados utilizados posteriormente na análise comparativa com os scripts automatizados desenvolvidos manualmente;
- Tradução de artigos científicos do idioma inglês para o português, com o objetivo de auxiliar na compreensão do conteúdo;
- Apoio na adequação de trechos textuais à sintaxe LaTeX;
- Revisão linguística e gramatical do texto em português e do abstract em inglês.

A autora assume responsabilidade pelo conteúdo apresentado neste trabalho, tendo revisado, validado, adaptado e complementado todas as informações produzidas pelas ferramentas de inteligência artificial. O uso dessas ferramentas ocorreu exclusivamente como apoio ao processo de pesquisa e escrita, não substituindo a análise crítica, a interpretação dos resultados, a condução do experimento ou a elaboração das conclusões científicas apresentadas.

Referências

ANDRADE, L. M. de et al. Testes de software em caixa de silício: Uma nova abordagem baseada em inteligência artificial. *ARACÊ*, v. 7, n. 6, p. 35011–35037, 2025. Citado na página 10.

BARREIROS, L. et al. Visual studio code e produtividade no desenvolvimento de software. *Revista de Engenharia de Software*, 2025. Citado na página 19.

BARUFFALDI, S. et al. *Identifying and measuring developments in artificial intelligence: making the impossible possible*. Paris: OECD, 2020. Acesso em: 22 out. 2025. Disponível em: <https://www.oecd.org/en/publications/identifying-and-measuring-developments-in-artificial-intelligence_5f65ff7e-en.html>. Citado na página 9.

BERNARDO, P. C.; KON, F. A importância dos testes automatizados. *Engenharia de Software Magazine*, v. 1, n. 3, p. 54–57, 2008. Citado na página 9.

CABROL, M.; ÁVALOS, R. S. *Quem tem medo da inteligência?: Possibilidades e riscos da inteligência artificial no Estado digital*. Washington: BID, 2021. Acesso em: 22 out. 2025. Disponível em: <http://dx.doi.org/10.18235/0003012>. Citado na página 9.

CARVALHO, V. F. d.; CAGNIN, M. I.; PAIVA, D. M. B. *Avaliação de Acessibilidade de Web Sites de Governos Estaduais do Brasil*. 2025. Citado 2 vezes nas páginas 9 e 27.

COHN, M. *Succeeding with Agile: Software Development Using Scrum*. [S.l.]: Addison-Wesley, 2009. Citado na página 17.

Cucumber. *Cucumber Documentation*. 2024. Acesso em: 12 abr. 2026. Disponível em: <https://cucumber.io/docs>. Citado na página 22.

Cypress Documentation. *Cross-origin Testing*. 2026. Acesso em: 12 abr. 2026. Disponível em: <https://docs.cypress.io/app/guides/cross-origin-testing#cross-origin-iframes>. Citado na página 41.

DIAS, J. D. C. *Teste de Software com IA: Um Mapeamento Sistemático da Literatura*. 2023. Acesso em: 22 out. 2025. Citado na página 9.

ELIAS, R. *Gherkin e Cucumber — mapeando os testes automatizados*. 2023. Acesso em: 12 abr. 2026. Disponível em: <https://medium.com/opanehtech/gherkin-e-cucumber-mapeando-os-testes-automatizados-53232bf26e79>. Citado na página 20.

FAUSTINO, N. d. O. *Processo automatização de teste para aplicações WEB: estudo de caso*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, 2025. Acesso em: 19 abr. 2026. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/45852/1/ProcessoAutomatiza%c3%a7%c3%a3oTeste.pdf>. Citado na página 22.

Google. *Gemini: visão geral*. 2025. Acesso em: 12 abr. 2026. Disponível em: <https://gemini.google/br/overview/?hl=pt-BR>. Citado na página 24.

IGNáCIO, A. C.; OLIVEIRA, L. d. S.; FRANCEZ, M. P. M. Eficiência do uso da inteligência artificial no desenvolvimento de software. *Advances in Global Innovation & Technology*, v. 2, n. 2, p. 06–16, 2024. Disponível em: <https://doi.org/10.29327/2384439.2.2-1>. Citado na página 10.

INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD. *Syllabus: ISTQB Certified Tester Foundation Level (CTFL) – versão 4.0*. Bruxelas, 2025. Acesso em: 03 jul. 2025. Disponível em: <https://www.istqb.org/downloads/send/2-foundation-level-documents/368-ctfl_syllabus_v4-0.html>. Citado 5 vezes nas páginas 13, 14, 15, 16 e 17.

JUNIOR, J. L. et al. *A Case Study on Test Case Construction with Large Language Models: Unveiling Practical Insights and Challenges*. 2023. Acesso em: 18 fev. 2026. Disponível em: <https://sol.sbc.org.br/index.php/cibse/article/view/28465/28275>. Citado 7 vezes nas páginas 9, 10, 30, 33, 34, 35 e 64.

KHETARPAL, R. Modern web testing with cypress. *QA Engineering Review*, 2021. Citado na página 19.

LI, Y. et al. Large-scale, independent and comprehensive study of the power of llms for test case generation. *arXiv preprint arXiv:2407.00225v3*, 2025. Citado na página 10.

MACHADO, R. Introdução ao cypress. *DevMedia*, 2017. Citado na página 19.

MAIA, e. a. Aplicações de inteligência artificial em testes de software. *Revista ou Evento não informado*, 2023. Citado 2 vezes nas páginas 26 e 27.

Microsoft. *Visual Studio Code Documentation*. 2025. Acesso em: 12 abr. 2026. Disponível em: <https://code.visualstudio.com/docs>. Citado na página 19.

MUGHAL, A. H. *Advancing BDD Software Testing: Dynamic Scenario Re-usability and Step Auto-complete for Cucumber Framework*. 2024. Acesso em: 12 abr. 2026. Disponível em: <https://arxiv.org/abs/2402.15928>. Citado na página 20.

MULLER, G. *O que é Teste de Software? Por que é necessário?* 2020. <https://cwi.com.br/blog/o-que-e-teste-de-software-por-que-e-necessario/>. Acesso em: 01 jun. 2025. Citado na página 9.

MWAURA, J. Understanding cypress and dom manipulation. *International Journal of Software Engineering*, 2021. Citado na página 19.

NASCIMENTO, J. R. d. *Exploração de técnicas de engenharia de prompt para aprimorar os resultados do uso de LLM no TCMRio*. Dissertação (Mestrado) — Instituição não informada, 2024. Citado 2 vezes nas páginas 24 e 25.

NASCIMENTO, J. R. d. *Exploração de técnicas de engenharia de prompt para aprimorar os resultados do uso de LLM no TCMRio*. [S.l.]: Instituição não informada, 2025. Citado na página 25.

NUNES, E. H. D. C. et al. *Diagnóstico da Acessibilidade Digital em Serviços Públicos Prioritários usando a Lei de Acesso à Informação*. 2025. Citado na página 27.

OLIVEIRA, M. L. d. S. *Um estudo sobre a aplicação de LLMs no teste de software*. Dissertação (Mestrado) — Instituição não informada, 2025. Citado 8 vezes nas páginas 26, 29, 30, 33, 34, 35, 47 e 64.

PEIXOTO, M. D. L. C. et al. Turning manual tasks into actions: Assessing the effectiveness of gemini-generated selenium tests. In: *Proceedings of the 2nd ACM/IEEE International Conference on AI Software (AIware)*. [S.l.: s.n.], 2025. Citado 7 vezes nas páginas 10, 24, 31, 33, 34, 35 e 65.

SOUSA, A. S. *Teste Baseado em Modelo em Aplicativos Móveis: Uma Avaliação com as Ferramentas GraphWalker e Appium*. Dissertação (Mestrado) — Universidade Federal de São Carlos, São Carlos, 2025. Acesso em: 19 abr. 2026. Disponível em: <https://repositorio.ufscar.br/server/api/core/bitstreams/a830250b-8c52-45c1-9945-3369cfdf43c3/content>. Citado na página 22.

SOUSA, H. N. F. d. et al. *Um experimento comparativo da eficácia de diferentes LLM na geração de cenários Gherkin*. 2025. Citado na página 20.

SOUSA, R. E. d. *Um estudo comparativo entre ferramentas de automação de testes: selenium e cypress*. 2023. Acesso em: 01 jun. 2025. Citado na página 9.

TAGAWA, T. Y. D.; AGUIAR, I. C.; VILLANI, L. Eficácia e impacto na experiência do cliente: um estudo sobre os modelos generativos chatgpt e gemini. *Revista Processando o Saber*, v. 17, p. 197–211, 2025. Citado na página 24.

TAKY, A. Cypress testing framework overview. *Software Testing Journal*, 2021. Citado na página 19.

WANG, G. et al. Do advanced language models eliminate the need for prompt engineering in software engineering? *ACM Transactions on Software Engineering and Methodology*, 2025. Accepted for publication. Citado 2 vezes nas páginas 25 e 26.

YIN, R. K. *Case Study Research and Applications: Design and Methods*. 6. ed. Thousand Oaks: SAGE Publications, 2017. Citado na página 36.